ZAKŁADY URZADZEŃ KOMPUTEROWYCH
M E R A E L Z A B
ul. Kruczkowskiego 39
41-808 Z A B R Z E

BASIC – MERITUM

PODRĘCZNIK PROGRAMOWANIA I UŻYTKOWANIA

44990001

Spis treści

Rozdział 1. Ogólne informacje o systemie MERITUM ... 3

Rozdział 2. Podstawowa obsługa techniczna systemu z modelem MERITUM-I ... 5

2.1 Włączanie systemu ... 6
2.2 Obsługa ... 6
2.3 Magnetofon jako urządzenie wejścia-wyjścia ... 6
2.4 Uwagi dotyczące użytkowania magnetofonu ... 7

Rozdział 3. Komunikacja użytkownika z komputerem ... 9

3.1 Organizacja i wykorzystanie ekranu ... 10
3.2 Tryby pracy ... 11
3.3 Sposoby sterowania pracą komputera ... 12
3.4 Informacje o błędach ... 14
3.5 Uruchamianie programów ... 14
3.6 Uwagi ... 14

Rozdział 4. Opis zleceń komputera MERITUM-I dla trybu bezpośredniego ... 15

Rozdział 5. Opis wersji języka BASIC dla modelu MERITUM-I ... 21

5.1 Informacje ogólne o składni ... 22
5.2 Definicje typów zmiennych ... 22
5.3 Operatory ... 24
5.4 Kolejność wykonywania operacji ... 26
5.5 Definicje funkcji matematycznych ... 27
5.6 Funkcje pochodne ... 30
5.7 Skróty ... 30
5.8 Instrukcje ... 30
5.8.1 Instrukcje wejścia-wyjścia ... 31
5.8.2 Instrukcje organizacyjne i sterujące przebiegiem wykonywania programu ... 40
5.8.3 Instrukcje specjalne ... 52
5.8.4 Instrukcje operacji na łańcuchach ... 59

Rozdział 6. Praca w trybie edycji ... 66

6.1 Zlecenia sterujące przebiegiem sesji edycji ... 67
6.2 Klawisze funkcyjne edycji ... 69
6.3 Zlecenia edycji ... 70

Rozdział 7. Praktyczne uwagi dotyczące optymalizacji programów ... 74

Dodatek 1. Wykaz błędów sygnalizowanych przez komputer

Dodatek 2. Mapa pamięci systemu MERITUM-I

Dodatek 3. BASIC - MERITUM zestawienie instrukcji

Dodatek 4. Tabela kodów sterujących, znaków ASCII i znaków graficznych

OGÓLNE INFORMACJE O SYSTEMIE MERITUM

MERITUM to nazwa całej rodziny komputerów osobistego użytku /tzw. komputerów osobistych, ang.: personal computers/ i opartych o nie systemów przetwarzania danych.
Podstawową cechą systemu MERITUM jest możliwość modularnej rozbudowy tak konfiguracji sprzętowej jak i oprogramowania. Komputer MERITUM I – pierwszy z proponowanych modeli – jest modelem bazowym, właściwym MERITUM proponowanej koncepcji.

MERITUM - I jest typowym komputerem osobistym: małym, lekkim i funkcjonalnym. Umieszczony jest we wnętrzu obudowy typowej klawiatury. Konstrukcja jego oparta jest o efektywny mikroprocesor Z-80. Pamięć systemu zawiera 14 kbajtów pamięci EPROM oraz 16 lub 17 kilobajtów pamięci RAM. Klawiatura ma typowy układ klawiszy /typ QWERTY/.

W oparciu o komputer MERITUM - I zestawić można tani prosty w obsłudze i stosunkowo efektywny system MERITUM - I w skład którego wchodzą:

1. Komputer MERITUM - I umieszczony w obudowie klawiatury oraz zasilacz sieciowy umieszczony w oddzielnej obudowie.

2. Standardowy telewizor czarno-biały /lub prosty monitor ekranowy np. typu Neptun 156 w obudowie telewizora Neptun 150/ w charakterze monitora /klawiatura + telewizor lub monitor stanowią konsolę operatorską/.

3. Magnetofon kasetowy wraz ze zwykłymi kasetami jako trwały nośnik informacji /tzw. pamięć masowa – zewnętrzna – dla przechowywania programów i zbiorów danych/.

4. Dodatkowo – dowolna drukarka o interfejsie szeregowym lub równoległym dla otrzymania trwałej kopii. /Komputer dysponuje typowym interfejsem szeregowym oraz trzema równoległymi rejestrami we-wy w oparciu o które zbudować można dowolny interfejs równoległy/.

Model MERITUM - I programowany jest w rozszerzonej wersji języka BASIC oraz posiada bogate oprogramowanie umożliwiające pisanie, uruchamianie i poprawianie programów pisanych w BASIC'u. Wyposażony jest również w pełne oprogramowanie umożliwiające efektywne korzystanie z wymienionych powyżej czterech podstawowych składników systemu. Wersja języka BASIC dla MERITUM - I zawiera z jednej strony np. zestaw efektywnych instrukcji operowania na zmiennych łańcuchowych z drugiej umożliwia np. komunikację z programami tworzonymi w języku wewnętrznym mikroprocesora Z-80. Zastosowanie ekranu TV lub monitora jako konsoli wyjściowej daje szerokie możliwości prezentacji informacji w tym w formie graficznej.

Komputer MERITUM - I jest również podstawą do tworzenia następnych generacji systemów MERITUM o znacznie szerszych możliwościach. Zmieniając bowiem znajdujące się w pamięciach EPROM oprogramowanie podstawowe oraz dołączając bardziej efektywne urządzenia peryferyjne /np. mini dyski elastyczne, telewizor kolorowy/ można będzie konfigurować kolejne wersje systemu: MERITUM - II, MERITUM - III itd.

W niniejszym podręczniku omówiono sposób użytkowania systemu MERITUM - I w zakresie efektywnego programowania w języku BASIC. Zakłada się że czytelnikowi znane są tak podstawowe pojęcia z dziedziny programowania jak: program, instrukcja i jej argument, zlecenie i jego argument itd..

PODSTAWOWA OBSŁUGA TECHNICZNA SYSTEMU Z MODELEM MERITUM-I

2.1 Włączanie systemu

READY
>_

Magnetofon zatrzymuje się klawiszami STOP lub PAUZA.
W momencie kiedy komputer wykryje błąd odczytu w miejsce pierwszej gwiazdki pokaże się znak "C". Operacja odczytu musi być wówczas powtórzona.
Użytkownik uruchamia wprowadzony program zleceniem RUN.
W celu zapisania programu na magnetofon należy wprowadzić zlecenie CSAVE z nazwą zbioru ujętą w cudzysłów

>CSAVE "NAZWA" –

Nazwa może zawierać do sześciu liter, z tym, że przy wyszukiwaniu zbiorów na taśmie podczas operacji odczytu /zlecenie CLOAD/ brana jest pod uwagę tylko pierwsza litera. Następnie należy wcisnąć równocześnie klawisze START i ZAPIS. Przyciskiem (ENTER) uruchamia się operacje zapisu informacji na kasetę. Kończy się ona komunikatem :

READY
>_

Po zapisaniu programu na taśmie należy sprawdzić poprawność zapisu zleceniem CLOAD ? Postępowanie przy zleceniu CLOAD ? jest identyczne jak przy zwykłym zleceniu CLOAD "NAZWA".
Komputer porównuje program zapisany na taśmie z programem w pamięci. W przypadku wystąpienia błędu pojawi się komunikat "BAD". Należy wówczas powtórzyć operację zapisu najlepiej na innym odcinku taśmy.

2.4 Uwagi dotyczące użytkowania magnetofonu

1. Taśmy magnetofonowe używane do przechowywania programów komputerowych mogą być standardowe z tym, że nie powinny być pozaginane i porysowane. Odcinki uszkodzone powinny być eliminowane.

2. Dla zwiększenia pewności zapisu można programy nagrywać dwu –, trzykrotnie.

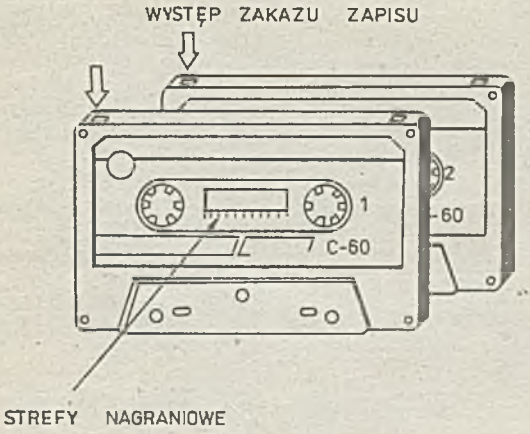
3. Programy o dużej objętości powinny być nagrywane na taśmach nowych i nieuszkodzonych. /Im dłuższy program tym większe prawdopodobieństwo błędu/.

4. Jeden błąd przy zapisie na kasetę może wyeliminować całe nagranie.

5. Należy pamiętać, że kasety posiadają odcinki rozbiegowe na obu swoich końcach.

6. Zaleca się stosowanie kaset jak najkótszych; 20 – 10 minutowych. Na każdej kasecie – bez specjalnej straty – może znajdować się wtedy jeden program nagrany wielokrotnie. Upraszcza to znacznie posługiwanie się magnetofonem jako pamięcią zewnętrzną.

7. W przypadku gdy na jednej długiej kasecie /np.: C–60/ chcemy nagrać kilka różnych programów, najlepiej rozpoczynać ich nagrywanie na początku każdej strefy nagraniowej zaznaczonej na kasecie.



Kolejność, liczbę programów i ich nazwę zapisać należy na kartce dołączonej do każdej kasety. Pełni ona rolę kartoteki programów.

8. Jeżeli zamierza się zapisywać ostateczną wersję programu, należy po zapisaniu programu wyłamać występ zakazu zapisu znajdujący się w tylnej ściance kasety.

9. Nie należy poddawać nagranych kaset działaniu pola magnetycznego.

10. W celu zwiększenia niezawodności magnetofonu należy utrzymywać głowice magnetofonu i mechanizmy napędzające taśmy w doskonałej czystości. Do czyszczenia głowic najlepiej wykorzystywać taśmy czyszczące typ KS-1 produkcji Stilon. Należy również często sprawdzać łatwość przesuwu taśmy w kasecie.

11 Dla uproszczenia posługiwania się dużą liczbą programów można przed każdym programem nagrać słowny komentarz opisujący jego działanie (przewód łączący magnetofon z komputerem winien być wówczas odłączony).

12. Magnetofon i kasety należy umieszczać w najdalszej możliwej odległości od odbiornika TV i źródeł zasilania.

KOMUNIKACJA UŻYTKOWNIKA Z KOMPUTEREM

3.1 Organizacja i wykorzystanie ekranu

Po uruchomieniu komputera i telewizora /jak opisano wyżej/ na ekranie pojawi się napis

OBSZAR PAO ? – (tzn. obszar pamięci operacyjnej?)

Użytkownik powinien:

– przycisnąć klawisz funkcyjny (ENTER) jeśli chce pracować w języku BASIC i mieć do dyspozycji całą dostępną pamięć RAM /w MERITUM - I jest to 16 kbajtów tj. 15 572 bajty/, lub

– zarezerwować fragment pamięci jeśli chce ładować programy w języku wewnętrznym /przy użyciu zlecenia SYSTEM/.

Po wykonaniu powyższych czynności na ekranie pojawi się napis:

MERA ELZAB – ITM BASIC PC
READY
>–

Oznacza to, że system jest gotowy do pracy i oczekuje na zlecenia.

Ekran może być zorganizowany na dwa sposoby:
1. 16 linii x 64 znaki w linii /normalny tryb pracy/
2. 16 linii x 32 znaki w linii.

Dla wprowadzenia organizacji o 32 znakach w linii należy przycisnąć jednocześnie klawisze funkcyjne (SHIFT) oraz (→)

Dla powtórnego wprowadzenia organizacji o 64 znakach należy nacisnąć klawisz funkcyjny CLEAR /jednocześnie zerowany jest ekran/.

Zmianę organizacji ekranu można również uzyskać "wewnątrz" programów lub poprzez wykonanie pewnych zleceń. CLS lub NEW (oprócz innych funkcji) wywołuje organizację o 64 znakach a PRINT CHR$(23) organizację o 32 znakach w linii.

Monitor zaopatrzony jest w specjalny znak / – / wskazujący miejsce wprowadzenia kolejnego znaku. Jest to tzw. kursor.

W normalnym trybie pracy na ekranie mieszczą się 1024 komórki znakowe /znaki/ /od 0 – 1023/, które ponumerowane są w sposób następujący:

– pierwsza linia od góry: komórki od 0 – 63 licząc od strony lewej do prawej

– następne linie analogicznie:
64 – 127
128 – 191
itd.
– ostatnia linia od 960 – 1023.

Każda z komórek podzielona jest na 6 równych części, które mogą być wyświetlone w dowolnej kombinacji /znaki semigraficzne/. Dzięki temu możliwe jest przedstawienie na ekranie informacji w postaci semigraficznej. Specjalna grupa instrukcji umożliwia adresowanie /wyświetlanie/ dowolnej z tych 6 części.
Istnieją instrukcje umożliwiające sekwencyjny /PRINT/ lub swobodny /PRINT @ pozycja/ dostęp do każdej komórki i wyprowadzenie dowolnego znaku alfanumerycznego lub semigraficznego.

Linie podzielone są na pola znakowe. W normalnym trybie pracy w linii znajdują się 4 pola 16-komórkowe. Istnieje możliwość przesunięcia kursora do początku każdego z pól /znak średnika w liście elementów instrukcji PRINT/ lub do 0, 8, 16, 24, 32, 40, 48, 56 znaku w każdej linii /klawisz funkcyjny→/. Jest to realizacja tzw. tabulacji poziomej.

W ogólnym przypadku, jeżeli użytkownik kiedykolwiek zamierza napisany na ekranie tekst /linia programu, zlecenie itd./ wprowadzić do komputera /np. dla zapamiętania numerowanej linii programu lub uruchomienia akcji napisanego na ekranie zlecenia/ musi przycisnąć klawisz (ENTER). /Wyjątek stanowią zlecenia wewnątrz trybu edycji/.
W większości przypadków w przykładach pomijać będziemy umieszczanie w tekście klawisza (ENTER). W niektórych przypadkach będzie on umieszczany dla przypomnienia lub podkreślenia jego roli.

3.2 Tryby pracy

Komputer może pracować w czterech trybach pracy:

1. Tryb bezpośredni /kalkulatorowy/

2. Tryb edycji

3. Tryb wykonywania programu

4. Tryb systemowy.

Ad.1. Po włączeniu i naciśnięciu klawisza (ENTER) komputer znajduje się w trybie bezpośrednim, na ekranie pojawia się znak >. Znak > wyprowadzony przez komputer, zawsze oznacza że jest on w trybie bezpośrednim i można wprowadzać zlecenie,nienumerowaną lub numerowaną linię programu. Tryb ten umożliwia:

a) wykonywanie natychmiastowych akcji w tym obliczeń /"praca kalkulatorowa"/ – poprzez wprowadzenie do komputera nienumerowanej linii programu w języku BASIC,

b) pisanie programów poprzez wprowadzanie do pamięci programu komputera ciągu numerowanych linii w języku BASIC,

c) sterowanie przez użytkownika pracą całego systemu poprzez natychmiastowe wykonywanie przez komputer tzw. zleceń /np. uruchom program – RUN, wprowadź program z kasety magnetofonowej do pamięci – CLOAD itp./.

