3. Nieco o danych

Zakładam, że już z grubsza wiemy, jak przygotować program do uruchomienia pod DOSem (kompilacja, linkowanie, itp.). Teraz przyszła pora na kilka słów o obsłudze danych. A dokładnie o ich wgrywaniu, trzymaniu w programie, obsłudze po kompilacji.

Zacznijmy od czegoś prostego – idealnym przykładem jest tu zestaw fontów. Zajmuje on zawsze (w standardzie) 1 kilobajt pamięci, jego umiejscowienie też nie jest bez znaczenia. W TB XL powinien zawsze znajdować się na początku strony pamięci, a właściwie na początku takiej strony, której adres w pamięci jest podzielny przez cztery. Dlaczego właśnie tak? Przyjrzyjmy się nieco bliżej. Zestaw fontów to 1024 bajty, jedna strona pamięci to 256 bajtów, stąd jeden zestaw zajmuje cztery strony pamięci.

Kolejna rzecz – to zmiana zestawu znaków w programie. Wykonujemy ją jedną prostą instrukcją: ***POKE 756,adres_zestawu_znaków***. Wiadomo, że instrukcja POKE pozwala nam na wprowadzanie jednej wartości ośmiobitowej, to jest liczby z zakresu 0-255. Wskazuje on nam początek strony, na której znajduje się nasz zestaw znaków.

Przyjrzyjmy się bliżej – standardowy zestaw znaków znajduje się na 224 stronie pamięci (POKE 756,224) – czyli pod adresem 57344. Trudny adres do zapamiętania, nieprawdaż? Ale wykonajmy jedną instrukcję pod TB XL:

```
?HEX$(57344)
```

Otrzymamy wynik E000. Jest to nasz adres w postaci szesnastkowej. Jeśli teraz zamienicie dwie pierwsze cyfry z tego adresu na system dziesiętny to uzyskacie zaskakujący wynik:

```
?$E0
```

Jest to bowiem 224, czyli to co wpisujemy pod POKE. Sprawa staje się prostsza do zrozumienia i zapamiętania. Ja w swoich programach ustaliłem sobie nieomalże stały adres na fonty – większość z nich mieści się pod adresem $9C00 (mała podpowiedź: POKE 756,$9C – tak, w TB XL dane możemy podawać szesnastkowo, poprzedzając je znakiem dolara). Przykłady w tym artykule dotyczące zestawu znaków postaram się podawać zawsze ustawione pod ten adres pamięci.

Dobrze, załóżmy, że przygotowaliśmy sobie zestaw znaków pod dowolnym edytorem fontów. Przydałoby się go użyć, czyli wprowadzić do pamięci komputera. Można to zrobić na kilka sposobów, podam trzy najczęściej spotykane (przynajmniej tak mi się wydaje). Wszelkie wczytywanie danych (za wyjątkiem danych poprzedzonych nagłówkiem binarnym) wymagają

otwarcia odpowiedniego kanału do ich obsługi – a dobra praktyka programistyczna nakazuje na koniec nieużywany kanał zamknąć. Przyjrzyjmy się dwu poniższym przykładom (zakładamy, że plik z fontami jest nazwany FONT.FNT i jest w stacji D:):

Styl ATARI BASIC:

```
OPEN #1,4,0,"D:FONT.FNT":FOR I=0 TO
1023:GET #1,A:POKE $9C00+I,A:NEXT
I:CLOSE #1
```

UWAGA! Pamiętamy, że gdy chcemy to wczytać w Atari Basicu adres szesnastkowy zmieniamy na dziesiętny!

Styl Turbo Basic XL:

```
OPEN #1,4,0,"D2:*FNT":BGET
#1,$9C00,1024:CLOSE #1
```

Proponuję wypróbować oba. Wykonują dokładnie to samo, jak widać instrukcja ***BGET*** w TB XL nie dość, że jest wydajniejsza (szybsza) od zwykłego ***GET***, to jeszcze nie wymaga użycia ani pętli (ilość bajtów podajemy wprost), ani zmiennej pośredniej (A).

Pokrótce, co się dzieje:

OPEN #1,4,0,"D:FONT.FNT" – otwiera nam kanał #1 na operacje wejścia/wyjścia (tu wejście)

FOR I=0 TO 1023:GET #1,A:POKE $9C00+I,A:NEXT I lub ***BGET #1,$9C00,1024*** – pobiera z wejścia (stacja D:) 1024 bajty danych i wpisuje je pod adres $9C00

CLOSE #1 – zamyka pierwszy kanał wejścia/wyjścia

Po dokładny opis instrukcji zapraszam do literatury. Tak wczytany zestaw danych możemy już wykorzystać (***POKE 756,$9C***). Ale wcześniej wspomniałem o trzech metodach, a podałem dwie, o co chodzi?

