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.

BASIC XE

presents NEW WHAT in (to) BASIC-u?

The present (hereby) elaboration presents shortened description of language of programming for computers BASIC XE ATARI XL/XE. New describe instructions relatively to standard here BASIC-a.

SPIS TREŚCI

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

CONTENTS

and manner of their use describe on part according to following table

EXIT	49	**EXTEND**	14	**FAST**	10	**FIND**	30
HEX$	32	**HITCLR**	37	**HSTICK**	32	**INVERSE**	22
LEFT$	31	**LOCAL**	6	**LOMEM**	13	**LVAR**	14
MID$	31	**MISSILE**	36	**MOVE**	43	**NORMAL**	22
NUM	8	**PEN**	32	**PMADR**	37	**PMCLR**	36
PMCOLOR	35	**PMGRAPHICS**	34	**PMMOVE**	35	**PMWIDTH**	36
PROCEDURE	46	**PROTECT**	27	**RANDOM**	44	**RENAME**	27
RENUM	9	**RGET**	24	**RIGHT$**	31	**RPUT**	23
SET	12	**SORTDOWN**	41	**SORTUP**	41	**SYS**	13
TAB	17	**TRACE**	11	**TRACEOFF**	11	**UNPROTECT**	27
USING	17	**VSTICK**	33	**WHILE**	28	**CP**	51

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.

BASIC XE Certainly < obvious > it includes (reach; make) all available instructions in language - BASIC WADOMOŚCI BASIC XE Atari BASIC, microcomputer is furnished to which (who) each Atari. Several additional offered become (stay) below served by capability BASIC XE fastest execution of program. New procedures of operations assure (provide) fast execution of program in connection with instruction very zmiennopozycyjnych FAST. There is ok BASIC XE. Two for six cases (together; times) fastest than standard Atari BASIC, over twice fast than for 64 BASIC Commodore and junior for IBM (P. Capability of utilization of whole memory 130XE. It allows whole full utilization available in (to) memory BASIC XE Atari 130XE. Perfected editor. Does not have meaning at utilization interpretera BASIC XE, instructions are written that letters small, big, as well as in reverse graphics (video) Inverse. Besides, instructions of numbering of lines of automatic programs are in (to) its (his) time of writing available and along with for line program (proper cancellation (appealing)) przenumerowywania przenumerowaniem. LETTER OF (LIST OF) program is presented at execution of instruction with its (his) revealing structure - noose, podprogramów etc. widen capabilities of operations on texts. It eases programer from each necessity of maximum definition of alternate (changeable; variable) text length instruction BASIC XE DIM. Besides, it gives new capability to operate on texts, in (to) analogous it as to microsoft available BASIC. It enables fast and effective programing of operation of processing of text. Instructions for operating graphics Player/Missile. Package of instruction owns enabling simple utilization capability animation movable object (BASIC XE P/M ang. It allows creation of interesting game without necessity of acquaintance of map of memory) that Player/Missile Graphics and operating system Atari. Simple access to harbor manipulatorów. Special instructions allow fast and effective lecture of site type manipulatorów paddle and joystick connected for proper harbors of computers. Widen messages about errors. Besides, short description of error serves in case of (chance of) pronouncement of error of execution except application of number of error BASIC XE, which (who) has been discovered.

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

[1] Patrz dodatek A.

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.

As start up simple be for work at use approach language very BASIC XE interpretera BASIC XE. Switch will suffice (will enough) to put in proper site / (BASIC XE Atari BASIC if become (stay) for interior of computer) BASIC XE wmontowany or put kartridż and including computer next. Full utilization of capability given is with external memory by at cooperation possible exclusively (station disk BASIC XE or tape recorder). It belongs to place cassette in such case (chance) in tape recorder or in station of diskette about logical number diskette 1 BASIC XE and including computer. If it is not furnished system with station of diskette or tape recorder does not acts , it can be taken advantage BASIC XE, but without following instruction BSAVE, (ALL, DEL, EXIT, FAST, LOCAL, LVAR, PRO (MOVE EDURE, RENUM, ROET, RPUT, SORTUP, SORTDOWN, fastest arithmetic procedures and does not concern instruction enabling utilization diagram (graphics) ( ) P/M HITCLR. Note it does not acts with no so called correctly BASIC XE 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:

Keywords and it serve symbols in language list of keyword below BASIC XE and by taken advantage symbol BASIC XE. New differ instructions relatively to thick letter Atari BASIC

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$**	**MISSILE**	**MOVE**	NEW	NEXT
NORMAL	NOT	NOTE	**NUM**	ON	OPEN
OR	PADDLE	PEEK	**PEN**	PLOT	**PMADR**
PMCLR	**PMCOLOR**	**PMGRAPHICS**		**PMMOVE**	**PMWIDTH**
POINT	POKE	POP	POSITION	**PROCEDURE**	
PROTECT	PTRIG	RAD	**RANDOM**	READ	REM
RENAME	**RENUM**	RESTORE	RETURN	**RGET**	**RIGHT$**
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.

Notes about accepted (catched on) notacji and square parentheses in description of format of function summaries and positions include (reach; make) instruction, which (who) can, but must not be used. If three full stops take a stand in square parenthesis, e.g. [zm1. It means], that given position can take a stand many times. If in record of format of function or several positions served it among vertical lines instruction take a stand mean, that it belongs to choose one of they. If given describes function podrozdział, but not instruction, it has been checked off (noted) in title podrozdziału. It admits instead of some whole inscribing name of instruction summaries application < applicable > BASIC XE. It serve admissible summaries in such case (chance) in round parentheses in title describing given instruction podrozdziału. In description of format of function and instruction as well as sometimes it be of service (use) in text of description summaries. Meaning of applicable summary has been served below zm numeric - alternate (changeable; variable) (or text _ wyr num - formulation numeric _ zm num - alternate (changeable; variable) numeric _ text wyr. - Formulation text _ text zm - alternate (changeable; variable) text number _ line - number line _ package spec there description takes a stand in text - specification ( ) package couple – parameter description meaning above-mentioned deadline (date), where they have been used they and define. Now instructions will be described BASIC XE, does not have which (who) among instruction ATARI BASIC or they are, but operation has been expanded them.

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

It is used instruction for declaring capacity of numeric table DIM, alternate (changeable; variable) text and text tables. Instruction reserves area for numeric tables for _ + 1 DIM wyr num1 if table is one-dimension, or _ + 1 poem wyr num1, for (after) each _ + 1 element wyr num2. Plus one, as indices are begun for numeric tables from zero. Area reserves for alternate (changeable; variable) text for at most _ sign DIM wyr num. Place reserves for text tables for about length _ text _ sign each DIM wyr num1 wyr num2. Following example shows use of instruction DIM. Numeric table has been declared in line about 101 elements 100 called A1. Inclusive table numeric has been declared solution in line for 4 columns 110 7 poems called. It has been declared in line about length 120 alternate (changeable; variable) text 25 signs called maximum $ Imie and text table 40 signs 100 line of text called city 100 6,3 25 inclusive < include (reach; make) > $ 100 ( ) 110 solution ( ) 120 $ ( ) Dim A1 Dim Dim Imie, cities 100,40 $ ( ) note area can reserve for alternate (changeable; variable) text automatically BASIC XE. Informations are served SET on this theme in description of operation of instruction _ 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 1234ó07, 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.

Format acts _ [LOCAL zm num1, _ zm num2, interest example] have 100 temperature 110 amount Local Local, X, it allows writing to universal manner instruction Y LOCAL podprogramów and simple (easy) utilization of their enables in future with optional program. In association with instructions GOSUB and it allows defining local for PRO (instruction EDURE LOCAL podprogramu or procedures have (internal) alternate (changeable; variable) numeric. They are treated as local after meeting instruction in lithium of instruction for meeting instruction alternate (changeable; variable) numeric served LOCAL EXIT. It signifies about it (become alternate (changeable; variable) local? It means, that change of numeric value alternate (changeable; variable) served is changed in instruction in borders between only LOCAL LOCAL and besides, changes instructions not causing EXIT. Sometimes it is possible to escape conflicts among in program about same names sideline effects using instruction (called) using alternate (changeable; variable) LOCAL podprogramami. Following simple example shows operation of instruction LOCAL test 10 = 1234567 20 40 Print 10,Test Gosub 30 40 test Print 20,Test End Local 50 test = 0.54321 Print 40,Test will display number of line in program on screen of monitor 60 instruction included (reach; make) Print 50,Test Exit PRINT, they were executed in which (who) and alternate (changeable; variable) value test. It will allow to follow him (it) to simple manner operation. Display is effect of operation of program on screen following (step) information line includes (reach; make) attribution of alternate (changeable; variable) value 10 1234567 40 1234567 50 0.54321 20 1234567 10 exemplary program, this value is displayed next. Line it for about start in line 20 jump 40 podprogramu, but after return with (from) next display value alternate (changeable; variable) test podprogramu. It interest start line 40 program part < frequent >. This line includes (reach; make) alternate (changeable; variable) declaration test as alternate (changeable; variable) local, alternate (changeable; variable) value is displayed test next. Line 50 this alternate (changeable; variable) attribution test of new value but display of alternate (changeable; variable) value next test. But now there is alternate (changeable; variable) local. Line includes (reach; make) instruction 60 EXIT, that causes alternate (changeable; variable) come-back test of primary (original) value , i.e. had that in moment of meeting instruction such LOCAL and return with (from) podprogramu, likewise as instruction RETURN. Employment of instruction POPE and use enables from (with) alternate (changeable; variable) local in (to) not only LOCAL podprogramie. However, it does not effect readability of program advantageously. Notes 1. Instruction can be used with (from) evoked instruction LOCAL podprogramami GOSUB or type ((PRO ( ) ALL EDURE. It allows to accommodate existing programs to simple manner to use alternate (changeable; variable) local. Besides, induction is fast instruction podprogramu GOSUB, than at use (ALL, if program acts in mode FAST. 2. They do not change values beyond parenthesis of instruction alternate (changeable; variable) LOCAL and EXIT, it is possible to see that in former example. Unsubscribed alternate (changeable; variable) value totals (take away; amount to) test in line 40 12345ó7, although other value transmit this alternate (changeable; variable) locally. It does not cause use of instruction as local alternate (changeable; variable) declared LOCAL wyzerowania. It is possible value zero it (them) attributing memorial < they (their) > wyzerować. 3. Usage in (to) same name alternate (changeable; variable) declared podprogramach as local allows to bypass alternate (changeable; variable) about different names in program 128 " " limitation application at most. We have each require for example in program from which (who) 10 10 alternate (changeable; variable) podprogramów, that gives alternate (changeable; variable) 100 including. In (to) all calling alternate (changeable; variable) just the same podprogramach and we have instruction in program 10 different alternate (changeable; variable) applying (use) only LOCAL. 4. Instruction causes dispatch of numeric value alternate (changeable; variable) on pile LOCAL BASIC-a XE. Execution of instruction causes collecting from pile of alternate (changeable; variable) numeric value EXIT and come-back primary (original) value memorial < they (their) > (i.e. such, as before execution of instruction) LOCAL. 5. Instruction can be used instruction in (to) only finishing LOCAL podprogramach EXIT but not RETURN, POPE OF former alternate (changeable; variable) value behind exception of collecting from pile before instruction

instruction RETURN. It is possible to find on this theme in description of instruction more information POPE.

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.

Format start [ ] [NUM, step] example 50 NUM NUM NUM, automatic numbering of line enables at writing of program in language 1 50,1 instruction NUM NUM BASIC XE. It allows on fastest inscribing program and concentration (aggregation) enables on proper (suitable) content of program, as after each inscription of line BASIC XE and number unsubscribes on screen accepting of digital line RETURN nastopnej and cursor places on position of start of text of program in line. If it will not be served initial number (start) and in examples step of numeration first line (step) ( ), it will start numeration from number of last line of program in memory increased currently 0 10 finding (be placed) BASIC XE, number about 10 after each introduction of line boosting (accrue). If there was not in memory of program currently, then numbering will be started in such case (chance) from line about number 10. If initial number has been served start ( ), however, it has not been served step of numeration start in examples from line about served number numbering (step) (second (other) line) BASIC XE and it will boost numbers about 10 next. If step of numeration has been served step in examples ( ) (third line), however, it has not been served it start from line about greatest number from number of last line in memory of program increased about initial number numeration (start) currently finding (be placed) (step). If equal (start) as well as they have been served step start in examples from line about number ( ) (last line) number (start) and next number will be behind each introduction of new line former number step + ( ). Note start ( ) neither be as equal step serve zero ( ) not leg. Automatic numeration of line will be interrupted in (to) following four cases (chances) and it will expect on instruction in direct mode BASIC XE 1) if immediately digital will be pressed after number of line RETURN. 2) If syntactic error will ascertain in introduced line BASIC XE. 3) If line about number, which (who) would be generated exist by automatic numberer in program currently already. 4) If number of line, which (who) has been generated by automatic numberer currently be great than 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.

Format [DEL nr1,] example nr2 erases lines of (ropes of) programs in memory currently 100 1000,1999 instruction finding (be placed) DEL DEL DEL. If one number of line will be served be cancel in examples about served number first line one line only ( ) ( ) nr1. If two numbers of lines serve ( ) nr1 and ( ) separated comma nr2 lines (ropes) become (stay) from number for number - canceled ( ) ( ) inclusive nr1 nr2.

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.

Format start [ ] [RENUM, step] example 100 RENUM RENUM RENUM, in memory 20 currently start 1000,5 instruction program finding (be placed) transmitting (be suitable) first line program number ( ) RENUM RENUM przenumerowuje and number for each next line about boosting (accrue) (step) if serve number has been attribute value 10 (start) it (him), if step serve step has been attribute value 10 ( ) it (him) renumeracji. Note start ( ) neither can be served as equal step zero ( ). All cancellation (appealing) in instructions taking a stand (e.g. in (to) GOSUB, GOTO etc. they will be modified), if they are constant (solid) numeric. If number of line is served as numeric formulation in cancellation (appealing) (e.g. become (stay)) not GOTO 10wA przenumerowany. Notes 1. If program is executed he (its; his; it) speed of execution not depend in mode on manner in program line FAST ponumerowania. 2. If instruction takes a stand have not been change in program with served numbers of lines

through execution of instruction LETTER (LIST) these numbers RENUM. 3. It does not change instruction numbers of lines RENUM, if she (it) takes a stand as cancellation (appealing) and it is preceded as numeric formulation for line about number served cancellation (appealing). For example, execution of instruction RENUM if line will take a stand in program he (it) 10 100,3 * X Gosub Y, it will cause change on cancellation (appealing) for line about proper number 200 100, however, unchanged remaining 200, as it is preceded as numeric formulation for line about number served cancellation (appealing) (3 *) Y. HE (IT) can result in instructions of types such situation. 4. If it takes a stand in program for line cancellation (appealing), which (who) does not exist (e.g. 50 GOTO, where line does not take a stand in program 50), they will not be changed they.

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.