W szczególności możliwe są zlecenia wprowadzające komputer w pozostałe tryby pracy:

– zlecenie RUN – w tryb wykonywania programu
– zlecenie EDIT – w tryb edycji
– zlecenie SYSTEM – w tryb systemowy

Ad.2. Tryb ten umożliwia edycję /inaczej: redagowanie, poprawianie/ zawartości linii programu. Możliwa jest zmiana fragmentu wymagającego poprawy bez konieczności przepisywania całej linii. Komputer automatycznie przechodzi również do tego trybu przy wykryciu błędu syntaktycznego - SN w pewnej linii w trakcie wykonywania programu. Wskazywany jest jednocześnie numer błędnej linii. W trybie edycji użytkownik dysponuje całą grupą zleceń umożliwiających efektywne poprawianie tekstu. W celu wyjścia z trybu edycji należy przycisnąć klawisz Q, E lub (ENTER).

Ad.3. W trybie tym komputer wykonuje program /w następstwie zlecenia RUN/ po wyzerowaniu zmiennych numerycznych i przypisaniu zmiennym łańcuchowym łańcuchów pustych.

Ad.4. W trybie tym możliwe jest ładowanie do pamięci i uruchamianie programów w języku wewnętrznym /maszynowym/. Programy te /w tym i zbiory/ mogą być następnie przetwarzane przez programy napisane w BASIC'u lub mogą być traktowane jako zupełnie niezależne programy.

3.3 Sposoby sterowania pracą komputera

Istnieją trzy podstawowe sposoby sterowania pracą komputera przez użytkownika:

a) Wprowadzanie zleceń /w trybie bezpośrednim lub trybie edycji/ poprzez napisanie nazwy zlecenia przy użyciu zwykłych klawiszy alfanumerycznych /np. NEW w trybie bezpośrednim, 5 D w trybie edycji itd./. Opis zleceń dopuszczalnych w trybie bezpośrednim zawarto w rozdziale 4. Opis zleceń dopuszczalnych w trybie edycji zawarto w rozdziale 6.

b) Spowodowanie wykonywania programu w języku BASIC.
Wykonywanie programu w języku BASIC to główne przeznaczenie komputera MERITUM - I. Opis składni języka BASIC przedstawiono w rozdziale 5. Podstawowym elementem składni są instrukcje sterujące przebiegiem wykonywania programu.

W szczególności jako instrukcje odbierane są przez komputer wszystkie zlecenia trybu bezpośredniego /z wyjątkiem zlecenia CONT/ umieszczone w programie jako jego linie.

Treść linii programu /w tym i słów stanowiących instrukcje, funkcje itp. np. FOR, IF, SIN,... itd/, należy pisać przy pomocy zwykłych klawiszy alfanumerycznych. Wykonywanie danej linii programu odbywać się może:

– w trybie bezpośrednim /linia nienumerowana/

– w trybie wykonywania programu /jako jedna z ciągu linii numerowanych/.

W obu przypadkach działanie komputera polega na analizie źródłowej postaci programu /tzn. takiej formy w jakiej wprowadził program użytkownik/ linia po linii i natychmiastowym wykonywaniu czynności wskazanych przez napotkane i zdekodowane elementy składni /instrukcje, funkcje, wyrażenia/. Jest to realizacja programu z przeplataniem faz tłumaczenia i wykonywania poszczególnych linii programu czyli na tzw. zasadzie interpretacji /w odróżnieniu od zasady kompilacji tzn. całościowego przetłumaczenia wszystkich linii na program w języku wewnętrznym celem późniejszego jego wykonania/.
Tak więc MERITUM - I wyposażony jest w interpretator języka BASIC.

Przypomina się, że wprowadzenie do pamięci komputera napisanej na ekranie linii celem jej zapamiętania lub natychmiastowego wykonania odpowiadającej jej akcji odbywa się zawsze po naciśnięciu klawisza (ENTER)

c) Przyciśnięcie jednego z klawiszy funkcyjnych /nie alfanumerycznych/

Opis akcji podejmowanych przez komputer po naciśnięciu różnych klawiszy funkcyjnych w różnych trybach pracy zamieszczono na następnej stronie.

Tryb:	Opis klawisza lub kombinacji klawiszy	Podejmowana akcja:
Edycji	Opis funkcji klawiszy używanych w trybie edycji przedstawiono wraz z opisem grupy podzleceń edycyjnych w rozdziale 6. /ze względów dydaktycznych/	
Systemowy	(ENTER)	Wprowadzenie do komputera informacji pisanych na ekranie przez użytkownika /np. adres startowy programu w języku wewnętrznym/
Bezpośredni	(ENTER)	Komputer analizuje napisaną na ekranie linię i odpowiednio ją wykonuje /np. zlecenie/. Jeśli jest to numerowana linia programu komputer zapamiętuje linię w pamięci programu. Jeśli jest to nienumerowana linia programu – komputer wykonuje ją natychmiast.
	(↓)	Nowa linia. Przesunięcie kursora w dół do następnej linii ekranu monitora.
	(SHIFT)(→)	Przełączenie monitora na format 32 znaki w linii.
	(CLEAR)	Zerowanie ekranu i przełączenie go na format 64 znaki w linii.
	(:)	Oddzielenie instrukcji lub wyrażeń języka BASIC w celu umieszczenia ich większej ilości w jednej linii programu np.: PRINT "1 LINIA PROGRAMU": PRINT "2 LINIA PROGRAMU" ENTER Jedna linia programu może zawierać maksimum 255 znaków
	(←)	Cofnięcie kursora i skasowanie ostatnio wprowadzonego znaku.
	(SHIFT)(←)	Skasowanie zawartości całej linii i cofnięcie kursora do początku linii.
	(→)	Przesunięcie kursora do następnej pozycji tabulacji poziomej. Pozycje tabulacji znajdują się (do organizacji ekranu 64 znaki w linii) kolejno w miejscu 0, 8, 16, 24, 32, 40, 48 i 56 znaku w każdej z linii.
	(SHIFT)(@)	Zatrzymanie listowania programu wywołanego zleceniem LIST, wznowienie – naciśnięcie dowolnego klawisza.
Wykonywania programu	(BREAK)	Zatrzymanie wykonywania programu. Wznowienie wykonywania możliwe jest poprzez wprowadzenie zlecenia CONT.
	(SHIFT)(@)	Pauza. Zatrzymanie wykonywania programu łącznie z "zatrzymaniem" ekranu podczas listowania programu zleceniem LIST /traktowanym jako instrukcja/. Wznowienia działania dokonuje się poprzez naciśnięcie dowlnego klawisza.
	(ENTER)	Jeśli w trakcie wykonywania programu komputer oczekuje na wprowadzenie informacji to przyciśnięcie klawisza powoduje wprowadzenie napisanego na ekranie łańcucha danych.

3.4 Informacje o błędach

W trakcie realizacji programu komputer wyprowadza na ekran informacje o typie wykrytego błędu poprzez podanie dwuliterowego kodu błędu. Pełny wykaz kodów błędów i ich znaczeń zamieszczono w Dodatku 1. W momencie wykrycia błędu syntaktycznego (SN ERROR) system przechodzi do trybu edycji w danej linii przy czym lokalizację błędu pozostawia się użytkownikowi.

3.5 Uruchamianie programów

Dla ułatwienia uruchamiania /zatrzymania, powtórnego uruchamiania, śledzenia itd./ programów napisanych w języku BASIC przewidziane są następujące możliwości:

- przerwanie wykonywania programu klawiszem (BREAK) lub przy napotkaniu w programie instrukcji STOP,
- po zatrzymaniu programu możliwe jest np. sprawdzenie wartości zmiennych /przy użyciu instrukcji PRINT/ lub uruchomienie programu z nowymi wartościami,
- wznowienie wykonywania programu poprzez wprowadzenie zlecenia CONT;
- śledzenie przebiegu wykonywania programu przy użyciu zleceń TRON i TROFF w trybie bezpośrednim /wprowadzenie zlecenia TRON i uruchomienie programu zleceniem RUN/ lub przy użyciu TRON i TROFF jako instrukcji wewnątrz programu.

3.6 Uwagi:

1. Przy pisaniu zleceń dotyczących ostatnio wprowadzonej, poprawionej, listowanej ... itd. linii programu zamiast numeru tej linii /parametr zlecenia/ użyć można dla uproszczenia znaku kropki / . /.

OPIS ZLECEŃ KOMPUTERA MERITUM-I DLA TRYBU BEZPOŚREDNIEGO

W rozdziale tym zawarte są opisy: sposobów wprowadzania zleceń oraz akcji jakie komputer podejmuje po przyjęciu zlecenia w trybie bezpośrednim. Zlecenia opisane są w porządku alfabetycznym. Parametry zleceń występują bezpośrednio /bez spacji/ za zleceniem chociaż wstawienie spacji przed parametrami jest również akceptowane przez komputer. Grupa podzleceń dla trybu edycji opisana jest w rozdziale 6.

AUTOnumer linii, krok

Włącza automatyczną numerację linii programu od linii określonej poprzez "numer linii", przy czym kolejne linie numerowane są narastającymi liczbami różniącymi się o "krok". Jeśli pominie się parametr "krok" to wartość kroku ustalona zostaje na 10. Jeśli pominie się oba parametry to pierwsza linia przyjmuje wartość 10 a następne kolejno 20, 30 ... itd. W wyniku przyjęcia tego zlecenia przez komputer po każdym naciśnięciu klawisza (ENTER) podawany będzie numer następnej linii. Naciśnięcie klawisza (BREAK) kończy działanie automatycznej numeracji. Klawisz ten może być również użyty dla zachowania dotychczasowej zawartości, uprzednio wprowadzonej już linii o tym samym numerze co generowany aktualnie w procesie auto-numeracji /za numerem linii pojawia się w takim przypadku gwiazdka/.

Uwaga: Zakończenie wprowadzania programu w trybie auto-numeracji i spowodowanie przejścia do trybu bezpośredniego />/ odbywa się poprzez naciśnięcie klawisza (BREAK)

CLEARn

Zeruje wszystkie zmienne /w przypadku zmiennych łańcuchowych – przypisuje im łańcuchy puste/ oraz rezerwuje n bajtów na łańcuchy tekstów. Komputer po załączeniu a także po zleceniu RUN automatycznie wykonuje CLEAR 50.

CLOAD"nazwa"

Ładuje program /w języku BASIC/ o wymienionej "nazwie" z kasety magnetofonowej do pamięci komputera. Program znajdujący się dotychczas w pamięci jest niszczony. Przy przeszukiwaniu zawartości kasety od dowolnego miejsca pomijane są wszystkie kolejne programy /zbiory/ o innych nazwach. Ładowany jest pierwszy napotkany program o danej nazwie przy czym nazwy rozróżniane są z dokładnością do pierwszej litery. Wykonanie CLOAD z pominięciem "nazwy" powoduje załadowanie pierwszego napotkanego programu. W momencie wykrycia sekwencji startowej programu w prawym górnym rogu pojawiają się dwie gwiazdki "**", przez cały czas prawidłowego odczytu informacji z kasety druga gwiazdka miga. W trakcie ignorowania programu o niezgodnej nazwie po napotkaniu początku tego programu w miejscu pierwszej gwiazdki pojawia się pierwsza litera jego nazwy. Po wykonaniu CLOAD"nazwa" (ENTER) należy uruchomić funkcję odczytu magnetofonu.

CLOAD?"nazwa"

Porównuje program znajdujący się aktualnie w pamięci komputera z programem o podanej "nazwie" znajdującym się na kasecie magnetofonowej.

W przypadku napotkania jakiejkolwiek różnicy pojawi się komunikat "BAD" (z ang. - źle). W trakcie czytania programu z kasety na ekranie pojawiają się gwiazdki jak w zleceniu CLOAD. Aktualna zawartość pamięci komputera nie jest niszczona. Uwagi odnośnie znaczenia i występowania lub braku parametru "nazwa" są takie same

jak w zleceniu CLOAD. Zlecenie służy najczęściej do sprawdzenia poprawności przepisania programu na taśmę magnetofonową. Po wykonaniu CLOAD?"nazwa" (ENTER) należy uruchomić odczyt magnetofonu. Przy sprawdzaniu poprawności zapisu należy pamiętać o konieczności przewinięcia taśmy.

Uwaga: 1. Jeśli w trakcie wykonywania zlecenia CLOAD lub CLOAD? na ekranie nie pojawią się dwie gwiazdki lub po pojawieniu się dwu gwiazdek jedna z nich nie będzie migać należy przerwać czytanie przyciskiem (RESET) po czym powtórzyć zlecenie /przewijając uprzednio taśmę/.

Uwaga: 2. Przy organizacji ekranu 16 linii x 32 znaki – druga gwiazdka nie jest widoczna na ekranie.

CONT

Wznawia dalsze działanie programu zatrzymanego uprzednio klawiszem (BREAK) lub po wystąpieniu w programie instrukcji STOP /patrz opis instrukcji STOP/. Każde inne wykonanie zlecenia CONT traktowane jest jako błąd. CONT wraz z klawiszem (BREAK) i instrukcją STOP stanowi najprostszy mechanizm do uruchamiania programów. Przed wprowadzeniem CONT, w czasie przerwy w programie można zmieniać wartości zmiennych lecz nie wolno zmieniać treści programu.

CSAVE"nazwa"

Zapisuje program znajdujący się aktualnie w pamięci komputera na kasetę magnetofonową i nadaje mu "nazwę" mogącą zawierać dowolne znaki alfanumeryczne różne od cudzysłowu /"/. Ze względów praktycznych /patrz opis CLOAD/ wygodnie jest używać nazw będących pojedynczym znakiem alfanumerycznym. Magnetofon należy włączyć na zapis przed przyciśnięciem klawisza (ENTER)

DELETEnumer linii – numer linii

Usuwa z programu wszystkie linie od pierwszego podanego "numeru linii" do drugiego podanego "numeru linii" włącznie. Numer ostatniej usuwanej linii musi być użyty w programie. Pominięcie drugiego parametru powoduje usunięcie jednej określonej numerem linii /DELETEnumer linii/. Pominięcie pierwszego parametru powoduje usunięcie wszystkich linii programu od początku aż do linii o podanym numerze włącznie/DELETE – numer linii/. Użycie postaci DELETE . / z kropką zamiast numeru linii/ w trakcie wprowadzania lub edycji linii powoduje usunięcie tej linii.

Przykłady

DELETE 10 – 110	usunięcie linii od numeru 10 /włącznie/ do numeru 110 /włącznie/
DELETE 12	usunięcie linii o numerze 12
DELETE – 25	usunięcie wszystkich linii od początku programu do linii o numerze 25.

EDITnumer linii

Wprowadza komputer w tryb edycji /inaczej tryb poprawiania lub redagowania/ linii o podanym numerze. Po przejściu do trybu edycji można wykonywać całą grupę zleceń opisanych w rozdziale 6.

LISTnumer linii – numer linii

Wyświetla na ekranie wszystkie linie programu aktualnie przechowywanego w pamięci od pierwszego podanego "numeru linii" do drugiego podanego "numeru linii" włącznie. Przy pominięciu pierwszego parametru /LIST – numer linii/ wyświetlone są wszystkie linie aż do wymienionej włącznie. Przy pominięciu drugiego parametru i myślnika wyświetlona jest jedna żądana linia /LIST numer linii/, a przy pozostawieniu myślnika /LIST numer linii –/ również wszystkie następne do końca programu. Przy pominięciu obu parametrów /LIST/ linie programu wyświetlane są w sposób ciągły, od początku do końca programu zatrzymując wyświetlanie na ostatniej linii programu. Dla "zatrzymania" ekranu należy wówczas nacisnąć jednocześnie (SHIFT) i (@), a ponowne uruchomienie ciągłego wyświetlania wywołuje się przyciskając dowolny klawisz. Przy wykonywaniu LIST. /kropka zamiast parametru/ wyświetlona zostaje ostatnio wprowadzana lub poprawiana linia.

