
BASIC XE
ATARI

BASIC XE

CO NOWEGO W BASIC-u?

Opracowanie niniejsze stanowi skrótowy opis języka programowania BASIC XE dla komputerów ATARI XL/XE.

Opisano tu nowe w stosunku do standardowego BASIC-a instrukcje.

SPIS TREŚCI

Poszczególne instrukcje i sposób ich użycia opisano na stronach wg poniższej tabeli:

BASIC XE - WADOMOŚCI PODSTAWOWE

BASIC XE zawiera oczywiście wszystkie instrukcje dostępne w języku Atari BASIC, w który każdy mikrokomputer Atari jest wyposażony. Poniżej podanych zostało kilka dodatkowych oferowanych przez BASIC XE możliwości:

Szybsze wykonanie programu.

Nowe procedury operacji zmiennopozycyjnych w połączeniu z instrukcją FAST zapewniają bardzo szybkie wykonanie programu. BASIC XE jest ok. dwa do sześciu razy szybszy niż standardowy Atari BASIC, ponad dwukrotnie szybszy niż BASIC dla Commodore 64 i dla IBM P(Junior.

Możliwość wykorzystania całej pamięci 130XE.

BASIC XE pozwala na pełne wykorzystanie całej dostępnej w Atari 130XE pamięci.

Udoskonalony edytor.

Przy wykorzystaniu interpretera BASIC XE nie ma znaczenia, jakimi literami pisane są instrukcje: małymi, dużymi, czy też w grafice odwrotnej (Inverse Video). Ponadto do dyspozycji są instrukcje automatycznego numerowania linii programu w czasie jego pisania oraz przenumerowywania programu (wraz z przenumerowaniem odpowiednich odwołań do linii). Przy wykonaniu instrukcji LIST program przedstawiony jest z uwidocznieniem jego struktury - pętli, podprogramów itp.

Poszerzone możliwości operacji na tekstach.

BASIC XE uwalnia programistę od każdorazowej konieczności określania maksymalnej długości zmiennej tekstowej instrukcją DIM. Ponadto daje nowe możliwości operowania na tekstach, w tym analogiczne jak dostępne w Microsoft BASIC. Umożliwia to szybkie i efektywne zaprogramowanie operacji przetwarzania tekstów.

Instrukcje do operowania grafiką Player/Missile.

BASIC XE posiada zbiór instrukcji umożliwiających proste wykorzystanie możliwości animacji ruchomych obiektów P/M (ang. Player/Missile Graphics) co pozwala na tworzenie interesujących gier bez konieczności znajomości mapy pamięci i systemu operacyjnego Atari.

Prosty dostęp do portów manipulatorów.

Specjalne instrukcje pozwalają na szybki i efektywny odczyt położenia manipulatorów typu paddle i joystick podłączonych do odpowiednich portów komputera.

Poszerzone komunikaty o błędach.

W wypadku wystąpienia błędu wykonania BASIC XE oprócz podania numeru błędu podaje ponadto krótki opis błędu, który został wykryty.

Jak uruchomić BASIC XE

Przystąpienie do pracy przy użyciu interpretera języka BASIC XE jest bardzo proste. Wystarczy ustawić w odpowiednim położeniu przełącznik BASIC XE / Atari BASIC (jeżeli BASIC XE został wmontowany do wnętrza komputera) lub włożyć kartridż[1] i następnie włączyć komputer. Pełne wykorzystanie możliwości dawanych przez BASIC XE możliwe jest wyłącznie przy współpracy z pamięcią zewnętrzną (stacją dysków lub magnetofonem). W takim wypadku należy umieścić kasetę w magnetofonie lub dyskietkę BASIC XE w stacji dyskietek o numerze logicznym 1 i włączyć komputer. Jeżeli system nie jest wyposażony w stację dyskietek lub magnetofon, BASIC XE może być wykorzystywany, ale bez poniższych instrukcji:

BSAVE, (ALL, DEL, EXIT, FAST, LOCAL, LVAR, MOVE PRO(EDURE, RENUM, ROET, RPUT, SORTUP, SORTDOWN, szybszych procedur arytmetycznych i instrukcji umożliwiających wykorzystanie grafiki P/M (nie dotyczy HITCLR).

Uwaga: BASIC XE nie działa poprawnie z żadnym tzw. TRANSLATOREM.

Słowa kluczowe i symbole w języku BASIC XE

Poniżej podano listę słów kluczowych i symboli wykorzystywanych przez BASIC XE. Pismem pogrubionym wyróżniono nowe w stosunku do Atari BASIC instrukcje:

ABS	ADR	AND	ASC	ATN	**BGET**
BLOAD	**BPUT**	**BSAVE**	**BUMP**	BYE	**CALL**
CHR$	CLOAD	CLOG	CLOSE	CLR	COLOR
CONT	COS	**CP**	CSAVE	DATA	DEG
DEL	**DIM**	**DIR**	DOS	**DPEEK**	**DPOKE**
DRAWTO	**ELSE**	END	**ENDIF**	**ENDWHILE**	ENTER
ERASE	**ERR**	**EXIT**	EXP	**EXTEND**	**FAST**
FIND	FOR	FRE	GET	GOSUB	GOTO
GRAPHICS	**HEX$**	**HITCLR**	**HSTICK**	IF	INPUT
INT	**INVERSE**	**LEFT$**	LEN	LET	LIST
LOAD	**LOCAL**	LOCATE	LOG	**LOMEM**	LPRINT
LVAR	**MID$**	**MISSILE**	**MOVE**	NEW	NEXT
NORMAL	NOT	NOTE	**NUM**	ON	OPEN
OR	PADDLE	PEEK	**PEN**	PLOT	**PMADR**
PMCLR	**PMCOLOR**	**PMGRAPHICS**		**PMMOVE**	**PMWIDTH**
POINT	POKE	POP	POSITION	**PROCEDURE**	
PROTECT	PTRIG	RAD	**RANDOM**	READ	REM
RENAME	**RENUM**	RESTORE	RETURN	**RGET**	**RIGHT$**

[1] Patrz dodatek A.

RND	**RPUT**	RUN	SAVE	**SET**	
SETCOLOR	SGN	SIN	**SORTDOWN**	**SORTUP**	SOUND
SQR	STATUS	STEP	STICK	STOP	STR$
STRIG	**SYS**	**TAB**	THEN	TO	**TRACE**
TRACEOFF	TRAP	**UNPROTECT**	**USING**	USR	VAL
VSTICK	**WHILE**	XIO			

!	"	#	$	%	&	(	)	w	/	+	-	.
<	>	<=	>=	<>	=	^	;	:				

Uwagi o przyjętej notacji i skrótach

Nawiasy kwadratowe w opisie formatu funkcji i instrukcji zawierają pozycje, które mogą, lecz nie muszą być użyte. Jeżeli w nawiasie kwadratowym występują trzy kropki, np. [zm1....] oznacza to, że dana pozycja może wystąpić wielokrotnie.

Jeżeli w zapisie formatu funkcji lub instrukcji występuje kilka pozycji podanych pomiędzy pionowymi kreskami oznacza to, że należy wybrać jedną z nich.

Jeżeli dany podrozdział opisuje funkcję, a nie instrukcję, zostało to zaznaczone w tytule podrozdziału.

Zamiast wpisywania całych nazw niektórych instrukcji BASIC XE dopuszcza stosowanie skrótów. W takim wypadku dopuszczalne skróty podano w nawiasach okrągłych w tytule podrozdziału opisującego daną instrukcję.

W opisie formatów funkcji i instrukcji a także niekiedy w tekście opisu posłużono się skrótami. Poniżej podane zostało znaczenie stosowanych skrótów:

zm - zmienna (numeryczna lub tekstowa

wyr_num - wyrażenie numeryczne

zm_num - zmienna numeryczna

wyr_tekst. - wyrażenie tekstowe

zm_tekst - zmienna tekstowa

nr_linii - numer linii

spec _zbioru - specyfikacja (opis) zbioru

par – parametr

Opis znaczenia powyższych terminów występuje w tekście tam, gdzie zostały one użyte i zdefiniowane.

Zostaną teraz opisane instrukcje BASIC XE, których nie ma wśród instrukcji ATARI BASIC lub są, ale ich działanie zostało rozszerzone.

DIM

Format: DIM { tablica_numeryczna (wyr_num1,[wyr_num2]) | zmienna_tekstowa (wyr_num) | tablica tekstowa (wyr_numl,wyr_num2) }[,..]

Instrukcja DIM jest używana do deklarowania objętości tablic numerycznych, zmiennych tekstowych i tablic tekstowych.

Dla tablic numerycznych instrukcja DIM rezerwuje obszar dla wyr_num1+1 jeżeli tablica jest jednowymiarowa, lub wyr_num1+1 wierszy, każdy po wyr_num2+1 elementów. Plus jeden, ponieważ indeksy dla tablic numerycznych zaczynają się od zera.

Dla zmiennych tekstowych DIM rezerwuje obszar dla co najwyżej wyr_num znaków. Dla tablic tekstowych DIM rezerwuje miejsce dla wyr_num1 tekstów o długości wyr_num2 znaków każdy.

Poniższy przykład pokazuje użycie instrukcji DIM. W linii 100 zostaje zadeklarowana tablica numeryczna o 101 elementach nazwana A1. W linii 110 zostaje zadeklarowana tablica numeryczna zawierająca 7 wierszy po 4 kolumny nazwana Rozwiązanie. W linii 120 zadeklarowana została zmienna tekstowa o długości maksymalnie 25 znaków nazwana Imie$ i tablica tekstowa zawierająca 100 linii tekstu po 40 znaków nazwana Miasta$

```
100 Dim A1(100)
110 Dim Rozwiązanie(6,3)
120 Dim Imie$(25),Miasta$(100,40)
```

Uwaga: BASIC XE może automatycznie zarezerwować przestrzeń dla zmiennych tekstowych. Informacje na ten temat podane są w opisie działania instrukcji SET 11,wyr_num

LOCAL

Format: LOCAL zm_num1 [,zm_num2,...]

Przykład:

```
100 Local Temperatura
110 Local Suma, X, Y
```

Instrukcja LOCAL pozwala na pisanie podprogramów w sposób uniwersalny i umożliwia łatwe wykorzystanie ich w przyszłości z dowolnym programem. W stowarzyszeniu z instrukcjami GOSUB i PRO(EDURE instrukcja LOCAL pozwala na definiowanie lokalnych dla podprogramu lub procedury (wewnętrznych) zmiennych numerycznych. Po napotkaniu instrukcji LOCAL zmienne numeryczne podane w licie instrukcji traktowane są jako lokalne aż do napotkania instrukcji EXIT. (o to znaczy: zmienne stają się lokalne? Oznacza to, że zmiana wartości zmiennej numerycznej podanej w instrukcji LOCAL zmienia się tylko w granicach między LOCAL i EXIT nie powodując zmiany poza tymi instrukcjami. Używając instrukcji LOCAL można

uniknąć konfliktów (zwanych niekiedy efektami ubocznymi) pomiędzy podprogramami w programie używającym zmiennych o tych samych nazwach.

Poniższy prosty przykład pokazuje działanie instrukcji LOCAL:

```
10 Test=1234567: Print 10,Test
20 Gosub 40: Print 20,Test
30 End
40 Local Test: Print 40,Test
50 Test=0.54321: Print 50,Test
60 Exit
```

Instrukcje PRINT zawarte w programie wyświetlą na ekranie monitora numer linii, w której były wykonane i wartość zmiennej Test. Pozwoli to w prosty sposób prześledzić jego działanie. Efektem działania programu jest wyświetlenie na ekranie następujących informacji:

```
10 1234567 40 1234567 50 0.54321 20 1234567
```

Linia 10 przykładowego programu zawiera przypisanie wartości zmiennej, następnie wartość ta jest wyświetlana. Linia 20 to skok do podprogramu o początku w linii 40, a następnie po powrocie z podprogramu wyświetlenie wartości zmiennej Test. Linia 40 to początek interesującej części programu. Linia ta zawiera deklarację zmiennej Test jako zmiennej lokalnej, następnie wyświetlana jest wartość zmiennej Test. Linia 50 to przypisanie zmiennej Test nowej wartości a następnie wyświetlenie wartości zmiennej Test. Ale teraz jest to zmienna lokalna. Linia 60 zawiera instrukcję EXIT, co powoduje przywrócenie zmiennej Test jej pierwotnej wartości, tj. takiej jaką miała w momencie napotkania instrukcji LOCAL i powrót z podprogramu, analogicznie jak instrukcja RETURN.

Zastosowanie instrukcji POP i LOCAL umożliwia korzystanie ze zmiennych lokalnych nie tylko w podprogramie. Nie wpływa to jednak korzystnie na czytelność programu.

Uwagi:

1. Instrukcja LOCAL może być używana z podprogramami wywoływanymi instrukcją GOSUB lub (ALL (typu PRO(EDURE). Pozwala to w prosty sposób przystosować istniejące programy do wykorzystania zmiennych lokalnych. Ponadto wywołanie podprogramu instrukcją GOSUB jest szybsze, niż przy użyciu (ALL, jeżeli program działa w trybie FAST.
2. Zmienne nie zmieniają wartości poza nawiasem instrukcji LOCAL i EXIT, co można zobaczyć w poprzednim przykładzie. Wypisana wartość zmiennej Test w linii 40 wynosi 1234ó67, mimo że tej zmiennej nadano lokalnie inną wartość. Użycie instrukcji LOCAL nie powoduje wyzerowania zmiennych zadeklarowanych jako lokalne. Można je wyzerować przypisując im wartość zero.

3. Używanie w podprogramach tych samych nazw zmiennych zadeklarowanych jako lokalne pozwala "ominąć" ograniczenie stosowania co najwyżej 128 zmiennych o różnych nazwach w programie. Dla przykładu mamy w programie 10 podprogramów z których każdy wymaga 10 zmiennych, co daje łącznie 100 zmiennych. Nazywając zmienne we wszystkich podprogramach tak samo i stosując instrukcję LOCAL mamy tylko 10 różnych zmiennych w programie.
4. Instrukcja LOCAL powoduje wysłanie wartości zmiennej numerycznej na stos BASIC-a XE. Wykonanie instrukcji EXIT powoduje pobranie ze stosu wartości zmiennych numerycznych i przywrócenie im pierwotnej wartości (tj. takiej, jak przed wykonaniem instrukcji LOCAL).
5. Instrukcja LOCAL może być stosowana tylko w podprogramach kończących się instrukcją EXIT a nie RETURN, za wyjątkiem pobrania instrukcją POP poprzednich wartości zmiennych ze stosu przed instrukcją RETURN. Więcej informacji na ten temat znaleźć można w opisie instrukcji POP.

NUM

Format: NUM [start][,krok]

Przykłady:

```
NUM
NUM 50
NUM ,1
NUM 50,1
```

Instrukcja NUM umożliwia automatyczne numerowanie linii przy pisaniu programu w języku BASIC XE. Pozwala to na szybsze wpisywanie programu i umożliwia skupienie się na właściwej treści programu, ponieważ BASIC XE po wpisaniu każdej linii i zaakceptowaniu jej klawiszem RETURN wypisuje na ekranie numer nastopnej linii i umieszcza kursor na pozycji początku tekstu programu w linii. Jeżeli nie zostanie podany numer początkowy (start) i kroku numeracji (krok) (pierwsza linia w przykładach), to BASIC XE rozpocznie numerację od numeru ostatniej linii programu znajdującego się aktualnie w pamięci powiększonego 0 10, zwiększając numer o 10 po wprowadzeniu każdej linii. Jeżeli aktualnie w pamięci programu nie było, to numerowanie rozpocznie się w takim wypadku od linii o numerze 10. Jeżeli numer początkowy (start) został podany, natomiast krok numeracji (krok) nie został podany (druga linia w przykładach) BASIC XE rozpocznie numerowanie od linii o podanym numerze i będzie zwiększał następnie numery o 10. Jeżeli krok numeracji (krok) został podany (trzecia linia w przykładach), natomiast numer początkowy (start) nie został podany to numeracja rozpocznie się od linii o numerze większym od numeru ostatniej linii aktualnie znajdującego się w pamięci programu powiększonego o (krok). Jeżeli zarówno (start) jak i

(krok) zostały podane (ostatnia linia w przykładach) numerowanie rozpocznie się od linii o numerze (start) i za każdym wprowadzeniem nowej linii numer następnej będzie numerem poprzedniej + (krok).

Uwaga: (start) ani (krok) nie noga być podane jako równe zeru.

W czterech poniższych wypadkach automatyczna numeracja linii zostanie przerwana i BASIC XE będzie oczekiwał na instrukcję w trybie bezpośrednim:

1) Jeżeli klawisz RETURN zostanie naciśnięty natychmiast po numerze linii.

2) Jeżeli BASIC XE stwierdzi błąd syntaktyczny we wprowadzonej linii.

3) Jeżeli linia o numerze, który zostałby aktualnie wygenerowany przez numerator automatyczny istnieje już w programie.

4) Jeżeli numer linii, który został aktualnie wygenerowany przez numerator automatyczny jest większy niż 32767.

DEL

Format: DEL nr1 [,nr2]

Przykłady:

```
DEL 100
DEL 1000,1999
```

Instrukcja DEL kasuje linie programu znajdującego się aktualnie w pamięci. Jeżeli podany jest tylko jeden numer linii (nr1) (pierwsza linia w przykładach) skasowana zostanie jedna linia o podanym numerze. Jeżeli podano dwa numery linii (nr1) i (nr2) oddzielone przecinkiem - skasowane zostaną linie od numeru (nr1) do numeru (nr2) włącznie.

RENUM

Format: RENUM [start][,krok]

Przykłady:

```
RENUM
RENUM 100
RENUM ,20
RENUM 1000,5
```

Instrukcja RENUM przenumerowuje program znajdujący się aktualnie w pamięci nadając pierwszej linii programu numer (start) i zwiększając numer dla każdej następnej linii o(krok) Jeżeli nie podano numeru (start) zostaje mu przypisana wartość 10, jeżeli nie podano kroku renumeracji (krok) zostaje mu przypisana wartość 10.