Format acts example FAST every time, program must search during normal execution of program from start 100 language FAST FAST interpreter BASIC XE in order to recover line about definite number, if instruction must execute GOTO, GOSUB, FOR or this way, majority acts on different computers (language exactly) WHILE interpreterów BASIC. However, operation can become (stay) change in this range through use of instruction language interpretera BASIC XE FAST. When instruction has been met carry in memory so called program currently finding (be placed) FAST BASIC XE prekompilację. It means, that each number of line, cancellation (appealing) takes a stand for which (who), physical address has been attributed, she (it) is placed under which (who) in memory of program. In execution time of program next, when has to execute instruction BASIC XE GOTO, GOSUB, FOR or WHILE, program must not review in memory about proper number in turn searching for line, immediately jump executes for proper address only. Notes 1. If number of line in instruction served GOTO or GOSUB etc. there is not constant (solid), only alternate (changeable; variable) or numeric formulation, it has not been calculated address in time prekompilacji and execution of such instruction proceeds with normal speed. 2. Execution of all of following instruction destroys effects prekompilacji, or mode interrupts FAST return to direct mode 3 ENTER LETTER (LIST) RUN (FLEECE) " specification package " DEL EXTEND LOAD LVAR SAVE. Line of program should take a stand before first in program inclusive instructions instruction FAST GOSUB, FOR, (ALL, WHILE, LOCAL. Best if it takes a stand in first line of program FAST. 4. Serve make difficult creation of program above note in execution time finishing loading fish-plate (instruction ENTER) and in mode working FAST. One exists expensive (dear) < way >, which (who) enables creation of program in mode acting FAST and using with fish-plates. Main program can not be in noose, podprogramie neither local zone alternate (changeable; variable) when it executes instruction ENTTER. Then, first line of finished loading segment can be instruction FAST and it will not cause it revolt of problem.

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].

Format TRACE. execute examples TRACEOFF they enable enclosure 100 it two instruction TRACE TRACEOFF or exclusion of (switching off of) following of execution of program execute program (number line). Is displayed number of line in mode on screen in square parentheses executable currently TRACE. Notes 1. First line of program can not be followed. 2. Instruction executed in direct mode (if following is showed active) be as 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.

Format SET _ wyr num1, _ example wyr num2 SET allows change of operation in some situations 2,128 100 SET instruction SET language 5,A interpretera BASIC XE. It means function _ Wyr num1, which (who) has to be changed, value means has been change according to which (who) function _ wyr num2. Following table serves possible to change all functions instruction 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	Edycja i listing programu w standardzie ATARI BASIC (wyłącznie duże litery).

	1		Standard BASIC XE.
6	0	0	Pojawiają się opisy błędów.
	1		Pojawia się tylko numer błędu (ATARI BASIC).
7	0	0	P/M znikają za poziomymi brzegami ekranu.
	1		P/M odbijają się od poziomych brzegów ekranu
8	0	1	Liczba parametrów nie jest wysyłana na stos podczas wykonania USR.
	1		Liczba parametrów jest odkładana na stosie.
9	0	0	Po wykonaniu instrukcji ENTER następuje powrót interpretera do trybu bezpośredniego.
	1		Po ENTER sygnalizowany jest błąd 32.
10	0	0	Cztery „missile” są niezależne.
	1		„Missile” poruszają się razem, ich atrybuty (kolory itp.) pozostają niezależne
11	1...255	40	Automatyczna rezerwacja długości zm. tekst.
	0		Brak rezerwacji (jak ATARI BASIC).
12	0	1	Przy LIST nie działa formatowanie (wcięcia).
	1		Formatowanie jest włączone.
13	0	1	VAL nie akceptuje liczb hex. (błąd 18).
	1		Liczby hexadecymalne są akceptowane.
14	0	0	PRINT USING obcina liczby jeśli są dłuższe niż wyspecyfikowano w formacie.
	1		Sygnalizowany jest błąd nr 23.
15	0	0	W trybie EXTEND tylko ADR(„tekst”) powoduje wystąpienie błędu nr 3.
	1		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.

Function ( ) format SYS (_) example SYS wyr num 100 0 ( ) = 0 SET 0 If SYS THEN, function is as (serve) for verification of status of system 128 (SYS BASIC XE i.e. SET possible to modifying behind assistance of instruction function). There is number of function of system _ Wyr num as it describe in former point.

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.

Format LOMEM address example LOMEM it is as (serve) for reservation of memory below normal area for program 128 ( ) + 1024 instruction (area address) LOMEM DPEEK LOMEM. This area can become (stay) used e.g. as memory for storage of screen, or for in engine code podprogramów. Utility of this instruction can be limited enough, as they exist other reserve (such, it does not use with which (who)) address BASIC XE. Note there program deletes of memory currently finding (be placed) LOMEM.

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.

Format CLR example CLR it erases in table of alternate (changeable; variable) value 100 all alternate (changeable; variable) (CLR CLR ang.) Variable Value Table And ration of memory erases executed for tables instruction DIM. It does not erase instruction from table of alternate (changeable; variable) name alternate (changeable; variable) names CLR, that makes NEW only. After instruction CLR, if they are used arithmetic tables, text, or alternate (changeable; variable) text, at first defined must become (stay) them dimension (measurements) instruction 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.

(LVAR LV.) Format [" specification package "] example LVAR " LVAR P for declared package " instruction all name alternate (changeable; variable) LVAR listuje. Each alternate (changeable; variable) name is preceded take a stand in (to) which (who) list of number of line of program (list cancellation (appealing)). Above-mentioned example will cause from list of cancellation (appealing) on printer name alternate (changeable; variable) wylistowanie. If it omit specification of package it will be showed on screen - list. Notes 1. They are differred through alternate (changeable; variable) text sign added < addition > " $ ", however, tables by sign added < addition > " (". 2. There to be last dry (LVAR or in line sole) instruction.

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.

Format EXTEND example EXTEND EXTEND if it works on unused instruction ( ) EXTEND Atari 130XE BASIC XE just as Atari BASIC. It is equivalent from the point of view of many programs in (to) " normal " mode BASIC XE Atari BASIC. Certainly < obvious > it is fastest considerably and many additional capabilities owns, but it administers memory just as Atari BASIC. Instruction causes EXTEND, that it proceeds with (from) with (from) for mode compatible " normal " mode ( ) " widen " BASIC XE Atari BASIC. Program is placed in mode widen in language in (to) additional memory) BASIC XE 64k Atari 130XE. Program can occupy whole take advantage independently of data that memory (table ó4k, alternate (changeable; variable) text etc.), which (who) be placed in basic memory. Then, it is possible to use instruction as line of program in direct mode (not) equal EXTEND, if program is placed in memory, as well as then,, when does not have program in memory. Execution of instruction causes displacement of program currently in memory to additional finding (be placed) EXTENID ó4k. Execution of instruction is in mode return to normal mode NEW sole expensive (dear) < way > EXTEND or booting (embarkation) ( ) program LOAD, which (who) was written in basic mode.

In moment of boating of (loading of) program also, which (who) has been written proceed in mode for mode widen automatically EXTEND BASIC XE. Recording of (writing down of) he (it) is instruction for processing program written in mode widen on program in normal mode for external memory sole LETTER (LIST) expensive (dear) < way > working (tape recorder, station of disk) and secondary introduction in basic mode instruction ENTER. Notes 1. Instruction can be used in direct mode only EXTEND, it can not take a stand as instruction in line of program. 2. Passage is possible for mode widen on computer only Atari 130XE or compatible (e.g.) Atari 256XT. If additional memory does not exist or it is inaccessible, attempt of (test of) execution of instruction causes error EXTEND 60 Error, " memory note " Extended Available. 3. It checks agreements (conventions) according to established by ( ) BASIC XE Atari Corporation if (or) it is not occupied additional memory. If it is occupied (e.g. by DOS for mode widen 2.5 include (reach; make) set up (bet; found)) not proceed RAMDISK BASIC XE and error exerts number 40. If early than installation in relation to checking off (noting) additional memory software (programming) (zajętości e.g. DOS does not show initial versions 2.5), that it uses additional memory, there instruction will be executed formerly information causing disruption finding (be placed) EXTEND. 4. Additional memory fulfills (perform) from bottom program beginning " " BASIC XE. As it show first on second (other) drawing in addition ex- < b > ok. It is placed in block 0 program 16k, in block next etc. it write over ok. 16k As it leaves min. (face) BASIC XE. In each block $ 100 byte free and never are placed among blocks in (to) integrity in one block not " column " line program (given line program always). 5. Deducting ok. Values in additional memory 1 bottom border free area $ 400 that exert (show) function ( ) receive remaining < remain > ad FRE. Block can become (stay) taken advantage for example in (to) integrity 3 e.g. for storage of several graphic screen, if 1 ( ) FRE at least it will exert - $ 400 free byte 16k. Addition D and service manual includes (reach; make) informations about equipment part of switching block of memory Atari 130XE.

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.

( ) Format INPUT I. [# channel] INPUT zmienna1, zmienna2. [” Text ”] example 100 INPUT X INPUT but 4 $ (;) 100 INPUT X, Y, with (from) (4), ex- < b > $ 100 # 4 INPUT, but 5,9 $ ( ) 100 " number # INPUT, name » ", number ( ) X, name is as (serve) for introduction of alternate (changeable; variable) value $ ( ) instruction X INPUT and attributing of alternate (changeable; variable) . It will be attributed instruction first give introduce INPUT zmiennej1, second (other) zmiennej2 etc. if it is introduced one instruction more INPUT than there can be served glades (commas) in one one alternate (changeable; variable) value separated 1inii, or with clicking of digital after application of next alternate (changeable; variable) value introduce one at a time ( ) RETURN. Will indicate it in second (other) case (accidentally) double query (BASIC XE?), That alternate (changeable; variable) value next belongs to serve. If values several alternate (changeable; variable) text are introduced one instruction INPUT introduced must become (stay) each text as separate line -. If numeric values alternate (changeable; variable) are introduced one instruction INPUT and must take a stand on end of alternate (changeable; variable) pull introduced text alternate (changeable; variable) text instruction INPUT. Curious capability shows fifth of above-mentioned examples. Text will be displayed interrogation (query) formulate in quote served after instruction on screen about data to manner pictorial directly allowing more INPUT, at use standard than "? ". Notes 1. It is possible to cause, that error of execution will ascertain BASIC XE, instead of more wczytywać than SET one alternate (changeable; variable) instruction using _ INPUT 4,wyr num. Is possible to change it about alternate (changeable; variable) value also (standard "? SET in description of instruction closest informations SET ") using _ ( ) 2,wyr num. It belongs to escape during capability - introduction more than alternate (changeable; variable) one each instruction INPUT - application instruction INPUT and use is prescribed for data recorded (written down) on disk in

this case (chance) (PRINT RGET and) RPUT. 2. As you can see, it allows instruction third of above-mentioned examples for elements of tables application < applicable > (numeric BASIC XE INPUT and text), that is not admissible in standard Atari BASIC. It facilitates widen creation of program very and it allows their boosts of efficiency.

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.

Format FIG FIG [# kan,] _ example wyr num it is as (serve) for introduction of space for definite position by on served channel FIG # 2,20 100 FIG 12 FIG _ wyr num. If it has not been served channel on screen – FIG gun. First column has number zero. Notes 1. Numerator of column is remembered for each fix-up separately and it is erased on zero after passage for new line each time. Numerator is stored in (to) see, (ex- < b > ( ) Aux6 IO OS Reference Manual. 2. If value is smallest _ wyr num than current position is executed passage for new line and spaces are deduced (moved out) for next position wyspecyfikowanej.

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.