Przykłady:

LIST 20 – 70	wyświetlenie linii od 20 do 70 włącznie
LIST – 70	wyświetlenie wszystkich linii od początku programu do linii 70 włącznie
LIST 20	wyświetlenie linii 20
LIST 20 –	wyświetlenie wszystkich linii od linii 20 /włącznie/ do końca programu

LLISTnumer linii – numer linii

Działa jak instrukcja LIST, przy czym informacje wprowadzane są na drukarkę.

NEW

Usuwa wszystkie linie programu aktualnie przechowywanego w pamięci komputera oraz zeruje wszystkie zmienne /zmiennym łańcuchowym przypisane zostają łańcuchy puste/ przy niezmienionej rezerwacji obszaru przydzielonego na łańcuchy przez ostatnie zlecenie CLEAR.

RUNnumer linii

Rozpoczyna wykonywanie programu znajdującego się aktualnie w pamięci od linii o podanym numerze przy jednoczesnym wykonaniu akcji zlecenia CLEAR. Jeśli parametr zostanie pominięty /RUN/ wykonywanie programu rozpoczyna się od linii o najniższym numerze. Gdy zlecenie to użyte jest wewnątrz programu jako instrukcja to może być wykorzystane jako wygodny sposób kończenia programu z jednoczesnym zerowaniem zmiennych i natychmiastowym ponownym rozpoczęciem jego wykonywania. /przypadek programów o działaniu ciągłym/.
Rozpoczęcie wykonywania programu realizuje także zlecenie GOTO numer linii, przy czym nie są zerowane zmienne /nie jest realizowane automatycznie zlecenie CLEAR/.

SYSTEM

Wprowadza komputer w tryb systemowy. W tym trybie pracy możliwe jest ładowanie programów /lub zbiorów danych/ i wykonywanie programów napisanych w języku wewnętrznym /tzw. języku maszynowym/ rozpoznawanym bezpośrednio przez strukturę sprzętową systemu /inaczej na poziomie systemu/. Po naciśnięciu klawisza (ENTER) komputer wyświetla gwiazdkę i pytajnik /*?/. Następnie wprowadza się nazwę programu /zbioru/ bez cudzysłowu i komputer ładuje go z kasety po czym wyprowadza w nowej linii znaki *?. /Uwaga: rozpoznawana jest jedynie pierwsza litera nazwy/. Wprowadzenie w tej sytuacji znaku "/ " oraz adresu dziesiętnego spowoduje rozpoczęcie wykonywania programu /oczywiście po przyciśnięciu (ENTER)/. Jeśli pominie się adres to program wykonywany jest od adresu zawartego w części organizacyjnej zapisu programu na taśmie magnetofonowej. Po wywołaniu trybu systemowego można od razu wprowadzić znak "/" i adres dla uruchomienia jakiegoś programu.

TROFF

Wyłącza funkcję śledzenia /TRON/

TRON

Uaktywnia funkcję śledzenia przebiegu działania programu. Śledzenie TRACE programu polega na wyświetlaniu efektów akcji podjętej przez komputer w wyniku realizacji każdej linii programu. Numery wykonywanych kolejno linii umieszczone są po lewej stronie wewnątrz pary nawiasów, a na prawo od nich ewentualne wizualne efekty działania linii programu. Naciśnięcie (SHIFT) i (@) powoduje przerwanie wykonywania programu i "zatrzymanie" ekranu. Naciśnięcie dowolnego klawisza wznawia wykonywanie programu. Przy użyciu kombinacji zleceń TRON i TROFF wewnątrz programu w charakterze instrukcji śledzone są wyłącznie linie programu zawarte pomiędzy nimi.

Przykłady

Przykład 1

```
10 PRINT"START"
20 X = 2
30 X = X * 1.57
40 PRINT"PETLA"
50 STOP
60 GOTO 40
```

Po napisaniu:

```
> TRON (ENTER)
> RUN  (ENTER)
```

na ekranie pokażą się kolejno:

```
<10> START
<20> <30><40> PETLA
<50>
BREAK IN 50
READY
>
```

Kontynuujemy wykonanie zleceniem CONT

<60><40> PETLA
<50>
BREAK IN 50
READY
> ltd.

Przykład 2
100 TRON
110 X = X * 1.37
120 TROFF
Po napisaniu RUN (ENTER) na ekranie pojawi się
<110><120>

Przy takim użyciu kombinacji instrukcji TRON i TROFF należy pamiętać, aby w innym fragmencie programu nie nastąpił skok do linii 110, gdyż funkcja załączenia śladowania nie zostanie wykonana.

Uwaga: Jako zlecenia /w trybie bezpośrednim/ mogą też być użyte wszystkie instrukcje wersji języka BASIC dla systemu MERITUM za wyjątkiem instrukcji: INPUT i INPUT #.

OPIS WERSJI JĘZYKA BASIC DLA MODELU MERITUM-I

5.1 Informacje ogólne o składni

Można wyróżnić następujące elementy składni języka BASIC – MERITUM - I:

a) definicje typu zmiennych /prostych, tablicowych, łańcuchowych/ – opisane w pkt. 5.2

b) definicje operatorów /relacji, arytmetycznych, logicznych/ – opisane w pkt. 5.3

c) określenie kolejności wykonywania operacji – opisane w pkt. 5.4

d) definicje funkcji matematycznych – opisane w pkt. 5.5

e) sposób tworzenia funkcji pochodnych – opisane w pkt. 5.6

f) definicje skrótów – opisane w pkt. 5.7

g) instrukcje:

- wejścia-wyjścia opisane w pkt. 5.8.1
- organizacyjne i sterujące przebiegiem wykonywania programu – opisane w pkt. 5.8.2
- specjalne – opisane w pkt. 5.8.3
- operacji na łańcuchach – opisane w pkt. 5.8.4

Linie programu wprowadzane do pamięci komputera dla późniejszej realizacji programu /w trybie wykonywania programu/ opatrywane są numerami /liczby naturalne/. Kolejno wykonywane linie programu winny mieć numery narastające. Numeracja linii może być dowolna (gdy wprowadzane są bez użycia zlecenia AUTO), co 10 lub zgodnie z parametrem "krok" zlecenia AUTO od dowolnego numeru początkowego. Komputer zapewnia automatyczną numerację. Linie programu wykonywane w trybie bezpośrednim wprowadzane są bez numeru linii. W jednej linii programu umieścić można wiele instrukcji, przy czym muszą być one oddzielone znakiem dwukropka (:).

5.2 Definicje typów zmiennych

Zasady tworzenia nazw zmiennych są następujące:

- muszą zaczynać się od litery A ÷ Z,
- składają się z liter lub cyfr 0 ÷ 9,
- nie mogą zawierać w sobie ciągu znaków będących tzw. nazwą zarezerwowaną.
  Nazwami zarezerwowanymi są wszystkie nazwy zleceń, instrukcji, funkcji itd. używane w tej wersji języka.

Nazwy zmiennych mogą być dłuższe od dwóch znaków lecz tylko dwa pierwsze znaki będą używane przez komputer do rozróżniania nazw.

Przykłady

Przykład 1.
C C7 CB CC F6 D MU XZ X1 XPT
Są to poprawne nazwy oznaczające różne zmienne.

Przykład 2
ADD ADR AD
Są to poprawne nazwy oznaczające tą samą zmienną.

Przykład 3
ZIF
Jest to błędna nazwa, ponieważ zawiera słowo kluczowe IF.

Wyróżnia się następujące typy zmiennych:

1. Całkowite /ang. integer/

2. Pojedynczej precyzji /ang. single precision/

3. Podwójnej precyzji /ang. double precision/

4. Łańcuchowe /ang. string/

Pierwsze trzy typy używane są do przechowywania wartości numerycznych z różną dokładnością. Czwarty typ oznacza łańcuchy /sekwencje/ znaków w kodzie ASCII maksymalnie do 255 znaków.

Istnieją następujące znaki deklaracji typu zmiennych:

% – całkowita /wszystkie liczby większe od –32769 i mniejsze od +32768/ np.: M%, W5%

! – pojedynczej precyzji /6 cyfr znaczących/ np.: B!, KG!, R1!

– podwójnej precyzji /16 cyfr znaczących/ np.: B#, AA#, D1#

D – podwójnej precyzji w notacji naukowej /dla wprowadzania stałych lub podczas wyprowadzania dużych lub małych liczb/ np.: "X# = 3.1456D + 15" co daje wartość $3.1456 * 10^{15}$

$ – łańcuch /do 255 znaków/ np.: A5$, XZ$, TA$
Przykładowy łańcuch: "ITM MERITUM ZUK".

Zmienne bez znaków deklaracji traktowane są jako zmienne pojedynczej precyzji. Można to zmienić instrukcją DEF. Te same nazwy zmiennych mogą być użyte dla zmiennych różnych typów i komputer będzie nadal je rozróżniał według deklaracji zmiennych np.: X$, X%, X# są różnymi nazwami zmiennych. Instrukcja DIM służy do definiowania tablic na początku programu. W zależności od typu użytej zmiennej tablica może zawierać wartości całkowite, podwójnej precyzji, łańcuchy, etc. Dowolna poprawna nazwa zmiennej może być użyta do oznaczania tablicy.

Przykłady

G#(M) – jednowymiarowa tablica zawierająca liczby podwójnej precyzji,

T%(M,N) – dwuwymiarowa tablica zawierająca liczby całkowite,

TABS(I,J,K) – trójwymiarowa tablica przechowująca wartości będące łańcuchami.

5.3 Operatory

Operatory dzielimy na:

1. Operatory arytmetyczne:
   - \+ dodawanie
   - – odejmowanie
   - * mnożenie
   - / dzielenie
   - ↑ potęgowanie /może być też [/

2. Operatory logiczne:
   AND – iloczyn logiczny
   OR – suma logiczna
   NOT – negacja /inwersja/

Przykład

```
10 IF X = 10 OR Y = 100 THEN 100
.
.
.
100 PRINT "KONIEC"
```

Jest to przykład użycia operatorów logicznych w zdaniu logicznym. Operatory logiczne mogą również służyć do wykonywania operacji Boole'owskich, tzn. do operacji na poszczególnych bitach. Jest to szczególnie istotne przy współpracy programów napisanych w języku BASIC z programami napisanymi w języku wewnętrznym. Przykładowo po wczytaniu przy pomocy instrukcji INP /patrz. 5.8.3/ z zewnątrz do komputera zawartości rejestru określającego binarnie stany pewnych urządzeń czy zdarzeń /tzw. brama wejściowa ang. "port"/ możliwe jest badanie stanu poszczególnych bitów, porównywanie ich z wzorcami itd. Przy takim wykorzystaniu operatorów logicznych komputer dokonuje konwersji ich argumentów na szesnastobitowe liczby całkowite ze znakiem w zapisie dopełnieniowym do 2 /zakres od –32768 do +32767/. Po dokonaniu określonych operacji na tych liczbach wyniki przybierają taką samą formę /gdy przekroczony zostaje argument pojawia się błąd FC/. Operacje przeprowadzane są bit po bicie tzn. każdy bit wyniku tworzony jest na podstawie bitów na tej samej pozycji z obu argumentów w/g poniższych tablic wartości funkcji logicznych.

ARG 1 AND ARG 2		
ARG 1	ARG 2	WYNIK
0	0	0
0	1	0
1	0	0
1	1	1

NOT ARG	
ARG	WYNIK
0	1
1	0
–	–
–	–

ARG 1 OR ARG 2		
ARG 1	ARG 2	WYNIK
0	0	0
0	1	1
1	0	1
1	1	1

W poniższej tabelce podano odpowiednie przykłady. Pozycje w binarnym zapisie liczb liczone są od prawej /bit 0/ do lewej /bit 15/. Bit 15 jest najstarszy.

L/P	ARG 1	OPERATOR	ARG 2		WYNIK	ARG 1 binarnie ARG 2 binarnie WYNIK binarnie
1.	63	AND	1	=	1	....0111111 .....000001 000001
2.	4	AND	-1	=	4	.....0100 .....1111 0100
3.	8	OR	8	=	8	.....01000 .....01000 01000
4.	4	OR	2	=	6	.....0100 .....0010 0110
5.		NOT	0	=	-1	.....0000 1111
6.		NOT	1	=	-2	.....0001 1110

Uwaga: nieznaczące zera zostały pominięte, "...." oznacza "ta sama wartość aż do bitu 15-go".

3. Operatory relacji:

= równy

<> różny

< mniejszy

<= mniejszy lub równy

> większy

>= większy lub równy

Przykłady

Przykład 1
50 IF A <> 115 THEN PRINT "NIE TRAFIONO"

Przykład 2
200 Y = (A > 100) AND (B < = 130)

W wyniku obliczenia tej zależności komputer przypisuje zmiennej Y wartość – 1 jeśli prawa strona jest prawdziwa tzn. obydwa wyrażenia są prawdziwe. W przeciwnym razie Y = 0

4. Operatory łańcuchowe

= równość łańcuchów

<> nierówność łańcuchów

+ składanie dwóch łańcuchów

< poprzedza alfabetycznie

<= poprzedza alfabetycznie lub równy

> następuje alfabetycznie

>= następuje afabetycznie lub równy

Przykłady

"ZX"<"ZY" – łańcuch ZX poprzedza alfabetycznie łańcuch ZY, bo X poprzedza Y
"OLA">"ALA" – łańcuch OLA następuje alfabetycznie po łańcuchu ALA, bo O następuje po A.

Pozostałe relacje zostaną zobrazowane przykładami w punkcie 5.8.4.

5.4 Kolejność wykonywania operacji

Kolejność wykonywania działań w wyrażeniach /matematycznych, logicznych, relacjach lub będących ich kombinacjami/ jest następująca:

↑ potęgowanie np. X ↑ 2
– zmiana znaku np. – X
* /
+ –
<, >, =, <=, >=, <>
NOT
AND
OR

Jeżeli wyrażenie zawiera więcej symboli na tym samym poziomie to porządek wykonywania działań jest od lewej do prawej. Kolejność działań może być zmieniona przez użycie nawiasów. Jako pierwsze wykonywane są operacje wewnątrz najbardziej zagłębionej pary nawiasów.

5.5 Definicje funkcji matematycznych

Poniżej opisane są funkcje arytmetyczne, trygonometryczne i konwersji dostępne w tej wersji języka BASIC. Argument wszystkich funkcji musi być ujęty w nawias. Argument funkcji może być zmienną numeryczną /np. X3/, wyrażeniem /np. 4 *A/ lub stałą /np.78/. Funkcje matematyczne dają wynik w postaci wartości o pojedynczej precyzji z dokładnością do sześciu miejsc dziesiętnych. Funkcje konwersji /CINT, CDBL, itd/ dają wynik, którego dokładność zależy od określonej funkcji. Funkcje ABS, FIX i INT dają wartość o dokładności zależnej od dokładności argumentów.