Uwaga: (start) ani (krok) nie mogą być podane jako równe zeru.

Wszystkie odwołania występujące w instrukcjach (np. w GOSUB, GOTO itd.) zostaną zmodyfikowane, jeżeli są stałymi numerycznymi. Jeżeli w odwołaniu numer linii podany jest jako wyrażenie numeryczne (np. GOTO 10wA) nie zostanie przenumerowany.

Uwagi:

1. Jeżeli program wykonywany jest w trybie FAST szybkość jego wykonania nie zależy od sposobu ponumerowania linii w programie.
2. Jeżeli w programie występuje instrukcja LIST z podanymi numerami linii numery te nie zostaną zmienione poprzez wykonanie instrukcji RENUM.
3. Instrukcja RENUM nie zmienia numerów linii, jeżeli występuje ona jako odwołanie i poprzedzona jest odwołaniem do linii o numerze podanym jako wyrażenie numeryczne. Na przykład wykonanie instrukcji RENUM jeżeli w programie wystąpi linia:

   ```
   10 On X Gosub 100,3*Y,200
   ```

   spowoduje zmianę 100 na odwołanie do linii o odpowiednim numerze, natomiast 200 pozostanie niezmienione, ponieważ poprzedzone jest odwołaniem do linii o numerze podanym jako wyrażenie numeryczne (3*Y). Taka sytuacja może wyniknąć w instrukcjach typu ON ...
4. Jeżeli w programie występuje odwołania do linii, które nie istnieje (np. GOTO 50, gdzie linia 50 nie występuje w programie), nie zostaną one zmienione.

FAST

Format: FAST

Przykłady:

```
FAST
100 FAST
```

Podczas normalnego wykonania programu interpreter języka BASIC XE musi za każdym razem przeszukiwać program od początku aby odnaleźć linię o określonym numerze, jeżeli musi wykonać instrukcję GOTO, GOSUB, FOR lub WHILE (większość interpreterów języka BASIC na różnych komputerach działa właśnie w ten sposób). Działanie interpretera języka BASIC XE w tym zakresie może jednak zostać zmienione poprzez użycie instrukcji FAST. Kiedy napotkana zostaje instrukcja FAST BASIC XE przeprowadza tzw. prekompilację programu aktualnie znajdującego się w pamięci. Oznacza to, że każdemu numerowi linii, do którego występuje odwołanie, przypisany zostaje fizyczny adres, pod którym znajduje się ona w pamięci programu.

Następnie w czasie wykonania programu, gdy BASIC XE ma wykonać instrukcję GOTO, GOSUB, FOR lub WHILE, nie musi przeglądać kolejno programu w pamięci poszukując linii o

odpowiednim numerze, tylko natychmiast wykonuje skok do odpowiedniego adresu.

Uwagi:

1. Jeżeli numer linii podany w instrukcji GOTO lub GOSUB itd. nie jest stałą, tylko zmienną lub wyrażeniem numerycznym, adres nie zostaje obliczony w czasie prekompilacji i wykonanie takiej instrukcji odbywa się z normalną prędkością.
2. Wykonanie każdej z poniższych instrukcji niszczy efekty prekompilacji, czyli przerywa tryb FAST:

   DEL
   ENTER
   EXTEND
   LIST
   LOAD
   LVAR
   RUN "specyfikacja zbioru"
   SAVE
   powrót do trybu bezpośredniego
3. Linia programu zawierająca instrukcje FAST powinna wystąpić przed pierwszą w programie instrukcja GOSUB, FOR, (ALL, WHILE, LOCAL. Najlepiej jeżeli FAST występuje w pierwszej linii programu.
4. Podane powyżej uwagi utrudniają stworzenie programu doładowującego nakładki w czasie wykonania (instrukcją ENTER) i pracującego w trybie FAST. Istnieje jedna droga, która umożliwia stworzenie programu działającego w trybie FAST i korzystającego z nakładek. Program główny nie może być w pętli, podprogramie ani strefie zmiennych lokalnych kiedy wykonuje instrukcję ENTTER. Wtedy pierwsza linia doładowywanego segmentu może być instrukcją FAST i nie spowoduje to powstania problemów.

TRACE / TRACEOFF

Format: TRACE ... TRACEOFF

Przykłady:

```
100 TRACE
TRACEOFF
```

Te dwie instrukcje umożliwiają załączenie lub wyłączenie śledzenia wykonania programu (numerów linii wykonywanego programu). W trybie TRACE numer linii aktualnie wykonywanej jest wyświetlany na ekranie w nawiasach kwadratowych.

Uwagi:

1. Pierwsza linia programu nie może być śledzona.

2. Instrukcja wykonana w trybie bezpośrednim (jeżeli śledzenie jest aktywne) jest pokazywana jako [32768].

SET

Format: SET wyr_num1,wyr_num2

Przykłady:

```
SET 2,128
100 SET 5,A
```

Instrukcja SET pozwala na zmianę działania interpretera języka BASIC XE w niektórych sytuacjach. Wyr_num1 oznacza funkcję, która ma być zmieniona, wyr_num2 oznacza wartość wg której funkcja zostaje zmieniona. Poniższa tabela podaje wszystkie funkcje możliwe do zmiany instrukcją SET.

wyr_num1	wyr_num2	domyślne	znaczenie
0	0 1 128	0	Klawisz BREAK działa normalnie. Naciśnięcie BREAK powoduje błąd nr 1. BREAK jest ignorowany przez BASIC XE, ale niektóre procedury systemowe mogą rozpoznać jego wciśnięcie.
1	1...128	10	Ustawienie tabulatora dla przecinka w instrukcji PRINT.
2	0...255	63	Kod znaku wyświetlanego po instr. INPUT.
3	0 1	0	Pętle FOR są wykonywane co najmniej raz. Pętle mogą być nie wykonywane (ANSI).
4	0 1	1	INPUT nie czyta wielu zmiennych (błąd 8) INPUT może czytać wiele zmiennych kolejno.
5	0 1	1	Edycja i listing programu w standardzie ATARI BASIC (wyłącznie duże litery). Standard BASIC XE.
6	0 1	0	Pojawiają się opisy błędów. Pojawia się tylko numer błędu (ATARI BASIC).
7	0 1	0	P/M znikają za poziomymi brzegami ekranu. P/M odbijają się od poziomych brzegów ekranu
8	0 1	1	Liczba parametrów nie jest wysyłana na stos podczas wykonania USR. Liczba parametrów jest odkładana na stosie.
9	0 1	0	Po wykonaniu instrukcji ENTER następuje powrót interpretera do trybu bezpośredniego. Po ENTER sygnalizowany jest błąd 32.
10	0 1	0	Cztery „missile” są niezależne. „Missile” poruszają się razem, ich atrybuty

			(kolory itp.) pozostają niezależne
11	1...255 0	40	Automatyczna rezerwacja długości zm. tekst. Brak rezerwacji (jak ATARI BASIC).
12	0 1	1	Przy LIST nie działa formatowanie (wcięcia). Formatowanie jest włączone.
13	0 1	1	VAL nie akceptuje liczb hex. (błąd 18). Liczby hexadecymalne są akceptowane.
14	0 1	0	PRINT USING obcina liczby jeśli są dłuższe niż wyspecyfikowano w formacie. Sygnalizowany jest błąd nr 23.
15	0 1	0	W trybie EXTEND tylko ADR(„tekst”) powoduje wystąpienie błędu nr 3. ADR(„tekst”) zawsze podaje prawidłowy adres tekstu.

SYS (funkcja)

Format: SYS(wyr_num)

Przykład:

```
100 If SYS(0)=0 THEN SET 0, 128
```

Funkcja SYS służy do sprawdzania statusu systemu BASIC XE (tj. funkcji możliwych do modyfikowania za pomocą instrukcji SET). Wyr_num jest numerem funkcji systemu jak opisano w punkcie poprzednim.

LOMEM

Format : LOMEM adres

Przykład:

```
LOMEM DPEEK(128)+1024
```

Instrukcja LOMEM służy do rezerwowania pamięci (przestrzeni adresowej) poniżej normalnej przestrzeni dla programu. Przestrzeń ta może zostać użyta np. jako pamięć do przechowywania ekranu, lub dla podprogramów w kodzie maszynowym. Użyteczność tej instrukcji może być dość ograniczona, ponieważ istnieją inne zarezerwowane (takie, z których BASIC XE nie korzysta) adresy.

Uwaga: LOMEM usuwa z pamięci aktualnie znajdujący się tam program.

CLR

Format: CLR

Przykład:

```
100 CLR
```

CLR kasuje wszystkie zmienne w tablicy wartości zmiennych (ang. Variable Value Table) i kasuje przydział pamięci dla tablic wykonany instrukcją DIM. Instrukcja CLR nie kasuje nazw zmiennych z tablicy nazw zmiennych, co robi tylko NEW. Po wykonaniu instrukcji CLR, jeżeli używane są tablice arytmetyczne, tekstowe, lub zmienne tekstowe, musi zostać najpierw określony ich wymiar instrukcją DIM.

LVAR (LV.)

Format: LVAR ["specyfikacja zbioru"]

Przykład:

```
LVAR "P: "
```

Instrukcja LVAR listuje wszystkie nazwy zmiennych do zadeklarowanego zbioru. Każda nazwa zmiennej jest poprzedzona listą numerów linii programu w których występuje (lista odwołań). Powyższy przykład spowoduje wylistowanie nazw zmiennych z listy odwołań na drukarkę. Jeżeli pominięto specyfikację zbioru - lista zostanie pokazana na ekranie.

Uwagi:

1. Zmienne tekstowe są wyróżnione poprzez dodanie znaku "$", natomiast tablice przez dodanie znaku "(".
2. LVAR susi być ostatnią (lub jedyną) instrukcją w linii.

EXTEND

Format: EXTEND

Przykład: EXTEND

Jeżeli nie użyto instrukcji EXTEND (na Atari 130XE) BASIC XE pracuje podobnie jak Atari BASIC. Z punktu widzenia wielu programów BASIC XE w "normalnym" trybie jest równoważny Atari BASIC. Jest oczywiście znacznie szybszy i posiada wiele dodatkowych możliwości, ale zarządza pamięcią podobnie jak Atari BASIC.

Instrukcja EXTEND powoduje, że BASIC XE przechodzi z "normalnego" trybu (kompatybilnego z Atari BASIC) do trybu "poszerzonego". W trybie poszerzonym program w języku BASIC XE znajduje się w dodatkowych 64k pamięci Atari 130XE). Program może zajmować całe ó4k pamięci bez względu na dane jakie wykorzystuje (tablice, zmienne tekstowe itd.), które znajdują się w podstawowej pamięci.

Instrukcji EXTEND można użyć w trybie bezpośrednim (nie jako linii programu) zarówno wtedy, jeżeli program znajduje się w pamięci, jak i wtedy, kiedy w pamięci nie ma programu. Wykonanie instrukcji EXTENID powoduje przeniesienie programu aktualnie znajdującego się w pamięci w dodatkowe ó4k. W trybie EXTEND jedyną drogą powrotu do trybu normalnego jest wykonanie instrukcji NEW lub załadowanie (LOAD) programu, który był napisany w trybie podstawowym.

Także w momencie ładowania programu, który napisany został w trybie EXTEND BASIC XE automatycznie przechodzi do trybu poszerzonego. Jedyną drogą do przetworzenia programu napisanego w trybie poszerzonym na program pracujący w trybie normalnym jest zapisanie go instrukcją LIST do pamięci zewnętrznej (magnetofon, stacja dysków) i ponowne wprowadzenie instrukcją ENTER w trybie podstawowym.

Uwagi:

1. Instrukcja EXTEND może być użyta tylko w trybie bezpośrednim, nie może wystąpić jako instrukcja w linii programu.
2. Przejście do trybu poszerzonego jest możliwe tylko na komputerze Atari 130XE lub kompatybilnym (np. Atari 256XT). Jeżeli dodatkowa pamięć nie istnieje lub jest niedostępna, próba wykonania instrukcji EXTEND powoduje błąd:

   Error 60,"Extended Memory Not Available".
3. BASIC XE sprawdza (wg ustalonej przez Atari Corporation umowy) czy dodatkowa pamięć nie jest zajęta. Jeżeli jest zajęta (np. zawiera RAMDISK założony przez DOS 2.5) BASIC XE nie przechodzi do trybu poszerzonego i wykazuje błąd nr 40. Jeżeli wcześniejsze niż ustalenia odnośnie zaznaczania zajętości dodatkowej pamięci oprogramowanie (np. początkowe wersje DOS 2.5) nie pokazuje, że używa dodatkowej pamięci, instrukcja EXTEND zostanie wykonana powodując zniszczenie znajdującej się tam poprzednio informacji.
4. BASIC XE wypełnia dodatkową pamięć programem zaczynając "od dołu". Jak pokazano na drugim rysunku w dodatku B pierwszych ok. 16k programu znajduje się w bloku 0, następne w bloku itd. Powyżej napisano ok. 16k ponieważ BASIC XE pozostawia min. $100 bajtów wolnych w każdym bloku i nigdy nie "łamie" linii programu pomiędzy blokami (dana linia programu zawsze w całości znajduje się w jednym bloku).
5. Odejmując ok. $400 ad wartości jaką wykazuje funkcja FRE(1) otrzymuje się dolną granicę pozostałej wolnej przestrzeni w pamięci dodatkowej. Dla przykładu blok 3 może zostać w całości wykorzystany np. do przechowywania kilku ekranów graficznych, jeżeli FRE(1)-$400 wykaże co najmniej 16k wolnych bajtów. Dodatek D oraz instrukcja obsługi Atari 130XE zawiera informacje na temat sprzętowej strony przełączania bloków pamięci.

INPUT (I.)

Format: INPUT { [#kanał] / ["tekst"] } zmienna1, zmienna2...

Przykłady:

```
INPUT X
100 INPUT A$(4;)
100 INPUT X, Y, Z(4), B$
100 INPUT #4, A$(5,9)
100 INPUT "Nr#, Nazwa» ",Numer(X), Nazwa$(X)
```

Instrukcja INPUT służy do wprowadzania wartości zmiennych i przypisywania ich zmiennym. Pierwsza dana wprowadzana instrukcji INPUT zostanie przypisana zmiennej1, druga zmiennej2 itd. Jeżeli jedną instrukcją INPUT wprowadzana jest więcej niż jedna zmienna wartości mogą być podane w jednej 1inii oddzielone przecinkami, lub wprowadzane pojedynczo (z naciśnięciem klawisza RETURN po podaniu wartości kolejnej zmiennej). W drugim przypadku BASIC XE wskaże podwójnym znakiem zapytania (??), że podać należy wartość następnej zmiennej. Jeżeli jedną instrukcją INPUT wprowadzane są wartości kilku zmiennych tekstowych - każdy tekst musi zostać wprowadzony jako oddzielna linia. Jeżeli jedną instrukcją INPUT są wprowadzane wartości zmiennych numerycznych i tekstowych zmienne tekstowe muszą wystąpić na końcu ciągu zmiennych wprowadzanych instrukcją INPUT.

Piąty z powyższych przykładów pokazuje ciekawą możliwość. Tekst w cudzysłowie podany bezpośrednio po instrukcji INPUT zostanie wyświetlony na ekranie pozwalając sformułować zapytanie o dane w sposób bardziej obrazowy, niż przy użyciu standardowego "?".

Uwagi:

1. Można spowodować, że BASIC XE stwierdzi błąd wykonania, zamiast wczytywać więcej niż jedną zmienną instrukcją INPUT używając SET 4,wyr_num. Można także zmienić znak zapytania o wartość zmiennej (standardowo "?") używając SET 2,wyr_num (bliższe informacje w opisie instrukcji SET).

   W miarę możliwości należy unikać:

   - wprowadzania więcej niż jednej zmiennej każdą instrukcją INPUT
   - stosowania instrukcji INPUT i PRINT do danych zapisanych na dysku (w tym wypadku zalecane jest użycie RGET i RPUT).

2. Jak widać w trzecim z powyższych przykładów BASIC XE pozwala na stosowanie instrukcji INPUT do elementów tablic

(numerycznych i tekstowych), co nie jest dopuszczalne w standardowym Atari BASIC. To poszerzenie bardzo ułatwia tworzenie programów i pozwala na zwiększenie ich efektywności.

TAB

Format: TAB [#kan,] wyr_num

Przykłady:

```
TAB #2,20
100 TAB 12
```

TAB służy do wprowadzenia spacji aż do pozycji określonej przez wyr_num na podany kanał. Jeżeli kanał nie został podany – TAB działa na ekranie. Pierwsza kolumna ma numer zero.

Uwagi:

1. Licznik kolumn jest pamiętany oddzielnie dla każdego urządzenia i jest kasowany na zero każdorazowo po przejściu do nowej linii. Licznik przechowywany jest w Aux6 IO(B (zob. OS Reference Manual).
2. Jeżeli wartość wyr_num jest mniejsza niż aktualna pozycja wykonywane jest przejście do nowej linii i następnie wyprowadzane są spacje aż do wyspecyfikowanej pozycji.

TAB (funkcja)

Format: TAB(wyr_num)

Przykład:

```
PRINT #3;"kolumny:";TAB(20);K1;TAB(30);K2
```

Funkcja TAB działa analogicznie, jak opisana powyżej instrukcja TAB. Jako funkcja może ona natomiast być używana w instrukcji PRINT i PRINT USING. Pozwala to w prosty i przejrzysty sposób zaplanować postać wyprowadzanych informacji.

Uwagi:

1. Jeżeli wartość wyr_num jest mniejsza niż aktualna pozycja wykonywane jest przejście do nowej linii i następnie wyprowadzane są spacje aż do wyspecyfikowanej pozycji.
2. Funkcja TAB może być używana jedynie w instrukcjach PRINT i PRINT USING.

PRINT USING

Format: PRINT [#kan | ; | , | USING wyr_tekst, wyr1 [,wyr2...]