acts FIG (function) format FIG (_) example wyr num # 3 PRINT; " column "; FIG (20); K1; FIG (30); it acts how (as) described function FIG likewise K2, as however, she (it) can be used in instruction function PRINT and PRINT USING. It allows to simple and plan clear manner form of deduced (moved out) information. Notes 1. If value is smallest _ wyr num than current position is executed passage for new line and spaces are deduced (moved out) for next position wyspecyfikowanej. 2. Function can be used in instructions FIG only PRINT and 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
```

put format PRINT USING [# PRINT kan; _ text USING wyr, [wyr1, wyr2.], Format allows to define PRINT USING, data have to be deduced (moved out) according to which (who). There is text formulation _ text Wyr, value defines which (who) format of deducing of (moving out of) data and it consists from one or several fields of formats. Each field of (tail of) format defines manner of deducing of (moving out of) value one formulation. First field of (tail of) format defines manner of deducing of (moving out of) value of first formulation etc. signs formatting, it can be used for defining field of format which (who) # & * + $, %! And meaning of all of this sign will be discussed in farthest / (part < frequent >). There be not signs as separation of fields of formats signs formatting be treated and they are unsubscribed among fields. Note at least one correctly must include (reach; make) defined field of (tail of) format _ text wyr. It will unsubscribe text in opposite case (chance) behind case (together; time) _ case (together; time) searching for field format BASIC XE wyr. Format of introduction of number following signs be as (serve) for creation of field of format for unsubscribing number and has following (step) meaning # fulfill (perform) space & fulfill (perform) zero * star (christmas) wype3nij. Decimal full stop, it put sign glades (commas) + (+ / before -) / for (after) only - sign (before for (after) -) / $ sign dollar # $ and * if deduced (moved out) number includes (reach; make) figures less than it declare fields has been remove in field of format in (to) number right < law (as of right) > to the end formatting but free places will be fulfilled (performed) dependent on kind of signs signs formatting. If deduced (moved out) number includes (reach; make) figures more than it become (stay) in field of format unsubscribed last figures of numbers what declare so many (so much) < rear > total (take away; amount to) length field format. It picture following (step) example number on publishing format form 123 # # # # 123 123 & & & & 0123 123 * * * * * 123 1234 # # # # 1234 12345 # # # # 2345 note it is possible to cause instruction SET 14,1, that in case (chance) when deduced (moved out) number includes (reach; make) figures more than it will not be cut it in field of format at unsubscribing definite , but error of execution will take a stand number 23. Full stop ( ) full stop defines place of situating of (placing of) decimal full stop in description of field of format taking a stand. All positions include (reach; make) figures after decimal full stop in (to) deduce (move out) form of number taking a stand. If deduced (moved out) number includes (reach; make) after full stop figures less than in description of field of format wyspecyfikowano, additional places will be fulfilled (performed) zeros. If deduced (moved out) number includes (reach; make) after comma figures more than in field of format wyspecyfikowano, deduced (moved out) number will be rounded. Note it is not treated second (other) full stop as sign in same field of format taking a stand formatting only as sign separating field format. Several examples serve below number on publishing format form 12.488 # # #. # # 12.49 123.4 # # #. # # 123.40 2.35 * *. * *. 2.35. Glades (commas) ( ) place means in field of format glades (commas), where has be put in (to) deduce (move out) form of number glades (commas). If it will take a stand in field of format glades (commas) earlier than first figure of deduced (moved out) number, this place will occupy dependent on chosen description of field of format in (to) deduce (move out) form of number glade (comma) proper sign. Note glades (commas) can take a stand before decimal full stop only (if it is used decimal full stop). It will be treated as sign in opposite case of (chance of) glade (comma) separating field format. Here, several examples number on publishing format form 5216 # #, # # # 5,216 3 *, * * * * * * * * * 3 4175 #, # # #. 4,175. + And - plus means in field of format, that sign of number has to be deduced (moved out) (+ if number is positive, - if negative). Minus means, that sign (there has to be deduced (moved out) -) if number is negative, however, space (sign span) if number is positive. Sign can take a stand on start of number in first field

of format, before first figure of number directly or after last deduced (moved out) figure of number directly. If take a stand can be on first position of field of format of deduced (moved out) number sign field of (tail of) format describe has e.g. number on publishing format form 43.7 + # # #. # # + 43.70 - 43.7 - # # #. # # - 43.70 23.58 - & & &. & & 023.58 - 23.58 & & &. & & - 023.58 If it has to take a stand sign before first figure of deduced (moved out) number in description of field of format before decimal full stop all positions directly (if decimal full stop takes a stand must include (reach; make) sign) formatting " + " or " - ", as on examples number on publishing format form 3.75 + + +. # # + 3.75 3.75 - - -. # # 3.75 - 3.75 - - -. # # Can take a stand after deduced (moved out) number - 3.75 sign too directly. Sign must take a stand as last sign in this case (chance) after decimal full stop in field of format number on publishing format form 43.17 * * *. * * + * 43.17 + 43.17 & & &. & & - 043.17 - 43.17 # # #. # # - 43.17 Sign of dollar - $ ( ) sign of dollar can take a stand on start of number in first field of format, or before first figure of number directly. Several examples of uses of signs of dollars serve on constant (solid) position below number on publishing format form 34.2 $ # #. # # $ 34.20 34.2 + $ # #. # # + $ 34.20 34.2 - $ # #. # # $ 34.20 - 34.2 + $ # # #. # # On alternate (changeable; variable) position before first figure of number - $ 34.20 example use sign dollar ( ) number on publishing format form 34.2 $ $ $ $ $. # # $ 34.20 34.2 + $ $ $ $ $. # # + $ 34.20 - 72692.41 $ $ $, $ $ $. # # + $ 72,692.41 - Note 1. One sign can take a stand on alternate (changeable; variable) position before first figure of number only (always). 2. Use +, - or strange effects can cause in incorrect position $. Format of deducing of (moving out of) text following signs be as (serve) for format of deducing (moving out) text it means %, that text has to be unsubscribed with slip for right end field specyfikowanego. It means, that text has to be unsubscribed with slip for left end field specyfikowanego. If number of sign of deduced (moved out) text is greatest than in description of field of format wyspecyfikowano, it will be cut text as on example initial signs of texts (become (stay) unsubscribed only) text of format of unsubscribed text " " % % % % % % % % " " BASIC XE BASIC XE BASIC XE! " " % % % % % " " BASIC XE BASIC XE BASIC BASIC XE! Insert sign BASIC it does not finish sign of oblique line field of format (/), but it causes, that sign will be unsubscribed next in field of format after it, following examples show that value format form deduced (moved out) 4084463099 (# # # /) # # # / 408 - # # # # ( ) 446-3099 ” ” % / / BASIC .%. % / / .% ./.%. Note B.A.S.I.C if it is for deducing (moving out) value more than in (to) _ text field format wyspecyfikowano wyr, the next time, unsubscribing will be continued as in example from given format using " # # # # " PRINT USING, unsubscribing will cause 25,19,7 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.

Format NORMAL/INVERSE example NORMAL INVERSE 100 150 NORMAL NORMAL INVERSE NORMAL and they allow to change operation of instruction INVERSE PRINT, LPRINT and PRINT USING. Everything it will be unsubscribed how (as) in proper instructions after instruction in (to) such form NORMAL, however, they will be converted (exchanged; turned into) all signs after instruction during unsubscribing on video INVERSE inverse. Signs in this case (accidentally), which (who) e.g. they were as ordinary signs have been deduce (move out) in instruction in (to) primary (original) video PRINT inverse. Note it returns for mode after passage for direct mode BASIC XE NORMAL, or RUN (FLEECE) by program of instruction after all.

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.

Format BPUT # BPUT kan, _ wyr num1, _ [wyr num2, bank sends contents of fragment of memory on external fix-up by opened instruction] instruction (earlier) channel BPUT wyspecyfikowany OPEN. Block is started from definite address value (fragment) memory _ wyr num1 and length has in bytes ( ) definite value _ wyr num2. It is possible to define number

of bank of memory in mode in description of instruction also (close information) EXTEND EXTEND. Note optional address can define in memory _ Wyr num1, e.g. address of text start alternate (changeable; variable) at use of function found (been placed) ADR. Served records (write down) program in following example on disk in graphic mode memory of screen 8 100 8 Graphics address = ($ 58) 110 " screen Dpeek Print Wypelnianie. It " 120 = 0 (40 * 160) For Sbajt - " screen " 130 address + 1:Rem wypelnij Poke Sbajt, 256 ( ) 140 190 " screen Random Next Sbajt Print wypelniony. Now recording (writing down). " 160 # 1 Close # 1,8,0 Open, " (" D:GR8.S " Channel prepared " 170 # 1 Rem Bput, address, 40 * 160 180 # 1 190 " record Close Print zakonczony. " 200 Note End does not include (does not reach; does not make) information of amount of record (write down) data recorded (written down) instruction package BPUT. Record of block is prescribed about constant (solid) length, in order to facilitate later (late) 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 #l: 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.

define format BGET # BGET kan, _ wyr num1, _ [wyr num2, bank define for memory from address by with (from)] beginning (begin) _ _ byte channel BGET wczytuje wyr num1 wyr num2 wyspecyfikowanego. It is possible, just as in instruction BPUT, define number of bank of memory, if it is placed in mode BASIC XE EXTEND. Following example shows manner of reading of graphic screen in mode 8 of disk for memory of screen directly. 100 = ($ 58) 110 # L Graphics 8:Adres Dpeek Close # 1,4,0 Open, " (" D:GR8.S " preparation " 120 " Rem Print Wczytywanie. " 130 # * 160 140 # 1 130 " Bget 1,Adres,40 Close Print wczytywanie zakonczone. " 160 Note End they are not checked errors related with served address and length block wczytywanego.

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
```

include (reach; make) format RPUT # RPUT kan, [wyr, wyr. It allows deducing (moving out) on external fix-up through opened about constant (solid) length] instruction (instruction) channel record RPUT OPEN. Each formulation occupies one field (tail) in record. Numeric field (tail) consists from one byte, which (who) defines type of field as numeric and six bytes include (reach; make) numbers recorded (written down) in code ex- < b > ((D as) zmiennopozycyjne. Text field (tail) includes (reach; make) one byte as text defining type field, two bytes defining length LENA (LINEN) alternate (changeable; variable), two bytes defining size alternate (changeable; variable) DIM and signs of texts byte inclusive < include (reach;

make) > DIM. It means, that there to be instruction alternate (changeable; variable) record (write down) not sauce instruction RPUT wczytywane INPUT, if it has been recorded (written down) value instruction more RPUT than alternate (changeable; variable) one. Following example shows use for record from fields 20 records folding (consist) " surname " RPUT, " " Imie, give " street ", " city ", code, phone 100 20,30 surname $ ( ) DIM, 20,30 $ ( ) Imie, street 20,30 20,30 $ ( ) 110 city $ ( ) DIM, code (20), phone (20) 120 # 1 Close # 1,8,0 Open, it 20 140 " " 130 = 1 " surname » " D:ZNAJOMI.DAT For Recnr Input, surname $ (Recnr;) 150 " » " Input Imie, $ (Imie Recnr;) 160 " street » " Input, street $ (Recnr;) 170 " city » " Input, city $ (Recnr;) 180 " code » " Input, code ( ) 190 " phone » " Recnr Input, phone ( ) 200 Recnr Print it give " served Print " 210 surname $ (Print Recnr;) $ (Print Imie Recnr;) 220 street $ (Print Recnr;) city $ (Print Recnr;) 230 " # # / Print Using - # # # ", code ( ) 240 " # # / Recnr Print Using - # # / - # # / - ", phone ( ) 250 " ( ) Recnr Print Zapisac T/N? ", $ ($ = " ") ($ = " ") Odp 2B0 If Odp y Or Odp Y " Record " 260 # $ (Rem RPUT Rput 1,Nazwisko Recnr;), $ (Imie Recnr;) 270 # $ (Rput 1,Miasto Recnr;), code ( ) Recnr, phone ( ) 280 Recnr Else " record again Print Wprowadz. " 140 290 300 # Goto Endif Next Recnr 3l0 Close 1:Print " Executed " 320 Print 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
```

give format RGET # RGET kan, alternate (changeable; variable) [, alternate (changeable; variable) allows reading record about constant (solid) length record (write down) from channel instruction opened] previously (instruction) RGET RPUT OPEN and attribution of read value eats proper alternate (changeable; variable) (numeric and text). Notes alternate (changeable; variable), which (who) must be recorded (be written down) value be same type that wczytywana. If they are alternate (changeable; variable) text wczytywane - dimension (measurements) alternate (changeable; variable) DIM, value must be the same for which (who) be wczytywana, as record (write down) alternate (changeable; variable). It is not possible for numeric tables instruction value wczytywać RET and text tables. It belongs to use in such case (chance) alternate (changeable; variable) subsidiary and attribute alternate (changeable; variable) value read next proper elements of tables. Following example shows use of instruction for reading from fields 20 records folding (consist) " surname " BGET, " " Imie, " street ", " city ", code, phone 100 20,30 surname $ ( ) DIM, 20,30 $ ( ) Imie, street 20,30 20,30 $ ( ) 110 city $ ( ) DIM, code (20), phone 30 (20) 120 $ ( ) DIM Tnazw, 30 $ ( ) Timie, 30 $ ( ) Tulica, 30 $ ( ) 130 # 1 Tmiasto Close # 1,4,0 Open, it 20 150 " " 140 = 1 # $ D:ZNAJOMI.DAT For Recnr Rget 1,Tnazw, $ Timie, $ Tulica, $ Tmiasto, Tkod, 160 surname $ (Ttelefon Recnr;) = $ Tnazw $ (Imie Recnr;) = $ 170 street $ (Timie Recnr;) = $ Tulica city $ (Recnr;) = $ 180 code ( ) = Tmiasto Recnr Tkod phone ( ) = 190 200 # 1 Recnr Ttelefon Next Racnr Close Print " read Print. Now read give company ltd. " 210 " Rem, eats it " 220 " mozna wypisac Input pokazac? ", 230 < > > 20 310 240 320 250 Recnr If Recnr 0:If Recnr Then Gosub Else " Show all record " 260 = Rem For Recnr and it 20 270 320 280 290 300 220 310 320 surname $ (Gosub Next Recnr Endif Goto End Print Recnr;) $ (Print Imie Recnr;) 330 street $ (Print Recnr;) city $ (Print Recnr;) 340 " # # / Print Using - # # # ", code ( ) 350 " # # / Recnr Print Using - # # / - # # / - ", phone ( ) 360 Recnr 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.

Format BSAVE _ BSAVE wyr num1, _ wyr num2, " specification package " example $ 680 BSAVE, $ 6FF, contents of fragment of memory allows to record (to write down) on external fix-up " " instruction D:FLIP.BIN BSAVE. It is performed record in standard with inclusive title address of boating (loading) DOS (Atari and address of end of block. Block can be uploaded place to this instruction next just BLOAD. Address of start of block defines _ Wyr num1, address of end of block of memory defines for recording (writing down) _ wyr num2. So, become (stay) recorded (written down) _ _ + 1 byte wyr num2-wyr num1. Note block of memory records (write down) as one segment with one title BSAVE. It does not record (does not write down) address of start, or it will not be started up block recorded (written down) after booting (embarkation) from DOS this instruction directly automatically.

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))

Format BLOAD " specification package " example BLOAD it allows in standard " " package binary recorded (write down) DOS BLOAD D:FLIP.BITY BLOAD wczytywanie Atari, or e.g. packages built (created) instruction BSAVE, if (or) in engine code written at use procedures MAC/65. Notes 1. They are not checked addresses served during practice of instruction in title of package BLOAD. Inattentive use can cause disruption important in given moment of contents of memory BLOAD. 2. It allows to load binary packages with (from) many segments folding (consist) BLOAD. If package owns title with addresses INIT and they are ignored RUN (FLEECE). 3. Package, which (who) can become (stay) started up by be definite address start execution RUN (FLEECE) SET 8,0 but = (($)) 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.

acts format DIR acts [" specification _ package "] example DIR 100 " DIR D * " $ " .COM DIR DIR FILE DIR D2:TEST? Display causes letters of (lists of) packages on screen of monitor according to serve specification " execution instruction .BAS DIR. It acts just as instruction to DOS DIR XL and but to DOS 2.5. If it serve specification have been display on all packages _ package finding (be placed) D1 i.e. instruction acts how (as) at so, *. * ". All packages will be displayed in first example on finding (be placed) D1, which (who) own THAT " " wyróżnik. Second (other) example causes display on letters of (lists of) all packages finding (be placed) D1. Third example shows text use alternate (changeable; variable). There is correct, but correct specification of package must include (reach; make) alternate (changeable; variable) text. Error will take a stand in opposite case (chance). Compact (tight) example causes display on letters of (lists of) packages (finding (be placed) D2, name is five-letter which (who) and are started from with (from) " TEST " " " wyróżnikiem BAS. Note if it takes a stand must be in program last DIR or in line sole instruction.

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

acts format " specification _ package ” example PROTECT PROTECT " PROTECT D * " 100 " .COM PROTECT D2 package allows to indemnify not concern before accidental cancel accidental formatting of disk " ( ) PROGRAM.EXE PROTECT. Package indemnified must be instruction before its (his) cancel PROTECT odbezpieczony. Instruction acts to DOS instruction so as PRO PROTECT XL and instruction to DOS 2.5 ( ) F LOCK

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.

(UNPROTECT UNP. acts) format " specification _ package " example UNPROTECT 100 " UNPROTECT D * " .COM UNP. " D2 *. * ". It allows (UNPROTECT odbezpieczyć e.g. for enabling cancel before cancel) package indemnified (e.g. in language instruction) PROTECT BASIC XE. Instruction acts how (as) instruction so UNPROTECT, and to DOS 2.5 ( ) G UNLOCK.

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.

Format RENAME " specification _ package RENAME, name of package ” example " RENAME D2:STARY.BAS, it allows change of name of disk package " NOWY.BAS RENAME. Among old glades (commas) and indispensable is new name of package. Note new name can not include (reach; make) specification of fix-up _ package (Dn). It is not prescribed use of star (christmas) in names renamed packages (*).

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"

acts format ERASE " specification package " example ERASE " ERASE D it allows cancel unindemnified before cancel of package * " " " instruction .BAK ERASE D2:TEST.BAS ERASE. It acts as ERA to DOS XL and ( ) DOS 2.5 D DELTE. First example causes cancel from disk in station with (from) all package " TANK " D1 wyróżnikiem. Second (other) example causes cancel from disk under name package " " Dl 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
```