ABS(x)

Oblicza wartość bezwzględną argumentu
ABS(x) = x dla x > = 0 i ABS(x) = –x dla x < 0

```
100 X=3 : Y = –8
110 IF ABS(X–Y) > 4 PRINT "WYGRALES"
```

INT(x)

Oblicza wartość całkowitą argumentu. Obliczona wartość całkowita musi być równa lub mniejsza od wartości argumentu. Zakres argumentu nie jest ograniczony do przedziału –32768 ÷ 32767
INT(3,8) = 3
INT(–3,8) = –4

CINT(x)

Oblicza największą liczbę całkowitą nie większą od wartości argumentu. Zakres argumentu jest ograniczony do zakresu –32768 ÷ 32767. Funkcją tą można posłużyć się do przyspieszenia wykonania operacji na argumentach podwójnej precyzji lub całkowitych, bez zmiany dokładności argumentu, jeżeli użytkownika interesuje tylko wynik całkowity.

LOG(x)

Oblicza logarytm naturalny argumentu ($\log_e(x)$). W celu wyznaczenia logarytmu przy innej podstawie stosuje się wzór:

$$\log_b(x) = \log(x) / \log(b)$$

Przykład

```
10 FOR X = 1TO 5
20 Z = LOG(X) / LOG(10)
30 PRINT X,Z
40 NEXT X
```

RANDOM

Służy do uruchomienia generatora liczb pseudolosowych i ma charakter instrukcji. Jeżeli w programie użyta jest funkcja RND, umieszczenie RANDOM na początku programu zapewni generowanie sekwencji liczb pseudolosowych każdorazowo po uruchomieniu programu.

SQR(x)

Oblicza pierwiastek kwadratowy z argumentu. Działa podobnie jak X↑(1/2) lecz szybciej.

SGN(x)

Przybiera wartość 1 dla x dodatniego; −1 dla x ujemnego oraz wartość 0 dla x = 0

X = SGN(+3.8) w wyniku otrzymujemy X = 1
Z = SGN(−2.5) w wyniku otrzymujemy Z = −1

FIX(x)

Wyznacza część całkowitą argumentu. Wynik jest wartością całkowitą.

Dla x> = 0 FIX(x) = INT(x)
Dla x < 0 FIX(x) = INT(x) +1
FIX(3.8) daje w wyniku 3, a FIX(−3.8) daje −3.

EXP(x)

Podnosi stałą "e" do potęgi argumentu.
Jest to funkcja odwrotna do funkcji LOG(x).

x = EXP(LOG(x))

RND(x)

Podaje liczbę pseudolosową z zakresu [0 ÷ 1] lub z zakresu liczb całkowitych większych od zera w zależności od argumentu przy pomocy bieżącej pseudolosowej liczby bazowej /generowanej wewnętrznie i niedostępnej dla użytkownika/.
RND(0) oblicza wartość o pojedynczej precyzji z przedziału [0 ÷ 1]. RND (argument) oblicza wartość całkowitą z przedziału [1 ÷ argument], przy czym argument musi być mniejszy od 32768.

Przykład
100 Z = RND(23)
Instrukcja ta oblicza liczbę całkowitą pseudolosową dodatnią i mniejszą od 24.
110 X = RND(23.5)
Instrukcja ta oblicza liczbę pseudolosową z tego samego przedziału, ponieważ RND uwzględnia część całkowitą argumentu.

CDBL(x)

Wyznacza odpowiednik podwójnej precyzji argumentu x. Wyznaczona wartość zawiera 17 cyfr, lecz tylko cyfry zawarte w argumencie będą znaczące. CDBL jest stosowana w celu forsowania operacji podwójnej precyzji na argumentach całkowitych lub pojedynczej precyzji.
100 FOR X% = 1 TO 5 : PRINT 1/CDBL(X%), : NEXT

CSNG(x)

Wyznacza odpowiednik pojedynczej precyzji argumentu x. Z argumentu o podwójnej precyzji pozostawia 6 cyfr znaczących z zaokrągleniem typu 4/5 cyfry najmniej znaczącej.
CSNG(.55555555555555556) = 0.555556
CSNG(0.66666666666666666) = 0.666666

Funkcje trygonometryczne

SIN(x)

Oblicza sinus argumentu /argument musi być w radianach/. Jeżeli x jest wyrażone w stopniach, to wtedy należy użyć wyrażenia:
SIN(X*.0174533)

COS(x)

Oblicza cosinus argumentu /argument musi być w radianach/. Jeżeli x jest wyrażone w stopniach, to wtedy należy użyć wyrażenia:
COS(X*.0174533)

TAN(x)

Oblicza tangens argumentu /argument musi być w radianach/. Jeżeli x jest wyrażone w stopniach, to wtedy należy użyć wyrażenia:
TAN(X*.0174533)

ATN(x)

Oblicza arctangens /w radianach/ argumentu x. Aby otrzymać wynik w stopniach należy pomnożyć ATN(x) przez 57.29578

5.6 Funkcje pochodne

Możliwe jest utworzenie wielu innych funkcji przy użyciu funkcji opisanych w rozdziale 5.5. Zestawiono je poniżej:

```
SEC(X) = 1/COS(X)
CSC(X) = 1/SIN(X)
COT(X) = 1/TAN(X)
ARCSIN(X) = ATN(X/SQR(-X*X+1))
ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708
ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
ARCCOT(X) = -ATN(X)+1.5708
SINH(X) = (EXP(X)-EXP(-X))/2
COSH(X) = (EXP(X)+EXP(-X))/2
TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
SECH(X) = 2/(EXP(X)+EXP(-X))
CSCH(X) = 2/(EXP(X)-EXP(-X))
COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
ARCSINH(X) = LOG(X+SQR(X*X+1))
ARCCOSH(X) = LOG(X+SQR(X*X-1))
ARCTANH(X) = LOG((1+X)/(1-X))/2
ARCSECH(X) = LOG((SQR(-X*X+1)+1)/X)
ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
ARCCOTH(X) = LOG((X+1)/(X-1))/2
```

5.7 Skróty

Stosuje się dwa typy skrótów:
? dla PRINT
' dla : REM

5.8 Instrukcje

Uwagi wstępne:

- w prezentowanej wersji BASIC'a parametry instrukcji występować mogą bezpośrednio (bez spacji) za treścią słowa kluczowego lub mogą być od niego oddzielone 1 lub większą liczbą spacji,
- początek pierwszej instrukcji w linii może być oddzielony 1 lub większą liczbą spacji od numeru linii,
- maksymalna długość linii programu wynosi 255 znaków,

– wszystkie zmienne przybierają początkowo typ pojedynczej precyzji,
– wymagany zadeklarowany obszar pamięci przyporządkowany jest automatycznie dla łańcuchów i tablic niezależnie od tego czy są one używane czy nie.

5.8.1 Instrukcje wejścia-wyjścia.

Grupa instrukcji wejścia-wyjścia umożliwia wymianę informacji pomiędzy wszystkimi podzespołami systemu /komputer, klawiatura, monitor, magnetofon, drukarka/. Instrukcje dotyczą ładowania programów z kasety do komputera i odwrotnie, listowania programów na monitorze i drukarce, wyprowadzania wyników i wprowadzania danych /np. z klawiatury/ itd.

PRINTlista elementów

Wyprowadza elementy wymienione w "liście elementów" na ekran. Lista elementów tworzona jest poprzez oddzielenie elementów przecinkami lub średnikami. Użycie średnika powoduje, że elementy wyprowadzane są bez odstępu. Użycie przecinka powoduje wydruk kolejnego elementu w następnym polu linii /patrz rozdział 3.1/. Elementami mogą być: teksty ujęte w cudzysłowy /stałe łańcuchowe/, zmienne łańcuchowe, stałe numeryczne /liczby/, zmienne lub wyrażenia zawierające kombinacje wszystkich powyższych elementów. Wszystkie liczby wyprowadzane są ze spacją na końcu, a liczby dodatnie dodatkowo ze spacją na początku /bez znaku+/. Jeśli po ostatnim elemencie z listy występuje średnik, kursor zatrzymuje się w aktualnej pozycji /i następne wyprowadzanie rozpoczyna się od tego miejsca/, w przeciwnym razie kursor przechodzi w dół do początku następnej linii ekranu. W liście elementów wystąpić też mogą tzw. modyfikatory /np. TAB/. Instrukcje tego typu wyróżniono oddzielnie /np. PRINT TAB/.

Przykłady

Przykład 1
```
10 X = 3
20 PRINT 2 "X;","; 30 ;"–PUNKT PROSTEJ"
>RUN
   6 , 30–PUNKT PROSTEJ
```

Przykład 2
```
30 PRINT "POLE 1","POLE 2","POLE 3","POLE 4","POLE 1 ETC"
>RUN
POLE 1 ·          POLE 2          POLE 3          POLE 4
POLE 1 ETC
```
Istnieją cztery 16- znakowe pola w linii /patrz rodział 3.1/.

Przykład 3
```
50 PRINT,,,"POLE 4"
>RUN
                                                  POLE 4
```

Przykład 4
```
100 PRINT "KURSOR POZOSTAJE";
200 PRINT " W TEJ SAMEJ LINII"
300 PRINT "TERAZ W NOWEJ LINII"
400 ?"SKROT DLA INSTRUKCJI PRINT"
>RUN
KURSOR POZOSTAJE W TEJ SAMEJ LINII
TERAZ W NOWEJ LINII
SKROT DLA INSTRUKCJI PRINT
```

Uwaga: przed i za łańcuchami nie są automatycznie wstawiane żadne spacje.

PRINT@pozycja, lista elementów

Wyprowadza na ekran elementy wymienione w "liście elementów" od określonej "pozycji" komórki znakowej ekranu. Pozycja musi być liczbą z przedziału 0 ÷ 1023 /dokładniejsze dane o organizacji obszaru roboczego monitora przedstawiono w rozdziale 3.1/. Znak @ musi występować bezpośrednio po PRINT.

PRINT TAB(wyrażenie)

Przesuwa kursor w bieżącej linii do pozycji określonej "wyrażeniem". Jeśli jego wartość jest większa od 63 to przesunięcie nastąpi do linii następnych /maksymalna wartość wyrażenia wynosi 255/. Parametrem zlecenia mogą być wyrażenia numeryczne co ułatwia drukowanie wykresów funkcji matematycznych, tabel itp. Modyfikator TAB może być wielokrotnie użyty w liście instrukcji PRINT. Za modyfikatorem TAB nie trzeba umieszczać znaków rozdzielających. Jeśli kursor znajduje się poza określoną pozycją to modyfikator TAB jest ignorowany.

Przykłady

Przykład 1
```
10 PRINT "ZNAK1"; TAB(9) "ZNAK10"; TAB(19) "ZNAK20"
>RUN
ZNAK1     ZNAK10     ZNAK20.
```

Przykład 2
```
10 PRINT TAB(X↑2)X↑2
```

Uwaga: Jeśli w liście elementów instrukcji PRINT występuje element TAB (wyrażenie) to nie trzeba rozdzielać go średnikiem od następnego elementu.

PRINT USING łańcuch; lista elementów

Wyprowadza elementy wymienione w "liście elementów" z określeniem formatu wyprowadzenia. Najczęstsza postać instrukcji to:

PRINT USING łańcuch; wartość

powodująca wydrukowanie "wartości" w formacie określonym poprzez "łańcuch". W łańcuchu mogą być użyte następujące znaki tworzące tzw. specyfikację formatu:

#	Określa pozycję każdej cyfry danej wartości numerycznej. Ilość Użytych zanków = określa liczbę pozycji pola numerycznego. Jeżeli liczba pozycji pola numerycznego jest większa niż liczba cyfr w wartości numerycznej wówczas nieużywane pozycje pola na lewo od liczby będą wyświetlane jako spacje, a na prawo od kropki dziesiętnej jako zera.
+	Jeżeli znak "+" umieszczony jest na początku lub na końcu pola, będzie on wyprowadzany zgodnie ze specyfikacją jako + dla liczb dodatnich lub jako – dla ujemnych.
–	Jeżeli znak "–" jest umieszczony na końcu pola spowoduje on dopisywanie znaku ujemnego za wszystkimi liczbami ujemnymi oraz spacji za dodatnimi.
.	Punkt dziesiętny może być umieszczony w dowolnym miejscu pola numerycznego określonego znakami # . W przypadku, gdy nie wszystkie cyfry na prawo od punktu dziesiętnego mieszczą się w polu ma miejsce zaokrąglanie liczby.
,	Jeżeli przecinek jest umieszczony w dowolnym miejscu pomiędzy cyfrą i punktem dziesiętnym, powoduje on oddzielenie każdej trójki cyfr na lewo od przecinka /począwszy od przecinka/. Przecinek stanowi dodatkową pozycję w polu numerycznym.
$$	Dwa znaki dolara umieszczone na początku pola będą działać jako ruchomy znak dolara tzn. znak ten będzie zajmował pierwszą pozycję poprzedzającą liczbę.
**	Dwie gwiazdki umieszczone na początku pola powodują, że wszystkie niewykorzystane pozycje na lewo od przecinka dziesiętnego będą zaznaczone gwiazdkami. Dwie gwiazdki stanowią dwie więcej pozycje w polu.
**$	Jeżeli te trzy znaki zostaną użyte na początku pola, wówczas nie zajęte pozycje z lewej strony liczby będą zaznaczone znakiem *, a znak $ zajmie pierwszą pozycję poprzedzającą liczbę.
% spacje %	Specyfikacja pola znakowego o długości większej niż 1 znak. Długość pola jest równa liczbie spacji +2.
!	Powoduje użycie pierwszego znaku łańcucha o bieżącej wartości.

W poniższej tabeli przedstawiono przykłady specyfikacji formatu PRINT USING AS; A. Do AS wprowadzono uprzednio odpowiedni łańcuch specyfikacji formatu.

Lp.	Łańcuch specyfikacji formatu AS	Wartość zmiennej do wydruku A	Wydruk instr. PRINT USING AS;A	Uwagi
1	##.##	424.24	%424.24	% drukowany jest zawsze gdy pole jest za małe.
2	###.##	424.24	424.24	
3	####.##	424.24	424.24	
4	##.##	42.279	42.28	Liczba została zaokrąglona.
5	+##.##	42.3	+42.3	
6	+##.##	–42.3	–42.3	
7	##.#+	42.3	42.3+	
8	##.#+	–42.3	42.3–	
9	##.#–	42.3	42.3	
10	##.#–	–42.3	42.3–	
11	$$##.#	42.3	$42.3	
12	**##.#	42.3	**42.3	
13	**##.#	4235.1	4235.1	
14	##,###.#	42351.3	42,351.3	
15	####.#	42351.3	42,351	
16	!	MERIT	M	
17	% %	MERIT	MERI	Liczba drukowanych znaków równa jest 2+2 /liczba spacji pomiędzy znakami %/ =4
18	%%	MERIT	ME	Drukowane są 2 znaki gdyż brak spacji pomiędzy znakami %.