PRINT USING pozwala zdefiniować format, według którego mają być wyprowadzane dane. Wyr_tekst jest wyrażeniem tekstowym, którego wartość określa format wyprowadzania danych i składa

się z jednego lub kilku pól formatu. Każde pole formatu określa sposób wyprowadzania wartości jednego wyrażenia. Pierwsze pole formatu określa sposób wyprowadzenia wartości pierwszego wyrażenia itd. Znaki formatujące, które mogą być używane do definiowania pola formatu to # & * + $, . % ! oraz / (znaczenie każdego z tych znaków zostanie omówione w dalszej części). Znaki nie będące znakami formatującymi są traktowane jako rozdzielenie pól formatu i są wypisywane pomiędzy polami.

Uwaga: wyr_tekst musi zawierać co najmniej jedno poprawnie zdefiniowane pole formatu. W przeciwnym wypadku BASIC XE będzie wypisywał wyr_tekst raz za razem poszukując pola formatu.

Formatowanie wprowadzania liczb: poniższe znaki służą do tworzenia pola formatu do wypisywania liczb i mają następujące znaczenie:

wypełnij spacją
& wypełnij zerem
* wype3nij gwiazdka
. kropka dziesiętna
, postaw przecinek
+ znak (+/-) przed / po
- znak (tylko -) przed / po
$ znak dolara

$ oraz *: Jeżeli wyprowadzana liczba zawiera mniej cyfr niż zadeklarowano w polu formatu to liczba zostanie przesunięta w prawo do końca pola formatującego a wolne miejsca zostaną wypełnione znakami zależnymi od rodzaju znaków formatujących. Jeżeli wyprowadzana liczba zawiera więcej cyfr niż zadeklarowano w polu formatu to zostanie wypisanych tyle ostatnich cyfr liczby ile wynosi długość pola formatu. Obrazują to następujące przykłady:

liczba	*format*	*postać na wydruku*
123	####	123
123	&&&&	0123
123	****	*123
1234	####	1234
12345	####	2345

Uwaga: Instrukcję SET 14,1 można spowodować, że w wypadku gdy wyprowadzana liczba zawiera więcej cyfr niż określono w polu formatu to nie zostanie ona obcięta przy wypisywaniu, lecz wystąpi błąd wykonania numer 23.

(kropka): kropka występująca w opisie pola formatu określa miejsce usytuowania kropki dziesiętnej. Wszystkie pozycje występujące po kropce dziesiętnej w wyprowadzanej postaci

liczby zawierają cyfry. Jeżeli wyprowadzana liczba zawiera mniej cyfr po kropce niż wyspecyfikowano w opisie pola formatu, dodatkowe miejsca wypełnione zostaną zerami. Jeżeli wyprowadzana liczba zawiera więcej cyfr po przecinku niż wyspecyfikowano w polu formatu, wyprowadzana liczba zostanie zaokrąglona.

Uwaga: Druga kropka występująca w tym samym polu formatu nie jest traktowana jako znak formatujący tylko jako znak oddzielający pola formatu. Poniżej podano kilka przykładów:

liczba	*format*	*postać na wydruku*
12.488	###.##	12.49
123.4	###.##	123.40
2.35	**.**.	2.35.

(przecinek): przecinek w polu formatu oznacza miejsce, gdzie ma być postawiony przecinek w wyprowadzanej postaci liczby. Jeżeli przecinek w polu formatu wystąpi wcześniej niż pierwsza cyfra wyprowadzanej liczby, to miejsce przecinka w wyprowadzanej postaci liczby zajmie odpowiedni znak zależny od wybranego opisu pola formatu.

Uwaga: Przecinek może wystąpić tylko przed kropką dziesiętną (jeżeli kropka dziesiętna jest używana). W przeciwnym wypadku przecinek zostanie potraktowany jako znak oddzielający pola formatu. Oto kilka przykładów:

liczba	*format*	*postać na wydruku*
5216	##,###	5,216
3	*,****	*****3
4175	#,###.	4,175.

+ oraz - : Plus w polu formatu oznacza, że ma być wyprowadzony znak liczb (+ jeżeli liczba jest dodatnia, - jeżeli ujemna). Minus oznacza, że znak (-) ma być wyprowadzony jeżeli liczba jest ujemna, natomiast spacja (znak odstępu) jeżeli liczba jest dodatnia. Znak może wystąpić na początku liczby w pierwszym polu formatu, bezpośrednio przed pierwszą cyfrą liczby lub bezpośrednio po ostatniej wyprowadzanej cyfrze liczby.

Jeżeli znak ma wystąpić na pierwszej pozycji pola formatu wyprowadzanej liczby pole formatu opisane może być np.:

liczba	*format*	*postać na wydruku*
43.7	+###.##	+ 43.70
-43.7	-###.##	- 43.70
23.58	-&&&.&&	023.58

```
-23.58    &&&.&&         -023.58
```

Jeżeli znak ma wystąpić bezpośrednio przed pierwszą cyfrą wyprowadzanej liczby to wszystkie pozycje w opisie pola formatu przed kropką dziesiętną (jeżeli kropka dziesiętna występuje) muszą zawierać znak formatujący "+" lub "-", jak na przykładach:

liczba	*format*	*postać na wydruku*
3.75	+++.##	+3.75
3.75	---.##	3.75
-3.75	---.##	-3.75

Znak może też wystąpić bezpośrednio po wyprowadzanej liczbie. W tym wypadku znak musi wystąpić po kropce dziesiętnej jako ostatni znak w polu formatu:

liczba	*format*	*postać na wydruku*
43.17	***.**+	*43.17+
43.17	&&&.&&-	043.17
-43.17	###.##-	43.17-

$ (znak dolara): Znak dolara może wystąpić na początku liczby w pierwszym polu formatu, lub bezpośrednio przed pierwszą cyfrą liczby. Poniżej podano kilka przykładów użycia znaku dolara na stałej pozycji:

liczba	format	postać na wydruku
34.2	$##.##	$34.20
34.2	+$##.##	+$34.20
34.2	-$##.##	$34.20
-34.2	+$###.##	-$ 34.20

Przykłady użycia znaku dolara na zmiennej pozycji (przed pierwszą cyfrą liczby):

liczba	*format*	*postać na wydruku*
34.2	$$$$$.##	$34.20
34.2	+$$$$$.##	+ $34.20
-72692.41	$$$,$$$.##+	$72,692.41-

Uwagi:

1. Tylko jeden znak może wystąpić na zmiennej pozycji (zawsze przed pierwszą cyfrą liczby).
2. Użycie +, - lub $ w nieprawidłowej pozycji może spowodować dziwne efekty.

Formatowanie wyprowadzania tekstów: poniższe znaki służą do formatowania wyprowadzania tekstu:

% oznacza, że tekst ma być wypisany z przesunięciem do prawego końca specyfikowanego pola.

! oznacza, że tekst ma być wypisany z przesunięciem do lewego końca specyfikowanego pola.

Jeżeli liczba znaków wyprowadzanego tekstu jest większa niż wyspecyfikowano w opisie pola formatu, to tekst zostanie obcięty (zostaną wypisane jedynie początkowe znaki tekstu) jak na przykładach:

tekst	*format*	*wypisany tekst*
"BASIC XE"	%%%%%%%%%	BASIC XE
"BASIC XE"	!!!!!!!!!	BASIC XE
"BASIC XE"	%%%%%	BASIC
"BASIC XE"	!!!!!	BASIC

Wstawianie znaków: znak kreski ukośnej (/) nie kończy pola formatu, ale powoduje, że następny po nim znak będzie wypisany w polu formatu, co pokazują poniższe przykłady:

wartość	*format*	*postać wyprowadzana*
4084463099	(###/)###/-####	(408)446-3099
”BASIC”	%/.%/. %/.%./.%/.	B.A.S.I.C

Uwaga: Jeżeli wartości do wyprowadzenia jest więcej niż wyspecyfikowano w wyr_tekst pól formatu, wypisywanie będzie kontynuowane korzystając po raz kolejny z danego formatu jak w przykładzie:

```
PRINT USING "####", 25,19,7
```

spowoduje wypisanie:

```
25 19 7
```

NORMAL/INVERSE

Format: NORMAL
INVERSE

Przykłady:

```
NORMAL
```

```
100 NORMAL
150 INVERSE
```

NORMAL i INVERSE pozwalają zmienić działanie instrukcji PRINT, LPRINT i PRINT USING. Po wykonaniu instrukcji NORMAL wszystko zostanie wypisane w takiej postaci, jak w odpowiednich instrukcjach. Natomiast po wykonaniu instrukcji INVERSE wszystkie znaki podczas wypisywania zostaną zamienione na inverse video. W tym przypadku znaki, które np. w instrukcji PRINT były pierwotnie w inverse video zostaną wyprowadzone jako znaki zwykłe.

Uwaga: BASIC XE powraca do trybu NORMAL po przejściu do trybu bezpośredniego, lub po wykonaniu przez program instrukcji RUN.

BPUT

Format: BPUT #kan, wyr_num1, wyr_num2 [,bank]

Instrukcja BPUT wysyła zawartość fragmentu pamięci na urządzenie zewnętrzne przez wyspecyfikowany (otwarty wcześniej instrukcją OPEN) kanał. Blok (fragment) pamięci rozpoczyna się od adresu określonego wartością wyr_num1 i ma długość (w bajtach) określoną wartością wyr_num2. W trybie EXTEND można także określić numer banku pamięci (bliższe informacje w opisie instrukcji EXTEND).

Uwaga: Wyr_num1 może określać dowolny adres w pamięci, np. adres początku zmiennej tekstowej znaleziony przy użyciu funkcji ADR. Podany w poniższym przykładzie program zapisuje na dysku pamięć ekranu w trybie graficznym 8:

```
100 Graphics 8: Adres=Dpeek($58)
110 Print "Wypelnianie ekranu..."
120 For Sbajt=0 To (40*160)-1:Rem "wypelnij ekran"
130 Poke Adres+Sbajt,Random(256)
140 Next Sbajt
190 Print "Ekran wypelniony. Teraz zapisywanie..."
160 Close #1: Open #1,8,0, "D:GR8.S(R": Rem "kanał
przygotowany"
170 Bput #1, Adres, 40*160
180 Close #1
190 Print "Zapis zakonczony." 200 End
```

Uwaga: zapisany instrukcją BPUT zbiór nie zawiera informacji o ilości zapisanych danych. Zalecany jest zapis bloków o stałej długości, aby ułatwić późniejsze wczytywanie.

BGET

Format: BGET #kan, wyr_num1, wyr_num2 [,bank]

BGET wczytuje do pamięci poczynając od adresu określonego przez wyr_num1 wyr_num2 bajtów z wyspecyfikowanego kanału. Można, podobnie jak w instrukcji BPUT, określić numer banku pamięci, jeżeli BASIC XE znajduje się w trybie EXTEND. Poniższy przykład pokazuje sposób wczytania ekranu graficznego w trybie 8 z dysku bezpośrednio do pamięci ekranu.

```
100 Graphics 8:Adres=Dpeek($58)
110 Close #1: Open #1,4,0,"D:GR8.S(R": Rem "Przygotowanie"
120 Print "Wczytywanie..."
130 Bget #1,Adres,40*160
140 Close #1
130 Print "wczytywanie zakonczone." 160 End
```

Uwaga: Nie są sprawdzane błędy związane z podanym adresem i długością wczytywanego bloku.

RPUT

Format: RPUT #kan, wyr [, wyr. . .]

Instrukcja RPUT pozwala na wyprowadzenie na urządzenie zewnętrzne poprzez otwarty (instrukcją OPEN) kanał rekordów o stałej długości. Każde wyrażenie zajmuje jedno pole w rekordzie. Pole numeryczne składa się z jednego bajta, który określa typ pól jako numeryczny i sześciu bajtów zawierających liczby zapisane w kodzie B(D (jako zmiennopozycyjne). Pole tekstowe zawiera jeden bajt określający typ pola jako tekstowego, dwa bajty określające długość LEN zmiennej, dwa bajty określające rozmiar DIM zmiennej i DIM bajtów zawierających znaki tekstu. Oznacza to, że zmienne zapisane instrukcją RPUT nie sosy być wczytywane instrukcją INPUT, jeżeli została zapisana instrukcją RPUT wartość więcej niż jednej zmiennej.

Poniższy przykład pokazuje użycie RPUT do zapisu 20 rekordów składających się z pól "Nazwisko", "Imie", "Ulica", "Miasto", Kod, Telefon:

```
100 DIM Nazwisko$(20,30),Imie$(20,30),Ulica$(20,30)
110 DIM Miasto$(20,30),Kod(20), Telefon(20)
120 Close #1: Open #1,8,0,"D:ZNAJOMI.DAT"
130 For Recnr=1 To 20
140 Input "Nazwisko» ",Nazwisko$(Recnr;)
150 Input "Imie» ",Imie$(Recnr;)
160 Input "Ulica» ",Ulica$(Recnr;)
170 Input "Miasto» ",Miasto$(Recnr;)
```

```
180 Input "Kod» ", Kod(Recnr)
190 Input "Telefon» ", Telefon(Recnr)
200 Print :Print "Podane dane:"
210 Print Nazwisko$(Recnr;):Print Imie$(Recnr;)
220 Print Ulica$(Recnr;): Print Miasto$(Recnr;)
230 Print Using "##/-###”,Kod(Recnr)
240 Print Using "##/-##/-##/-",Telefon(Recnr)
250 Print "Zapisac (T/N)?",Odp$
2B0 If (Odp$="y") Or (Odp$="Y"):Rem "zapis RPUT"
260 Rput #1,Nazwisko$(Recnr;),Imie$(Recnr;)
270 Rput #1,Miasto$(Recnr;),Kod(Recnr),Telefon(Recnr)
280 Else :Print "Wprowadz rekord ponownie.": Goto 140
290 Endif
300 Next Recnr
3l0 Close #1:Print :Print "Wykonane"
320 End
```

RGET

Format: RGET #kan, zmienna [,zmienna]

RGET pozwala na wczytanie rekordów o stałej długości zapisanych instrukcją RPUT z kanału uprzednio otwartego (instrukcją OPEN) i przypisanie wczytanych wartości odpowiednim zmiennym (numerycznym i tekstowym).

Uwagi:

Zmienna, której wartość jest wczytywana musi być tego samego typu co zapisana.

Jeżeli wczytywane są zmienne tekstowe - wymiar DIM zmiennej, do której wartość jest wczytywana musi być taki sam, jak zapisanej zmiennej.

Nie można wczytywać instrukcją RET wartości do tablic numerycznych i tablic tekstowych. W takim wypadku należy użyć zmiennych pomocniczych i następnie przypisać wartość wczytanych zmiennych odpowiednim elementom tablic.

Poniższy przykład pokazuje użycie instrukcji BGET do wczytania 20 rekordów składających się z pól "Nazwisko", "Imie", "Ulica", "Miasto", Kod, Telefon:

```
100 DIM Nazwisko$(20,30),Imie$(20,30),Ulica$(20,30)
110 DIM Miasto$(20,30),Kod(20),Telefon(20)
120 DIM Tnazw$(30),Timie$(30),Tulica$(30),Tmiasto$(30)
130 Close #1: Open #1,4,0,"D:ZNAJOMI.DAT"
140 For Recnr=1 To 20
150 Rget #1,Tnazw$,Timie$,Tulica$,Tmiasto$,Tkod,Ttelefon
160 Nazwisko$(Recnr;)=Tnazw$: Imie$(Recnr;)=Timie$
```

```
170 Ulica$(Recnr;)=Tulica$: Miasto$(Recnr;)=Tmiasto$
180 Kod(Recnr)=Tkod: Telefon(Recnr)=Ttelefon
190 Next Racnr
200 Close #1: Print : Print "Wczytane. "
210 Rem "Teraz dane sa wczytane, mozna je wypisac"
220 Input "Nowy rekord pokazac? ",Recnr
230 If Recnr<>0:If Recnr>20 Then 310
240 Gosub 320
250 Else :Rem "Pokaz wszystkie rekordy"
260 For Recnr=i To 20
270 Gosub 320
280 Next Recnr
290 Endif
300 Goto 220
310 End
320 Print Nazwisko$(Recnr;):Print Imie$(Recnr;)
330 Print Ulica$(Recnr;):Print Miasto$(Recnr;)
340 Print Using "##/-###", Kod(Recnr)
350 Print Using "##/-##/-##/-",Telefon(Recnr)
360 Return
```

BSAVE

Format: BSAVE wyr_num1,wyr_num2,"specyfikacja zbioru"

Przykład:

```
BSAVE $680,$6FF,"D:FLIP.BIN"
```

Instrukcja BSAVE pozwala zapisać zawartość fragmentu pamięci na urządzenie zewnętrzne. Zapis dokonywany jest w standardzie Atari DOS (z nagłówkiem zawierającym adres ładowania i adres końca bloku. Blok może być następnie załadowany instrukcją BLOAD w to samo miejsce. Wyr_num1 określa adres początku bloku, wyr_num2 określa adres końca bloku pamięci do zapisania. Zapisanych zostanie więc wyr_num2-wyr_num1+1 bajtów.

Uwaga: BSAVE zapisuje blok pamięci jako jeden segment z jednym nagłówkiem. Nie zapisuje adresu startu, czyli blok zapisany tą instrukcją po załadowaniu bezpośrednio z DOS-a nie zostanie automatycznie uruchomiony.

BLOAD

Format: BLOAD "specyfikacja zbioru"

Przykład:

```
BLOAD "D:FLIP.BITY"
```

BLOAD pozwala na wczytywanie zbiorów binarnych zapisanych w standardzie Atari DOS, czyli np. zbiorów utworzonych instrukcją BSAVE, czy procedur w kodzie maszynowym napisanych przy użyciu MAC/65.

Uwagi:

1. Podczas wykonywania instrukcji BLOAD nie są sprawdzane adresy podane w nagłówku zbioru. Nieuważne użycie BLOAD może doprowadzić do zniszczenia istotnej w danej chwili zawartości pamięci.
2. BLOAD pozwala ładować zbiory binarne składające się z wielu segmentów. Jeżeli zbiór posiada nagłówek z adresami INIT i RUN są one ignorowane.
3. Zbiór, który są określony adres początku wykonania RUN może zostać uruchomiony przez: SET 8,0: A=USR(Dpeek($2E0))

DIR

Format: DIR ["specyfikacja_zbioru"]

Przykłady:

```
100 DIR "D: *.COM"
DIR
DIR FILE$
DIR "D2:TEST?.BAS"
```

Wykonanie instrukcji DIR powoduje wyświetlenie na ekranie monitora listy zbiorów według podanej specyfikacji. Działa podobnie jak instrukcja DIR w DOS XL oraz A w DOS 2.5. Jeżeli nie podano specyfikacji_zbioru zostaną wyświetlone wszystkie zbiory znajdujące się na D1 tj. instrukcja działa tak, jak przy DIR "D1:*.* ".