Format WHILE/ENDWHILE it allows to simple instructions _ [ ] WHILE wyr num ENDWHILE WHILE and organize elegant manner conditional nooses. Instructions placed in parenthesis WHILE and they are executed long so ENDVHILE, till positive can be different from zero _ be (wyr num or negative). Before each execution of instruction included (reached; made) in noose (or value is calculated before execution of noose each time) _ wyr num and decision is taken if (or) noose has become (stay) executed if (or) not. However, noose will be executable < execute > has not been execute for example without end 1 (noose infinite (left-over)) 0 not once WHILE WHILE. Following program shows employment of noose WHILE 100 = 5 Rmax = 8 Kmax = 0 Rzad column = 0 it find = 0 purpose = 0 matrix (l05 Dim Rmax, notes notes) 110 < (find (be placed)) 120 column = column + 1 130 column < (find (be placed)) 140 matrix (Kmax While Rzad Kmax And While Kmax And If Rzad, column) = purpose find (be placed) = 1 150 column = column + 1 160 170 = + 1 180 190 find (be placed) Then Endwhile Rzad Rzad Endwhile If it find " " Print; purpose; on " "; 200 " position (" Print; Rzad-1; ", " column 1; ") " 210 Else purpose Print; it find " " 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
```

Format IF/ELSE/ENDIF employment enables instructions instructions useful _ [ ] [[ ]] very IF wyr num ELSE ENDIF BASIC XE, structural conditional instruction IF/ELSE/ENDIF. If it is genuine from zero _ (different) wyr num, instructions will be executed lines of (ropes of) programs for instruction ( ) ELSE, however, instructions among ELSE and they will be omitted ENDIF. If there is instructions among false _ (equal zero) _ wyr num wyr num but they will be omitted ELSE, however, among ELSE but they will be executed ENDIF. If it is possible be omit chain of instruction by not wanted finishing ELSE ELSE ENDIF. Then operation of described instruction looks as well as with this difference instruction IF-THEN, that pronouncement many line is admissible. Note keyword does not take a stand in instruction THEN IF/ELSE/ENDIF. Following program shows use IF/ELSE/ENDIF 100 If and < 2 110 " it " Print; 120 2 > 3 130 " computer " If Print; there is 140 3 < 4 150 " " If Print; 160 170 " spoiled Else Print! " 180 190 200 " Program " Endif Else Print; there is 210 4 > 5 220 " " If Print; 230 5 < 6 240 " strange " 250 260 270 " " If Print Endif Else Print dziala; 280 6 > 7 290 " If Print zle. " 300 310 " Correctly " 320 330 340 350 360 " good for nothing Else Print Endif Endif Endif Else Print! " 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
```

Function ( ) format ERR number of error allows to research (_) function it ERR wyr num and number of line, it has taken a stand in which (who), if personal procedure of attendance of (service activity of) error is written at use of instruction ( ) TRAP. If use it be about equal value 1 value of function number of line of program _ wyr num, error has taken a stand in which (who). If there is value equal zero value of function _ be number error wyr num. Is indefinite (vague) for value of argument remaining < survivor > _ value function wyr num. Example 100 110 " executioner sine " 120 Deg Print Cosecans For and it 180 steppe 15 130 = 0 " # # # # Print Using. # # # # # ", And, (Sin and), 140 200 150 " # # # # Trap Print Using. # # # # ", (1/Sin And) 160 Next and for line 170 200 180 End Rem przejscie if 190 (Rem Sin and it is equal zero). 200 1 " " 210 ( ) + 10 Print Nieokreslony Goto Err

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
```

Function ( ) format FIND (_ FIND wyr tekst1, _ wyr tekst2, _) example wyr num (" " PRINT FIND ABCDXXXABC, " " BC,) function fiesta fast N FIND and for definition effective instrument, if (or) served fragment of text takes a stand in served text. Text searchs from position _ beginning _ + 1 researching FIND wyr tekst1 wyr num, if (or) fragment takes a stand in this text _ wyr tekst2. If served fragment of text takes a stand in served text, then position is value of function, fragment is started on which (who). If fragment does not take a stand, then zero is value of function. They will be unsubscribed in above-mentioned example following (step) value 2 if = 0 N or 1 9 if > = 2 N and < 9 0 N if two examples of employment of functions serve > = 9 below N FIND 10 " change Input, cancellation, printing ", he (it) 20 (" " AS Find ZKD, but 1,1 $ ( ), it write 0) 100,200,300 30 10 10 " Goto Goto Input cos - ", but it lena (linen) $ 20 st. = 0 (For but $) - 2 30 = (F Find but $, " but ", it does not take a stand st.) + 1 40 = 1 " ' ' If F Then Print AH neither ' ' AC. " 50 End If But $ (F, on) = " ex- < b > " " ' ' F Then Print AB poz. "; = St. + 1 60 F-1:St If but $ (F, on) = " " " ' ' F C Then Print AC poz. "; = St. + 1 70 st. F-1:St Next

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".

Function $ ( ) format LEFT $ (_ text LEFT wyr, _) example wyr num 10 but $ = $ (" " LEFT ABCDE, 3) 20 $ (" " PRINT LEFT ABCD, it gives off text from text from left part 5) function $ _ _ sign starting LEFT wyr wyr num. If it is great _ wyr num than amount of sign is value of function not take a stand in text whole text error of execution _ text ( ) wyr. In first served example for but it will be put $ " ABC ", has been unsubscribe in second (other) " " 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".

Function $ ( ) format MID $ (_ text MID wyr, _ wyr num1, _) example wyr num2 but $ = $ (" " MID ABCDEFG, it allows giving off fragment of text from interior of text 2,4) $ _ text MID wyr. Text is given off from definite position by about length _ _ wyr num1 wyr num2. If equal zero is take a stand in text error _ (not has position zero) wyr num1. If it is great _ wyr num1 than length of text does not have error - (and empty (hollow) text is result of operation of function). It can be optional all-out positive number _ Wyr num2. If it is great _ + _ + 1 wyr num1 wyr num2 than length _ text wyr, fragment of text is value of function from position text _ to the end _ text wyr num1 wyr. In above-mentioned example but value will be attributed $ " " 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"

Function $ ( ) format RIGHT $ (_ text RIGHT wyr, _) example wyr num but $ = $ (" 123456 " RIGHT, it gives off text from text from right part 4) function $ _ _ sign starting RIGHT wyr wyr num. If it is great _ wyr num than amount of sign in text whole text is not take a stand error of execution - value function _ text ( ) wyr. In above-mentioned example but value will be attributed $ " 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$.

Function $ ( ) format STR $ (_) example STR wyr num but it gives text representation of value in result 650 $ = $ ( ) _ STR wyr num. Above-mentioned example will cause putting for but $ text " 650 ". Note one function can take a stand in instruction of logical comparison only $ CHR or $ 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.

Function $ ( ) format HEX answer $ (_) example HEX wyr num 5000 $ ( ) " $ " PRINT HEX PRINT; 32 $ ($ ( ) RIGHT HEX, second (other) example answer on text in record 2) function $ convert (exchange; turn into) value _ number (show HEX wyr num czterocyfrowej heksadecymalnym as it is possible to get double-digit number). Note it is not added sign on start of text " $ " 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.

Function ( ) format PEN (_) example PEN wyr num on 100 " " PRINT Pioro swietlne; 0 ( ) PEN; ", "; contents of register of illuminating pen reads in 1 ( ) function PEN PEN GTIA. If value totals (take away; amount to) zero be read _ horizontal wyr num współrzędna, if it totals (take away; amount to) be read 1 _ vertical wyr num współrzędna.

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
```

Function ( ) format HSTICK dependent on horizontal site of joystick accepts values (_) function HSTICK wyr num HSTICK. It defines postage (harbors) _ Wyr num, joystick is include (switch on) values of functions for which (who) (0-1) be following (step) rod shows following program to left 0 rod in neutral position in (to) 1 use of function - 1 rod right < law (as of right) > wychylony wychylony HSTICK direction 10 0 = ( ) 20 direction = Hstick If in (to) left - 1 „ Then Print. Stop < alloy > ” 30 direction = 0 „ IF Then Print. In (to) ” 40 direction = 1 (right < law (as of right) > If Then Print. ” 50 10 Goto

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
```

Function ( ) format VSTICK dependent on vertical site of rod of joystick accepts values (_) function VSTICK wyr num VSTICK. Joystick defines be include (switch on) for which (who) postage (harbors) _ (0-1) Wyr num. Values of functions are following (step) - 1 if rod is to bottom 0 wychylony if rod is in neutral position 1 if rod is show following program use of function upwards wychylony VSTICK direction 10 0 = ( ) 20 direction = Vstick If in (to) - 1 " Then Print dol. Stop < alloy > " 30 direction = 0 " If Then Print. In (to) " 40 direction = 1 " If Then Print gore. " 50 10 Goto

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".

It reconciling of curious features of computers DIAGRAM (GRAPHIC ARTIST; GRAPHICS) capability animation image PLAYER/MISSILE Atari. It is possible to execute animations different manners, however, sometimes it is possible to get fairest effects graphics of object call applying (use) exactly (too) P/M SPRITE's. Manner of movable remembering underway object is fitted diagrams (graphics) after two-dimension screen for linear structure essence ( ) memory FRAME screen P/M jednowymiarowości and proper equipment solutions. Defined four objects of types enables from for Atari P P0 P3, which (who) move independently and they are definite < define > in registers of colors of objects which (who) colors independent ( ) P/M and from on screen colors remaining < survivor >. All of can be presented from for in (to) normal largeness objects P0 P3 either (or) in (to) double width (latitude) either (or) in (to) double largeness (likewise as letters in graphic modes 0,1 and 2). It can be objects of types with (from) all of associate object of type m P, from for properly M1 M4. Colors of objects of types are answer m like objects of types which (who) P, however, manner of moving is independent on object P and from other objects .m moving of object P/M and match deals with testing of clash GTIA. Capabilities of utilization are big diagrams (graphics) P/M and miscellaneous, can concern animation not only, and also e.g. indices in like net of screen in exemplars defining of personal graphic sign not containing etc. does not have in whole of instruction factory language enabling creation Wmontowany interpreter Atari BASIC and animations of objects P/M and utilization diagrams (graphics) P/M. Convenient instructions for creation and however, it owns animation P/M BASIC XE. Instructions in this chapter describe and functions allow utilization of capability diagrams (graphics) microcomputer Player/Missile Atari. Size and color and changes of sites of objects are independent on current graphic mode on screen P/M. Each

object it on screen group of cell of (cellular phone of) memory vertical strip respondent < answer > P. Maybe which (who) is available one of 128 color in language each object have P BASIC XE. Point means object in horizontal strip about value in definite color 1 representing bit ( ) P zaświecony, however, bit means point about value without color in place 0 (color background screen, object is placed in which (who) currently) P. Object due to employed in microcomputers in (to) by change of contents of proper register solution equipment can be removed horizontal < level (horizon) > (instruction Atari P POKE or) PMOVE. Change of site of vertical object on screen in block of memory on screen this slip representing strip (instruction) P PMMOVE. Find complete in issued by firm technical operation diagram (graphics) possible book " 400/800 hardware " opity P/M Atari Atari 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.

(PMGRAPHICS PMG.) Format _ example PMGRAPHICS wyr num it is as (serve) for enclosing (encompassing) instruction PMG.2 or diagrams (graphics) excluding (switching off) P/M. Value can have 0,1 _ Wyr num or 2, that means properly 0 it exclude (switch off) - 1 P/M it enclose (encompass) in (to) resolution of one line 2 - P/M it enclose (encompass) resolution one line in (to) resolution of two (two) line - (P/M PMG. It means 1), that one byte of strip occupies one television line as line in graphics 8 (P/M or 15). Resolution two (two) line (PMG. It means 2), that one byte of strip occupies two television lines (ropes) as lines in graphics 7 ( ) P/M. Besides, amount of memory depends on mode wanted for safe keeping of object diagrams (graphics) P/M P/M. It requires resolution one line more memory twice than resolution two lines (ropes). It show on following drawing 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 - - - - - - - - - - - + $ 180 M1 M2 M3 M4 M1 M2 M3 M4 - - - - - - - - - - - + $ 300 - - - - - - - - - - - PMBASE - - - - - - - - - - - PMBASE - - - - - - - - - - - note it indicates summit of (top of) inclusive memory ($) MEMTOP 2E5 Missile. It is not indispensable due to function for programing in language above-mentioned map PMADR BASIC XE znajomość.

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.

(PMCOLOR PMCO.) Format PMCOLOR pmnum, _ wyr num1, _ example wyr num2 it is identical in operation 2,12,8 instruction PMCOLOR PMCOLOR as SETCOLOR but it concerns for registers of colors of objects P/M. Color defines _ Wyr num1, _ he (its; his; it) level (horizon) brightness wyr num2. Note registers of colors have indefinite (vague) contents after initiation of system P/M.

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.

Format PMMOVE [PMMOVE pmnum, _] [wyr num1; _] example wyr num2 0,120 PMMOVE; 1 1,180 4 PMMOVE PMMOVE; - 3 if object has been defined through (P/M POKE, MOVE, GET, BGET or he (it) can be relocated after screen) MISSILE. There is for display inconditional (ruthless) left edges _ (horizontal) position " strip " Wyr num1. It can be from range from 0 for 255. It does not cause change of largeness of object see, change of site of left edge ( ) P PMWIDTH, object is widen to right part. Relative slip defines in vertical _ Wyrr num2. It allows determination of relative slip in vertical from BASIC XE about 255 points to bottom for 255 about 255 points - 255 ( ) (upwards). Sign is coordinated vertical slips with values of functions + / VSTICK. For example, 2 PMMOVE; 2 ( ) cause remove object upwards VSTICK P2 or to bottom (either (or) remaining in same site depending on currently read in site of joystick). Note SET, it makes it possible at meeting edge of screen change of behavior of object _ wyr num P/M. Description of instruction see, 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.

(MISSILE MIS.) Format MISSILE pmnum, _ wyr num1, _ example wyr num2 reactivation of object of type enables be .m number of object 4,48,3 instruction (4-7) MISSILE Pmnum, inconditional (ruthless) vertical site of start of object defines m _ wyr num1, for example, height of object about height on position from mountain (top) 3 points 64 points _ define vertical .m 4,64,3 cause place object m wyr num2 MISSILE. Recurring use of instruction causes removal of (deposition of) object for object about same number from former site m MISSILE.

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.

(PMWIDTH PMW.) Format PMWIDTH pmnum, _ example wyr num it is possible to choose largeness of optional object 1,2 instruction PMWIDTH PMWIDTH P/M. Largeness of object defines _ Wyr num and value can have 1,2 or 4. Note sizes of objects can become (stay) boost (accrue) instruction P/M PMWIDTH, but without change of resolution. Obtainment of greatest object is with several objects about big resolution by creation of compound (composite) object possible P, that show in second (other) example.

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

(PMCLR PMC.) Format example PMCLR pmnum 4 instruction memory storing object type PMCLR PMCLR zeruje P or part of memory erases diagrams (graphics) depending on mode .m chosen instruction storing chosen object PMCLR P/M PMGRAPHICS. Note employment causes cancel of all object from 4 for 7 m pmnum, but not only chosen. In order to cancel single object possible execute m e.g. 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.

Function ( ) format BUMP (BUMP pmnum, _) example wyr num access to equipment register of clash of object gives 4,1 8,0 ( ) ex- < b > = ( ) function IF BUMP THEN BUMP BUMP P/M. Value accepts 1 if clash has taken a stand, value 0 if clash has not took a stand. Second (other) of parameters can define object of type P, or field (tail) (strip) object type P. It serve correct combinations of parameters in function below BUMP object P 0-3,0-3 - object ( ) object P BUMP P 0-3,8-11 - field (tail) object ( ) object m P BUMP 4-7,0-3 - object ( ) object m P BUMP 4-7,8-11 - field (tail) object ( ) note P BUMP 1. (BUMP p,) p Where value 0 is from 0 for 3 always has p, i.e. object can not enter to clash < self-service store > P. 2. Register of clash belongs to cancel for avoiding of error after lecture.

HITCLR

Format: HITCLR

Przykład:

```
100 HITCLR
```

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

Format HITCLR example HITCLR register of clash erases 100 instruction HITCLR HITCLR, which (who) can be tested function 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.

Function acts ( ) format PMADR ( ) example PMADR pmnum address serves in memory of object of type 0 = ( ) function P0 PMADR PMADR P or it facilitates operating at use of instruction .m objects P/M MOVE, POKE, BGET, BPUT etc. note m ( ) PMADR where address is number of object give in same result for all objects of types m m .m use (4-7) POKE and use of instruction is one of way for graphics to definition of appearance of object PEEK P/M P POKE. It is possible to place object in connection with (from) to simple manner in memory PMADR P P/M it 10 52 20 = 48 For Loc Read but 0 ( ) + Poke Pmadr Loc, but 30 40 date $ 99 Next Loc, $ BB, $ FF, $ BB, can be as (serve) for lecture $ 99 function PEEK and e.g. safe keeping of contents of memory defining object. Use allows instruction for graphics fast recording (writing down) for its (his) memory of object of appearance use MOVE P/M MOVE, or on overwriting contents of memory of object to other place, giving e.g. capability of change of appearance of fast object. For example, (Move Adr but $), 2 ( ) Pmadr, object overwrites in (to) double resolution 128 with (from) P but for area $ P2. However, 1 ( ) Poke Pmadr, $ FF 1 ( ) Move Pmadr, setup of all bit of object causes on value on screen 1 1 ( ) + 1,127 (strip) Pmadr P1. Use BGET and fulfillment of memory of object enables for graphics instruction BPUT P/M BGET e.g. from disk directly. For example, 0 # ( ) Bget 3,Pmadr, reading will cause as channel from package on disk opened in mode $ 80 # 3 object P0 PMG. 2. However, 4 # ( ) Bget 4,Pmadr, reading will cause from all disk of object $ 500 P and in mode m PMG.1. Use for graphics USR P/M as induction is with transfer of parameter through employment of function in engine code possible USR podprogramu, capability of utilization exists in language in engine code at operating graphics BASIC XE podprogramów P/M. For example, but = (Usr Pmmigaj, 2 ( ) Pmadr, it can be as (serve) for induction in engine code at $ 80) (placed podprogramu Pmmigaj in order to cause blinking of object about size 128 bytes P2. Exemplary programs with graphics 100 2,0,0 P/M Setcolor " " 110 2 Rem Ciagle GR.0 Pmgraphics " two line (rope) " 120 = 0 Rem rozdzielczosc P/M Wielk = 48 Y " initialization " 130 0 Rem Pmclr 4 Pmclr " cancellation Rem P0 and " 140 0,13,8 M0 Pmcolor 0 " object green " 150 = ( ) Rem P P Pmadr " address " Rem P0 l60 For and it = + + + 4 P Y P Y " object " 170 Rem ladowanie P Read V1 DATE " see, " 180 Rem Poke and, V1 " record " 190 Rem Next and it 200 = 1 " object "

210 For X 120:Rem petla poruszajaca P Pmmove 0,X in (to) " movement (traffic) horizontal < level (horizon) > " 220 about Rem Sound, + X X, 0,15 " " 230 240 Rem troche halasu Next X Missile 4,Y,1 on summit (top) " object m " 250 + 2,1 Rem P Missile 5,Y it in (to) 255 " " 270 = 127 Rem nastepny srodku P For X " object m " 280 Rem petla poruszajaca Pmmove 4,X " object " 290 300 ( ) Rem M0 Sound 0,255-X,10,15 If X&7 in (to) " that eight horizontal < level (horizon) > " 310 Rem punktow Missile 5,Y,5 it is necessary it " Rem zobaczyc! " 320 Endif " Thesis " 330 340 6,9 350 = + 2 Rem moze byc ELSE Next X Pmmove Wielk wielk " change " 360 > 4 = 0 370 Rem wielkosci If Wielk Then Wielk Pmwidth 0,Wielk " new " 380 4 Rem wielkosc Pmclr " cancellation m " 390 200 Rem obiektow Goto " Rem and from " 400 410 " * * * definition object * * * ” 420 " 84218421 " 430 " $ 99 * * * * " 440 " $ * * * * * * " 450 " $ * * * * * * * „ 460 " $ * * * * * * " 470 " $ 99 * * * * " 480 date poczatku Rem Rem P Rem Rem Rea BD Rem FF Rem BD Rem S99, EBD, it acts program above-mentioned faster 3FF,EBD,E99, if it will be recorded (written down) in lines for several instructions. However, readable has been presented so as more be. 100 0 Graphics 110 2 Pmgraphics 0 Pmclr 1 2,0,0 Pmclr 12U Setcolor 0,12,8 Pmcolor 0 1,12,8 130 = = ( ) Pmcolor P Pmadr L = ( ) P1 Pmadr " address object Rem P0 and " 140 = = P1 V0 60:Vs V0 " Rem poczatkowa poz. Vertical " 150 = 110 H0 " Rem poczatkowa poz. It horizontal " 160 = + 7 For Loc V0-8 V0 " object Rem P podw. rozdz. " 170 180 + Read X Poke P0 Loc, (/ $ 0100) 190 + Int X Poke P1 Loc, & $ 200 210 " Animation " 220 = 40 X ff Next Loc Rem Let Promien 230 " Deg While 1:Rem petla nieskonczona! 15 " 240 = ( ) C Random Pmcolor 0,C,8 It 355 steppe 5 250 executioner = 0 Pmcolor 1,C,8 For executioner in degrees executioner " " 260 = + * ( ) 270 = 280 = + * ( ) 290 0 Rem Vn V0 Promien Sin at1 Vzmiana Vn-Vs Hn H0 Promien Cos Pmmove, Hn; Vzmiana 1 Pmmove, + 8 Hn; 300 " movement (traffic) " 310 = 320 Vzmiana Rem dwoch obiektow P Vs Vn Sound 0,Hn,10,12 330 executioner 340 " Sound 1,Vn,10,12 Next Rem zakreslone kolo! " 350 360 370 " * * * Definition object * * * " 380 " 84218421 | 84218421 " 390 " $ * * | * * " 400 " $ * * | * * " 410 " $ * | * " 420 " $ 2004 * | * " 430 " $ * | * " 440 " $ * * * * | * * * * " 450 " $ * * * | * * * " 460 " $ * * * * | * * * * " 470 " $ 8001 * | * " 480 " $ 9009 * * | * * " 490 " $ 4812 * * | * * " 500 " $ * * * * | * * * * " 510 " $ 2004 * | * " 520 " $ * | * " 530 " $ * * | * * " 540 " $ * * | * * " 550 650 date $ Endwhile Rem Rem Rem Rem 03C0 Rem 0C30 Rem 10D8 Rem Rem 40D2 Rem 4E72 Rem BA51 Rem BE71 rem Rem Rem Rem 47E2 Rem Rem 10D8 Rem 0C30 Rem 03C0 Rem 03C0, $ 0C30, $ 10D8, $ 2004, $ 40D2, $ 4E72, $ BA51, $ 660 date $ 8001 BE71, $ 9009, $ 4812, $ 47E4, $ 2004, $ 1008, $ 0C30, introduced (presented) program uses with function $ over 03C0 SIN and COS. It is possible to speed up him (it) calculate with calculated instead of in time of course of underway noose operation wanted values each time objects considerably using table earlier it (them).

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
```

define format SORTUP/SORTDOWN IT in (to) in (to) table [[_ _] [SORTUP USING num1 num2; in (to) _ num3, IT in (to) in (to) in (to) _]] table [[_ _] [num4 SORTDOWN USING num1 num2; in (to) _ num3, in (to) _]] example num4 IT min. (face) text $ SORTUP Tablica1 SORTDOWN Tablica1 USING Max SORTUP USING; 1, IT 4 10 note text $ 5 SORTDOWN USING in (to) _ num3, can be used in (to) at classification of text table _ only num4. Can not be used for classification of numeric table. It sorts elements of tables according to codes all right growing sign SORTUP ATASCII or according to value depending on kind of table (text or numeric). However, it sorts all right diminishing SORTDOWN. If serve IT in (to) in (to) range first _ _ (num1 num2 and have been sorted third example) all element table. If it serve range of classification first equal as well as last element, in (to) (_ num1 and define must become (stay) in (to) _) num2 and IT divide keyword if it define start and end of fragment of text (; in (to) _ num3, classification has proceed in (to) according to which (who) _) num4, this classification is executed using whole text. If it has been defined fragment start (fourth example) equal as

well as become (stay) must be serve end of text serve must (; in (to) _ num3, in (to) _) num4. Even if serve IT must become (stay) in (to) in (to) range use _ _ keyword num1 num2 USING and precede specification of fragment of text (third example). Notes 1. If it surpasses served fragment of text length currently compared text these positions only, which (who) take a stand be take into account in text. 2. If two compared texts have different length and it are treated longest text as greatest on positions in shortest text taking a stand equal be (e.g. " abc " < " ") abcCf, that is intuitively correct. 3. Multidimensional tables numeric can not be sorted instructions SORTUP and SORTDOWN. Sort can be one-dimension tables numeric only. Example of classification of text serve below. It is possible to introduce texts for text table for 20 line of text about length for each 20 signs. Line finishes introduction of text 30 if empty (hollow) text will be introduced 10 20,20 text $ ( ) 20 Dim For and = and it " text > " 20:Input, text $ (and; text) 30 lena (linen) ($ (If and;)) Then Next and it it 40 1 50 = 1 text $ (Sortup TekstS Using I-1 For J I-1:Print J;) 60 run (fleece) Next J

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.

Function ( ) format DPEEK (_ [DPEEK wyr num, bank]) example " address PRINT VNTP - "; it is similar as function ($ 82) operation function DPEEK DPEEK PEEK, but collecting of value of word causes two (two) bytes from location ( ) _ wyr num and _ + 1 wyr num. There is addresses at taking contents of inclusive cell (cellular phone) useful particularly. Lecture of address would look like this: for example at use of instruction VNTP PEEK " address Print VNTP - "; as you can see, use is considerably simpler (simple) 130 131 ( ) + ( ) * 128 Peek Peek DPEEK.

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.

Format DPOKE _ DPOKE wyr num1, _ [wyr num2, bank] example 88 DPOKE, it is similar as instruction $ 8000 operation instruction DPOKE POKE, but recording of (writing down of)

contents of word causes two (two) bytes about location ( ) _ wyr num and _ + 1 wyr num. There is addresses at recording (writing down) contents of inclusive cell (cellular phone) useful particularly. Value must be integer from 0 for 65535 _ wyr num2. Address has been recorded (written down) in above-mentioned example for cells (cellular phones) 88 $ 8000 and 89. Execution of this operation would be complicated behind assistance of instruction more considerably POKE.

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.

Format MOVE _ MOVE wyr num1, _ wyr num2, _ [wyr num3, bank] example $ MOVE DOOD, $ 8000, $ 400 note precaution is indicated this instruction at use. She (it) causes slip of served number of byte from one location for second (other) of speed of program in engine code. Maybe they are checked no addresses drive (cause) that e.g. for hook-up of system or for disruption in memory of program finding (be placed). Initial address of block defines _ Wyr num1, which (who) has overwritten become (stay), _ initial address record wyr num2, for overwriting _ amount byte (length block) wyr num3. Sign defines order be overwrite in that bytes _ wyr num3 with (from) positive negative ( ) for with (from) - > ( ) (+ debt 1) for with (from) - > (+ debt 1) (+ 1) for with (from) - > (+ 1) (+ debt 2) for - > (+ debt 2). With (from) (+ debt 1) for with (from) - > (+ debt 1) ( ) for - > ( ) if length of block it for overwriting for which

(who) with bottom part of block block be number positive follow (step) record can cover źródłowego. If length of block it for overwriting for which (who) with overhead part of block block be number negative follow (step) record can cover źródłowego. Note perform slip among banks not can automatically MOVE. In order to at first execute such operation belong to remove for basic memory block but for other bank next.

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.

Function ( ) format RANDOM (_ [RANDOM wyr num1, _]) example wyr num2 99 10 ( ) = (RANDOM Y RANDOM, all-out number pseudofate is from definite partition by 20) value function _ wyr num1 and _ wyr num2. If it serve as in first example be generate from partition from 0 for only _ ( ) number _ inclusive wyr num1 wyr num1-1. If it serve _ wyr num1 and _ (wyr num2 as it is generated in second (other) example from partition from for) number _ _ inclusive wyr num1 wyr num2. Function uses with about even schedule generator number pseudofate hardwarowego.

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.

Why apply before description of instruction for operating type type several preliminary note podprogramy PROCEDURE podprogramami PROCEDURE padano and examples, which (who) will be allowed to orientate for utility that this instruction. Majority met (faced each other) dialect of language makes it possible by creation only evoked BASIC podprogramów GOSUB. Then, program looks use with so as following exemplary program podprogramów 20 alternate (changeable; variable) = 100 30 min. (face) = 10 = 5 ~ 0 Max 100 40 result = 50 min. (face) = 10 * alternate (changeable; variable) Gosub NUm = 90 * alternate (changeable; variable) Max 100 60 = 70 > alternate (changeable; variable) * 90 80 " Gosub Wynik2 Num If wynik2 Wynik1 Then Print, rather * Jestes konserwaty ny. " Reckless 90 " rather End Print Jestes. " 100 " " 110 End Rem Podprogram Print It serve from " " 120 min. (face) Print liczbe Print; for " "; Max; 130 " > " Input Wlacznie, 140 > = min. (face) < = 150 Num If Num And Num ax Then Return Inverse do not know (do not be able) you " Print czytac? This number is served range " 160 " affectation Print. " In small programs 170 100 Normal Goto, for example, like above-mentioned, instruction is completely sufficing (enough) GOSUB. If program is big instructions as obvious becomes less 3250 more GOSUB but following of program and apprehension of its (his) operation hard more. Atari BASIC And they allow record BASIC XE it serve = 100 20 = 100 30 min. (face) = = 10 1V Value 10:Max somewhat it serve program possible make (become) through use of name more readable Gosub podprogramów. However, it can become (stay) on this purpose alternate (changeable; variable) in program necessity of declaring that take advantage cause decrease (fall off; belittle) amount alternate (changeable; variable) that (maximum number all alternate (changeable; variable) total (take away; amount to) 128). In order to escape this difficulty and make clear capability, it offers procedures structural record of algorithm ( ) BASIC XE PROCEDURE. Name is in this case (chance) constant (solid) text, or additional requirement alternate (changeable; variable) not. It from range from range 20 rates = 100 30 " serve " 10,90 50 " serve " 10 * rate Call Using Wynik1 Call Using, it 90 * rate 70 < rate * Wynik2 If Wynik2 Wynik1 it write $ = " conservative " 80 Else it write be $ = " reckless " 90 95 " rather % % % % % % % % % % % % / Endif Print Using. ", It write $ it serve from range 100 " " min. (face) End Procedure Using, 110 rate Max Local rates = 1 + 90 120 rate < min. (face) rate > 130 rate < > 1 + 90 While Or Max If 140 Print Inverse do not know (do not be able) to read you " Print? This number is served range " 150 „ affectation Print. „ 160 170 Normal Endif Print It serve from " " 180 min. (face) Print liczbe Print; for " "; Max; 190 " > " Input Wlacznie, from range 200 210 rate line 30 include (reach; make) induction procedure called " serve " Num Endwhile Exit. This determination of parameter sent in line for procedure 30 USING. Alternate (changeable; variable) names take a stand in line for (after) 100 USING. This transmission of parameter is for procedure. Instruction finishes procedure in line 210 EXIT and it transfers as parameter largeness of alternate (changeable; variable) rate. Alternate (changeable; variable) name is served IT in line of induction of procedure behind word, value of parameter will transfer for which (who) from procedure. Above-mentioned examples are allowed to orientate in (to) utility of procedure.