Trzecia metoda jest pozornie trudniejsza, ale ja ją preferuję ze względu na późniejszą łatwość dołączania danych do kompilacji. Jeśli chcę mieć (finalnie) program w jednym bloku, skompilowany, wczytywany pod DOSem – wskazane by było, aby wiedział, skąd brać niezbędne dane (a takimi są fonty). A przykro by było, aby do prawidłowego działania program musiał je wczytywać po uruchomieniu (potrzebny DOS lub jego odpowiednik, wszystkie pojedyncze pliki należy trzymać razem, itp.).

Do czego zmierzam? Jest taka użyteczna instrukcja ***BLOAD***, która wczytuje dane binarne, koniecznie poprzedzone nagłówkiem. Nasze fonty

oczywiście takiego nie mają, ale w poprzednim artykule wspominałem o programach dodatkowych. Skorzystajmy z Super Packera, po jego załadowaniu wczytajmy nasze fonty do jego bufora (opcja F, ustalamy adres na 9C00, zapisujemy dane – dane binarne preferuję z rozszerzeniem *.DAT – więc ja podaję FONTY.DAT). Mamy w ten sposób przygotowany plik binarny do wgrania w programie.

Uwaga! Za pomocą Super Packera możemy w ten sposób przygotować sobie kilka bloków danych, na przykład grafikę, tekst, itp., a następnie zapisać to w jednym pliku.

Wróćmy do naszych fontów. W TB XL wpisujemy:

```
BLOAD"D:FONTY.DAT"
```

Teraz spróbujmy: ***POKE 756,$9C***. Hura, jest!

Jak widać, po odpowiednim przygotowaniu danych jest to bardzo proste. Oczywiście metoda nie ogranicza się do fontów, a zasadniczo do wszystkich danych zapisanych binarnie – ba, część programów od razu tworzy takie gotowe pliki (na przykład Chaos Music Composer – zarówno muzykę, jak i player wczytacie w ten sposób). Bardzo często w ten sposób przygotowuję sobie grafikę (bądź w postaci pełnoekranowej, bądź w postaci bloków pamięci). Przy odrobinie wprawy jest to dużo wygodniejsze (i szybsze) niż analogiczne wczytywanie danych instrukcjami TB XL (zapominamy o OPEN, CLOSE, ilości bajtów do wczytania podczas transmisji danych, itp., itd.). Dodatkowo – wszystkie potrzebne dane możemy zamknąć w jednym pliku (jedno BLOAD zamiast kilku/kilkunastu OPEN, BGET). A przy pisaniu większych programów jet to wielka oszczędność (czasu i miejsca).

Szykując kolejne dane należy pamiętać o jednej głównej zasadzie – nie mogą one nachodzić na siebie, bo będą się wzajemnie zamazywać (w zależności od położenia w pliku docelowym). Zdradliwym jest też fakt, że niekiedy przed kompilacją dane muszą być w innym miejscu niż po niej – ale to się zdarza zazwyczaj w większych programach.

Wracając do wczytywania i zapisu danych – proponuję zapoznać się z literaturą na ten temat, gdyż jest on zbyt obszerny na potrzeby tego artykułu. Zasadniczo tworząc programy plikowe trzymam się zasady, że wszystko wczytuję poza listingiem, bezpośrednio z edytora (pomijam numery linii), lub staram się takie rzeczy trzymać w oddzielnej procedurze, którą potem usuwam. O samych procedurach i więcej o danych będę jeszcze wspominał w kolejnych częściach tego opracowania.

Na zakończenie przypomnę tylko, jak w TB XL uruchamiać muzyki z CMC, skoro już o tym programie wspomniałem:

```
X=USR($adres_playera,nr_subsongu,
$adres_muzyki)
```

oraz jak wyciszyć muzykę:

```
X=USR($adres_playera)
```

Oczywiście jest to odwołanie do procedury maszynowej, przez co mamy podany przykład jej wywołania. Jak pewnie spostrzegawczy zauważyli – adresy są tu podane szesnastkowo, jednak nic nie stoi na przeszkodzie, aby je podawać dziesiętnie. Tylko czy jest sens – wywołanie takie jak tutaj nie wymaga konwersji liczb, co jest zaletą TB XL, a jest dokładnie w taki sposób ustawiane przez program.