W pierwszym przykładzie zostaną wyświetlone wszystkie zbiory znajdujące się na D1, które posiadają wyróżnik "COM". Drugi przykład powoduje wyświetlenie listy wszystkich zbiorów znajdujących się na D1. Trzeci przykład pokazuje użycie zmiennej tekstowej. Jest to poprawne, ale zmienna tekstowa musi zawierać prawidłową specyfikację zbioru. W przeciwnym wypadku wystąpi błąd. (zwarty przykład powoduje wyświetlenie listy zbiorów znajdujących się na D2, których nazwa jest pięcioliterowa i rozpoczyna się od "TEST" z wyróżnikiem "BAS".

Uwaga: Jeżeli DIR występuje w programie musi być ostatnią lub jedyną instrukcją w linii.

PROTECT

format PROTECT "specyfikacja_zbioru”

Przykłady:

```
PROTECT "D: *.COM"
100 PROTECT "D2: PROGRAM.EXE"
```

PROTECT pozwala zabezpieczyć zbiór przed przypadkowym skasowaniem (nie dotyczy przypadkowego sformatowania dysku). Zbiór zabezpieczony instrukcją PROTECT musi być przed jego skasowaniem odbezpieczony. Instrukcja PROTECT działa tak jak instrukcja PRO w DOS XL i instrukcja F(LOCK) w DOS 2.5

UNPROTECT (UNP.)

Format: UNPROTECT "specyfikacja_zbioru"

Przykłady:

```
100 UNPROTECT "D:*.COM"
UNP. "D2: *.*".
```

UNPROTECT pozwala odbezpieczyć (np. w celu umożliwienia skasowania) zbiory zabezpieczone przed skasowaniem (np. instrukcją PROTECT w języku BASIC XE). Instrukcja UNPROTECT działa tak, jak instrukcja UNP w DOS XL i G(UNLOCK) w DOS 2.5.

RENAME

Format: RENAME "specyfikacja_zbioru, nazwa zbioru”

Przykład:

```
RENAME "D2:STARY.BAS,NOWY.BAS"
```

RENAME pozwala na zmianę nazwy zbioru dyskowego. Przecinek pomiędzy starą i nową nazwą zbioru jest konieczny.

Uwaga: Nowa nazwa_zbioru nie może zawierać specyfikacji urządzenia (Dn:). Użycie gwiazdek (*) w nazwach przemianowywanych zbiorów nie jest zalecane.

ERASE

Format: ERASE "specyfikacja zbioru"

Przykłady:

```
ERASE "D:*.BAK"
ERASE "D2:TEST.BAS"
```

Instrukcja ERASE pozwala na skasowanie niezabezpieczonych przed skasowaniem zbiorów. Działa jak ERA w DOS XL i D(DELTE) DOS 2.5. Pierwszy przykład powoduje skasowanie z dysku w stacji D1 wszystkich zbiorów z wyróżnikiem "BAK". Drugi przykład powoduje skasowanie z dysku Dl zbioru pod nazwą "TEST.BAS"

WHILE/ENDWHILE

Format: WHILE wyr_num

```
        [instrukcje]
    ENDWHILE
```

WHILE pozwala w prosty i elegancki sposób zorganizować pętle warunkowe. Instrukcje umieszczone w nawiasie WHILE i ENDVHILE

są wykonywane tak długo, dopóki wyr_num jest różne od zera (może być dodatnie lub ujemne). Przed każdym wykonaniem instrukcji zawartych w pętli (czyli każdorazowo przed wykonaniem pętli) obliczana jest wartość wyr_num i podejmowana jest decyzja czy pętla ma zostać wykonana czy nie. Dla przykładu pętla WHILE 1 będzie wykonywana bez końca (pętla nieskończona) natomiast WHILE 0 nie zostanie wykonana ani razu. Poniższy program pokazuje zastosowanie pętli WHILE:

```
100 Rmax=5: Kmax=8: Rzad=0: Kolumna=0: Znaleziono=0: Cel=0
l05 Dim Macierz(Rmax,Kmax)
110 While Rzad<Kmax And (Not Znaleziono)
120  Kolumna=Kolumna+1
130  While Kolumna<Kmax And (Not Znaleziono)
140   If Macierz(Rzad,Kolumna)=Cel Then Znaleziono=1
150   Kolumna=Kolumna+1
160  Endwhile
170  Rzad=Rzad+1
180 Endwhile
190 If Znaleziono: Print "Znaleziono ";Cel;" na ";
200  Print "pozycji (";Rzad-1;","Kolumna-1;")"
210 Else: Print Cel;" nie znaleziono"
220 Endif
```

IF/ELSE/ENDIF

```
Format: IF wyr_num
          [instrukcje]
       [ELSE
          [instrukcje]]
        ENDIF
```

BASIC XE umożliwia zastosowanie bardzo użytecznej, strukturalnej instrukcji warunkowej IF/ELSE/ENDIF. Jeżeli wyr_num jest prawdziwe (różne od zera), zostaną wykonane instrukcje (linie programu) aż do instrukcji ELSE, natomiast instrukcje pomiędzy ELSE i ENDIF zostaną pominięte. Jeżeli wyr_num jest nieprawdziwe (równe zeru) to instrukcje pomiędzy wyr_num a ELSE zostaną pominięte, natomiast pomiędzy ELSE a ENDIF zostaną wykonane. Jeżeli ELSE nie jest potrzebne można pominąć ELSE kończąc łańcuch instrukcji przez ENDIF. Wówczas działanie opisanej instrukcji wygląda tak samo jak instrukcji IF-THEN z tą różnicą, że dopuszczalne jest wystąpienie wielu linii.

Uwaga: Słowo kluczowe THEN nie występuje w instrukcji IF/ELSE/ENDIF.

Poniższy program pokazuje użycie IF/ELSE/ENDIF:

```
100 If I<2
110   Print "Ten ";
120   If 2>3
130     Print "komputer ";
140     If 3<4
150       Print "jest ";
160   Else
170     Print "zepsuty!"
180   Endif
190 Else
200   Print "program ";
210   If 4>5
220     Print "jest ";
230     If 5<6
240       Print "dziwny"
250     Endif
260   Else
270     Print "dziala ";
280   If 6>7
290     Print "zle."
300       Else
310         Print "poprawnie"
320       Endif
330     Endif
340   Endif
350 Else
360   Print "do kitu!!! "
370 Endif
```

ERR (funkcja)

Format: ERR(wyr_num)

Funkcja ta pozwala zbadać numer błędu i numer linii, w której wystąpił, jeżeli pisana jest własna procedura obsługi błędów (przy użyciu instrukcji TRAP). Jeżeli użyto wyr_num o wartości równej 1 to wartością funkcji jest numer linii programu, w której wystąpił błąd. Jeżeli wartość wyr_num jest równa zeru to wartością funkcji jest numer błędu. Dla pozostałych wartości argumentu wyr_num wartość funkcji jest nieokreślona.

Przykład:

```
100 Deg
```

```
110 Print "Kat      Sinus          Cosecans"
120 For I=0 To 180 Step 15
130   Print Using "###    #.#####     ",I,Sin(i),
140   Trap 200
150   Print Using "####.####",1/Sin(I)
160 Next I
170 End
180 Rem przejscie do linii 200 jeżeli
190 Rem Sin(I) jest równy zeru.
200 Print "Nieokreslony"
210 Goto Err(1)+10
```

FIND (funkcja)

Format: FIND(wyr_tekst1, wyr_tekst2, wyr_num)

Przykład:

```
          PRINT FIND("ABCDXXXABC","BC",N)
```

Funkcja FIND fiest szybkim i efektywnym narzędziem do określania, czy podany fragment tekstu występuje w podanym tekście. FIND przeszukuje tekst wyr_tekst1 zaczynając od pozycji wyr_num+1 badając, czy w tym tekście występuje fragment wyr_tekst2. Jeżeli podany fragment tekstu występuje w podanym tekście, to wartością funkcji jest pozycja, na której fragment się rozpoczyna. Jeżeli fragment nie występuje, to wartością funkcji jest zero.

w powyższym przykładzie zostaną wypisane następujące wartości:

2 jeżeli N=0 lub 1

9 jeżeli N>=2 i N<9

0 jeżeli N>=9

Poniżej podano dwa przykłady zastosowania funkcji FIND:

```
10 Input "Zmiana, Kasowanie, Drukowanie",AS
20 On Find("ZKD",A$(1,1),0) Goto 100,200,300
30 Goto 10
```

```
10 Input "Napisz cos - ",A$
20 For St=0 To Len(A$) -2
30   F=Find(A$,"A",St)+1
40   If F=1 Then Print "Nie występuje 'AH' ani 'AC'.": End
50   If A$(F,F)="B" Then Print "'AB' na poz. ";F-1:St=St+1
60   If A$(F,F)="C" Then Print "'AC' na poz. ";F-1:St=St+1
70 Next St
```

LEFT$ (funkcja)

Format: LEFT$(wyr_tekst,wyr_num)

Przykłady:

```
10 A$=LEFT$("ABCDE",3)
20 PRINT LEFT$("ABCD",5)
```

Funkcja LEFT$ wydziela z tekstu wyr_tekst wyr_num znaków rozpoczynając od lewej strony. Jeżeli wyr_num jest większe niż ilość znaków w tekście wartością funkcji jest cały tekst wyr_tekst (nie występuje błąd wykonania).

W pierwszym podanym przykładzie do A$ zostanie podstawione "ABC", w drugim zostanie wypisane "ABCD".

MID$ (funkcja)

Format: MID$(wyr_tekst, wyr_num1, wyr_num2)

Przykład:

```
A$=MID$( "ABCDEFG",2,4)
```

MID$ pozwala na wydzielenie fragmentu tekstu z wnętrza tekstu wyr_tekst. Wydzielany jest tekst od pozycji określonej przez wyr_num1 o długości wyr_num2. Jeżeli wyr_num1 jest równe zero występuje błąd (nie ma pozycji zerowej w tekście). Jeżeli wyr_num1 jest większe niż długość tekstu - nie ma błędu (i wynikiem działania funkcji jest tekst pusty). Wyr_num2 może być dowolną całkowitą liczbą dodatnią. Jeżeli wyr_num1+wyr_num2+1 jest większe niż długość wyr_tekst, wartością funkcji jest fragment tekstu od pozycji wyr_num1 do końca tekstu wyr_tekst.

W powyższym przykładzie A$ zostanie przypisana wartość "BCDE".

RIGHT$ (funkcja)

Format: RIGHT$(wyr_tekst, wyr_num)

Przykład: A$=RIGHT$("123456",4)

Funkcja RIGHT$ wydziela z tekstu wyr_tekst wyr_num znaków rozpoczynając od prawej strony. Jeżeli wyr_num jest większe niż ilość znaków w tekście - wartością funkcji jest cały tekst wyr_tekst (nie występuje błąd wykonania).

W powyższym przykładzie A$ zostanie przypisana wartość "3456"

STR$ (funkcja)

Format: STR$(wyr_num)

Przykład:

```
A$=STR$(650)
```

Daje w wyniku tekstową reprezentację wartości wyr_num. Powyższy przykład spowoduje podstawienie do A$ tekstu "650".

Uwaga: W instrukcji porównania logicznego może wystąpić tylko jedna funkcja CHR$ lub STR$.

HEX$ (funkcja)

Format: HEX$(wyr_num)

Przykłady:

```
PRINT HEX$(5000)
PRINT "$";RIGHT$(HEX$(32),2)
```

Funkcja HEX$ zamienia wartość wyr_num na tekst odpowiadający czterocyfrowej liczbie w zapisie heksadecymalnym (drugi przykład pokazuje jak można uzyskać liczbę dwucyfrową).

Uwaga: Znak "$" nie jest dodawany na początku tekstu wynikowego.

PEN (funkcja)

Format: PEN(wyr_num)

Przykład:

```
100 PRINT "Pioro swietlne na ";PEN(0);",";PEN(1)
```

Funkcja PEN odczytuje zawartość rejestrów pióra świetlnego GTIA. Jeżeli wartość wyr_num wynosi zero czytana jest współrzędna pozioma, jeżeli wyr_num wynosi 1 czytana jest współrzędna pionowa.

HSTICK (funkcja)

Format: HSTICK(wyr_num)

Funkcja HSTICK przyjmuje wartości zależne od poziomego położenia joysticka. Wyr_num określa port, do którego włączony jest joystick (0-1) Wartości funkcji są następujące:

```
-1  drążek wychylony w lewo
 0  drążek w pozycji neutralnej
 1  drążek wychylony w prawo
```

Poniższy program pokazuje użycie funkcji HSTICK:

```
10 Kierunek=Hstick(0)
20 If Kierunek=-1 Then Print „ W lewo.”
30 IF kierunek=0 Then Print „Stop.”
40 If Kierunek=1 Then Print (W prawo.”
50 Goto 10
```

VSTICK (funkcja)

Format: VSTICK(wyr_num)

Funkcja VSTICK przyjmuje wartości zależne od pionowego położenia drążka joysticka. Wyr_num określa port do którego włączony jest joystick (0-1). Wartości funkcji są następujące:

-1 jeżeli drążek wychylony jest w dół
0 jeżeli drążek jest w pozycji neutralnej
1 jeżeli drążek jest wychylony w górę

Poniższy program pokazuje użycie funkcji VSTICK:

```
10 Kierunek=Vstick(0)
20 If Kierunek=-1 Then Print "W dol."
30 If Kierunek=0 Then Print "Stop."
40 If Kierunek=1 Then Print "W gore."
50 Goto 10
```

GRAFIKA PLAYER/MISSILE

Możliwości animacji obrazu to jedna z ciekawszych cech komputerów Atari. Animacje można wykonać różnymi sposobami, najlepsze efekty jednak można uzyskać stosując właśnie grafikę obiektów P/M (zwane też niekiedy SPRITE's). Istotą grafiki P/M jest sposób pamiętania ruchomego obiektu poruszającego się po dwuwymiarowym ekranie dostosowany do liniowej struktury (jednowymiarowości) pamięci RAM ekranu oraz odpowiednie rozwiązania sprzętowe. Atari umożliwia zdefiniowane czterech obiektów typu P od P0 do P3, które poruszają się niezależnie i których kolory (określone w rejestrach koloru obiektów P/M) są niezależne od siebie i od pozostałych kolorów na ekranie. Każdy z obiektów od P0 do P3 może być przedstawiany w normalnej wielkości albo w podwójnej szerokości albo w podwójnej wielkości (analogicznie jak litery w trybach graficznych 0,1 i 2). Z każdym z obiektów typu P może być stowarzyszony obiekt typu M, odpowiednio od M1 do M4. Kolory obiektów typu M są takie jak obiektów typu P którym odpowiadają, natomiast sposób poruszania się jest niezależny od obiektów P i od innych obiektów M. Poruszaniem obiektów P/M i testowaniem ich kolizji zajmuje się układ GTIA. Możliwości wykorzystania grafiki P/M są duże i różnorodne, mogą dotyczyć nie tylko animacji, ale i np. definiowania własnych znaków graficznych nie mieszczących się w siatce ekranu takich jak indeksy we wzorach itp. Wmontowany fabrycznie interpreter języka Atari BASIC nie ma w ogóle instrukcji umożliwiających tworzenie i animacje obiektów P/M i wykorzystanie Grafiki P/M. Wygodne instrukcje do tworzenia i animacji P/M posiada natomiast BASIC XE. Opisane w tym rozdziale instrukcje i funkcje pozwalają na wykorzystanie możliwości grafiki Player/Missile mikrokomputerów Atari. Rozmiar i kolor oraz zmiany położenia obiektów P/M na ekranie są niezależne od aktualnego trybu graficznego. Każdy obiekt P to grupa komórek

pamięci odpowiadająca pionowemu pasowi na ekranie. Każdy obiekt P może mieć jeden ze 128 kolorów które są dostępne w języku BASIC XE. W poziomym pasie reprezentującym obiekt P bit o wartości 1 oznacza punkt zaświecony (w określonym kolorze), natomiast bit o wartości 0 oznacza punkt bez koloru (kolor tła ekranu w miejscu, w którym obiekt P aktualnie się znajduje). Dzięki zastosowanym w mikrokomputerach Atari rozwiązaniom sprzętowym obiekt P może być przesuwany w poziomie przez zmiana zawartości odpowiedniego rejestru (instrukcja POKE lub PMOVE). Zmiana położenia pionowego obiektu P na ekranie to przesunięcie w bloku pamięci reprezentującym pas na ekranie (instrukcja PMMOVE).

Kompletny opity techniczny działania grafiki P/M znaleźć można w wydanej przez firmę Atari książce "Atari 400/800 Hardware Manual".

PMGRAPHICS (PMG.)

Format: PMGRAPHICS wyr_num

Przykład:

PMG.2

Instrukcja służy do załączania lub wyłączania grafiki P/M. Wyr_num może mieć wartość 0,1 lub 2, co oznacza odpowiednio:

0 - wyłącz P/M

1 - załącz P/M w rozdzielczości jednej linii

2 - załącz P/M w rozdzielczości dwóch linii

Rozdzielczość jednej linii (PMG. 1) oznacza, że jeden bajt pasa P/M zajmuje jedną linię telewizyjną (jak linia w grafice 8 lub 15). Rozdzielczość dwóch linii (PMG. 2) oznacza, że jeden bajt pasa P/M zajmuje dwie linie telewizyjne (jak linia w grafice 7). Od trybu grafiki P/M zależy ponadto ilość pamięci potrzebnej do przechowania obiektów P/M. Rozdzielczość jednej linii wymaga dwukrotnie więcej pamięci niż rozdzielczość dwie linie.