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:

```
l0 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.

finish PERCENT ( ) format PROCEDURE name [[PROCEDURE USING zm1, zm2.]] Example it calculate 1000 " " hour PROCEDURE USING, it unsubscribe message table ( ) 387 " PROCEDURE. " USING! Comas $ 4040 " " 137 note PROCEDURE Wyjscie if alternate (changeable; variable) is text, numeric table, or it must be preceded text name exclamation mark (!). Instruction defines start of procedure evoked instruction PROCEDURE CALL and it finish instruction EXIT. Name of procedure is optional constant (solid) text. It orders practitioner of (practice of) programing good < goods (right) >, in order to names of procedures were significant i.e. in order to it was possible to orientate that make after name given procedure. Address of return has been remembered (stored) in moment of induction of procedure on pile instruction CALL and next instruction is executed after procedure for (after) CALL. If procedure is used alternate (changeable; variable) values with parameters through ( ) USING zml, zm2. They are sent on pile, attribute is served values in instruction of induction next alternate (changeable; variable) this (see,) CALL and execution of program is transferred for procedure. Example serve below alternate (changeable; variable) value is sent test in moment of provocation of procedure called in line on pile 10 70 = 20 20 " " 1217 30 40 70 " test " 80 90 " test " X Call Using Print X End Procedure Using X Print X Exit X and value is attributed alternate (changeable; variable) next 204 X. Instruction will unsubscribe in line 204 PRINT 8O. It has been transferred after meeting instruction for next instruction for (after) steer EXIT CALL but former alternate (changeable; variable) value is collected from pile X and attributed this alternate (changeable; variable). Instruction will unsubscribe in line 30 20 PRINT. It means, that in connection with (from) (USING CALL and similar effect causes as employment of local segment alternate (changeable; variable)) PROCEDURE LOCAL. Alternate (changeable; variable) can be used changes of alternate (changeable; variable) values outside procedure in segment due to it not causing evoking procedure X X. Besides,, it is possible to transfer that from introduced (presented) example for procedure without acquaintance of alternate (changeable; variable) name inside of procedure result value also, this value will be attributed which (who). If alternate (changeable; variable) zml, they are transferred which (who) values for procedure not be alternate (changeable; variable) numeric (but < cure > e.g. numeric tables, alternate (changeable; variable) text or otherwise, crosses text tables) process. Information is indispensable for this explanation of problem, manner is placed alternate (changeable; variable) value to that on pile. BASIC XE As alternate (changeable; variable) value transferred treats contents of proper location in internal table of alternate (changeable; variable) value (VVTP ang. presents squares) Variable Value Table. Eight bytes presents her (it) for each alternate (changeable; variable) give - byte flag, alternate (changeable; variable) number presents (0-127) and six information bytes. Alternate (changeable; variable) value includes (reach; make) for alternate (changeable; variable) numeric six information bytes. However, in case of (chance of) numeric table, alternate (changeable; variable) text and information bytes include (reach; make) address in memory text tables and they give current characteristic. For example, maximum alternate (changeable; variable) size presents this characteristic in case (chance) alternate (changeable; variable) text DIM and current length of LENA (LINEN). However, proper (suitable) text contents alternate (changeable; variable) is placed under definite address of memory. In case of (accidentally of) numeric table and text information bytes include (reach; make) address and size DIM. Execution of instruction relies on dispatch on pile of eight byte collected with (from) CALL VVTP. So, it does not cause instruction dispatch of genuine value of numeric table CALL, alternate (changeable; variable) text and text tables but in (to) only bytes finding (be placed) VVTF. Be has been transfer for procedure value of described alternate (changeable; variable)

type alternate (changeable; variable) numeric if they will be preceded them names sign of exclamation mark (! In instruction of induction). It shows following (step) example it is nice $ " feed l0 Przyj. " $ = " Well < goods (right) > X? " 20 " Nicely " Call Using! $ 30 Przyj Print PrzyjS, $ 40 50 " Procedure " 60 " nicely " X End Rem Procedure Using! $ 70 $ X Print Przyj, Shows after instruction in line 1,5 70 $ 80 $ ( ) = " " 81 90 X X Marze Exit PRINT, that value has been copied for $ $ Przyj X, it appears on screen feed is nice. Feed is nice. However, execution of instruction causes unsubscribing in line on screen 30 (succeeding completion procedure instruction) PRINT EXIT it is nice mazhena (dream). Well < goods (right) >? So history < happen > as value has been copied for $ $ Przyj X, now address is placed in location $ $ Przyj X VVTP. So, change performed has caused change of value on inside of procedure beyond procedure $ $ X Przyj. Technical note conceals message of parameter is called in language of computer scientist for procedure in case (chance) alternate (changeable; variable) numeric by value message, it conceals in (to) by cancellation (appealing) cases (chances) remaining < survivor >.

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.

It requires note about utilization PROCEDURE BASIC XE, in order to parameters in induction ( ) CALL and they were in declaration of procedure same type ( ) PROCEDURE. Error takes a stand in opposite case (chance) number 24 " " USING Type Mismatch. It does not check BASIC XE, if (or) number of parameter in instruction of induction and the same is in declaration of procedure. If many parameters includes (reach; make) have been ignore far too < market > CALL. If value includes (reach; make) zero has been attribute not enough (little) parameters far too < market > scanting parameter CALL. It can cause pronouncement of error number 24. Sending as parameter belongs to be careful to procedure at its (her) change of value alternate (changeable; variable) text. It has been transferred length of alternate (changeable; variable) LENA (LINEN) with (from) for procedure current VVTP. If alternate (changeable; variable) length this will be changed inside of procedure at return for location in instruction copied ( ) new length not be automatically describing alternate (changeable; variable) being parameter EXIT VVTP CALL. For example, modification of line above example on 80 introduced (presented) unsubscribing will cause in line 80 30 $ = " " X XXX it is nice XXXenie. Well < goods (right) >? Change of contents has caused change of contents in line 80 $ $ X Przyj, but it has not been placed new length after omission of procedure in (to) on respondent position $ VVTP Przyj. Transfer of parameter is one of possible solution after procedure in instruction this problem EXIT. Program will act correctly if lines (ropes) will be changed except modification of line 80 20 else and 90 20 " nicely " Call Using! It $ Przyj! $ 90 Przyj Exit! Can take a stand in case of (chance of) attempt of (test of) transfer from procedure of value of text $ serious problem X or table, which (who) was not entrance (input) parameter. For example, start-up of program it 100 " percent " Call! But it $ 110 " percent " Call! Ex- < b > $ 120 Print but $, ex- < b > $ it write 300 " percent " 310 " > " End Procedure Input cos, line $ 320 Exit! Line will

cause unsubscribing by instruction in line 120 same text $ twice PRINT, behind second (other) case (together; time) introduced. When line is transferred overwrite from procedure with (from) $ be she (its; her; it) parameter (VVTP or address of start to location also) defining VVTP but behind first case (together; time) $ and behind second (other) case (together; time) ex- < b > $. So, but $ and they appeal (appeal) for same place in memory at practice of line 120 ex- < b > $. Alternate (changeable; variable) transfer is correct solution for procedure by USING and transfer with return by next EXIT. Note it must be first instruction in line PROCEDURE. It will not recover procedure in opposite case (chance) in program CALL. It belongs to call attention (to pay attention) also, in order to admit for accidental for procedure " passage " program, segment procedure instruction finishing previous < precede > GOTO, STOP < ALLOY >, END, RETURN or EXIT. Grouping of all procedure is prescribed after main program (evoking) ended (end) instruction 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.

Format EXIT [[EXIT par1, par2. be]] example become 390 10 * alpha 799 alpha EXIT EXIT, name $ 21990 EXIT! ( ) Odwrot, X, Can be become beta 835 parameter couple EXIT, formulations alternate (changeable; variable) be It . It be alternate (changeable; variable) . Note if which (who) are numeric table from parameters of couples, alternate (changeable; variable) text, it become text or text table must be preceded sign of exclamation mark - she (its; her; it) name (!). Execution causes following (step) action (function) EXIT 1. If there send be placed on pile value of parameter (e.g. by reason of devolution of parameter, or they have been retrieved by reason of use proper alternate (changeable; variable)) (LOCAL i.e. for overwrite) VVTP. 2. If they take a stand have been attribute IT for (after) for transfer in instruction after keyword parameters proper alternate (changeable; variable) detail EXIT CALL. 3. It checks EXIT, if (or) it has been evoked by podprogram CALL, if (or) by GOSUB. If it has been evoked return as to case of (chance of) instruction follow (step) by GOSUB RETURN. It does not cause use with parameters for transfer pronouncement of error EXIT, if it has been evoked by podprogram GOSUB. Parameters are ignored. Then, also, if amount of parameter transferred is with return instruction greatest EXIT than they are ignored additional parameters in proper instruction taking a stand for (CALL but proper parameters must be same type). Utilization of instruction is possible for omission of procedure without use POPE 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.