Przykłady:

Przykład 1

```
10 INPUT XS, YS, ZS, WS
20 PRINT USING "!"; XS; YS; ZS; WS
>RUN
? MERITUM, EWA, RI, ALA (ENTER)
MERA
```

Deklaracja "!" odnosi się tu do wszystkich łańcuchów w "liście elementów". Jeśli w deklaracji łańcucha specyfikacyjnego zamiast "!" umieścimy "! ! ! !" to wydruk zostanie wykonany z 2 spacjami pomiędzy pierwszymi literami /ogólnie z tyloma spacjami ile wstawiono ich w specyfikacji pomiędzy znakami !/ tzn:

```
M  E  R  A
```

Przykład 2 /fragment programu obsługi wpłat pieniężnych/

```
10 CLS
20 XS = "**#####,###.## ZLOTYCH"
30 INPUT "IMIE" ; IS
40 INPUT "NAZWISKO" ; NS
50 INPUT "SUMA WPLATY" ; S
60 CLS: PRINT "WPLACONO NA RACHUNEK ";
70 PRINT USING "!"; IS; ".";
80 PRINT NS
90 PRINT: PRINT USING XS; S
100 GOTO 100
```

Uwaga: przy wprowadzaniu programu można używać znaku ? zamiast słowa PRINT. W programie CLS oznacza zerowanie ekranu. Samo PRINT oznacza generację nowej linii. W linii 20 pokazano możliwość łączenia łańcucha specyfikacji pola oraz łańcucha tekstowego.

```
>RUN
IMIE? MALGORZATA (ENTER)
NAZWISKO? PODLIPIEWSKA (ENTER)
SUMA WPLATY? 3510.5 (ENTER)
WPLACONO NA RACHUNEK M.PODLIPIEWSKA
******3,510.50 ZLOTYCH
```

PRINT# – 1, lista elementów

Zapisuje wartości zmiennych określonych "listą elementów" na kasetę magnetofonową. Magnetofon musi być prawidłowo podłączony i ustawiony na zapis. Informacje, które mogą być przesłane na kasetę magnetofonową pojedynczą instrukcją PRINT#–1 nie mogą w sumie przekraczać 255 znaków, gdyż wszystkie następne znaki zostaną obcięte /255 znaków tworzy tzw. rekord informacji na kasecie magnetofonowej/. Przykładowo, jeśli łańcuch XS, jest dłuższy od 250 znaków to instrukcja PRINT#–1, XS, A # jest już nieprawidłowa, gdyż zmienna podwójnej precyzji A zajmuje 11 bajtów pamięci /patrz rozdz. 7/ i w sumie informacja wyspecyfikowana w liście elementów tej instrukcji przekroczy 255. Wystąpi wówczas błąd OD /out of data/ przy próbie zapisu danych instrukcją PRINT#–1.
#–1 określa numer urządzenia /w tym wypadku magnetofon nr 1/, gdyż w rozszerzonej wersji mogą być podłączone 2 magnetofony.

Przykład

```
10 X = 15350
20 BS = SUMA
30 PRINT # –1, X, BS, "KONCOWA"
```

Na taśmie magnetofonowej kolejno znajdą się bieżące wartości zmiennych X, BS /15350 SUMA/ oraz łańcuch KONCOWA. Przy odczycie tak zapisanych informacji z powrotem do komputera instrukcją INPUT#–1 jej lista elementów musi być zgodna co do typów, ilości i kolejności wystąpienia elementów z listą instrukcji PRINT#–1.

LPRINT

Jest to grupa instrukcji /LPRINT lista elementów, LPRINT TAB, LPRINT USING/ wyprowadzających informacje na drukarkę. Instrukcje są specyfikowane i zorganizowane w sposób analogiczny jak opisano powyżej dla grupy instrukcji PRINT. Istnieje szereg kodów kontrolnych sterujących pracą różnych drukarek np. 13 – powrót karetki, 10 – nowa linia i powrót karetki itd. Kody te różnią się dla różnych drukarek. Można je generować przy użyciu instrukcji PRINT CHR$(n), gdzie n jest odpowiednim kodem.

INPUT "komunikat";lista elementów

Powoduje wyprowadzenie zakończonego pytajnikiem "komunikatu" (jeśli jest) i zatrzymanie pracy komputera do momentu wprowadzenia określonej liczby wartości z klawiatury zgodnie z listą elementów. Wprowadzone wartości przypisane zostają poszczególnym nazwom zmiennych w/g listy. Elementy listy muszą być oddzielone przecinkami. W liście elementów mogą znaleźć się zmienne łańcuchowe lub numeryczne. Po napotkaniu tej instrukcji komputer wyświetla w nowej linii:

?_

Należy wówczas wprowadzić wszystkie wartości oddzielając je przecinkami /kończąc przyciśnięciem klawisza (ENTER) lub kolejno naciskając po każdej wartości klawisz (ENTER) Jeżeli łańcuch zawiera dwukropki, przecinki, lub początkowe blanki należy umieścić go w cudzysłowie. Przy wprowadzaniu wartości pojedynczo komputer wyświetla:

??_

co oznacza, że potrzebne są dalsze dane. Należy je wprowadzać tak długo, aż zostaną ustalone wszystkie zmienne i wtedy komputer przejdzie automatycznie do następnej linii w programie. W przypadku wprowadzenia większej liczby elementów danych niż jest to określone w liście komputer wyświetli komunikat:

? EXTRA IGNORED

i będzie kontynuował normalne wykonywanie programu. Przy próbie wprowadzenia wartości zmiennej niezgodnej z typem wyszczególnionym w liście /np. wprowadzenie zmiennej łańcuchowej dla zmiennej numerycznej/ komputer wyświetli:

? REDO
?_

i umożliwi wykonanie następnej próby wprowadzenia wartości danych odpowiedniego typu ale od pierwszej wartości wymienionej w liście. Zabronione jest wprowadzanie wyrażeń dla zmiennych numerycznych.

Przykłady

Przykład 1

```
10 CLS
20 INPUT A, AS, BS
30 PRINT "POPRAWNIE"
40 STOP
>RUN
? 25 (ENTER)
?? EWA, ADAM (ENTER)
POPRAWNIE
```

Komputer przyporządkował A= 25, AS= EWA, BS = ADAM

Przykład 2

```
100 INPUT "NAZWISKO"; NS
>RUN
NAZWISKO? NOWAK (ENTER)
```

Przykład 3

```
10 INPUT A, BS
>RUN
? 5 + 2 (ENTER)
? REDO
? 7 (ENTER)
?? "CUDZYSLOW, BO PRZECINEK" (ENTER)
```

We wprowadzonym łańcuchu BS był przecinek, a więc należało go ująć w cudzysłów. Wyrażenie 5 + 2 było niedopuszczalne.

INPUT # –1, lista elementów

Wczytuje określoną liczbę wartości przechowywanych na kasecie magnetofonowej i przyporządkowuje je do określonych nazw zmiennych wymienionych w "liście elementów". Magnetofon musi być właściwie podłączony i ustawiony na odczyt, taśma właściwie przewinięta. Struktura danych wprowadzonych musi być identyczna ze strukturą wyprowadzoną, która utworzyła rekord danych na kasecie tak pod względem liczby i typu jak również kolejności wystąpienia zmiennych.
Sygnalizowane są następujące błędy:

– jeśli wystąpi zły typ zmiennych (np. w/g listy elementów ma być wczytywany łańcuch, a na taśmie znajduje się aktualnie wartość numeryczna) – błąd złego pliku danych ('Bad File Data');

– jeśli na taśmie nie ma dostatecznej liczby wartości do "zapełnienia" zmiennych wymienionych w liście elementów – błąd braku danych ('Out of Data').

Znaczenie numeru urządzenia omówiono już przy opisie instrukcji PRINT # –1.

Przykład

Odczyt plików zapisanych na taśmę magnetofonową programem z przykładu zamieszczonego w opisie instrukcji PRINT # −1.

```
10 INPUT#−1, X, BS, SS
20 PRINT X, BS, SS
30 IF SS = "KONCOWA" END
40 GOTO 10
```

Program ten wczytywał będzie z magnetofonu i następnie drukował kolejne trójki wartości zmiennych X, BS, S$ aż do wystąpienia wartości S$ = KONCOWA

DATAlista elementów

Przechowuje wewnątrz programu wyszczególnione w "liście elementów" dane, które będą dostępne przy użyciu instrukcji READ. Elementy listy mogą być stałymi numerycznymi /wyrażenia są niedozwolone/ lub łańcuchowymi /łańcuchy zawierające dwukropki, przecinki lub początkowe blanki muszą być ujęte w cudzysłów/. Elementy listy oddzielone są przecinkami. Instrukcja READ czyta listę danych po kolei począwszy od pierwszego elementu w pierwszej instrukcji DATA kończąc na ostatnim elemencie w ostatniej instrukcji DATA. Instrukcje DATA mogą być umieszczone w dowolnym miejscu programu, w szczególności instrukcje READ czytające dane mogą występować przed instrukcjami DATA, z których danych korzystają. Wygodnie jest umieszczać instrukcje DATA w zwartych grupach jedna pod drugą.

READlista elementów

Odczytuje kolejne wartości danych z instrukcji DATA i przyporządkowuje je do określonych zmiennych wyszczególnionych w "liście elementów". Elementy listy oddzielone są przecinkami. Dane czytane są po kolei zaczynając od pierwszej wartości wyszczególnionej w pierwszej instrukcji DATA. Jeśli liczba elementów wszystkich instrukcji READ jest większa od liczby elementów wszystkich instrukcji DATA komputer wyświetli błąd Out of Data.

Przykłady

Przykład 1

```
10   READ A, B, C
70   READ KS, LS
```

```
100  DATA 2.132, 1800, 3
110  DATA ITM, ZUK
```

Przykład 2

```
50  DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
60  FOR J = 1 TO 4
70  READ X, Y
80  PRINT X, Y, X+Y
90  NEXT J
100 END
>RUN
1            2            3
3            4            7
5            6            11
7            8            15
```

Przykład 3

```
10 READ X$
20 IF X$ = "EWA" PRINT "JEST IMIE EWA" : END
30 GOTO 10
40 DATA "ADAM", "KLARA", "ZOFIA"
50 DATA "OLGA", "ALA", "EWA", "LOLA"
>RUN
JEST IMIE EWA
```

Program sprawdza czy na liście jest choć jedno imię EWA.

RESTORE

Powoduje, że następne wykonanie instrukcji READ rozpoczyna się od czytania pierwszego elementu w pierwszej instrukcji DATA. Umożliwia to ponowne używanie wartości z tych samych instrukcji DATA.

Przykład

```
10 READ A
20 RESTORE
30 READ B
40 PRINT A, B
50 DATA 20, 25
>RUN
20 20
```

Ponieważ wystąpiła instrukcja RESTORE, druga instrukcja READ pobrała pierwszy element z instrukcji DATA.

5.8.2 Instrukcje organizacyjne i sterujące przebiegiem wykonywania programu.

Kolejno przedstawione zostaną następujące grupy instrukcji:

a/ organizacyjne	REM, STOP, END, ON ERROR GOTO, ERROR, RESUME, CLEARn, LET
b/ definiujące:	DEFINT, DEFSNG, DEFDBL, DEFSTR, DIM
c/ sekwencji wykonania programu:	GOTO, ON . . . GOTO, GOSUB, ON . . . . GOSUB, RETURN, FOR – TO – STEP, NEXT
d/ warunkowe:	IF, THEN, ELSE

REM

Powoduje zignorowanie przez komputer pozostałej części linii programu /nie jest ona analizowana/. Dopuszcza się użycie zamiast słowa REM skrótu którym jest znak apostrofu / ' /. Instrukcja umożliwia pisanie komentarzy w programie. Za słowem kluczowym REM wystąpić może 252 dowolnych znaków alfanumerycznych. Jeśli REM wystąpi w linii programu zawierającej więcej instrukcji musi być instrukcją ostatnią.

Przykłady

Przykład 1

```
 10 REM        **********************
 20 REM        *                    *
 30 REM        *   PROGRAM – SZACHY *
 40 REM        *                    *
 50 REM        **********************
 60 REM
 70 REM        PROGRAMISTA: M. PODLIPIEWSKI
 80 REM
 90 REM        TEN PROGRAM BEDZIE DOBRZE KOMENTOWANY
100 REM                                                    /
110 REM        BOGATY, DOBRZE PRZYGOTOWANY KOMENTARZ /ISTOTNE INFORMACJE
120 REM        ROZPLANOWANE GRAFICZNIE/ ZNACZNIE ULATWIA PISANIE, URUCHAMIA-
130 REM        NIE I POZNIEJSZA ANALIZE /CZYTELNOSC/ DOKUMENTACJI PROGRAMU.
140 REM        JEST ON BARDZO ISTOTNYM ELEMENTEM NOWOCZESNEJ METODOLOGII
150 REM        PROJEKTOWANIA OPROGRAMOWANIA.
160 REM
170 REM
180 CLS        : REM TEN KOMENTARZ JEST ZBEDNY GDYZ CLS  JEST
190              REM "SAMOKOMENTUJACE SIE"
200 GOSUB 1100 : REM  WYWOLANIE PODPROGRAMU WYSWIETLENIA
210              REM SZACHOWNICY
220
```

Przykład 2

```
10 'TO TEZ JEST KOMENTARZ
20 'GDYZ APOSTROF JEST SKROTEM INSTRUKCJI REM
```

STOP

Przerywa wykonanie programu oraz drukuje komunikat BREAK IN numer linii. Wznowienie wykonania programu wykonuje się przy pomocy zlecenia CONT /patrz rozdział 4/ o ile program w czasie zatrzymania nie został zmieniony. Instrukcja STOP jest najczęściej wykorzystywana jako narzędzie do uruchamiania programu /po uzyskaniu poprawnej pracy programu jest ona usuwana/. Po zatrzymaniu programu można np. testować wartości zmiennych przy pomocy instrukcji PRINT.

Przykład

```
10 A = A+1
20 GOSUB 120
30 STOP
.
.
.
>RUN
BREAK IN 30
READY
```

Można obecnie przetestować czy procedura zmieniła wartość zmiennej A poprzez wykonanie:

```
>PRINT A
```

END

Kończy normalne wykonywanie programu.
Umieszczenie tej instrukcji w miejscu logicznego końca programu nie jest konieczne. Może ona w ogóle nie wystąpić w programie. Przy pomocy instrukcji END można natomiast powodować zakończenie obliczeń w miejscach różnych od logicznego końca programu / rozgałęzienia w programie, zabezpieczenia przed niepożądanym wejściem do danego fragmentu programu/.

Przykład

```
.
10 GOSUB 80
.
.
70 END
80 'TU ROZPOCZYNA SIE PODPROGRAM
.
.
150 RETURN
```

Takie użycie instrukcji END powoduje, że linia 80 /początek podprogramu/ dostępna jest jedynie poprzez skok GOSUB 80 co zabezpiecza przed przypadkowym wejściem do programu.

ON ERROR GOTOnumer linii

Powoduje skok do określonego poprzez "numer linii" fragmentu programu /rodzaj podprogramu/ po wystąpieniu błędu w programie zasadniczym o ile instrukcja użyta została przed miejscem wystąpienia błędu. Normalny komunikat o błędzie nie wystąpi a wykonanie programu nie zostanie przerwane. Działanie tej instrukcji kasowane jest poprzez użycie jej powtórnie z numerem linii = 0 /ON ERROR GOTO 0/, co przywraca normalną reakcję komputera na napotykane błędy. Podprogram "obsługi błędu" winien być zakończony instrukcją RESUME /poprzedzoną najczęściej instrukcją ON ERROR GOTO 0/. Wykaz sytuacji w których komputer reaguje na błędy w sposób normalny /przerywa wykonywanie programu i wyprowadza na ekran komunikat o błędzie/ znajduje się w Dodatku 1.

RESUMEnumer linii

Kończy program obsługi błędu poprzez określenie miejsca z którego ma być wznowione normalne wykonywanie programu w zależności od wartości parametru i tak program kontynuowany jest od:

– instrukcji w której wystąpił błąd – przy pominięciu parametru /RESUME/ lub gdy jest on równy 0 /RESUME0/;

– instrukcji następnej po tej w której wystąpił błąd – gdy jako parametr użyte jest słowo NEXT /RESUME NEXT/;

– określonej linii programu gdy parametrem jest numer linii istniejącej w programie /np. RESUME110/.