Pokazano to na poniższym rysunku:

```
            PMG. 2                    PMG. 1
+$400     -----------    +$800     -----------

          Player 3                  Player 3
+$380     -----------    +$700     -----------

          Player 2                  Player 2
+$300     -----------    +$600     -----------
          Player 1                  Player 1
+$280     -----------    +$500     -----------
```

```
          Player 0                  Player 0
+$200     -----------    +$400      -----------

          M1 M2 M3 M4               M1 M2 M3 M4
+$180     -----------    +$300      -----------

PMBASE    -----------    PMBASE     -----------
```

Uwaga: MEMTOP ($2E5) wskazuje na szczyt pamięci zawierającej Missile.

Dzięki funkcji PMADR do programowania w języku BASIC XE znajomość powyższej mapy nie jest konieczna.

PMCOLOR (PMCO.)

Format: PMCOLOR pmnum,wyr_num1,wyr_num2

Przykład: PMCOLOR 2,12,8

Instrukcja PMCOLOR jest identyczna w działaniu jak SETCOLOR ale odnosi się do rejestrów koloru obiektów P/M. Wyr_num1 określa kolor, wyr_num2 jego poziom jasności.

Uwaga: Rejestry koloru P/M mają nieokreśloną zawartość po inicjacji systemu.

PMMOVE

Format: PMMOVE pmnum [,wyr_num1][;wyr_num2]

Przykład:

```
PMMOVE 0,120;1
PMMOVE 1,180
PMMOVE 4;-3
```

Jeżeli obiekt P/M został zdefiniowany (poprzez POKE, MOVE, GET, BGET lub MISSILE) może on być przemieszczany po ekranie.

Wyr_num1 jest bezwzględną (poziomą) pozycją lewej krawędzi "pasa" do wyświetlenia. Może być z zakresu od 0 do 255. Zmiana wielkości obiektu P (zob. PMWIDTH) nie powoduje zmiany położenia jego lewej krawędzi, obiekt jest poszerzany w prawą stronę.

Wyrr_num2 określa względne przesunięcie w pionie. BASIC XE pozwala na określenie przesunięcia względnego w pionie od -255 (o 255 punktów w dół) do 255 (o 255 punktów w górę). Znak +/- przesunięcia pionowego jest skoordynowany z wartościami funkcji VSTICK. Na przykład PMMOVE 2; VSTICK(2) spowoduje przesuniecie obiektu P2 w górę lub w dół (albo pozostanie w

tym samym położeniu) w zależności od aktualnie odczytanego położenia joysticka.

Uwaga: SET, wyr_num daje możliwość zmiany zachowania obiektów P/M przy napotkaniu krawędzi ekranu. Zob. opis instrukcji SET.

MISSILE (MIS.)

Format: MISSILE pmnum,wyr_num1,wyr_num2

Przykład: MISSILE 4,48,3

Instrukcja umożliwia uaktywnienie obiektu typu M. Pmnum jest numerem obiektu (4-7), wyr_num1 określa bezwzględne pionowe położenie początku obiektu M, wyr_num2 określa pionową wysokość obiektu M. Na przykład MISSILE 4,64,3 spowoduje umieszczenie obiektu M o wysokości 3 punktów na pozycji 64 punkty od góry. Powtórne użycie instrukcji MISSILE dla obiektu M o tym samym numerze powoduje usunięcie obiektu z poprzedniego położenia.

PMWIDTH (PMW.)

Format: PMWIDTH pmnum,wyr_num

Przykład:

```
PMWIDTH 1,2
```

Instrukcją PMWIDTH można wybrać wielkość dowolnego obiektu P/M. Wyr_num określa wielkość obiektu i może mieć wartość 1,2 lub 4.

Uwaga: Rozmiary obiektów P/M mogą zostać zwiększone instrukcją PMWIDTH, ale bez zmiany rozdzielczości. Uzyskanie większego obiektu o dużej rozdzielczości jest możliwe przez stworzenie obiektu złożonego z kilku obiektów P, co pokazano w drugim przykładzie.

PMCLR (PMC.)

Format: PMCLR pmnum

Przykład:

```
PMCLR 4
```

Instrukcja PMCLR zeruje pamięć przechowującą obiekt typu P lub M. PMCLR w zależności od trybu grafiki P/M wybranego instrukcją PMGRAPHICS kasuje część pamięci przechowującą wybrany obiekt.

Uwaga: Zastosowanie pmnum od 4 do 7 powoduje skasowanie wszystkich obiektów M, a nie tylko wybranego. Aby skasować pojedynczy obiekt M można wykonać np. SET 7,O: PMMOVE n;255

BUMP (funkcja)

Format: BUMP (pmnum,wyr_num)

Przykład:

```
IF BUMP(4,1) THEN B=BUMP(8,0)
```

Funkcja BUMP daje dostęp do sprzętowego rejestru kolizji obiektów P/M. Przyjmuje wartość 1 jeżeli kolizja wystąpiła, wartość 0 jeżeli kolizja nie wystąpiła. Drugi z parametrów może określać obiekt typu P, lub pole (pas) obiektu typu P. Poniżej podano poprawne kombinacje parametrów w funkcji BUMP:

```
obiekt P - obiekt P       BUMP(0-3,0-3)
obiekt P - pole obiektu P     BUMP(0-3,8-11)
obiekt M - obiekt P       BUMP(4-7,0-3)
obiekt M - pole obiektu P     BUMP(4-7,8-11)
```

Uwagi:

1. BUMP(p,p) gdzie p jest od 0 do 3 zawsze ma wartość 0, tzn. obiekt P nie może wejść w kolizję sam ze sobą.
2. W celu uniknięcia błędów po odczycie należy skasować rejestr kolizji.

HITCLR

Format: HITCLR

Przykład:

```
100 HITCLR
```

Instrukcja HITCLR kasuje rejestr kolizji, który może być testowany funkcją BUMP.

PMADR (funkcja)

Format: PMADR(pmnum)

Przykład:

```
P0=PMADR(0)
```

Funkcja PMADR podaje adres w pamięci obiektu typu P lub M. Ułatwia to operowanie obiektami P/M przy użyciu instrukcji MOVE, POKE, BGET, BPUT itp.

Uwaga: PMADR(m) gdzie m jest numerem obiektu M (4-7) daje w wyniku ten sam adres dla wszystkich obiektów typu M.

Użycie POKE i PEEK dla grafiki P/M

Jedną z dróg do zdefiniowania wyglądu obiektów P jest użycie instrukcji POKE. W połączeniu z PMADR można w prosty sposób umieścić obiekt P w pamięci P/M:

```
10 For Loc=48 To 52
20 Read A: Poke Pmadr(0)+Loc,A
30 Next Loc
40 Data $99,$BB,$FF,$BB,$99
```

Funkcja PEEK może służyć do odczytu i np. przechowania zawartości pamięci definiującej obiekt.

Użycie MOVE dla grafiki P/M

Użycie instrukcji MOVE pozwala na szybkie zapisanie do pamięci obiektu jego wyglądu, lub na przepisanie zawartości pamięci obiektu w inne miejsce, dając np. możliwość szybkiej zmiany wyglądu obiektu. Na przykład:

```
Move Adr(A$),Pmadr(2),128
```

przepisuje obiekt P w podwójnej rozdzielczości z A$ do obszaru P2.

Natomiast:

```
Poke Pmadr(1),$FF: Move Pmadr(1),Pmadr(1)+1,127
```

powoduje ustawienie wszystkich bitów obiektu P1 na wartość 1 (pas na ekranie).

Użycie BGET i BPUT dla grafiki P/M

Instrukcja BGET umożliwia wypełnienie pamięci obiektu np. bezpośrednio z dysku. Na przykład:

```
Bget #3,Pmadr(0),$80
```

spowoduje wczytanie ze zbioru na dysku otwartego jako kanał #3 obiektu P0 w trybie PMG. 2. Natomiast:

```
Bget #4,Pmadr(4),$500
```

spowoduje wczytanie z dysku wszystkich obiektów P i M w trybie PMG.1.

Użycie USR dla grafiki P/M

Ponieważ poprzez zastosowanie funkcji USR możliwe jest wywołanie podprogramu w kodzie maszynowym z przekazaniem parametrów, istnieje możliwość wykorzystania w języku BASIC XE podprogramów w kodzie maszynowym przy operowaniu grafiką P/M. Na przykład:

```
A=Usr(Pmmigaj,Pmadr(2),$80)
```

może służyć do wywołania podprogramu w kodzie maszynowym (umieszczonego pod adresem Pmmigaj aby spowodować miganie obiektu P2 o rozmiarze 128 bajtów.

Przykładowe programy z grafiką P/M

```
100 Setcolor 2,0,0: Rem "Ciagle GR.0"
110 Pmgraphics 2: Rem "rozdzielczosc P/M dwie linie"
120 Wielk=0: Y=48: Rem "inicjalizacja"
130 Pmclr 0: Pmclr 4: Rem "kasowanie P0 i M0"
140 Pmcolor 0,13,8: Rem "obiekt P zielony"
150 P=Pmadr(0): Rem "adres P0"
l60 For I=P+Y To P+Y+4: Rem "ladowanie obiektu P"
170 Read V1: Rem "zob. DATA"
180 Poke I,V1: Rem "zapis"
190 Next I
200 For X=1 To 120:Rem "petla poruszajaca obiektem P"
210   Pmmove 0,X: Rem "ruch w poziomie"
220   Sound O,X+X,0,15: Rem "troche halasu"
230 Next X
240 Missile 4,Y,1: Rem "obiekt M na szczycie P"
250 Missile 5,Y+2,1: Rem "nastepny w srodku P"
270 For X=127 To 255: Rem "petla poruszajaca obiektem M"
280   Pmmove 4,X: Rem "obiekt M0"
290   Sound 0,255-X,10,15
300     If(X&7): Rem "co osiem punktow w poziomie"
310      Missile 5,Y,5: Rem "to trzeba zobaczyc!"
320     Endif : Rem "moze byc tez ELSE"
330 Next X
340 Pmmove 6,9
350 Wielk=wielk+2: Rem "zmiana wielkosci"
360 If Wielk>4 Then Wielk=0
370 Pmwidth 0,Wielk: Rem "nowa wielkosc"
380 Pmclr 4: Rem "kasowanie obiektow M"
390 Goto 200: Rem "i od poczatku"
400 Rem
410 Rem "***definicja obiektu P ***”
420 Rem "    84218421 "
430 Rem "$99 *  **  * "
440 Rea "$BD * **** * "
450 Rem "$FF ******** „
460 Rem "$BD * **** * "
470 Rem "$99 *  **  * "
```

```
480 Data S99,EBD,3FF,EBD,E99
```

Program powyższy działa szybciej, jeżeli zapisany zostanie w liniach po kilka instrukcji. Tak jak został przedstawiony jest jednak bardziej czytelny.

```
100 Graphics 0:
110 Pmgraphics 2: Pmclr 0: Pmclr 1
12U Setcolor 2,0,0: Pmcolor 0,12,8: Pmcolor 1,12,8
130 P==Pmadr(0): P1=Pmadr(l): Rem "adresy obiektowi P0 i P1"
140 V0=60:Vs=V0: Rem "poczatkowa poz. pionowa"
150 H0=110: Rem "poczatkowa poz. pozioma"
160 For Loc=V0-8 To V0+7: Rem "obiekt P podw. rozdz."
170   Read X
180   Poke P0+Loc,Int(X/$0100)
190   Poke P1+Loc, X&$ff
200 Next Loc
210 Rem "animacja"
220 Let Promien=40: Deg
230 While 1:Rem "petla nieskonczona!"
240   C=Random(15): Pmcolor 0,C,8: Pmcolor 1,C,8
250   For Kat=0 To 355 Step 5: Rem "w stopniach"
260     Vn=V0+Promien*Sin(Kat)at1
270     Vzmiana=Vn-Vs
280     Hn=H0+Promien*Cos(Kat)
290     Pmmove 0, Hn; Vzmiana: Pmmove 1 , Hn+8; Vzmiana
300     Rem "ruch dwoch obiektow P"
310     Vs=Vn
320     Sound 0,Hn,10,12: Sound 1,Vn,10,12
330   Next Kat
340   Rem "zakreslone kolo!"
350 Endwhile
360 Rem
370 Rem "*** definicja obiektu ***"
380 Rem "      84218421|84218421   "
390 Rem "$03C0       **|**         "
400 Rem "$0C30     **  |  **       "
410 Rem "$10D8    *    |    *      "
420 Rem "$2004   *     |     *     "
430 Rem "$40D2  *      |      *    "
440 Rem "$4E72  * ***  |  *** *    "
450 Rem "$BA51 *  * *  |  * *  *   "
```

```
460 Rem "$BE71 *  ***  |  ***  *   "
470 rem "$8001 *       |       *   "
480 Rem "$9009 * *     |     * *   "
490 Rem "$4812  *  *   |   *  *    "
500 Rem "$47E2  *   ***|***   *    "
510 Rem "$2004   *     |     *"
520 Rem "$10D8    *    |    *      "
530 Rem "$0C30     **  |  **       "
540 Rem "$03C0       **|**         "
550 Rem
650 Data $03C0,$0C30,$10D8,$2004,$40D2,$4E72,$BA51,$BE71
660 Data $8001,$9009,$4812,$47E4,$2004,$1008,$0C30,$03C0
```

Powyżej przedstawiony program korzysta z funkcji SIN i COS. Jego działanie można znacznie przyspieszyć używając tablic z obliczonymi wcześniej potrzebnymi wartościami zamiast obliczać je każdorazowo w czasie przebiegu pętli poruszającej obiektami.

SORTUP/SORTDOWN

Format:

SORTUP tablica [USING[w_num1 TO w_num2][;w_num3,w_num4]]

SORTDOWN tablica [USING[w_num1 TO w_num2][;w_num3,w_num4]]

Przykłady:

```
SORTUP Tablica1
SORTDOWN Tablica1 USING Min TO Max
SORTUP Tekst$ USING; 1, 4
SORTDOWN Tekst$ USING 5 TO 10
```

Uwaga: w_num3,w_num4 mogą być używane jedynie przy sortowaniu tablic tekstowych. Nie mogą być używane do sortowania tablic numerycznych.

SORTUP sortuje elementy tablicy w porządku rosnącym według kodów ATASCII znaków lub według wartości w zależności od rodzaju tablicy (tekstowa lub numeryczna). SORTDOWN natomiast sortuje w porządku malejącym. Jeżeli nie podano zakresu w_num1 TO w_num2 (pierwszy i trzeci przykład) wszystkie elementy tablicy zostają posortowane.

Jeżeli podano zakres sortowania to zarówno pierwszy jak i ostatni element, (w_num1 i w_num2) muszą zostać określone i rozdzielone słowem kluczowym TO

Jeżeli nie określono początku i końca fragmentu tekstu (;w_num3,w_num4) według którego ma się odbywać sortowanie, to sortowanie jest wykonywane używając całego tekstu. Jeżeli

fragment został określony (czwarty przykład) to zarówno początek jak i koniec tekstu muszą zostać podane (muszą być podane ;w_num3,w_num4). Nawet jeżeli nie podano zakresu w_num1 TO w_num2 słowo kluczowe USING musi zostać użyte i poprzedzić specyfikację fragmentu tekstu (trzeci przykład).

Uwagi:

1. Jeżeli podany fragment tekstu przekracza długością aktualnie porównywany tekst to tylko te pozycje, które występują w tekście są brane pod uwagę.
2. Jeżeli dwa porównywane teksty mają różną długość i na pozycjach występujących w tekście krótszym są jednakowe to tekst dłuższy jest traktowany jako większy (np. "abc"<"abcCf"), co jest intuicyjnie poprawne.
3. Tablice numeryczne wielowymiarowe nie mogą być sortowane instrukcjami SORTUP i SORTDOWN. Sortowane mogą być jedynie tablice numeryczne jednowymiarowe.

Poniżej podano przykład sortowania tekstów. Do tablicy tekstowej Teksty można wprowadzić do 20 linii tekstu o długości do 20 znaków każda. Linia 30 kończy wprowadzanie tekstów jeżeli wprowadzony zostanie tekst pusty:

```
10 Dim Tekst$(20,20)
20 For I=i To 20:Input "Tekst> ",Tekst$(I;)
30   If Len(Tekst$(I;)) Then Next I
40 Sortup TekstS Using 1 To I-1
50 For J=1 To I-1:Print Tekst$(J;):Next J
60 Run
```

DPEEK (funkcja)

Format: DPEEK(wyr_num[,bank])

Przykład:

```
PRINT "Adres VNTP - ";DPEEK($82)
```

Działanie funkcji DPEEK jest podobne jak funkcji PEEK, ale powoduje pobranie wartości słowa (dwóch bajtów) z lokacji wyr_num i wyr_num+1. Jest to szczególnie użyteczne przy pobieraniu zawartości komórek zawierających adresy. Dla przykładu odczyt adresu VNTP przy użyciu instrukcji PEEK wyglądałby następująco:

```
Print "Adres VNTP - ";Peek(130)+Peek(131)*128
```

Jak widać użycie DPEEK jest znacznie prostsze.

DPOKE

Format: DPOKE wyr_num1, wyr_num2 [,bank]

Przykład:

```
DPOKE 88,$8000
```

Działanie instrukcji DPOKE jest podobne jak instrukcji POKE, ale powoduje zapisanie zawartości słowa (dwóch bajtów) o lokacji wyr_num i wyr_num+1. Jest to szczególnie użyteczne przy zapisywaniu zawartości komórek zawierających adresy. Wartość wyr_num2 musi być liczbą całkowitą od 0 do 65535. W powyższym przykładzie adres $8000 został zapisany do komórek 88 i 89. Wykonanie tej operacji za pomocą instrukcji POKE byłoby znacznie bardziej skomplikowane.

MOVE

Format: MOVE wyr_num1,wyr_num2,wyr_num3 [,bank]

Przykład:

```
MOVE $DOOD,$8000,$400
```

Uwaga: Przy użyciu tej instrukcji wskazana jest ostrożność. Powoduje ona przesunięcie podanej liczby bajtów z jednej lokacji do drugiej z szybkością programu w kodzie maszynowym. Żadne adresy nie są sprawdzane co może doprowadzić np. do zawieszenia systemu lub do zniszczenia znajdującego się w pamięci programu.