Format CALL acts name [[CALL USING par1, par2. IT] [[zm1, zm2.] Example it calculate test

10 " " 720 " " CALL CALL USING! It read IT IT alternate (changeable; variable) result 800 number 1100 ( ) " " percent $ 7 CALL CALL USING, but it $ X if alternate (changeable; variable) or there is text parameter, text table or numeric table, there must be preceded sign of exclamation mark (!). Use of instruction has been discuss in former chapters CALL jut, where several exemplary programs serve also. Contrary to instruction PROCEDURE, name must be become text in which (who) in instruction name constant (solid) taking a stand can be equal CALL as well as alternate (changeable; variable) text. Technical notes 1. Possible degree is dependent on remaining free as pile of memory procedure zagnieżdżenia. So as GOSUB and four bytes consumes on remembering (storing) address of return instruction WHILE CALL and besides, twelve bytes for each transfer parameter. 2. It is comparable in normal mode without parameters with (from) in respect speed CALL GOSUB. It acts in mode faster FAST GOSUB than CALL, even bezparametrowe. In spite of it at holistic look (regard) on problem of induction podprogramu and transfers of parameters by use of procedure evoked CALL and it is profitable so.

CP

Format: CP

Działanie instrukcji CP jest identyczne jak DOS w Atari BASIC.

Format CP it is identical as DOS in (to) operation instruction CP CP 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.

ADDITION has has emerged but it has emerged as successor not be mistaken (mislead) in firm with (from) change BASIC BASIC ( ) ( ) XE BASIC XE OSS Operating Systems Software XL Turbo Basic XL. Several its (his) changes has has emerged during this development of (evolution of) language of programing. It was assigned for sale mainly in the form kartridża. It was assigned for cooperation with station of diskette initially version exclusively, version cooperates with tape recorder last also. It walks about manner of reading through instruction from external memory here, to ROM instruction not recorded (write down) kartridża. It can be for interior of computer too BASIC XE wmontowany, then, inclusion (participation) < include (switch on) > and exclusion (switching off) proceeds behind assistance of additional switch assembled from rear of computer most often. After embedding kartridża or DOS belongs to put diskette to station of diskette with (from) inclusion (participation) BASIC 2.5 inclusive additional instructions switch XE. Certainly < obvious > it belongs to put proper cassette in case of (accidentally of) tape recorder and put her (it) on start of program. When station of diskette is connected DOS for computer with put diskette with (from) 2.5, it will be uploaded after inclusion of (participation of) computer DOS.SYS and screen will appear with capability of choice of (election of) type of external memory. It will be begun it it after proper choice (election) process of boating (loading) 7.2 DISC START BASIC XE version Press OPTION boot with DRIVE Press SELECT boot with RECORDER Press use ONLY CARTRIDGE. I have alluded, that there is successor BASIC BASIC XE XL. There is in essence so, so, consistent is beyond instructions with (from) BASIC XL XE BLOAD, BSAVE, CALL, PROCEDURE, EXIT, EXTEND, $ HEX, HITCLR, INVERSE, $ LEFT, LOCAL, $ MID, $ RIGHT, SORTDOWN, SORTUP and USING, it lacks which (who).

DODATEK B

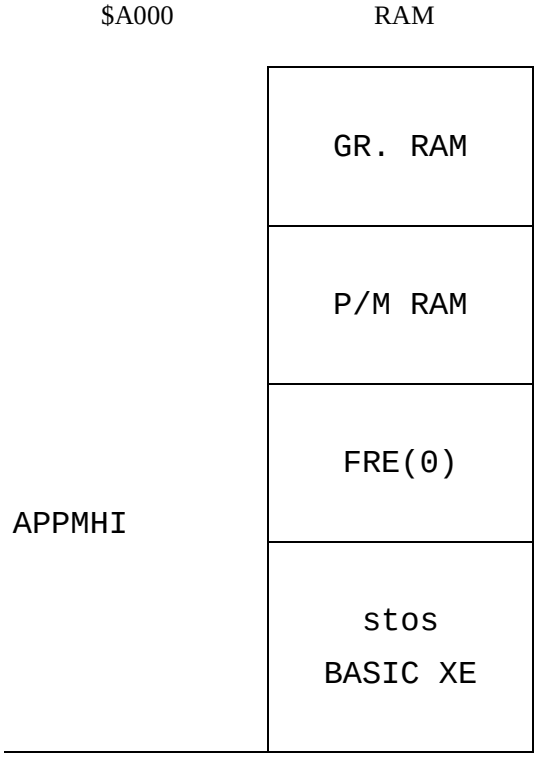
Mapa pamięci BASIC XE

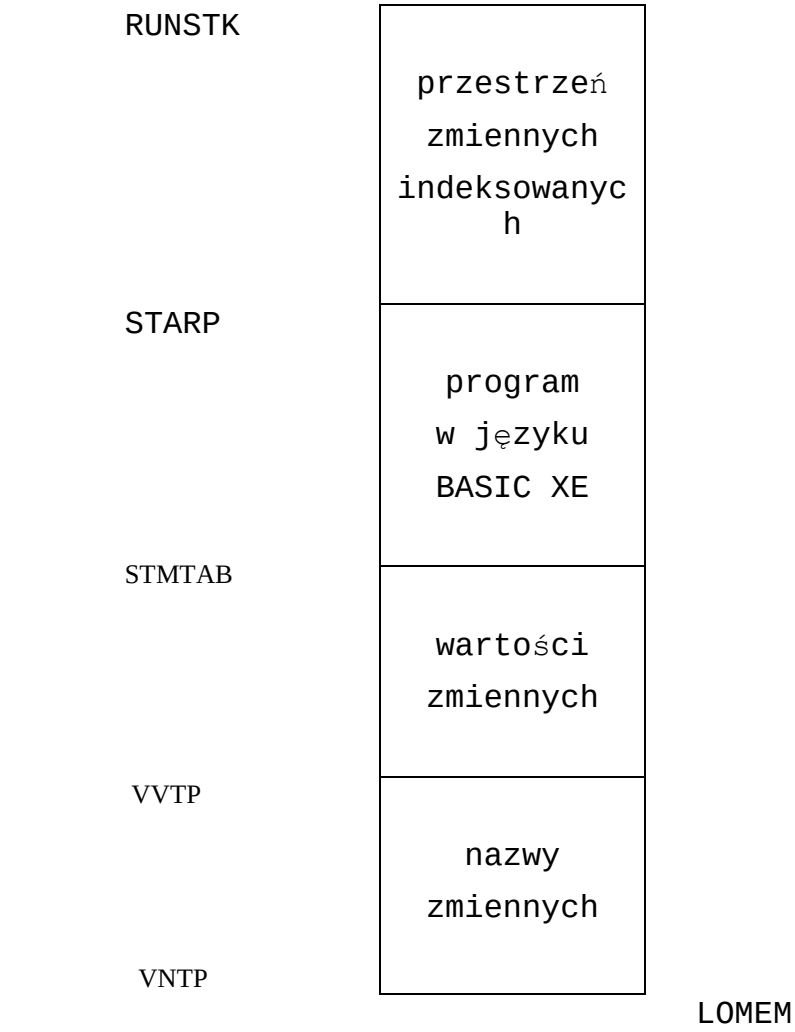
W dodatku podano ważniejsze informacje o wykorzystaniu pamięci przez BASIC XE. Występujące w opisie 'AtB' oznacza Atari BASIC, 'BXE' - BASIC XE. Należy podkreślić, że jedyne istotne z punktu widzenia programowania w języku BASIC XE lokacje to LOMEM, STOPLN, ERRSAV i PTABV. Znajomość tych adresów i wykorzystanie 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.

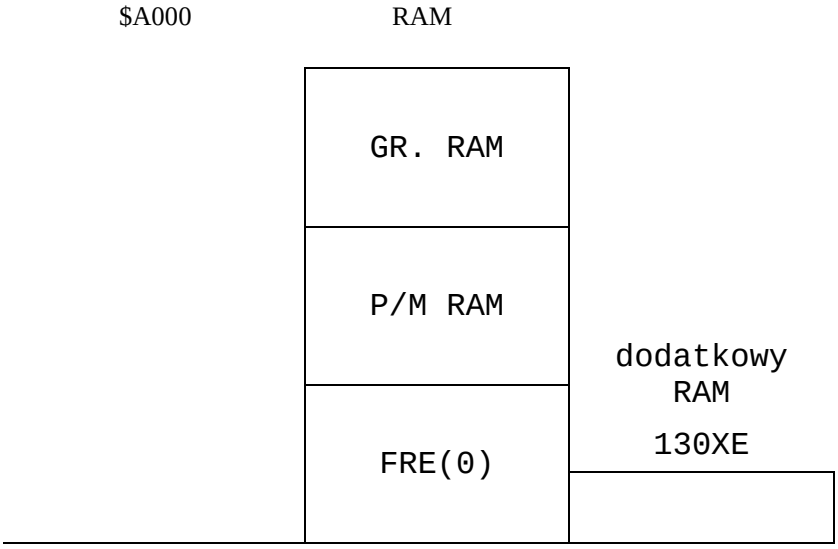
Adres	Nazwa	Znaczenie
$E-$F	APPMHI	systemowy wskaźnik wolnej pamięci
$20-$2F	ZIOCB	wykorzystane przez operacje zmiennopozycyjne
$43-$49	FMSZPG	wykorzystane przez operacje zmiennopozycyjne
$80,$81	LOMEM	wskaźnik dołu pamięci
$82,$83	VNTP	wskaźnik tablicy nazw zmiennych

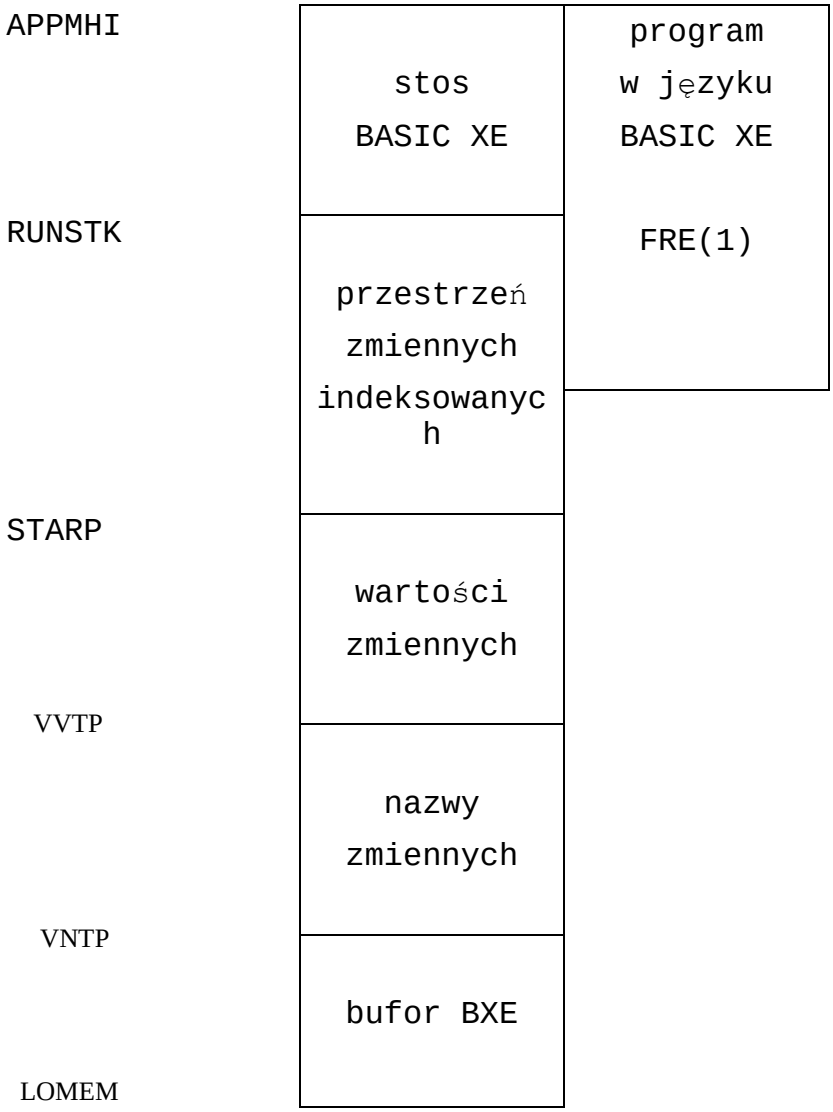
$84,$85	VNTD	wskaźnik końca tablicy nazw zmiennych + 1;
$86,$87	VVTP	wskaźnik tablicy wartości zmiennych
$88,$89	STMTAB	wskaźnik tablicy instrukcji
$8A,$8B	STMCUR	wskaźnik aktualnej instrukcji
$8C,$8D	STARP	wskaźnik tablicy wartości zmiennych indeksowanych
$8E,$8F	RUNSTK	wskaźnik stosu BASIC
$90,$91	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.

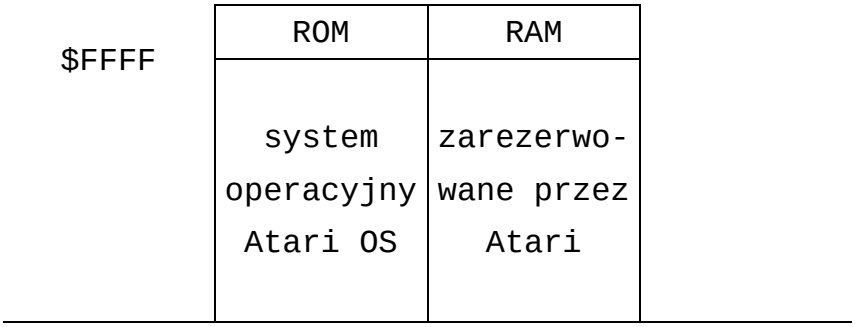


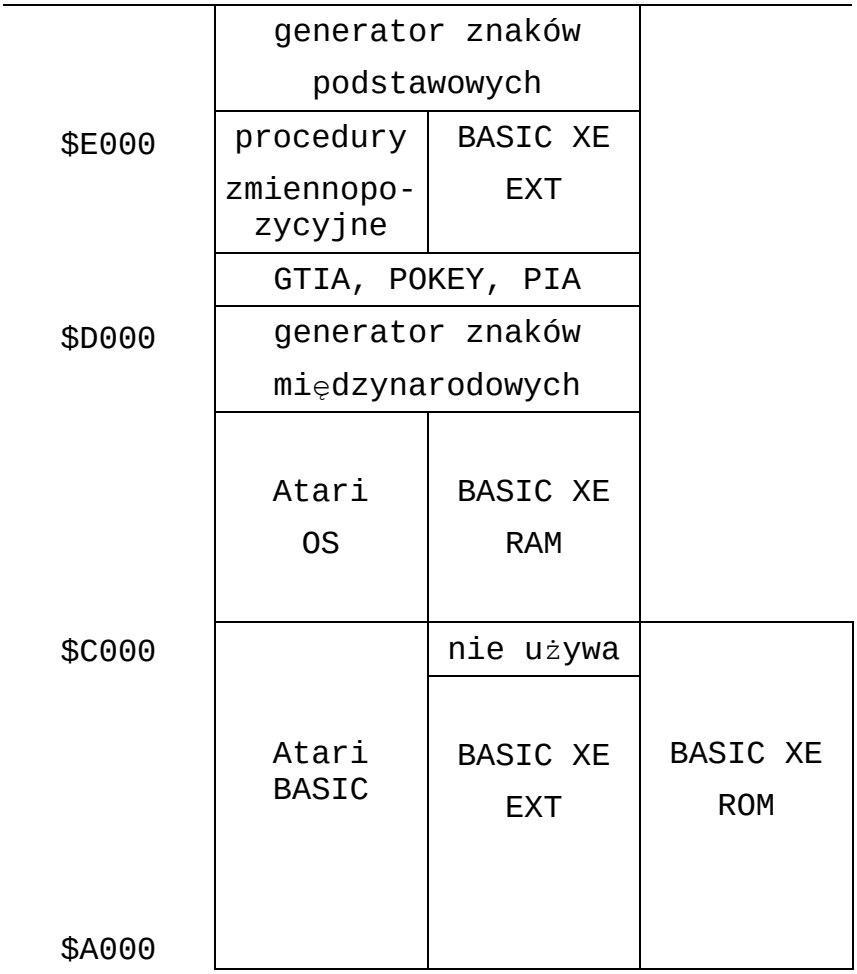

RUNSTK
przestrzeń zmiennych indeksowanych
STARP
program w języku BASIC XE
STMTAB
wartości zmiennych
VVTP
nazwy zmiennych
VNTP
LOMEM


$A000
RAM
GR. RAM
P/M RAM
FRE(0)
dodatkowy RAM 130XE



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.





ADDITION serve important informations of utilization of memory in addition by map of memory ex- < b > BASIC XE BASIC XE. It means in description taking a stand ' ' AtB Atari BASIC, ' ' BXE - BASIC XE. It belongs to underline, that index it from the point of view of programing in language sole important location BASIC XE LOMEM, STOPLN, ERRSAV and PTABV. Acquaintance of this address and however, utilization of instruction is not indispensable POKE, as functions are in language available BASIC XE and instructions on these cells (cellular phones) enabling operation. Note affectation on zero part detailed all cell of (cellular phone of) memory below $ 80 they are used by - $ FF BASIC XE. Address name meaning $ E - $ system index (indicator) free memory $ 20 F APPMHI by operations - $ taken advantage $ 43 2F ZIOCB zmiennopozycyjne by operations - $ 49 taken advantage $ 80 FMSZPG zmiennopozycyjne, $ 81 index (indicator) bottom memory $ 82 LOMEM, $ 83 index (indicator) table name alternate (changeable; variable) $ 84 VNTP, $ 85 index (indicator) end table name alternate (changeable; variable) + 1 VNTD; $ 86, $ 87 index (indicator) table value alternate (changeable; variable) $ 88 VVTP, $ 89 index (indicator) table instruction $ STMTAB 8A, $ 8 -ex- < b > index (indicator) current instruction $ STMCUR 8C, they index index of (indicator of) table of alternate (changeable; variable) value $ $ 8D STARP 8E, $ index (indicator) pile $ 90 8F RUNSTK BASIC, $ 91 index (indicator) summit (top) memory $ MEMTOP BA, $ index (indicator) line detention (halt) program $ register number error $ index (indicator) setup $ BB STOPLN C3 ERRSAV C9 PTABV tabulatora CB - $ D1 - - - - - by - unused $ ~ BXE D - $ register 1 $ D9 FR00 zmiennopozycyjny EO - $ register 2 $ 480 E3 FR1 zmiennopozycyjny - $ 57F - - - - - be by at realization of instruction relatively to - used BXE poszerzeniami AtB. $ 580 - $ 6F7 - - - - - By unused - normally BXE, but INPUT and can cause change of contents of area of this memory

ENTER. $ 680 - $ 6FF - - - - - By - unused BXE. Good place for in engine code podprogramów. By DOS $ used 700-LOMEM and some fix-up (handlery e.g.). Two following drawings show use of memory among LOMEM and start of area ($) cartridge A000. First of they if unused EXTEND, second (other) if it use EXTEND. $ FRAME A000 APPMHIRUNSTKSTARPSTMTAB VVTP VNTP GR. FRAMES 0 FRAME ( ) $ FRAME P/M FRE stosBASIC XE przestrzeńzmiennychindeksowanych programw językuBASIC XE wartościzmiennych nazwyzmiennych LOMEM A000 APPMHIRUNSTKSTARP VVTP VNTP LOMEM GR. Shows additional by from address of start of area of affixed (adjoined) unit FRAMES 0 1 utilization of memory ROM FRAME ( ) ( ) buffer next drawing (RAM130XE P/M FRE programw językuBASIC XEFRE stosBASIC XE przestrzeńzmiennychindeksowanych wartościzmiennych nazwyzmiennych BXE BASIC XE ang. For) $ cartridge FFFF, or to the end address area. Then they are used areas meant by ' ' BASIC XE EXT BASIC XE, when diskette of inclusive expansion was placed during inclusion of (participation of) computer in station of disk D1. $ $ $ $ $ ROM FRAME generator FFFF E000 D000 C000 A000 systemoperacyjnyAtari OS zarezerwo-wane przezAtari znakówpodstawowych proceduryzmiennopo-zycyjne BASIC XEEXT GTIA, POKEY, generator unused PIA znakówmiędzynarodowych AtariOS BASIC XERAM Atari BASIC BASIC XEROM BASIC XEEXT

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.).

ADDITION is completely compatible with (from) in principle with (from) compatibility C Atari BASIC BASIC XE Atari BASIC. It means, that each program written acts in language correctly Atari BASIC if it will be uploaded and by started up BASIC XE. However, several exists differences introduced (presented) slight < money of exchange > below, which (who) can cause, that program will not act correctly. Alternate (changeable; variable) names from the point of view of more of keyword in language BASIC XE than it can happen in (to) Atari BASIC, that alternate (changeable; variable) names some will be interpreted incorrectly. It is possible to notice (to remark) these places on screen instruction LETTER (LIST) simply (easy) program ( ) listując or printer and employ in these places instruction (LET or change alternate (changeable; variable) name). For example, NUMBER is in (to) 100 correct line = 7 Atari BASIC. However, there is keyword in language NUM BASIC XE and in the following way < as follows > this line looks for (after) wylistowaniu 100 era = 7 Num if alternate (changeable; variable) above-mentioned era allude in program not take a stand line answer it is erroneous from the point of view of effects of operations of programs 100 0 that Num. There is solution of this problem as use allude in instructions of attribution LET. However, it belongs to call attention (to pay attention), that alternate (changeable; variable) names can not be with names of functions in language identical BASIC XE. It belongs to change alternate (changeable; variable) name in such case (chance) on other (e.g. change name on BUMP BMP or VBUMP etc.).

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
```