Przykłady

Przykład 1

```
 10 ' PROGRAM OBLICZA ODWROTNOSC R WPROWADZANEJ
 20 ' DANEJ D. DLA D=0 NALEZY PRZYJAC R=1
 30
 40 ON ERROR GOTO 110
 50 INPUT "WPROWADZ DANA";D
 60 R=1/D
 70 PRINT R
 80 GOTO50
 90 ' PODPROGRAM REAKCJI NA BLAD DZIELENIA
100 ' PRZEZ ZERO
110 R=1
120 PRINT "PRZYJETO ";
130 RESUME NEXT
```

Przykład 2 /inne rozwiązanie podprogramu obsługi błędu z Przykładu 1/

```
 10 ' PROGRAM OBLICZA ODWROTNOSC R WPROWADZANEJ
 20 ' DANEJ D. DLA D=0 NALEZY PRZYJAC R = 1
 30
 40 ON ERROR GOTO 110
 50 INPUT "WPROWADZ DANA";D
 60 R=1/D
 70 PRINT R
 80 GOTO50
 90 ' PODPROGRAM REAKCJI NA BLAD DZIELENIA
100 ' PRZEZ ZERO
110 R=1
120 PRINT "PRZYJETO " ; R
130 RESUME50
```

ERROR kod

Powoduje identyczną reakcję komputera jak na rzeczywisty błąd o określonym przez parametr "kodzie". /Patrz Dodatek 1/. Reakcja ta – to przerwanie wykonywania programu i wyprowadzenie na ekran komunikatu o błędzie np.:

```
? TM ERROR
READY
```

>– dla błędu "niewłaściwego typu danych"

CLEAR n

Zeruje wszystkie zmienne oraz rezerwuje n /n – liczba nieujemna/ bajtów dla przechowywania łańcuchów. Komputer po załączeniu automatycznie rezerwuje 50 bajtów. Właściwe operowanie instrukcją CLEAR jest elementem optymalizacji wykorzystania pamięci. Przykładowo jeśli w programie używa się łańcucha o długości 10 to należy wykonać CLEAR 10 celem zwolnienia 40 z 50 bajtów zarezerwowanych automatycznie. Jeśli długość użytego w programie łańcucha przekroczy zarezerwowany obszar – wystąpi błąd Out of String Space. /"poza zakresem zarezerwowanym dla łańcucha"/.

LET zmienna = wyrażenie

Przypisuje wartość umieszczoną po prawej stronie równości do zmiennej umieszczonej po lewej stronie równości /tzw. instrukcja przypisania/. W opisywanej wersji BASIC'a słowo kluczowe LET jest opcjonalne i może być pominięte.

Przykład :

```
10 LET XS = "PRZYKŁADY UZYCIA LET"
20 LET A = 2.56
30 LET A = A+Y1
40 A = A–1.3 : REM TO TEZ JEST INSTRUKCJA PRZYPISANIA
```

Grupa instrukcji definicji typów zmiennych

DEFINT
DEFSNG
DEFDBL
DEFSTR

lista zakresów literowych

Powoduje, że zmienna lub stała, której nazwa rozpoczyna się od dowolnej litery z "zakresów literowych" podanych w liście /elementy listy to pojedyncze litery lub zakresy literowe np. A – N; oddzielane są one przecinkami/ traktowana jest jako:

– **całkowita dla DEFINT**	/zakres –32768 do 32767/
– **pojedynczej precyzji dla DEFSNG**	/dokładność do 7 cyfr a wprowadzane z dokładnością do 6 cyfr/
– **podwójnej precyzji dla DEFDBL**	/dokładność do 17 cyfr, wprowadzone z dokładnością do 16 cyfr/
– **łańcuchowa dla DEFSTR**	/do 255 znaków o ile dostatecznie dużo bajtów zarezerwowano instrukcją CLEAR/

za wyjątkiem przypadku dodania do nazwy zmiennej znaku deklaracji innego typu /#, %/. Tak więc znaki deklaracji mają wyższy priorytet od w/w instrukcji. Grupa tych instrukcji umożliwia optymalizację programów tak pod względem zajętości pamięci jak też czasu wykonywania programu. Potrzeba ich stosowania wynika m.in. z następujących przesłanek:

a/ wszystkie niezadeklarowane zmienne /np. na początku programu/ przyjmują w tej wersji języka typ pojedynczej precyzji,

b/ wartości całkowite zajmują np. najmniej pamięci, a "arytmetyka" liczb całkowitych jest znacznie szybsza od arytmetyki liczb pojedynczej lub podwójnej precyzji – jednak typ całkowity obejmuje jedynie liczby z zakresu –32768 do 32767.

Instrukcje DEFINT oraz DEFDBL mogą być umieszczone w dowolnym miejscu programu lecz mogą zmieniać znaczenie jedynie zmiennych bez znaku deklaracji typu stąd umieszczane są z reguły na początku programu. Instrukcja DEFSNG jest używana przede wszystkim do redefiniowania zmiennych, które uprzednio zadeklarowano jako podwójnej precyzji lub całkowite /patrz w/w pkt. a/.

Przykłady

Przykład 1
10 DEFINT B, N, W–Z

Za linią 10 zmienne których nazwy rozpoczynają się od liter B, N, W, X, Y i Z /np. BB, N1, WAR, ZX81 itd./ będą zmiennymi całkowitymi, lecz B#, B1#, WA# będą nadal zmiennymi podwójnej precyzji.

Przykład 2

```
10 DEFINT A–D, L–N

100 DEFSNG A–D, L–N
```

Za linią 100 zmienne o nazwach zaczynających się od liter A, B, C, D, L, M, N będą znów traktowane jako pojedynczej precyzji /redefinicja/ lecz C% będzie zmienną całkowitą, a C# zmienną podwójnej precyzji.

Przykład 3

```
10 DEFSTR L–N
20 M1 = "POPRAWNE PRZYPORZADKOWANIE"
30 NU3 = "TEZ POPRAWNE PRZYPORZADKOWANIE"
   .
   .
100 PRINT L# : ' L# JEST JEDNAK ZMIENNA NUMERYCZNA PODWOJNEJ
110              ' PRECYZJI
```

DIM nazwa (dim 1, dim2, ... , dimK)

Deklaruje rozmiary tablicy /liczby elementów w każdym wymiarze tablicy/ o podanej "nazwie". Argumenty "dim" – określające liczby elementów w poszczególnych wymiarach mogą być liczbą lub wyrażeniem numerycznym. Przy braku instrukcji DIM komputer ustala każdy z wymiarów na 11 /wskaźnik 0 – 10/. W formie argumentu instrukcji DIM może również wystąpić cała lista elementów o powyższej strukturze oddzielonej przecinkami.

Przykłady

Przykład 1

```
10 DIM X(4), Y(4,5), ZS(20)
```

Instrukcja deklaruje następujące tablice:

- jednowymiarową tablicę X z elementami o indeksach 0÷4
- dwuwymiarową tablicę Y z elementami o indeksach 0,0÷4,5
- jednowymiarową tablicę łańcuchową z elementami /z których każdy jest łańcuchem/ o indeksach 0÷20.

Przykład 2

```
10 INPUT "WPROWADZ LICZBE DETALI";L
20 DIM T1 (L,4) : REM PRZYKŁAD ZMIENNEGO ROZMIARU TABLICY
```

Dla zmiany struktury tablicy należy najpierw wykonać instrukcję CLEAR.

GOTO numer linii

Przekazuje sterowanie w programie do linii o wymienionym w argumencie numerze.

Przykłady

Przykład 1: /GOTO jako skok bezwarunkowy/
10 GOTO 50

Przykład 2: /GOTO w połączeniu z instrukcją testującą tworzy skok warunkowy/
40 IF X < 0 THEN GOTO 10

Przykład 3: /GOTO w trybie bezpośrednim jako alternatywa zlecenia RUN lecz od linii o określonym numerze i z pominięciem automatycznego CLEAR/

```
(BREAK)          – przyciśnięcie klawisza w trakcie pracy programu
BREAK IN 50
READY
>PRINT X         – testowanie wartości zmiennej X
 7
>X=15            – zmiana wartości zmiennej X
>GOTO 50         – kontynuacja programu od linii 50 /można teraz uruchomić od dowolnej linii/
```

Przy przerwaniu wykonywania programu poprzez (BREAK) i wznawianiu jego działania przy pomocy zlecenia CONT – zmiana wartości zmiennych nie była zabroniona.

ON n GOTO 1-numer linii, 2-numer linii, ... k-numer linii

Przekazuje sterowanie w programie do linii o numerze wyszczególnionym w tym elemencie listy "numerów linii" który wskazuje część całkowita wyrażenia n. Jeśli nie ma takiego elementu, to sterowanie przechodzi do następnej linii programu. Jeśli część całkowita n=0 to uwzględniony jest pierwszy element z listy (jak dla =1). Jeśli n < 0 to zasygnalizowany zostanie błąd. Lista numerów linii zawierać może dowolną liczbę elementów.

Przykład
140 ON X1+1 GOTO 100, 120, 100, 530
Po obliczeniu wartości wyrażenia X1+1 komputer przejdzie do wykonania linii:
100 – jeśli część całkowita wartości wyrażenia = 1
120 – " " " " " = 2
100 – " " " " " = 3
530 – " " " " " = 4

GOSUB numer linii

Przekazuje sterowanie w programie do podprogramu rozpoczynającego się od linii o wyszczególnionym numerze. GOSUB może być poprzedzone instrukcją testującą dzięki czemu uzyskać można warunkowy skok do podprogramu. Podprogramy muszą być zakończone instrukcją RETURN.

Przykład

```
100 IF X < 0 THEN GOSUB 300
110 PRINT "WARTOSC X=" ; X

300 X=X+1 'POCZATEK PODPROGRAMU
310 RETURN
```

ON n GOSUB 1-numer linii, 2-numer linii, ... , k-numer linii.

Przekazuje sterowanie w programie do podprogramu rozpoczynającego się od linii o numerze wyszczególnionym w tym elemencie listy "numerów linii", który wskazuje część całkowita wyrażenia n. Jeśli nie ma takiego elementu, to sterowanie przechodzi do następnej linii programu. Jeśli część całkowita n=0 to uwzględniony jest pierwszy element z listy (jak dla =1). Jeśli n < 0 to zasygnalizowany zostanie błąd. Lista numerów linii zawierać może dowolną liczbę elementów.

Przykład – identyczny jak w opisie ON n GOTO . . . . przy czym słowo GOTO należy zamienić na GOSUB.

RETURN

Kończy podprogram i przekazuje sterowanie w programie do instrukcji następnej po ostatnio wykonywanej instrukcji GOSUB. Jeśli komputer napotka RETURN bez uprzedniego wykonania GOSUB – zasygnalizowany zostanie błąd.

FOR zmienna=wyrażenie1 TO wyrażenie2 STEP krok

sekwencja instrukcji programu

NEXT zmienna

Jest to struktura pętli iteracyjnej zbudowana z dwu instrukcji FOR-TO-STEP oraz NEXT /oddzielne numery linii/. Powoduje powtarzalne wykonywanie "sekwencji instrukcji programu" przy czym początkowa wartość zmiennej sterującej określona "wyrażeniem1" modyfikowana jest o wartość określoną przez "krok" każdorazowo po napotkaniu instrukcji NEXT, aż do osiągnięcia wartości końcowej określonej "wyrażeniem2".

Jeśli to nastąpi, sterowanie przekazywane jest do instrukcji następnej po NEXT. W przeciwnym wypadku kolejne wykonanie sekwencji rozpoczyna się od pierwszej instrukcji po FOR . . . . . Wyrażenia 1, 2 oraz krok mogą być w szczególności stałymi lub pojedynczymi zmiennymi. Modyfikacja wartości zmiennej sterującej polega na jej zwiększeniu lub zmniejszeniu w zależności od znaku wartości kroku.
Jeśli STEP nie wystąpi to komputer ustala wartość kroku na 1. Osiągnięcie wartości określonej "wyrażeniem2" interpretować należy następująco:
Zmienna sterująca osiągnęła wartość
– większą lub równą wartości "wyrażenia2" dla kroku dodatniego
– mniejszą lub równą wartości "wyrażenia2" dla kroku ujemnego

Wartości wyrażenia 1, 2 i kroku ustalone są i zapamiętywane przy pierwszym wykonaniu instrukcji FOR. Zmiana tych wartości wewnątrz pętli nie wpłynie już na sposób wykonania pętli FOR-NEXT. Zmienna sterująca nie może być jednak zmieniona gdyż spowoduje to błędne działanie pętli. Podanie "nazwy" zmiennej sterującej po słowie NEXT jest opcjonalne. Pętle mogą być wielokrotnie zagłębione /zagnieżdżane/. Podawanie "nazwy" po słowie NEXT ma jedynie ułatwić programiście zachowanie właściwej kolejności zagłębień. Musi być wtedy zachowana dyscyplina w ich używaniu tzn. zmienna sterująca z najgłębszej pętli musi wystąpić po pierwszym napotkanym słowie NEXT. Wygodne jest to również przy skokach na zewnątrz pętli FOR-NEXT. Przy zagłębianiu pętli zalecane jest stosowanie techniki wcięć w tekście programu /przesuwanie początku kolejno zagnieżdżonych pętli FOR-NEXT na prawo w stosunku do bloku poprzedniego/. Przy wystąpieniu w trakcie kończenia kolejno zagnieżdżonych pętli sekwencji słów NEXT typu:

```
    NEXT I
  NEXT J
NEXT K
```

można użyć słowa NEXT z listą zmiennych:

```
NEXT I,J,K
```

Przykłady

Przykład 1

```
10 FOR X=1 TO 2 STEP 0.6
20 PRINT X;
30 NEXT
>RUN
1 1.6
READY
>_
```

Wartość 2.2 nie jest już wydrukowana gdyż 2.2>2

Przykład 2

```
10 FOR I=0 TO -10 STEP -2
20 PRINT I;
30 NEXT I
>RUN
0 -2 -4 -6 -8 -10
READY
>_
```

Przykład 3

```
10 X=2
20 Y=5
30 FOR I=X TO Y-1
40 Y=0:X=17
50 PRINT I;
60 NEXT
>RUN
2 3 4
READY
>-
```

W powyższym przykładzie "wyrażenie1" jest zmienną, "wyrażenie2" jest wyrażeniem. Krok nie występuje więc =1. Zmiana wartości "wyrażenia2" wewnątrz pętli nie wpływa już na prawidłowe zakończenie pętli.

Przykład 4

```
10 FOR M=1 TO 2
20 PRINT "PETLA ZEWNETRZNA"
30      FOR N=1 TO 2 'TA PETLA PISANA Z WCIECIEM
40      PRINT"    PETLA WEWNETRZNA"
50      NEXT N
60 NEXT M
>RUN
PETLA ZEWNETRZNA
    PETLA WEWNETRZNA
    PETLA WEWNETRZNA
PETLA ZEWNETRZNA
    PETLA WEWNETRZNA
    PETLA WEWNETRZNA
READY
>-
```

W przykładzie pokazano: zagnieżdżenie pętli oraz użycie wcięć w treści programu. Zamiast linii50 i 60 można użyć instrukcję NEXT N,M.

IF wyrażenie THEN akcja1 ELSE akcja2

Powoduje przetestowanie "wyrażenia" /logicznego lub relacji/ i przekazuje sterowanie w programie do części "akcja1" jeśli wyrażenie jest prawdziwe lub do części "akcja2" gdy nie jest ono prawdziwe. Wyrażenie jest logicznie prawdziwe jeśli przyjmuje wartość niezerową. Może ono być tworzone z różnorodnych wyrażeń relacyjnych łączonych operatorami logicznymi. Jako "akcja" podana może być instrukcja, wiele instrukcji lub numer linii do której przekazać należy sterowanie /bez potrzeby użycia instrukcji GOTO przed numerem linii/. Jeśli "ELSE akcja2" zostanie pominięte to w przypadku gdy wyrażenie nie jest prawdziwe sterowanie przekazywane jest do następnej instrukcji w programie. Dodatkowo w pewnych przypadkach pominąć można słowo THEN zarówno w sytuacji gdy występuje ELSE jak też gdy ELSE zostało pominięte. /W tym ostatnim przypadku otrzymujemy najbardziej uproszczoną formę instrukcji typu "IF wyrażenie-akcja"/.