Wyr_num1 określa początkowy adres bloku, który ma zostać przepisany, wyr_num2 początkowy adres zapisu, wyr_num3 ilość bajtów do przepisania (długość bloku). Znak wyr_num3 określa porządek w jakim bajty są przepisywane:

```
     Dodatnie                          Ujemne
(z)        -> (do)              (z+dług-1) -> (do+dług-1)
(z+1)      -> (do+1)       (z+dług-2) -> (do+dług-2)
     .     .     .               .     .     .     .     .
     .     .     .               .     .     .     .     .
(z+dług-1) -> (do+dług-1)           (z)    -> (do)
```

Jeżeli długość bloku do przepisania jest liczbą dodatnią to blok do którego następuje zapis może pokryć się z dolną częścią bloku źródłowego.

Jeżeli długość bloku do przepisania jest liczbą ujemną to blok do którego następuje zapis może pokryć się z górną częścią bloku źródłowego.

Uwaga: MOVE nie może automatycznie dokonać przesunięcia pomiędzy bankami. Aby wykonać taką operację należy najpierw przesunąć blok do pamięci podstawowej a następnie do innego banku.

RANDOM (funkcja)

format: RANDOM(wyr_num1 [, wyr_num2])

Przykłady:

```
RANDOM(99)
```

```
Y=RANDOM(10, 20)
```

Wartością funkcji jest liczba pseudolosowa całkowita z przedziału określonego przez wyr_num1 i wyr_num2. Jeżeli podano tylko wyr_num1 (jak w pierwszym przykładzie) generowana liczba jest z przedziału od 0 do wyr_num1-1 włącznie. Jeżeli podano wyr_num1 i wyr_num2 (jak w drugim przykładzie) to generowana liczba jest z przedziału od wyr_num1 do wyr_num2 włącznie. Funkcja korzysta z hardwarowego generatora liczb pseudolosowych o rozkładzie równomiernym.

Dlaczego stosuje się podprogramy typu PROCEDURE

Przed opisaniem instrukcji do operowania podprogramami typu PROCEDURE padano kilka wstępnych uwag i przykładów, które pozwolą się zorientować co do użyteczności tych instrukcji.

Większość spotykanych dialektów języka BASIC daje możliwość tworzenia tylko podprogramów wywoływanych przez GOSUB. Program korzystający z podprogramów wygląda wtedy tak jak poniższy program przykładowy:

```
20 Zmienna=100
30 Min=10: Max=5~0: Gosub 100
40 Wyniki=NUm
50 Min=10*Zmienna: Max=90*Zmienna: Gosub 100
60 Wynik2=Num
70 If wynik2>Zmienna*Wynik1 Then 90
80 Print ",Jestes raczej konserwaty*ny.":End
90 Print "Jestes raczej ryzykantem.":End
100 Rem "Podprogram"
110 Print: Print "Podaj liczbe od"
120 Print Min;" do ";Max;
130 Input "Wlacznie> ",Num
140 If Num>=Min And Num<=ax Then Return
150 Inverse: Print"Nie umiesz czytac?! Ta liczba jest"
160 Print "poza podanym zakresem. ":Normal
170 Goto 100
```

W niewielkich programach, na przykład takich jak powyższy, instrukcja GOSUB jest całkiem wystarczająca. Jeżeli program jest duży instrukcje jak GOSUB 3250 staje się coraz mniej oczywiste a prześledzenie programu i zrozumienie jego działania coraz trudniejsze. Atari BASIC i BASIC XE pozwalają na zapis:

```
1V Podaj=100
20 Value=100
```

```
30 Min=10:Max=10: Gosub Podaj
```

Poprzez użycie nazw podprogramów można uczynić program nieco bardziej czytelnym. Jednakże powoduje to konieczność zadeklarowania na ten cel zmiennych co zmniejsza ilość zmiennych jakie mogą zostać wykorzystane w programie (maksymalna liczba wszystkich zmiennych wynosi 128). Aby uniknąć tej trudności i stworzyć możliwość przejrzystego, strukturalnego zapisu algorytmów BASIC XE oferuje procedury (PROCEDURE). W tym wypadku nazwa jest stałą tekstową, czyli nie potrzeba dodatkowych zmiennych.

```
20 Temp=100
30 Call "Podaj z zakresu" Using 10,90 To Wynik1
50 Call "Podaj z zakresu" Using 10*Temp,90*Temp To Wynik2
70 If Wynik2<Temp*Wynik1: Pisz$="konserwatysta"
80 Else: Pisz$="ryzykantem"
90 Endif
95 Print Using "Jesteś raczej %%%%%%%%%%%%/.", Pisz$: End
100 Procedure "Podaj z zakresu" Using Min, Max
110   Local Temp: Temp=1e+90
120   While Temp<Min Or Temp>Max
130     If Temp<>1e+90: Print
140       Inverse: Print"Nie umiesz czytać?! Ta liczba jest"
150       Print „poza podanym zakresem. „: Normal
160     Endif
170     Print: Print "Podaj liczbe od "
180     Print Min; " do "; Max;
190     Input "Wlacznie> ", Num
200   Endwhile
210 Exit Temp
```

Linia 30 zawiera wywołanie procedury o nazwie "Podaj z zakresu". USING w linii 30 to określenie parametrów przesyłanych do procedury. W linii 100 po USING występują nazwy zmiennych. Jest to przesyłanie parametrów do procedury.

W linii 210 instrukcja EXIT kończy procedurę i przekazuje jako parametr wielkość zmiennej Temp. W linii wywołania procedury za słowem TO podana jest nazwa zmiennej, do której przekazana zostanie wartość parametru z procedury.

Powyższe przykłady pozwalają zorientować się w użyteczności procedur.

PROCEDURE (PROC.)

Format: PROCEDURE nazwa [USING zm1 [, zm2. . .]]

Przykłady:

```
1000 PROCEDURE "Oblicz" USING Godzina,!Tablica()
387 PROCEDURE "Wypisz komunikat." USING !Kom$
4040 PROCEDURE "Wyjscie"
```

137

Uwaga: Jeżeli zmienna jest tekstem, tablicą numeryczną, lub tekstową to jej nazwa musi być poprzedzona wykrzyknikiem (!).

Instrukcja PROCEDURE określa początek procedury wywoływanej instrukcją CALL i kończącej się instrukcją EXIT. Nazwa procedury jest dowolną stałą tekstową. Dobra praktyka programowania nakazuje, aby nazwy procedur były znaczące tj. aby po nazwie można się było zorientować co dana procedura robi.

W momencie wywołania procedury instrukcją CALL na stosie zostaje zapamiętany adres powrotu i po zakończeniu procedury wykonywana jest następna instrukcja po CALL.

Jeżeli stosowana jest procedura z parametrami (poprzez USING) wartości zmiennych zml,zm2... są wysyłane na stos, następnie zmiennym tym przypisywane są wartości podane w instrukcji wywołania (zob. CALL) i wykonanie programu przekazywane jest do procedury. Poniżej podano przykład:

```
10 X=20
20 Call "Test" Using 1217
30 Print X
40 End
70 Procedure "Test" Using X
80 Print X
90 Exit
```

W momencie wywoływania procedury nazwanej "Test" w linii 70 wartość zmiennej X jest wysyłana na stos i następnie zmiennej X jest przypisywana wartość 204. Instrukcja PRINT w linii 8O wypisze 204. Po napotkaniu instrukcji EXIT sterowanie przekazane zostaje do instrukcji następnej po CALL a poprzednia wartość zmiennej X jest pobierana ze stosu i przypisywana tej zmiennej. Instrukcja PRINT w linii 30 wypisze 20.

Oznacza to, że USING (w połączeniu z CALL i PROCEDURE) powoduje efekt podobny jak zastosowanie segmentu zmiennych lokalnych LOCAL. Dzięki temu zmienna X może być użyta zewnątrz procedury nie powodując zmiany wartości zmiennej X w segmencie wywołującym procedurę. Ponadto, co także wynika z

przedstawionego przykładu można przekazywać wartość do procedury bez znajomości nazwy zmiennej wewnątrz procedury, której ta wartość zostanie przypisana.

Jeżeli zmienne zml,... , których wartości są przekazywane do procedury nie są zmiennymi numerycznymi (lecz np. tablicami numerycznymi, zmiennymi tekstowymi lub tablicami tekstowymi) proces przebiega inaczej. Dla wyjaśnienia tego problemu konieczna jest informacja, w jaki sposób wartość zmiennej jest umieszczana na stosie. BASIC XE jako wartość przekazywanej zmiennej traktuje zawartość odpowiedniej lokacji w wewnętrznej tablicy wartości zmiennych VVTP (ang. Variable Value Table Place). Stanowi ją osiem bajtów dla każdej zmiennej - bajt flag, numer zmiennej (0-127) i sześć bajtów informacyjnych. Dla zmiennych numerycznych sześć bajtów informacyjnych zawiera wartość zmiennej. Natomiast w wypadku tablic numerycznych, zmiennych tekstowych i tablic tekstowych bajty informacyjne zawierają adres w pamięci i charakterystykę aktualnych danych. Na przykład w wypadku zmiennych tekstowych charakterystykę tę stanowi maksymalny rozmiar zmiennej DIM i aktualna długość LEN. Natomiast właściwa zawartość zmiennej tekstowej znajduje się pod określonym adresem pamięci. W przypadku tablic numerycznych i tekstowych bajty informacyjne zawierają adres i rozmiar DIM.

Wykonanie instrukcji CALL polega na wysłaniu na stos ośmiu bajtów pobranych z VVTP. Tak więc instrukcja CALL nie powoduje wysłania prawdziwej wartości tablic numerycznych, zmiennych tekstowych i tablic tekstowych a jedynie bajtów znajdujących się w VVTF. Wartość opisanych typów zmiennych nie będących zmiennymi numerycznymi zostanie przekazana do procedury jeżeli ich nazwy zostaną poprzedzone znakiem wykrzyknika (!) w instrukcji wywołania. Pokazuje to następujący przykład:

```
10 Przyj$"Jedzenie jest przyjemne.": X$="Dobrze?"
20 Call "Przyjemnie" Using !Przyj$
30 Print PrzyjS,X$
40 End
50 Rem "Procedura"
60 Procedure "Przyjemnie" Using !X$
70   Print Przyj$,X$
80   X$(1,5)="Marze"
81   90 Exit
```

Po wykonaniu instrukcji PRINT w linii 70 widać, że wartość Przyj$ została skopiowana do X$, na ekranie ukazuje się:

Jedzenie jest przyjemne. Jedzenie jest przyjemne.

Natomiast wykonanie instrukcji PRINT w linii 30 (następujące po zakończeniu procedury instrukcją EXIT) powoduje wypisanie na ekranie:

Marzenie jest przyjemne. Dobrze?

Dzieje się tak ponieważ wartość Przyj$ została skopiowana do X$, adres Przyj$ znajduje się teraz w lokacji X$ VVTP. Dlatego zmiana dokonana na X$ wewnątrz procedury spowodowała zmianę wartości Przyj$ poza procedurą.

Uwaga techniczna: W języku informatyków przesłanie parametru do procedury w wypadku zmiennych numerycznych nosi nazwę przesłania przez wartość, w pozostałych wypadkach przesłania przez odwołanie.

Uwagi o wykorzystaniu PROCEDURE

BASIC XE wymaga, aby parametry w wywołaniu (CALL) i w deklaracji procedury (PROCEDURE) były tego samego typu. W przeciwnym wypadku występuje błąd nr 24 "USING Type Mismatch". BASIC XE nie sprawdza, czy liczba parametrów w instrukcji wywołania i w deklaracji procedury jest taka sama. Jeżeli CALL zawiera zbyt wiele parametrów zostają one zignorowane. Jeżeli CALL zawiera zbyt mało parametrów brakującym parametrom zostaje przypisana wartość zero. Może to spowodować wystąpienie błędu nr 24.

Przesyłając jako parametr do procedury zmienną tekstową należy zachować ostrożność przy zmienianiu jej wartości. Z VVTP aktualna długość LEN zmiennej zostaje przekazana do procedury. Jeżeli długość tej zmiennej zostanie zmieniona wewnątrz procedury przy powrocie (EXIT) nowa długość nie jest automatycznie kopiowana do lokacji VVTP opisującej zmienną będącą parametrem w instrukcji CALL. Na przykład modyfikacja linii 80 przedstawionego powyżej przykładu na:

```
80 X$="XXX"
```

spowoduje wypisanie w linii 30:

XXXenie jest przyjemne. Dobrze?

Zmiana zawartości X$ w linii 80 spowodowała zmianę zawartości Przyj$, ale po opuszczeniu procedury nowa długość nie została umieszczona w VVTP na pozycji odpowiadającej Przyj$.

Jednym z możliwych rozwiązań tego problemu jest przekazanie parametru po zakończeniu procedury w instrukcji EXIT. Program będzie działał poprawnie jeśli oprócz modyfikacji linii 80 zostaną jeszcze zmienione linie 20 i 90:

```
20 Call "Przyjemnie" Using !Przyj$ To !Przyj$
90 Exit !X$
```

Poważne problemy mogą wystąpić w wypadku próby przekazania z procedury wartości tekstu lub tablicy, który nie był parametrem wejściowym. Na przykład uruchomienie programu:

```
100 Call "Proc" To !A$
110 Call "Proc" To !B$
120 Print A$,B$:End
300 Procedure "Proc"
310   Input "Napisz cos> ",Linia$
320 Exit !Linia$
```

spowoduje wypisanie przez instrukcję PRINT w linii 120 dwa razy tego samego tekstu, wprowadzonego za drugim razem. Kiedy Linia$ jest przekazywana z procedury przepisywane są jej parametry z VVTP (czyli także adres początku) w lokację VVTP określającą A$ za pierwszym razem i B$ za drugim razem. Tak więc A$ i B$ odwołują się do tego samego miejsca w pamięci przy wykonywaniu linii 120.

Poprawnym rozwiązaniem jest przekazanie zmiennej do procedury przez USING i następnie przekazanie z powrotem przez EXIT.

Uwaga: PROCEDURE musi być pierwszą instrukcją w linii. W przeciwnym wypadku CALL nie odnajdzie procedury w programie. Należy także zwrócić uwagę, aby nie dopuścić do przypadkowego "przejścia" programu do procedury, kończąc segment poprzedzający procedurę instrukcją GOTO, STOP, END, RETURN lub EXIT. Zalecane jest zgrupowanie wszystkich procedur po programie głównym (wywołującym) zakończonym instrukcją END.

EXIT

Format: EXIT [par1 [,par2...]]

Przykłady:

```
390 EXIT 10*Alfa
799 EXIT Alfa, ! Nazwa$
21990 EXIT !Odwrot(),X,Beta
835 EXIT
```

Parametry par mogą być stałymi, wyrażeniami bądź zmiennymi.

Uwaga: Jeżeli któryś z parametrów par jest tablicą numeryczną, zmienną tekstową, stałą tekstową lub tablicą tekstową - jej nazwa musi być poprzedzona znakiem wykrzyknika (!).

EXIT powoduje wykonanie następujących czynności:

1. Jeżeli na stosie znajdują się wysłane tam wartości parametrów (np. w związku z przekazywaniem parametrów, lub w

związku z użyciem LOCAL) zostają one przywrócone odpowiednim zmiennym (tj. przepisane do VVTP).

2. Jeżeli po EXIT występują parametry do przekazania zostają one przypisane odpowiednim zmiennym wyszczególnionym w instrukcji CALL po słowie kluczowym TO.
3. EXIT sprawdza, czy podprogram został wywołany przez CALL, czy przez GOSUB. Jeżeli wywołany został przez GOSUB powrót następuje jak w wypadku instrukcji RETURN.

Użycie EXIT z parametrami do przekazania nie powoduje wystąpienia błędu, jeżeli podprogram został wywołany przez GOSUB. Parametry są ignorowane. Także wtedy, jeżeli ilość parametrów przekazywanych z powrotem instrukcją EXIT jest większa niż występująca po TO w odpowiedniej instrukcji CALL dodatkowe parametry są ignorowane (ale odpowiednie parametry muszą być tego samego typu).

Możliwe jest wykorzystanie instrukcji POP do opuszczenia procedury bez użycia EXIT.

CALL

Format: CALL nazwa [USING par1 [,par2..][TO zm1[,zm2..]

Przykład:

```
10 CALL "Test"
720 CALL "Oblicz" USING !Zmienne() TO Wynik
800 CALL "Wczytaj" TO Liczba
1100 CALL Proc$ USING 7, !A$ To X
```

Jeżeli zmienna lub parametr są tekstem, tablicą tekstową lub tablicą numeryczną, muszą być poprzedzone znakiem wykrzyknika (!). Użycie instrukcji CALL zostało jut przedyskutowane w poprzednich rozdziałach, gdzie także podano kilka przykładowych programów.

W przeciwieństwie do instrukcji PROCEDURE, w której nazwa musi być stałą tekstową nazwa występująca w instrukcji CALL może być zarówno stałą jak i zmienną tekstową.

Uwagi techniczne:

1. Możliwy stopień zagnieżdżenia procedur jest zależny od pozostałej wolnej jako stos pamięci. Tak jak GOSUB i WHILE instrukcja CALL zużywa cztery bajty na zapamiętanie adresu powrotu i ponadto dwanaście bajtów dla przekazania każdego parametru.
2. W trybie normalnym CALL bez parametrów jest porównywalne pod względem szybkości z GOSUB . W trybie FAST GOSUB działa szybciej niż CALL, nawet bezparametrowe. Mimo to przy całościowym spojrzeniu na problem wywołania podprogramu i

przekazania parametrów użycie procedur wywoływanych przez CALL i tak jest opłacalne.

CP

Format: CP

Działanie instrukcji CP jest identyczne jak DOS w Atari BASIC.

DODATEK A

Odmiany BASIC'a XE

BASIC XE powstał w firmie OSS (Operating Systems Software) jako następca BASIC'a XL (nie mylić z Turbo Basic XL). W miarę rozwoju tego języka programowania powstało kilka jego odmian. Głównie przeznaczony był do sprzedaży w formie kartridża.