Speed of execution of program is one of important advantage of language relatively to greatest speed of execution of program considerably BASIC XE Atari BASIC. Games certain situation (V! However, it can be ineligible). Employment of delay is sole solution in such case (chance) in program. It is possible to realize (to accomplish) delays e.g. procedures using waiting some time (sometimes) and provocation of its (her) from proper points of programs. Example of procedure serve demanded delay with parameter realizing below procedure it 1000 time

(sometimes) " wait " time (sometimes) 1010 wait 1020 wait = 0 Using Local For it wait form has 1030 exemplary line induction such procedure Next Exit it wait 100 " " 10 Call Using

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.

It changes no of location in published memory in (to) conflicts of utilization of memory manner of utilization following (step) book BASIC XE " " Atari Basic Reference Manual - Atari, Inc. Ce " de " " " – Atari Mapping The Atari COMPUTE! " Memory map " books Master - Educational Software, Inc. However, small chance to meeting of program made exclusively exists for operation at utilization Atari BASIC. It is caused by firm unpublished following (step) fact Atari 1. It uses certain location on zero part in some situations Atari BASIC. 2. Some locations have special meaning on zero part for Atari BASIC. 3. Addresses of some procedures included (reached; made) are other in constant (solid) memory in case (chance) inclusive ROM interpreter Atari BASIC and BASIC XE. Creation allude having at disposal great capability interpretera than Atari BASIC and besides, would not be without introduction of certain change in utilization of memory considerably fastest possible. As above-mentioned facts were not , as it allude , by firm published < publish > Atari, but requirement of utilization of non-typical memory gets (reach) at creation of program seldom, it from they, which (who) do not act be from the point of view of conflicts of utilization of memory at utilization rare correctly hugely interpretera BASIC XE.

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.

Automatic attribution of dimension (measurements) attributes dimension (measurements) alternate (changeable; variable) text 40 alternate (changeable; variable) text automatically BASIC XE, dimension (measurements) have not been declared which (who) instruction DIM. Has it on execution of program written in (to) no influence (income) Atari BASIC. If it would be desirable < covet > from that respects, then it belongs to remember, that use causes SET 11,0 giving up of attributing of automatic size undeclare alternate (changeable; variable) text.

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.

Form program program checking off (note) he (its; his; it) structure listowanego BASIC XE listuje. If there would be ineligible, then passage for to such manner listowania, as it make assure (provide) execution SET 12,0 Atari BASIC.

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
```

ADDITION allows capability of additional utilization of memory of FRAME whole utilization of memory D BASIC XE Atari 130XE. Additional memory of FRAME can be taken advantage as virtual disk ( ) ( ) 64K RAMDISK, as area of memory of program be in mode ( ) EXTEND, it be as memory store by program other informations taken advantage allowing < allow > (e.g. several switched screens etc.). It is possible to use in case of (chance of) computer with memory extended for in mode FRAME 256K BASIC XE EXTEND and build (create) virtual disk in (to) remaining < survivor > ( ) 128K RAMDISK, or as it serve as memory store by program over other informations taken advantage allowing < allow > (e.g. several switched screens etc.), unifying in mode working EXTEND, that is not possible in case (chance) Atari 130XE. Example of curious utilization of additional memory serve for storage of screen in graphic mode below ( ) 64K Atari 130XE. In order to take advantage this program be call one " PICTURE " indispensable or several drawings recorded (written down) on diskette in format by known program applicable < apply (use) > Micro-Illustrator. Other programs, for example, Koala Pad if (or) they use other format of record of memory of screen as package on diskette Atari Artist, but they make it possible record sealing wax as driving in of digital through driving in at drived in digitals INSERT (Micro-Illustrator CONTROL-SHIFT-INSERT CONTROL and) SHIFT. Packages of inclusive drawings can have optional names, but must have * wyróżnik .PIC. From the point of view of it most advantageously, that packages of inclusive drawings are enough big, place DOS on inclusive disk it (them) only and described program. Short comment serve for this program below it is used packages for remembering (storing) name of package read in from catalog of diskette for eight packages 180 alternate (changeable; variable) text $ ( ). Drawing it in format of program 190 7680 bytes of memory of screens Micro-Illustrator. State of (condition of) digital 200 START, SELECT and it is checked by research of contents of cell of (cellular phone of) memory $ OPTION D01F. If it is drived in digital of START significant bit least be zero. Lecture first in station 240 catalog of diskette part < frequent > D1. Read names eight packages can become (stay) 250 maximum. Verification of read name of package 260,270. If number of package is with images (offenses) greatest than 8 use as many sectors be on diskette busy - lecture and exit (outlet) from noose. Creation of name of package will be wanted for execution of instruction in (to) such form 280,290 that OPEN. Setup of alternate (changeable; variable) value depending on amount of read package 300,310 steering. Closure of (shutdown of) channel after lecture of catalog 320. Setup of color for monochrome images 360,370. If images (offenses) are to change graphic mode on 31 color belong and use proper SETCOLOR. Become (stay) 380,390 read in packages so many (so much) < rear >, what find in catalog. Lecture of image 400. Cells (cellular phones) $ 58, they include (reach; make) address of start of memory of screen $ 59. Placing of image 440 1 and in bank 2 0, 3 and in bank 4 1 etc. use of this instruction is safe 470 MOVE, as memory of screen is placed over $ 7FFF. If

maybe value of index (indicator) will be lowered not cross at execution of program operation correctly HIMEM! End 480,490 one package wczytywania. All packages have been read in from diskette 500 and in additional memory place. Become (stay) 570 this noose interrupted digital can only WHILE BREAK or RESET. Choice of (election of) image at utilization of function 600-620 RANDOM. Displacement of image from additional for memory of screen 670-700 memory 64K. START on driving in digital 740 desired < expectation >. But here, program 100 8,20 * * * * * * * * * * 110 * * 120 * PICTURE * 130 * * 140 * * * * * * * * * * * * 150 180 package $ ( ) Rem Rem Rem Rem Rem Rem Dim, 20 $ ( ) 190 size = 40 * 192 200 digital = $ Zbior d01f start with images (offenses) 230 = $ 01 210 220 all package 240 # 1 Rem Rem znajdz Rem Open, 6, 0, " D1 it 8 260 2,2 * " 290 image (offense) = 1 # $ 270 $ ( ) < > " " pope .PIC For Input 1,Zbior If Zbior Then image (offense) 300 280 package $ (Goto;) = " D1 ", 3,10 $ ( ) Zbior, image (offense) " " 290 package $ (; packages ((image (offense) Find;), " ", it read found (been placed) packages 8 350)) = " " 300 image (offense) 310 what = image (offense) 1 320 # 1 330 340 360 24 370 .PIC Next Close Rem Rem Rem Graphics Setcolor 2,6,O it what 390 image (offense) 1,6,8 380 image (offense) = 1 # $ (Setcolor 4,6,O:Setcolor For Open 1,4,0,Zbiory;) 400 # 1 Bget, ($ 58) Dpeek, it overwrite size for additional memory 410 image (offense) 430 image (offense) 1 420 440 bank = (( ) / 2) 450 address = $ 4000 460 = 0 address = $ 6000 470 ($ 58) Rem Rem Rem Int If Obraz&1 Then Move Dpeek, address, size, now it overwrite bank from additional memory to memory of screen about 480 660 image (offense) 1 # 1 490 image (offense) 500 510 image (offense) 520 530 old = = 0 540 550 560 570 1 600 image (offense) = old 610 image (offense) = ( ) 620 630 old = image (offense) 6 640 650 670 bank = (( ) / 2 680 address = $ 4000 690 = address = $ 6000 700 address Close Next Rem Rem mozna ogladac Rem O:Obraz Rem Rem petla nieskonczona Rem While While Random 1,lle Endwhile Rem Rem Rem Int If Obraz&1 Then Move, ($ 58) Dpeek, size, now bank 710 digitals 720 730 740 ( ) & start = 0 Rem Rem mozna ogladac Rem While Peek 790 Endwhile 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.

l60 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

ADDITION serve numbers of errors about errors in addition message E and messages after pronouncement of error on screen appearing (show) and description of reason of pronouncement of possible error. It takes a stand 1 note BREAK key TRAPed, if use SET 0,1 and digital has been drived in in execution time of program BREAK. Possible personal attendance of (service activity of) error through instruction TRAP. Whole available memory has been taken advantage 2 memory Full. Program belongs to modify number of instruction decreasing (fall off; belittle) or alternate (changeable; variable) dimension (measurements) index. give 3 value alternate (changeable; variable) Value Out of Range or formulations beyond admissible range. complicate alternate (changeable; variable) 4 man oversized Too Variables, i.e. more than admissible 128. complicate surpass declare dimension (measurements) alternate (changeable; variable) text 5 paste Acces String DIM. 6 But it give instruction after end in instruction DATE attempt (test) lecture (instruction) DATE READ READ. Number of line has taken a stand greatest than 32767 7 > 32767 Val or smallest than zero. Some other instructions (e.g. can cause pronouncement of error CULMINATING POINTS number 7) also. define value 8 read instruction INPUT/READ Type Mismatch INPUT or it does not answer

alternate (changeable; variable) typical < type > READ, has be assign which (who) (e.g. attempt of (test of) reading of text for alternate (changeable; variable) numeric) 9 attempt (test) use undeclared table DIMensioning or attempt of (test of) execution of instruction for alternate (changeable; variable) DIM, dimension (measurements) has been defined which (who) instruction already DIM. Formulation complicate 10 far too < market > Expression too Complex. Formulation belongs to divide on several instructions. Small has taken a stand in result of operation in procedure of arithmetics 11 too Overflow/Underflow zmiennopozycyjnej or oversized number (e.g. attempt of (test of) dividing by zero). Does not have in program of advisable line in instruction 12 note Line Found GOTO, GOSUB or IF - THEN. Without proper (suitable) instruction 13 met instruction NEXT without FOR NEXT FOR. This error can take a stand as result of use of unsuitable instruction POPE also. Line is met 14 far too < market > debt < long > Line too Long or Complex, or it complicate far too < market >, in order to there could be processed by BASIC XE. Dividing of line is solution on several shortest. It has taken a stand 15 note Line Found NEXT or RETURN when proper (suitable) instruction FOR or canceled become (stay) GOSUB. This error can take a stand e.g. in case (chance) when STOP < ALLOY > operation of program has been interrupted instruction, line become (stay) canceled but operation of program proceed with (resume) next instruction CONT. Without induction 16 met instruction instruction RETURN without GOSUB RETURN podprogramu GOSUB. This error can take a stand as result of use of unsuitable instruction POPE also. Consistent is not with syntax of language 17 line program (Bad Line BASIC XE e.g. 110 ERROR -.). 18 Notes but there is not figure first sign argument function Number VAL. It are not contained it can take a stand in available memory 19 loaded program (Too Big Load if e.g. it change LOMEM, use other DOS, memory occupying (deal with) more etc.). In instruction in (to) /you smallest than 0 20 bad number fix-up (Invalid Channel or greatest than 7). 21 Note format attempt (test) boating (loading) instruction package File LOAD LOAD, it has not been built (created) which (who) instruction SAVE. It takes a stand 22 USING String Too Big if determination of format is long in instruction nile 255 signs PRINT USING, or one field (tail) is long in format than 59 signs. Greatest is assigned value for unsubscribing 23 instruction USING Value Too Big PRINT USING than + 50 1E. It does not answer types of values in instruction for unsubscribing 24 format field (USING Type Mismatch PRINT USING e.g. attempt of (test of) unsubscribing of numeric value define as text on field (tail)). It does not answer has be attribute for which (who) 25 text length alternate (changeable; variable) text instruction RGET DIM Mismatch wczytywany RGET. 26 Record instruction RGET Type Mismatch wczytywany RGET and has be attribute alternate (changeable; variable) for which (who) not be same type. 28 Met instruction Invalid Structure ENDIF or without proper (suitable) instruction ENDWHILE IF or WHILE. 29 # Bad number object type P/M Out of Range P or proper (suitable) numbers are for objects of types from 0 for 3 .m P, for objects of types from 4 for 7 m. It use instruction without initialization 30 diagrams (graphics) note operating graphics instruction P/M Graphics Active P/M P/M PMG. 1 Or PMG. 2. be instruction 32 ENTER OF note end introduction program ENTER TRAPed. It takes a stand, if use SET 9,1. 34 Can _ NUM/RENUM Wyr num1 or in instruction _ wyr num2 NUM either (or) equal zero RENUM. Maximum number of line has been surpassed in time 35 can program (32767) NUM/RENUM przenumerowywania. be as text table 40 vice versa be It attempt (test) use alternate (changeable; variable) text String Type Mismatch. It be vice versa . Note be as program in mode 65 memory attempt (test) booting (embarkation) program recorded (write down) EXTENDed Available EXTEND or execution of instructions EXTEND if it lacks computer sufficing (enough) memory. 100 Note Extension Installed! Attempt of (test of) use of available instruction only if diskette was placed at switching on of computer in station D1 BASIC XE. Unexist 129 attempt (test) opening channel Channel Already OPEN, which (who) has been opened already. 130 But does not have serve fix-up in table of available fix-up Device Handler CIO. Bad symbol of fix-up serve, or it serve he (it) absolutely (e.g. it serve name of disk package without only Dn). allow from channel 131 attempt (test) lecture Write Only, which (who) has been opened for record only. Admissible is not served code of operation in (to) /you for given fix-up 132 Bad Device Cmd. It can take a stand in unsuitably used instruction XIO or OPEN. By channel 133 note attempt (test) execution operation Channel OPEN, it has not been opened which (who). For channel 135 attempt (test) record Read Only, which (who) has been opened for lecture only. 136 Achieved end package End-Of-File. Does not have data more. Limit of time has expired on answer for given fix-up 138 assigned Device Timeout. In general reasons of such situations it does not have such fix-up - it is connected fix-up for computer -

unsuitably does not have included (switched on) supplying - fix-up maybe besides, there is mean in case of (accidentally of) tape recorder out of order put defect tape be it - fix-up evil (poorly) < evil >, it be defect tape , bad setup of head, it be rotary speed. It does not answer 139 fix-up Device NAK. Unsuitable connection of fix-up reason can be, error in parameters steering or inefficiency of fix-up. Screen on position 141 attempt (test) execution operation graphic Position, which (who) does not exist in (to) data graphic mode. Is not execute in state (condition) 144 demand action (function) fix-up (Device Done e.g. attempt of (test of) record on indemnified diskette). Unexist 145 graphic mode attempt (test) use Invalid GR Mode. 147 But does not have sufficient amount of free memory for demanded graphic mode memory for GR Mode. Does not identify served number of station of disk # DOS l60 Invalid Drive. Bad number serve, or it is not included (switched on) station. Does not have buffers on opening of next package 161 allow man DOS more Too OPEN Files. Does not have place on diskette 162 more Disk Full. 165 Illegal name package Bad File Name. Informations be placed about names admitted (committed) by given DOS in (to) its (his) description of /service manual (disk operating system). For package indemnified before record 167 attempt (test) record File PROTECTed and cancel. Capacity of catalog of package of diskette has been surpassed 169 DIRectory Full. To record (to write down) packages not possible more. Does not have such package on diskette 170 note File Found. For unexisting place on diskette 171 culminating point attempt (test) execution instruction CULMINATING POINT Bad Value, or it has not been opened package in mode of record of /lecture (12). Literature 1. Collective work. Atari Basic. Language of programing and attendance of (service activity of) microcomputer Atari. Wyd. NOTES, warsaw 1987. 2. Adam Stawowy. Atari Basic XL Wyd. NOTES, warsaw 1987. 3. Mariush Giergiel. Atari Basic XE. Wyd. NOTES, warsaw 1987. 4. 400/800 ATARI Basic Reference Manual