Słowa THEN nie wolno jednak pominąć w sytuacji gdy:

a/ może to doprowadzić do niejednoznaczności w oddzieleniu części "wyrażenie" od części "akcja" np. w instrukcji:

```
10 IF X=Y THEN Y=0
```

słowo THEN nie może być pominięte,

b/ w części akcja występuje numer linii do której przekazane zostaje sterowanie /gdyż podobnie jak w sytuacji z pkt. a prowadzi to do niejednoznaczności/. Jeśli w takim przypadku THEN zostanie pominięte to numer linii w części "akcja" musi zostać poprzedzony słowem GOTO. Tak więc w przypadku podawania numeru linii dopuszczalne są jedynie następujące formy:

```
                IF wyrażenie THEN numer linii
lub
                IF wyrażenie GOTO numer linii
```

Instrukcje IF – THEN – ELSE mogą być zagnieżdżane tzn. w części "akcja1" lub "akcja2" może wystąpić następna struktura IF – THEN – ELSE. Dla zwiększenia czytelności programu wygodnie jest wtedy stosować technikę wcięć.

Przykłady

Przykład 1

```
40 INPUT X
50 IF X<=0 OR X>=100 THEN PRINT "POZA ZAKRESEM": PRINT X: END ELSE 40
```

Można to również napisać:

```
40 INPUT X
50 IF X<=0 OR X>=100
   THEN
   PRINT "POZA ZAKRESEM":
   PRINT X:
   ELSE 40
```

W przykładzie tym o ile X<0 lub X>100 jest prawdziwe to wykonana jest akcja zapisana po słowie THEN a jeśli nie jest to prawdziwe to wykonana jest akcja zapisana po słowie ELSE.

Przykład 2

```
10 INPUT X
20 IF X>=15 PRINT "X>=15"
30 PRINT "X<15"
40 GOTO 10
```

W przykładzie pominięto słowo THEN oraz ELSE. Gdy X<15 to sterowanie przechodzi bezpośrednio do linii 30. Oczywiście można linie 20, 30, 40 zapisać stosując pojedyńczą strukturę IF – THEN – ELSE.

Przykład 3

```
10 INPUT "WPROWADZ DWIE LICZBY";X,Y
20 IF X<=Y THEN IF X<Y PRINT X; ELSE PRINT "ZADNA NIE"; ELSE PRINT Y;
30 PRINT "JEST MNIEJSZA"
40 GOTO 10
```

Program wybiera i drukuje mniejszą z dwu wprowadzonych liczb. Pokazuje on zagnieżdżanie instrukcji IF – THEN – ELSE. Można go napisać w inny sposób pod względem graficznym /dodając "dla porządku" słowo THEN w instrukcji wewnętrznej/.

```
10 INPUT WPROWADZ DWIE LICZBY";X,Y
20 IF X<=Y
   THEN
      IF X<Y
      THEN
         PRINT X;
      ELSE
         PRINT "ZADNA NIE";
   ELSE
      PRINT Y;
30 PRINT "JEST MNIEJSZA"
40 GOTO 10
```

5.8.3 Instrukcje specjalne

Instrukcje specjalne obejmują instrukcje wykorzystywane w obsłudze błędów /ERL, ERR/, instrukcje obsługujące ekran a w szczególności realizujące grafikę /CLS, POS, SET, RESET, POINT/, instrukcje wejścia-wyjścia /INP,OUT/, instrukcje komunikacji z programem i danymi napisanymi na poziomie języka wewnętrznego /PEEK, POKE, USR/ i inne /MEM, VARPTR/. Instrukcje te zostaną opisane w przedstawionej powyżej kolejności. Używanie ich wymaga od programisty nieco większej wiedzy i praktyki tak w zakresie programowania jak i budowy komputera lecz są one w wielu wypadkach niezwykle wygodne.

ERL

Wyznacza numer linii w której wystąpił błąd i przypisuje go zmiennej o nazwie ERL. Jeśli przed pojawieniem się instrukcji ERL w programie nie wystąpił błąd to ERL=0. Jeśli błąd wystąpił to ERL=numerowi linii z błędem. Jeśli błąd wystąpił w trybie bezpośrednim to ERL przyjmuje wartość 65535. Instrukcja ERL używana jest najczęściej do obsługi błędów "dostępnych" przy pomocy instrukcji ON ERROR GOTO. /patrz 5.8.2/.

Przykład

```
 10 CLEAR 5
 20 ON ERROR GOTO 100
 30 INPUT "WPROWADZ KODOWA KOMBINACJE LITER";K$
 40 REM CIAG DALSZY PROGRAMU
 .
 .
 .
 90 END
100 IF ERL=30 THEN 120
110 ON ERROR GOTO 0
120 PRINT "POWTORZ ! – KOMBINACJA KODOWA MOZE ZAWIERAC MAKS. 5 LITER"
130 RESUME 30
```

Błąd w linii 30 powstać może gdy użytkownik wprowadzi więcej liter łańcucha niż zadeklarowano instrukcją CLEAR.

ERR/2+1

Wyznacza wartość kodu błędu i przypisuje go wyrażeniu ERR/2+1. Zmienna ERR przyjmuje więc wartość: ERR=2*(kod błędu – 1)

Wyrażenie ERR/2+1 najczęściej używane jest wewnątrz programów obsługi błędów wykrywanych dzięki umieszczeniu w programie instrukcji ON ERROR GOTO /patrz 5.8.2/. Kody błędów podano w dodatku 1.

Przykład

```
10 ON ERROR GOTO 200
20 DIM T(12)
30 N=1
40 READ T(N)
50 N=N+1
60 GOTO 40
70 REM WYJSCIE Z PETLI GDY KONIEC DANYCH
.
.
.
100 DATA 10, 9, 8, 7, 6
110 END
200 IF ERR/2+1=4 RESUME 70:REM"4" – KOD "OUT OF DATA"
210 ON ERROR GOTO 0
```

W powyższym przykładzie wyjście z pętli 40, 50, 60 możliwe jest gdy zabraknie danych w specyfikacji instrukcji READ.

CLS

Zeruje ekran i przesuwa kursor do lewego górnego rogu /gaszone są tak znaki alfanumeryczne jak i graficzne/.

POS(x)

Przypisuje wyrażeniu POS(x) liczbę z zakresu 0÷63 wskazującą bieżącą pozycję kursora na ekranie. Argument x jest sztuczny i może nim być dowolne wyrażenie numeryczne. Najwygodniej jest przyjąć x=0.

Przykłady

Przykład 1

```
10 PRINT TAB(10) POS(0)
```

Drukuje 10 za pozycją 10 kursora

Przykład 2

```
10 PRINT "TO" TAB(POS(0)+6) "SA" TAB(POS(0)+6) "POZYCJE";
20 PRINT TAB (POS(0)+1)"TABULACJI"
RUN
TO        SA        POZYCJE TABULACJI
```

SET(x,y)

Ustawia "punkt graficzny" w miejscu określonym współrzędnymi x i y. Jak już wspomniano w rozdz.3, dla celów grafiki każda komórka pola monitora podzielona została na 6 części /tzw. "punktów graficznych"/

tak więc na ekranie utworzona jest siatka zawierająca 128 /zmienna pozioma x/ x 48/zmienna pionowa y/ punktów graficznych przy pracy z ekranem zorganizowanym jako 16 linii x 64 znaki. Współrzędne x numerowane są od lewej do prawej a y od góry w dół. Argumenty x, y nie muszą być wartościami całkowitymi, ponieważ instrukcja uwzględnia jedynie część całkowitą. Mogą też być wyrażeniami numerycznymi, zmiennymi lub stałymi.

RESET(x,y)

Zeruje "punkt graficzny" o współrzędnych (x,y). Patrz opis instrukcji SET (x,y).

POINT(x,y)

Testuje " punkt graficzny " o współrzędnych (x, y). Jeśli dany punkt jest wyświetlany to wyrażenie POINT(x,y) przyjmuje wartość logicznej jedynki a w przeciwnym wypadku wartość logicznego zera.

Przykład
55 IF POINT(53,13) THEN SET(54,13) ELSE SET(53,13)
Jeśli "punkt" o współrzędnych (53.13) nie jest wyświetlony to należy go wyświetlić, a jeśli jest wyświetlony to należy wyświetlić następny w linii.

INP(port)

Wczytuje bajt danych z tzw. "portu" określonego w argumencie przyporządkowując wartość wczytanych danych wyrażeniu INP. Przez port /ang. brama/ rozumie się zewnętrzne urządzenie wejściowe lub wyjściowe typu rejestr, bufor, programowany element interfejsowy dużej skali integracji /np. USART, UART, PIO, CTC itd/. Argument port może przyjmować wartości 0÷255 /256 portów/.

Przykład
5 PRINT "WARTOSC WCZYTANA Z PORTU 4" , INP(4)

OUT port, wartość

Wysyła bajt danych o określonej w argumentach "wartości" do podanego "portu". Argument port może przyjmować wartości 0÷255.

PEEK(adres)

Odczytuje zawartość bajtu informacji z podanego w formie dziesiętnej "adresu" i przyporządkowuje ją wyrażeniu PEEK. Instrukcja pomocna jest przy łączeniu i współpracy programów napisanych w języku BASIC

z programami napisanymi w języku wewnętrznym /efektywnie tworzone programy zorientowane na przekazywanie bajtów informacji/.

Przykład
X=PEEK(16453)
Odczytuje wartość przechowywaną w komórce o adresie 16453 i przyporządkowuje tę wartość zmiennej X. Jeśli rzeczywisty adres komórki pamięci jest większy od 32767 należy zastosować następujący wzór do obliczenia argumentu "adres":

Adres w PEEK = –1 * (żądany adres – 32767)

POKE adres, wartość

Zapisuje "wartość" /z zakresu 0÷255/ określoną w argumentach do komórki pamięci o podanym dziesiętnie adresie. W Dodatku 2 przedstawiono zakresy adresów pamięci w systemie MERITUM-I, które można wykorzystywać do przechowywania wartości /danych/ przez użytkownika. Jeśli rzeczywisty adres komórki pamięci jest większy od 32767 należy zastosować następujący wzór do obliczenia argumentu "adres":

Adres w POKE = –1 * (żądany adres – 32767)

Instrukcję POKE używać można do znacznie szybszego niż przy pomocy instrukcji SET wyświetlania określonego obrazu graficznego. Jak już opisywano w rozdziale 3 każda z 1024 komórek ekranu /przy pracy ekranu w trybie normalnym/ może zostać podzielona na 6 części tzw. "punktów graficznych". W opisie instrukcji SET przedstawiono jak wyświetlać dany punkt graficzny. Kombinację punktów graficznych w danej komórce nazywać będziemy "znakiem graficznym". Znaki graficzne reprezentowane są przez jednobajtowe wartości w kodzie ASCII przy czym na najstarszym bicie b_7 musi być 1 logiczna, bit b_6 jest niewykorzystany a bity $b_5 \div b_0$ reprezentują poszczególne "punkty graficzne". Jeśli dany bit=1 to wyświetlony jest odpowiadający mu punkt graficzny w znaku graficznym. Przypisanie punktom graficznym poszczególnych bitów jest następujące:

b_0	b_1
b_2	b_3
b_4	b_5

Obszar roboczy /1024 komórki/ ekranu to fragment pamięci systemu od adresu 15360 do 16383 /patrz Dodatek 2/. Przechowywane w nim wartości odpowiadają zawartości poszczególnych komórek ekranu. Wartości te reprezentują:

1. Znaki alfanumeryczne o ile $b_7=0$ a na bitach $b_5 \div b_0$ zapisano właściwy kod ASCII znaku.

2. Znaki graficzne o ile wartości określone zostały w/g opisanej powyżej zasady /$b_7=1$ a na bitach $b_5 \div b_0$ kombinacja punktów graficznych/.

Zapełniając przy pomocy instrukcji POKE obszar roboczy monitora wartościami reprezentującymi właściwe znaki graficzne powodujemy około 6-krotne przyspieszenie "tworzenia" obrazu graficznego na ekranie w stosunku do efektu działania instrukcji SET i RESET, które wyświetlały lub zerowały jednokrotnie

1/6 pola znaku graficznego. Oczywiście przy pomocy instrukcji POKE można zapełniać obszar roboczy monitora wartościami reprezentującymi znaki alfanumeryczne /co jednak nie przyspiesza radykalnie wyprowadzania tekstów w stosunku do działania instrukcji PRINT/. Przy pomocy instrukcji PRINT CHR$(X) można /patrz 5.8.4/ wyświetlać na ekranie pojedyncze "znaki graficzne" obliczając reprezentujące je wartości X podobnie jak opisano powyżej. Przyspiesza to również wyświetlanie pojedynczego znaku graficznego. Użytkownik winien używać instrukcji POKE bardzo ostrożnie gdyż zapisanie dowolnej wartości w pewnych obszarach pamięci /program w RAM, dane w RAM z których korzystają programy umieszczone w pamięci ROM, stos itd./, może spowodować zniszczenie programu /tzw. "wypadnięcie systemu"/. Jeśli nieostrożne użycie POKE spowoduje "wypadnięcie systemu" to należy komputer wyzerować.

USR(x)

Wywołuje program /ma on w stosunku do programu w BASIC'u charakter procedury/ napisany w języku wewnętrznym mikroprocesora /Z-80/, którego adres szesnastkowy umieszczony został uprzednio w komórkach o adresach dziesiętnych 16527 /starszy bajt adresu/ i 16526 /młodszy bajt adresu/. Przekazywany do procedury parametr x umieszczony zostanie jako całkowita liczba dwubajtowa pod dziesiętnym adresem 2687 /i może być wykorzystany przez program w języku wewnętrznym/. Jeśli użytkownik nie chce przekazywać do programu w języku wewnętrznym żadnego parametru to x jest parametrem sztucznym /np. 0/. Parametr x może przyjmować wartości z zakresu –32768 do +32767. Niewłaściwe użycie instrukcji USR, która spowoduje skok w przypadkowe miejsce w pamięci RAM /gdy nie ustawiono właściwego adresu w w/w komórkach a w takim przypadku przyjmują one przypadkowe wartości/ lub gdy pod adresem załadowanym do w/w komórek nie znajduje się poprawny program w języku wewnętrznym, może spowodować "wypadnięcie systemu". Przy powrocie z procedury do programu głównego /w języku BASIC/ wyrażeniu USR(x) zostaje przypisywana wartość przekazana z procedury a gdy nie przesyła ona żadnej wartości – wartość argumentu x. Użycie instrukcji USR ma sens gdy programista zna zasady pisania programów w języku wewnętrznym mikroprocesora Z-80.
W obecnym modelu /MERITUM-I/ istnieje możliwość wczytania takiego programu z kasety /a także z innych urządzeń peryferyjnych np. czytnika taśmy perforowanej/ przy pomocy zlecenia SYSTEM lub wprowadzenia go do pamięci w kodach szestnastkowych przy pomocy instrukcji POKE.
W dalszych modelach użytkownik dysponował będzie bogatym oprogramowaniem uruchomieniowym dla tworzenia programów w języku wewnętrznym /asembler, program redagujący – edytor, program uruchamiający – debugger, program wprowadzająco-łączący/.
Dla wprowadzenia do komórek 16526 i 16527 adresu początku procedury można użyć instrukcji POKE. Zasady tworzenia procedur w języku wewnętrznym mogących współpracować z oprogramowaniem napisanym w języku BASIC są następujące:

a/ należy zarezerwować obszar w górnej części pamięci /najstarsze adresy -- patrz Dodatek 2/. Po określeniu liczby bajtów programu należy ją odjąć od najwyższego adresu pamięci, a otrzymany adres będzie adresem początkowym procedury. Adres ten należy podać po komunikacie OBSZAR PAO? przy inicjalizacji systemu.

b/ celem pobrania argumentu przekazanego w parametrze instrukcji USR do procedury należy wykonać instrukcję CALL 0A7FH, (H – zapis szesnastkowy)

c/ celem powrotu do programu głównego /w BASIC'u/ należy użyć instrukcji RET o ile nie będzie przekazywana żadna wartość,

d/ dla przekazania jakiejś wartości /wyniku działania procedury/ do programu w BASIC'u należy przekazywaną wartość załadować do pary rejestrów HL i wykonać instrukcję JMP 0A9AH,

e/ procedura nie może obciążyć stosu więcej niż 8-mioma poziomami /16 komórek stosu/.