Początkowo była to wersja przeznaczona do współpracy wyłącznie ze stacją dyskietek, zaś ostatnia jego wersja współpracuje również z magnetofonem. Chodzi tu o sposób doczytywania instrukcji z pamięci zewnętrznej, instrukcji nie zapisanych w ROM kartridża.

BASIC XE może być też wmontowany do wnętrza komputera, wtedy jego włączenie i wyłączenie odbywa się za pomocą dodatkowego przełącznika montowanego najczęściej z tyłu komputera.

Po włożeniu kartridża lub włączeniu BASIC'a XE przełącznikiem należy do stacji dyskietek włożyć dyskietkę z DOS 2.5 zawierającą dodatkowe instrukcje. W przypadku magnetofonu należy oczywiście włożyć odpowiednią kasetę i ustawić ją na początku programu.

Gdy do komputera podłączona jest stacja dyskietek z włożoną dyskietką z DOS 2.5, to po włączeniu komputera załaduje się DOS.SYS i pojawi się ekran z możliwością wyboru typu pamięci zewnętrznej.

```
BASIC XE version 7.2

Press OPTION to boot with DISC DRIVE
Press SELECT to boot with RECORDER
Press START  to use ONLY CARTRIDGE
```

Po dokonaniu odpowiedniego wyboru zacznie się proces ładowania.

Wspomniałem, że BASIC XE jest następcą BASIC'a XL. Tak jest w istocie, bowiem BASIC XL jest zgodny z XE poza instrukcjami: BLOAD, BSAVE, CALL, PROCEDURE, EXIT, EXTEND, HEX$, HITCLR, INVERSE, LEFT$, LOCAL, MID$, RIGHT$, SORTDOWN, SORTUP i USING, których nie posiada.

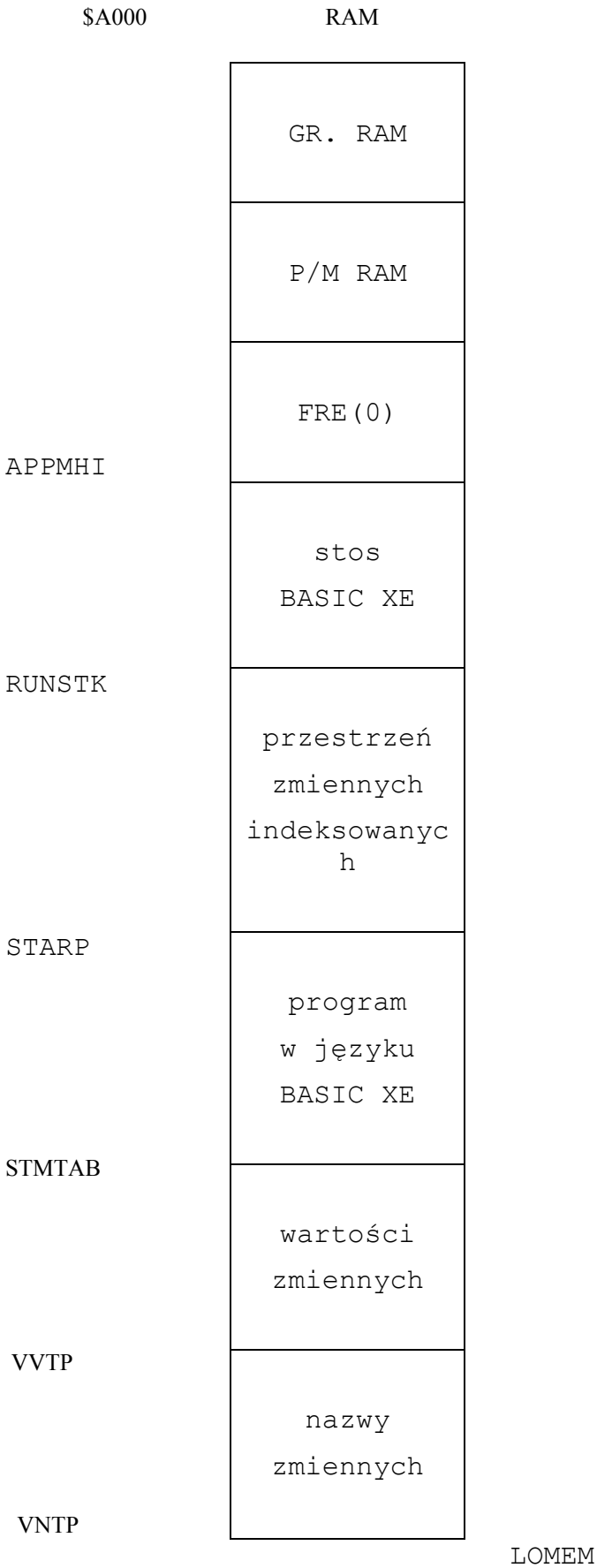
DODATEK B

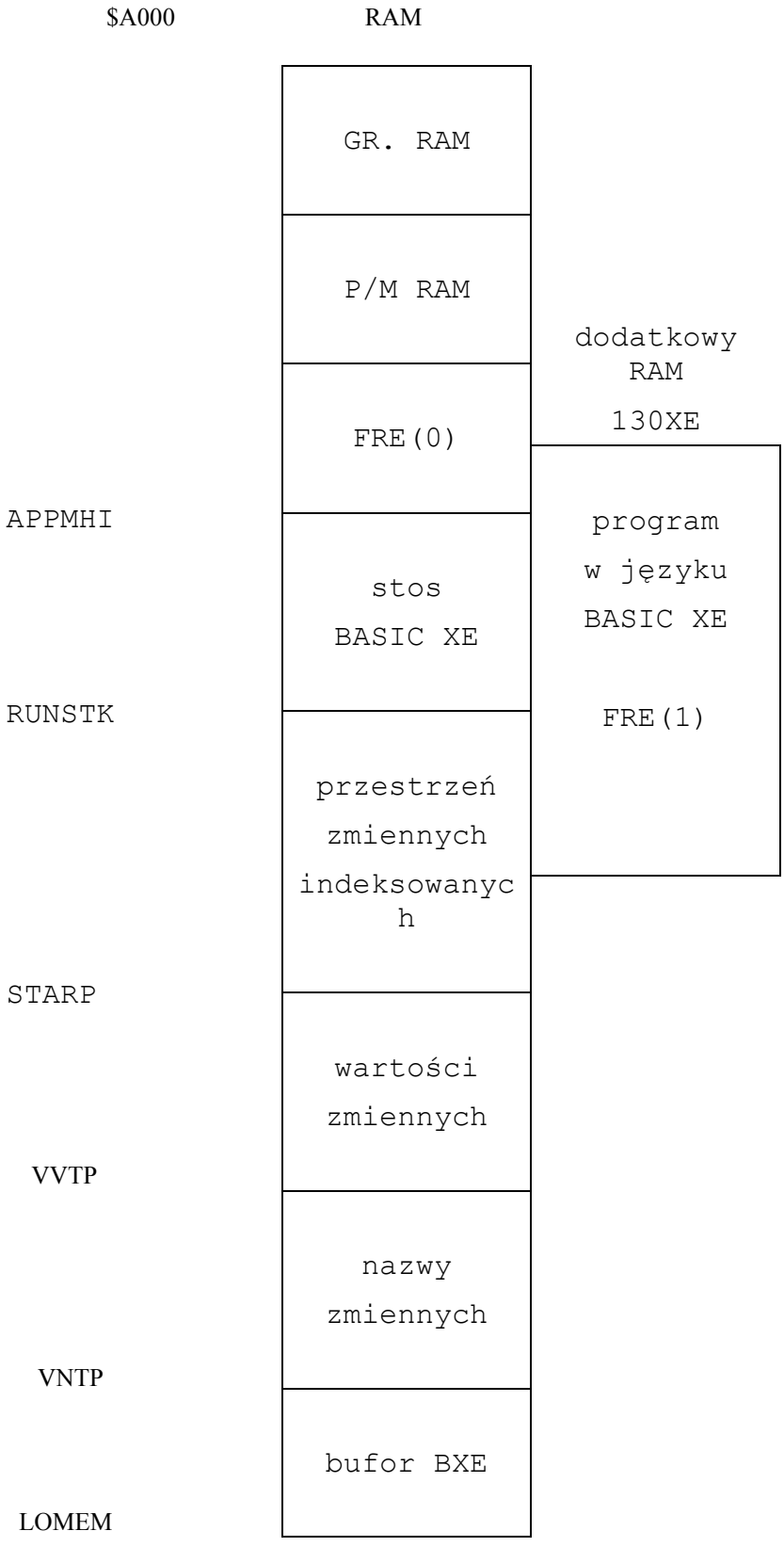
Mapa pamięci BASIC XE

W dodatku podano ważniejsze informacje o wykorzystaniu pamięci przez BASIC XE. Występujące w opisie 'AtB' oznacza Atari BASIC, 'BXE' - BASIC XE. Należy podkreślić, że jedyne istotne z punktu widzenia programowania w języku BASIC XE lokacje to LOMEM, STOPLN, ERRSAV i PTABV. Znajomość tych adresów i wykorzystanie instrukcji POKE nie jest jednak konieczne, ponieważ w języku BASIC XE są do dyspozycji funkcje i instrukcje umożliwiające operacje na tych komórkach.
Uwaga: Poza wyszczególnionymi poniżej wszystkie komórki pamięci na stronie zerowej $80-$FF są używane przez BASIC XE.

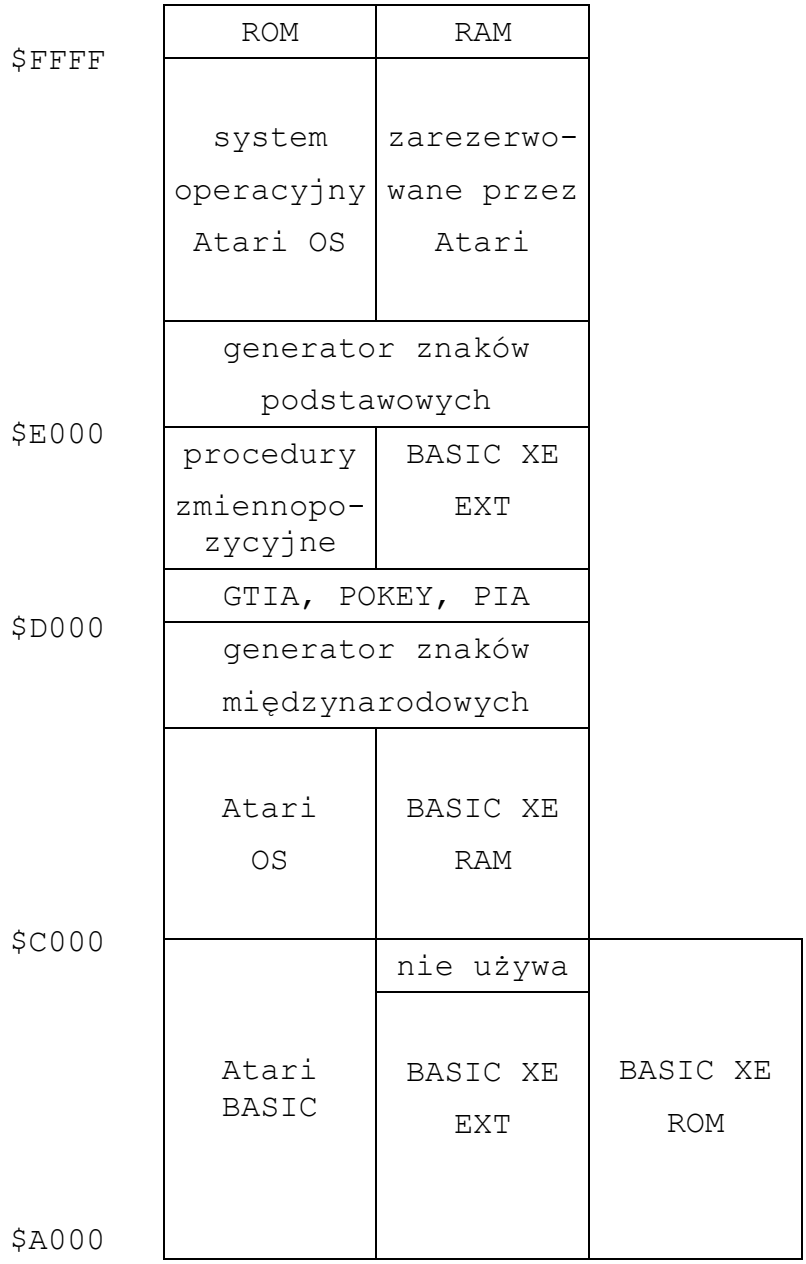
Adres	Nazwa	Znaczenie
$E-$F	APPMHI	systemowy wskaźnik wolnej pamięci
$20-$2F	ZIOCB	wykorzystane przez operacje zmiennopozycyjne
$43-$49	FMSZPG	wykorzystane przez operacje zmiennopozycyjne
$80,$81	LOMEM	wskaźnik dołu pamięci
$82,$83	VNTP	wskaźnik tablicy nazw zmiennych
$84,$85	VNTD	wskaźnik końca tablicy nazw zmiennych + 1;
$86,$87	VVTP	wskaźnik tablicy wartości zmiennych
$88,$89	STMTAB	wskaźnik tablicy instrukcji
$8A,$8B	STMCUR	wskaźnik aktualnej instrukcji
$8C,$8D	STARP	wskaźnik tablicy wartości zmiennych indeksowanych
$8E,$8F	RUNSTK	wskaźnik stosu BASIC
$90,$91	MEMTOP	wskaźnik szczytu pamięci
$BA,$BB	STOPLN	wskaźnik linii zatrzymania programu
$C3	ERRSAV	rejestr numeru błędu
$C9	PTABV	wskaźnik ustawienia tabulatora
$CB-$D1	------	nie używane przez BXE
$D~-$D9	FR00	rejestr zmiennopozycyjny 1
$EO-$E3	FR1	rejestr zmiennopozycyjny 2
$480-$57F	------	używane przez BXE przy realizowaniu instrukcji będących poszerzeniami w stosunku do AtB.
$580-$6F7	------	normalnie nie używane przez BXE, ale INPUT i ENTER mogą spowodować zmianę zawartości tego obszaru pamięci.
$680-$6FF	------	nie używane przez BXE. Dobre miejsce dla podprogramów w kodzie maszynowym.
$700-LOMEM		używane przez DOS i handlery niektórych urządzeń (np. R:).

Dwa poniższe rysunki pokazują użycie pamięci pomiędzy LOMEM i początkiem obszaru cartridge ($A000). Pierwszy z nich jeżeli nie użyto EXTEND, drugi jeżeli użyto EXTEND.


$A000
RAM
GR. RAM
P/M RAM
FRE(0)
APPMHI
stos
BASIC XE
RUNSTK
przestrzeń
zmiennych
indeksowanyc
h
STARP
program
w języku
BASIC XE
STMTAB
wartości
zmiennych
VVTP
nazwy
zmiennych
VNTP
LOMEM


$A000
RAM
GR. RAM
P/M RAM
FRE(0)
APPMHI
stos
BASIC XE
RUNSTK
przestrzeń
zmiennych
indeksowanyc
h
STARP
wartości
zmiennych
VVTP
nazwy
zmiennych
VNTP
bufor BXE
LOMEM
dodatkowy
RAM
130XE
program
w języku
BASIC XE
FRE(1)

Następny rysunek pokazuje wykorzystanie pamięci przez BASIC XE od adresu początku obszaru dołączanych modułów ROM (ang. cartridge) do $FFFF, czyli do końca przestrzeni adresowej. Obszary oznaczone 'BASIC XE EXT' są używane przez BASIC XE tylko wtedy, gdy podczas włączenia komputera w stacji dysków D1 znajdowała się dyskietka zawierająca rozszerzenia.



DODATEK C

Kompatybilność z Atari BASIC

W zasadzie BASIC XE jest w pełni kompatybilny z Atari BASIC. Oznacza to, że każdy program napisany w języku Atari BASIC działa poprawnie jeżeli zostanie załadowany i uruchomiony przez BASIC XE. Istnieje jednak kilka drobnych różnic przedstawionych poniżej, które mogą spowodować, że program nie będzie działał poprawnie.

Nazwy zmiennych

Ze względu na większą ilość słów kluczowych w języku BASIC XE niż w Atari BASIC może się zdarzyć, że nazwy niektórych zmiennych zostaną zinterpretowane niepoprawnie. Miejsca te można łatwo zauważyć listując program (instrukcja LIST) na ekran lub drukarkę i zastosować w tych miejscach instrukcja LET (lub zmienić nazwę zmiennej). Na przykład:

```
100 NUMER=7
```

jest poprawna linia w Atari BASIC. Natomiast NUM jest słowem kluczowym w języku BASIC XE i linia ta wygląda po wylistowaniu następująco:

```
100 Num Er=7
```

Jeżeli w programie nie występuje zmienna Er powyższa linia odpowiada:

```
100 Num 0
```

co jest błędne z punktu widzenia efektów działania programu. Rozwiązaniem tego problemu jest jak wspomniano użycie LET w instrukcjach przypisania. Należy jednak zwrócić uwagę, że nazwy zmiennych w języku BASIC XE nie mogą być identyczne z nazwami funkcji. W takim wypadku należy zmienić nazwę zmiennej na inną (np. zmienić nazwę BUMP na BMP lub VBUMP itp.).

Szybkość wykonania programu

Jedną z ważniejszych zalet języka BASIC XE w stosunku do Atari BASIC jest znacznie większa szybkość wykonania programu. V pewnych sytuacjach (gry!) może to jednak być niepożądane. Jedynym rozwiązaniem w takim wypadku jest zastosowanie opóźnień w programie. Opóźnienia można zrealizować np. używając procedury odczekującej pewien czas i wywoływanie jej z odpowiednich punktów programu. Poniżej podano przykład procedury z parametrem realizującej żądane opóźnienie:

```
1000 Procedura "Czekaj" Using Czas
1010   Local Czekaj
1020   For Czekaj=0 To Czas: Next Czekaj
1030 Exit
```

Przykładowa linia wywołania takiej procedury ma postać:

```
100 Call "Czekaj" Using 10
```

Konflikty wykorzystania pamięci

BASIC XE nie zmienia sposobu wykorzystania żadnej z lokacji w pamięci opublikowanej w następujących książkach:

"Atari Basic Reference Manual" - Atari, Inc.
"De Re Atari"
"Mapping The Atari" – COMPUTE! books
"Master Memory Map" - Educational Software, Inc.

Istnieje jednak niewielka szansa napotkania programu stworzonego wyłącznie do działania przy wykorzystaniu Atari BASIC. Spowodowane jest to niepublikowanymi przez firmę Atari następującymi faktami:

1. Atari BASIC używa pewnych lokacji na stronie zerowej w niektórych sytuacjach.
2. Niektóre lokacje na stronie zerowej mają specjalne znaczenie dla Atari BASIC.
3. Adresy niektórych procedur zawartych w pamięci stałej ROM zawierającej interpreter są inne w wypadku Atari BASIC i BASIC XE.

Stworzenie interpretera dysponującego większymi możliwościami niż Atari BASIC i ponadto znacznie szybszego nie byłoby bez wprowadzenia pewnych zmian w wykorzystaniu pamięci możliwe. Ponieważ powyższe fakty nie były, jak wspomniano, opublikowane przez firmę Atari, a potrzeba nietypowego wykorzystania pamięci rzadko zachodzi przy tworzeniu programów, te z nich, które ze względu na konflikty wykorzystania pamięci nie działają poprawnie przy wykorzystaniu interpretera BASIC XE są ogromnie rzadkie.

Automatyczne przypisanie wymiaru zmiennym tekstowym

BASIC XE przypisuje automatycznie wymiar 40 zmiennym tekstowym, których wymiary nie zostały zadeklarowane instrukcją DIM. Nie ma to żadnego wpływu na wykonanie programów napisanych w Atari BASIC. Jeżeli z jakichkolwiek względów byłoby to pożądane, to należy pamiętać, że użycie SET

11,0 powoduje zaniechanie automatycznego przypisywania rozmiaru niezadeklarowanym zmiennym tekstowym.

Postać listowanego programu

BASIC XE listuje program zaznaczając jego strukturę. Jeżeli byłoby to niepożądane, to przejście do listowania w taki sposób, jak robi to Atari BASIC zapewnia wykonanie SET 12,0.

DODATEK D

Możliwości wykorzystania dodatkowej pamięci RAM

BASIC XE pozwala na wykorzystanie całej pamięci Atari 130XE. Dodatkowa pamięć RAM (64K) może być wykorzystana jako dysk wirtualny (RAMDISK), jako obszar pamięci programu (w trybie EXTEND), bądź jako pamięć pozwalająca przechowywać inne informacje wykorzystywane przez program (np. kilka przełączanych ekranów itp.). W wypadku komputera z pamięcią rozbudowaną do 256K RAM można używać BASIC XE w trybie EXTEND i w pozostałych 128K utworzyć dysk wirtualny (RAMDISK), lub jak powyżej podano jako pamięć pozwalającą przechowywać inne informacje wykorzystywane przez program (np. kilka przełączanych ekranów itp.), pracując jednoczenie w trybie EXTEND, co w wypadku Atari 130XE nie jest możliwe.

Poniżej podano przykład ciekawego wykorzystania dodatkowej pamięci (64K Atari 130XE) do przechowywania ekranów w trybie graficznym. Aby wykorzystać ten program nazwany "OBRAZKI" konieczny jest jeden lub kilka rysunków zapisanych na dyskietce w formacie stosowanym przez znany program Micro-Illustrator. Inne programy, na przykład Koala Pad czy Atari Artist używają innego formatu zapisu pamięci ekranu jako zbiór na dyskietce, ale dają możliwość zapisu Lak jak Micro-Illustrator poprzez wciśnięcie CONTROL-SHIFT-INSERT (wciśnięcie klawisza INSERT przy wciśniętych klawiszach CONTROL i SHIFT). Zbiory zawierające rysunki mogą mieć dowolne nazwy, ale muszą mieć wyróżnik *.PIC. Najkorzystniej ze względu na to, że zbiory zawierające rysunki są dość duże, umieścić je na dysku zawierającym jedynie DOS i opisywany program. Poniżej podano krótki komentarz dla tego programu:

180 Zmienna tekstowa Zbiory$ jest używana do zapamiętania nazw zbiorów odczytanych z katalogu dyskietki (do ośmiu zbiorów).

190 Rysunek w formacie programu Micro-Illustrator to 7680 bajtów pamięci ekranu.

200 Stan klawiszy START, SELECT i OPTION jest sprawdzany przez badanie zawartości komórki pamięci $D01F. Jeżeli wciśnięty jest klawisz START to najmniej znaczący bit jest zerowy.

240 Odczyt pierwszej części katalogu dyskietki w stacji D1.

250 Maksymalnie mogą zostać przeczytane nazwy ośmiu zbiorów.

260,270 Sprawdzenie wczytanych nazw zbiorów. Jeżeli liczba zbiorów z obrazami jest większa niż 8 - odczyt jak wiele sektorów jest zajętych na dyskietce i wyjście z pętli.

280,290 Tworzenie nazw zbiorów w takiej postaci jaka będzie potrzebna do wykonania instrukcji OPEN.

300,310 Ustawienie wartości zmiennej sterującej w zależności od ilości przeczytanych zbiorów.

320 Zamknięcie kanału po odczycie katalogu.

360,370 Ustawienie kolorów dla obrazów monochromatycznych. Jeżeli obrazy są kolorowe należy zmienić tryb graficzny na 31 i użyć odpowiednich SETCOLOR.

380,390 Odczytanych zostanie tyle zbiorów, ile znaleziono w katalogu.

400 Odczyt obrazu. Komórki $58, $59 zawierają adres początku pamięci ekranu.

440 Umieszczenie obrazu 1 i 2 w banku 0, 3 i 4 w banku 1 itd.

470 Użycie tej instrukcji MOVE jest bezpieczne, ponieważ pamięć ekranu znajduje się powyżej $7FFF. Jeżeli przy wykonaniu programu obniżona zostanie wartość wskaźnika HIMEM operacja może nie przebiegać poprawnie!

480,490 Koniec wczytywania jednego zbioru.

500 Wszystkie zbiory zostały odczytane z dyskietki i umieszczone w dodatkowej pamięci.

570 Ta pętla WHILE może zostać przerwana tylko klawiszem BREAK lub RESET.

600-620 Wybór obrazu przy wykorzystaniu funkcji RANDOM.

670-700 Przeniesienie obrazu z dodatkowych 64K pamięci do pamięci ekranu .

740 Oczekiwanie na wciśnięcie klawisza START.

A oto program:

```
100 Rem ************
110 Rem *          *
120 Rem *  OBRAZKI *
130 Rem *          *
140 Rem ************
150 Rem
180 Dim Zbiory$(8,20),Zbior$(20)
190 Rozmiar=40*192
200 Klawisze=$d01f: Start=$01
210 Rem
220 Rem znajdz wszystkie zbiory z obrazami
230 Rem
240 Open #1, 6, 0, "D1:*.PIC"
290 For Obraz=1 To 8
260   Input #1,Zbior$
270   If Zbior$(2,2)<>" " Then Pop: Goto 300
280   Zbiory$(Obraz;)="D1:",Zbior$(3,10)," "
290   Zbiory$(Obraz;Find(Zbiory(Obraz;)," ",8))=".PIC"
300 Next Obraz
310 Ile=Obraz-1
```

```
320 Close #1
330 Rem
340 Rem wczytaj znalezione zbiory
350 Rem
360 Graphics 24
370 Setcolor 2,6,O: Setcolor 4,6,O:Setcolor 1,6,8
380 For Obraz=1 To Ile
390   Open #1,4,0,Zbiory$(Obraz;)
400   Bget #1, Dpeek($58), Rozmiar
410   Rem
420   Rem przepisz obraz do dodatkowej pamięci
430   Rem
440   Bank=Int((Obraz-1)/2)
450   Adres=$4000
460   If Obraz&1=0 Then Adres=$6000
470   Move Dpeek($58),Adres,Rozmiar,Bank
480   Close #1
490 Next Obraz
500 Rem
510 Rem teraz mozna ogladac obrazy
520 Rem
530 Stary=O:Obraz=0
540 Rem
550 Rem petla nieskonczona
560 Rem
570 While 1
600 While Obraz=Stary
610 Obraz=Random(1,Ile)
620 Endwhile
630 Stary=Obraz 6
640 Rem
650 Rem przepisz z pamięci dodatkowej do pamięci ekranu
660 Rem
670 Bank=Int((Obraz-1)/2
680 Adres=$4000
690 If Obraz&1=O Then Adres=$6000
700 Move Adres,Dpeek($58),Rozmiar,Bank
710 Rem
720 Rem teraz mozna ogladac
730 Rem
740 While Peek(Klawisze)&Start=0: Endwhile
790 Endwhile
```

DODATEK E

Komunikaty o błędach

W dodatku podano numery błędów i komunikaty ukazujące się po wystąpieniu błędu na ekranie oraz opis możliwej przyczyny wystąpienia błędu.

1 BREAK key not TRAPed

Występuje, jeżeli użyto SET 0,1 i w czasie wykonania programu wciśnięty został klawisz BREAK. Możliwa własna obsługa błędu poprzez instrukcję TRAP.

2 Memory Full

Została wykorzystana cała dostępna pamięć. Należy zmodyfikować program zmniejszając liczbę instrukcji lub wymiary zmiennych indeksowanych.

3 Value Out of Range

Wartość zmiennej lub wyrażenia poza dopuszczalnym zakresem.

4 Too Many Variables

Zbyt dużo zmiennych, tj. więcej niż dopuszczalne 128.

5 Acces Past String DIM

Przekroczony zadeklarowany wymiar zmiennej tekstowej.

6 No DATA to READ

Próba odczytu instrukcję READ po końcu danych w instrukcji (instrukcjach) DATA.

7 Val>32767

Wystąpił numer linii większy od 32767 lub mniejszy od zera. Niektóre inne instrukcje (np. POINT) mogą także spowodować wystąpienie błędu nr 7.

8 INPUT/READ Type Mismatch

Wartość wczytana instrukcję INPUT lub READ nie odpowiada typowi zmiennej, której ma być przyporządkowana (np. próba wczytania tekstu do zmiennej numerycznej)

9 DIMensioning

Próba użycia niezadeklarowanej tablicy lub próba wykonania instrukcji DIM dla zmiennej, której wymiar został już określony instrukcją DIM.

10 Expression too Complex

Zbyt skomplikowane wyrażenie. Należy podzielić wyrażenie na kilka instrukcji.

11 Overflow/Underflow

W wyniku operacji w procedurze arytmetyki zmiennopozycyjnej wystąpiła za mała lub zbyt duża liczba (np. próba dzielenia przez zero).

12 Line Not Found

Nie ma w programie linii wskazanej w instrukcji GOTO, GOSUB lub IF - THEN.

13 NEXT without FOR

Napotkana instrukcja NEXT bez właściwej instrukcji FOR. Błąd ten może także wystąpić jako wynik niewłaściwego użycia instrukcji POP.

14 Line too Long or Complex

Napotkana linia jest zbyt długa, lub zbyt skomplikowana, aby mogła być przetworzona przez BASIC XE. Rozwiązaniem jest podzielenie linii na kilka krótszych.

15 Line Not Found

Wystąpiło NEXT lub RETURN gdy właściwa instrukcja FOR lub GOSUB została skasowana. Błąd ten może wystąpić np. w wypadku gdy działanie programu zostało przerwane instrukcją STOP, linia została skasowana a następnie działanie programu wznowiono instrukcją CONT.

16 RETURN without GOSUB

Napotkana instrukcja RETURN bez wywołania podprogramu instrukcją GOSUB. Błąd ten może także wystąpić jako wynik niewłaściwego użycia instrukcji POP.

17 Bad Line

Linia programu nie jest zgodna ze składnią języka BASIC XE (np. 110 ERROR - ...).

18 Not a Number

Pierwszy znak argumentu funkcji VAL nie jest cyfrą.

19 Too Big To Load

Ładowany program nie mieści się w dostępnej pamięci (może wystąpić jeżeli np. zmieniono LOMEM, użyto inny DOS, zajmujący więcej pamięci itp.).

20 Invalid Channel

Zły numer urządzenia w instrukcji we/wy (mniejszy od 0 lub większy od 7).

21 File Not LOAD Format

Próba ładowania instrukcją LOAD zbioru, który nie został utworzony instrukcja SAVE.

22 USING String Too Big

Występuje jeżeli określenie formatu w instrukcji PRINT USING jest dłuższe nil 255 znaków, lub jedno pole w formacie jest dłuższe niż 59 znaków.

23 USING Value Too Big

Wartość przeznaczona do wypisania instrukcją PRINT USING jest większa niż 1E+50.

24 USING Type Mismatch

Format pól w instrukcji PRINT USING nie odpowiada typom wartości do wypisania (np. próba wypisania wartości numerycznej na pole określone jako tekstowe).

25 RGET DIM Mismatch

Tekst wczytywany instrukcją RGET nie odpowiada długością zmiennej tekstowej do której ma być przypisany.

26 RGET Type Mismatch

Rekord wczytywany instrukcja RGET i zmienna do której ma być przypisany nie są tego samego typu.

28 Invalid Structure

Napotkana instrukcja ENDIF lub ENDWHILE bez właściwej instrukcji IF lub WHILE.

29 P/M # Out of Range

Zły numer obiektu typu P lub M. Właściwe numery dla obiektów typu P są od 0 do 3, dla obiektów typu M od 4 do 7.

30 P/M Graphics Not Active

Użyto instrukcji operujących grafiką P/M bez inicjalizacji grafiki P/M instrukcją PMG. 1 lub PMG. 2.

32 ENTER not TRAPed

Koniec wprowadzania programu instrukcją ENTER. Występuje, jeżeli użyto SET 9,1.

34 Can't NUM/RENUM

Wyr_num1 lub wyr_num2 w instrukcji NUM albo RENUM równe zeru.

35 Can't NUM/RENUM

W czasie przenumerowywania programu przekroczony został maksymalny numer linii (32767).

40 String Type Mismatch

Próba użycia zmiennej tekstowej jako tablicy tekstowej bądź odwrotnie.

65 EXTENDed Memory Not Available

Próba załadowania programu zapisanego jako program w trybie EXTEND lub wykonania instrukcji EXTEND jeżeli komputer nie posiada wystarczającej pamięci.

100 Extension not Installed!

Próba użycia instrukcji dostępnej tylko jeżeli przy włączaniu komputera w stacji D1 znajdowała się dyskietka BASIC XE.

129 Channel Already OPEN

Próba otwarcia kanału, który został już otwarty.

130 No Device Handler

Nie ma podanego urządzenia w tabeli dostępnych urządzeń CIO. Podano zły symbol urządzenia, lub nie podano go wcale (np. podano tylko nazwę zbioru dyskowego bez Dn:).

131 Write Only

Próba odczytu z kanału, który został otwarty tylko do zapisu.

132 Bad Device Cmd

Podany kod operacji we/wy nie jest dopuszczalny dla danego urządzenia. Może wystąpić w niewłaściwie użytej instrukcji XIO lub OPEN.

133 Channel not OPEN

Próba wykonania operacji przez kanał, który nie został otwarty.

135 Read Only

Próba zapisu do kanału, który został otwarty tylko do odczytu.

136 End-Of-File

Osiągnięty koniec zbioru. Nie ma więcej danych.

138 Device Timeout

Skończył się limit czasu przeznaczony na odpowiedź dla danego urządzenia. Przyczyny takiej sytuacji to na ogół:

- nie ma takiego urządzenia
- urządzenie jest niewłaściwie podłączone do komputera
- urządzenie nie ma włączonego zasilania
- urządzenie jest niesprawne

W przypadku magnetofonu może to ponadto oznaczać źle ustawioną bądź wadliwą taśmę, złe ustawienie głowicy, bądź prędkości obrotowej.

139 Device NAK

Urządzenie nie odpowiada. Przyczyną może być niewłaściwe podłączenie urządzenia, błąd w parametrach sterujących lub niesprawność urządzenia.

141 Screen Position

Próba wykonania operacji graficznej na pozycji, która nie istnieje w danym trybie graficznym.

144 Device Done

Urządzenie nie jest w stanie wykonać żądanej czynności (np. próba zapisu na zabezpieczonej dyskietce).

145 Invalid GR Mode

Próba użycia nieistniejącego trybu graficznego.

147 No Memory for GR Mode

Nie ma dostatecznej ilości wolnej pamięci dla żądanego trybu graficznego.

160 Invalid Drive #

DOS nie rozpoznaje podanego numeru stacji dysków. Podano zły numer, lub stacja nie jest włączona.

161 Too Many OPEN Files

DOS nie ma więcej buforów pozwalających na otwarcie następnego zbioru.

162 Disk Full

Nie ma więcej miejsca na dyskietce.

165 Bad File Name

Nielegalna nazwa zbioru. Informacje na temat nazw dopuszczonych przez dany DOS (dyskowy system operacyjny) znajdują się w jego opisie/instrukcji obsługi.

167 File PROTECTed

Próba zapisu do zbioru zabezpieczonego przed zapisem i skasowaniem.

169 DIRectory Full

Przekroczona została pojemność katalogu zbiorów dyskietki. Nie można zapisać więcej zbiorów.

170 File Not Found

Nie ma takiego zbioru na dyskietce.

171 Bad Point Value

Próba wykonania instrukcji POINT dla nieistniejącego miejsca na dyskietce, lub zbiór nie został otwarty w trybie zapisu/odczytu (12).

Literatura:

1. Praca zbiorowa. Atari Basic. Język programowania i obsługa mikrokomputera Atari. Wyd. NOT, Warszawa 1987.
2. Adam Stawowy. Atari Basic XL Wyd. NOT, Warszawa 1987.
3. Mariusz Giergiel. Atari Basic XE. Wyd. NOT, Warszawa 1987.
4. ATARI 400/800 Basic Reference Manual