Przykłady

Przykład 1
Użytkownik chce wykonać skok do procedury w języku wewnętrznym znajdującej się w obszarze pamięci od adresu 7FF0H nie przekazując parametru do procedury.

```
.
.
.
100 POKE 16526,240
110 REM F0H=240 DZIESIETNIE
120 POKE 16527,112
130 REM 7FH=112 DZIESIETNIE
140 USR(0)
150 REM TU POWROT PO INSTRUKCJI RET W PROCEDURZE
```

Przykład 2
Użytkownik wywołuje procedurę przekazującą do programu głównego /w BASIC'u/ wartość, która powiększona o 4 stanowi zmienną A:

```
.
.
140 A=USR(0)+4
.
.
.
```

Przykład 3

```
.
.
.
5 USR(23)
.
.
.
```

Przekazuje do procedury w języku wewnętrznym wartość 23.

MEM

Wyznacza liczbę wolnych bajtów w pamięci /nieużywanych i niechronionych/. Może być użyta w trybie bezpośrednim np. w instrukcji PRINT MEM lub wewnątrz programu w celu zlikwidowania możliwych błędów "braku pamięci" /OM – Out of Memory/.

Przykład

```
.
.
.
.
.
5 IF MEM<80 THEN PRINT "UWAGA!: MASZ DO DYSPOZYCJI 80 BAJTOW"
10 REM DALSZA CZESC PROGRAMU
```

VARPTR (nazwa zmiennej)

Wyznacza wartość adresów w pamięci pod którym umieszczona jest wartość zmiennej o podanej "nazwie zmiennej". Przyjmując, że instrukcja daje w wyniku adres ADR to dla poszczególnych typów zmiennych odpowiednie ich wartości ulokowane są następująco /nawiasy oznaczają zawartość komórki pamięci spod adresu/:

a/ dla zmiennej całkowitej:

(ADR) – mniej znaczący bajt /w zapisie uzupełnieniowym do 2/
(ADR+1) – bardziej znaczący bajt /w zapisie uzupełnieniowym do 2/

b/ dla zmiennej pojedynczej precyzji:

(ADR) – mniej znaczący bajt
(ADR+1) – kolejny bajt
(ADR+2) – najbardziej znaczący bajt
(ADR+3) – wykładnik /powiększony o 128/

c/ dla zmiennej podwójnej precyzji:

(ADR) – najmniej znaczący bajt
(ADR+1) – następny bajt
(ADR+ . .) – . . . . . .
(ADR+6) – najbardziej znaczący bajt
(ADR+7) – wykładnik /powiększony o 128/

d/ dla zmiennej łańcuchowej:

(ADR) – długość łańcucha
(ADR+1) – mniej znaczący bajt adresu początku łańcucha
(ADR+2) – bardziej znaczący bajt adresu początku łańcucha

W przypadku b/ i c/ liczba przekazywana jest w znormalizowanej postaci wykładniczej. Kropka dziesiętna znajduje się przed najbardziej znaczącym bajtem, którego najstarszy bit jest bitem znaku.

Przykład
Jeśli zmienna A! przyjmie wartość 2 to będzie ona przechowywana w sposób następujący:
2 dziesiętnie = 10 binarnie a w postaci znormalizowanej jest to .1E2.
Wykładnik równy jest 128+2=130
Najbardziej znaczący bajt równy jest 10000000 lecz najstarszy bit zostaje zmieniony na 0 gdyż liczba jest dodatnia. Tak więc w poszczególnych komórkach pamięci znajdują się następujące wartości:

(ADR) – 0
(ADR+1) – 0
(ADR+2) – 0
(ADR+3) – 130

5.8.4 Instrukcje operacji na łańcuchach.

Wartością łańcuchową jest dowolny ciąg znaków alfanumerycznych /akceptowanych przez system MERITUM/. Do przechowywania wartości łańcuchowych użyta może być każda poprawna nazwa zmiennej zadeklarowana jako zmienna łańcuchowa poprzez instrukcję DEFSTR lub przy pomocy znaku deklaracji typu ($) – np. T$, TEKST$ itd. Pojedyncza zmienna łańcuchowa "przechować" może łańcuch o długości do 255 znaków. Stałe łańcuchowe /konkretne teksty złożone ze znaków alfanumerycznych przypisywane w danym momencie konkretnej zmiennej łańcuchowej/ ujęte są zawsze w cudzysłów /np. T$=" TEKST"/.

Np. przypisywanie : T$="TEKST NR 1"
porównywanie : IF T$<>"TEKST NR 2"

Wprowadzanie i wyprowadzanie stałych łańcuchowych odbywa się podobnie jak dla stałych numerycznych przy użyciu instrukcji INPUT, READ/DATA i PRINT. Przy wprowadzaniu łańcuchów instrukcją INPUT oraz w specyfikacji danych łańcuchowych instrukcji DATA znak cudzysłowu może być pominięty o ile stałe łańcuchowe nie zawierają: przecinków, dwukropków lub poprzedzających spacji. Zmienne lub stałe łańcuchowe nazywać będziemy krótko łańcuchami.

Przykłady

Przykład 1

```
10 INPUT "NAZWISKO IMIE";N$
20 INPUT "GODZINA: MINUTA";G$
>RUN
NAZWISKO IMIE? PODLIPIEWSKI JACEK
GODZINA: MINUTA?"22:30"
READY
```

W przypadku wprowadzania drugiej stałej łańcuchowej /czas aktualny/ cudzysłów był konieczny ze względu na występowanie w obrębie łańcucha dwukropka.

Przykład 2

```
. . . . . .

100 DATA KLIENT                    : REM BEZ CUDZYSŁOWU
110 DATA " PODLIPIEWSKI,JACEK"     : REM CUDZYSŁOW BO PRZECINEK
120 DATA "SUMA WPLATY:"            : REM CUDZYSLOW BO DWUKROPEK
130 READ X$,Y$,Z$
140 PRINT X$,Y$
150 PRINT Z$
160 PRINT S
>RUN
KLIENT PODLIPIEWSKI,JACEK
SUMA WPLATY 256.3
READY
>_
```

Łańcuchy mogą być argumentami wielu operacji jak np.:

1. Składanie łańcuchów oznaczane przez znak plus (+):

```
Przykład
100 XS="PRZYKLAD"
110 YS="SKLADANIA"
120 ZS="LANCUCHOW"
130 WS=XS+"-"+YS+"-"+ZS
140 PRINT WS
>RUN
PRZYKLAD-SKLADANIA-LANCUCHOW
```

2. Porównywanie łańcuchów do którego użyte mogą być następujące operatory relacji:

   =, <>, >, >=, <, <=

   Porównywanie łańcuchów ma na celu stwierdzenie ich równości, nierówności lub określenie następstwa alfabetycznego. Łańcuchy porównywane są znak po znaku od lewej strony do prawej. W przypadku równości wszystkie znaki łącznie z poprzedzającymi je spacjami muszą być identyczne. Przy porównywaniu brane są pod uwagę kody ASCII znaków i znak o kodzie stanowiącym mniejszą liczbę jest poprzedzającym. Przy porównywaniu łańcuchów o różnych długościach krótszy łańcuch poprzedza dłuższy gdy jego znaki są takie same jak w początkowej części dłuższego łańcucha /np. "X" poprzedza "XYZ"/.

3. Grupa operacji specjalnych w skład których wchodzą opisane poniżej instrukcje operacji na łańcuchach. Wśród nich wyróżnić można 2 grupy instrukcji:

   a/ tworzące i przetwarzające łańcuchy /w kolejności: LEFT$, RIGHT$, MID$, STRING$, ASC, CHR$, STR$, VAL, INKEY$,

   b/ organizacyjne /w kolejności: FRE, LEN/.

LEFT$ (łańcuch, n)

Tworzy nowy łańcuch składający się z n /stała lub wyrażenie numeryczne/ pierwszych znaków "łańcucha" /stała łańcuchowa lub wyrażenie łańcuchowe/ będącego argumentem.

```
Przykład
100 XS="MERITUM"
110 YS=LEFT$(XS,3)
120 PRINT YS+"A-ELZAB"
>RUN
MERA-ELZAB
READY
>_
```

RIGHT$(łańcuch, n)

Tworzy nowy łańcuch składający się z n /stała lub wyrażenie numeryczne/ ostatnich znaków"łańcucha" /stała łańcuchowa lub wyrażenie łańcuchowe/ będącego argumentem.

Przykład

```
100 X$="MERITUM"
110 Y$=RIGHT$ (X$, 4)
120 PRINT Y$
>RUN
ITUM
READY
>-
```

MID$(łańcuch, p, n)

Tworzy nowy łańcuch składający się z n znaków począwszy od znaku z pozycji p /n, p – stałe lub wyrażenie numeryczne/"łańcucha"/stała łańcuchowa lub wyrażenie łańcuchowe/ będącego argumentem. Jeśli w liście argumentów nie wyspecyfikowano argumentu n to tworzony jest łańcuch począwszy od pozycji p do końca.

Przykład

```
100 X$="MERITUM"
110 Y$=MID$(X$, 4,2)
120 Z$=RIGHT$(X$, 1)
130 W$=Y$+Z$+"–PRZEDS.POLONIJNO-ZAGR."
140 PRINT W$
>RUN
ITM – PRZEDS.POLONIJNO-ZAGR.
READY
>-
```

STRING$(n, arg)

Tworzy łańcuch składający się z n identycznych elementów będących:

– wyszczególnionym w argumencie "arg" znakiem ASCII – jeśli postać argumentu jest następująca: "znak ASCII" – np. STRING$(10, "*") daje "* * * * * * * * * *"

– znakiem ASCII /alfanumerycznym, sterującym lub graficznym/ odpowiadającym wyszczególnionej w argumencie "arg" liczbie – jeśli postać argumentu jest następująca: "liczba z zakresu 0÷255" np.: STRING$ (10,35) daje "# # # # # # # # # #" gdyż 35 jest kodem ASCII znaku "#".

ASC(łańcuch)

Wyznacza kod ASCII /dziesiętnie/ pierwszego znaku niepustego "łańcucha" będącego stałą łańcuchową lub wyrażeniem łańcuchowym.

Przykład

```
 90 X$="MAREK"
100 PRINT ASC(MID$ (X$, 2,2))
>RUN
65
READY
>_
```

CHR$(wyrażenie)

Tworzy łańcuch o długości 1 znaku ASCII, którego kod reprezentowany jest przez wartość /dopuszczalne 0÷255/ wyrażenia /stała, zmienna lub wyrażenie numeryczne/. CHR$ realizuje więc funkcję odwrotną do ASC. Przy pomocy CHR$ można wyświetlić dowolny ze znaków graficznych /kody 129÷191/ lub spowodować wykonanie funkcji sterującej przypisanej kodem sterującym /Kody 0÷31/ np.: PRINT CHR$(23) przełącza format ekranu na 32 znaki w linii. Możliwe też jest w ten sposób dołączenie do łańcuchów znaku cudzysłowu.

Przykład

```
10 X$=CHR$(34)'  34=KOD CUDZYSLOWU
20 PRINT,"WYNIKI OBLICZEN PROGRAMU";X$;"REAKTOR";X$
>RUN
                    WYNIKI OBLICZEN PROGRAMU "REAKTOR"
READY
>_
```

STR$(wyrażenie)

Tworzy łańcuch będący znakową reprezentacją wartości wyrażenia /stała, zmienna lub wyrażenie numeryczne/. Format łańcucha zawiera spację poprzedzającą a nie zawiera spacji końcowej: " łańcuch wartości wyrażenia" /PRINT wyprowadzał ze spacją końcową/. W przypadku ujemnej wartości wyrażenia pierwszym znakiem łańcucha w miejscu spacji początkowej jest znak minusa. Na tak utworzonym łańcuchu można przeprowadzać jedynie operacje łańcuchowe, podczas gdy na wyrażeniu będącym argumentem w STR$ dopuszczalne były jedynie operacje arytmetyczne.

Przykłady

Przykład 1

```
10 X=2.314
20 Z$=STR$(X):REM POWSTAJE LANCUCH" 2.314"
30 PRINT Z$
>RUN
 2.314
READY
>_
```

Przykład 2

```
10 X=2.314:Y=-X
20 PRINT STR$(X)
30 PRINT STR$(Y)
40 PRINT STR$(X-Y)
50 PRINT STR$(X)+STR$(Y)
>RUN
 2.314
-2.314
 4.628
 2.314-2.314
READY
>
```

VAL(łańcuch)

Wyznacza liczbę reprezentowaną przez znaki łańcucha /stała, zmienna, lub wyrażenie typu łańcuchowego/ przy czym:

- dla łańcuchów reprezentujących jedynie wartości liczbowe uwzględnione są wszystkie znaki łańcucha,
- dla łańcuchów mieszanych /tzn. wartość liczbową z następującym po niej tekstem np. "150 ZLOTYCH"/ uwzględnione są początkowe znaki łańcucha reprezentujące wartości liczbowe.

VAL działa więc odwrotnie w stosunku do STR$.

Przykłady

Przykład 1

```
10 INPUT"WPROWADZ CZESC CALKOWITA";X$
20 INPUT"WPROWADZ CZESC ULAMKOWA";Y$
30 PRINT VAL(X$+"."+Y$)
>RUN
WPROWADZ CZESC CALKOWITA?127
WPROWADZ CZESC ULAMKOWA?49
 127.49
READY
>
```

Przykład 2

```
10 A$="12"
20 B$="53"
30 PRINT "AKTUALNY CZAS="; VAL(A$+":"+B$)
>RUN
AKTUALNY CZAS=12:53
READY
>
```

INKEY$

W momencie pojawienia się sygnału strobu /cykliczny sygnał informujący o możliwości pobrania informacji/ z klawiatury tworzy jednoznakowy łańcuch odpowiadający znakowi naciśniętego klawisza klawiatury. Jeśli w czasie strobu nie został naciśnięty klawisz to wprowadzony jest łańcuch pusty /o długości zero/. Dzięki tej instrukcji możliwe jest wprowadzenie do komputera informacji bez potrzeby wstrzymania pracy programu i użycia klawisza (ENTER) Jest to szczególnie przydatne w programach tzw. gier "videokomputerowych". Ze względu na krótki czas trwania impulsu strobu instrukcję INKEY$ umieszcza się wewnątrz pętli co zapewnia wielokrotne "testowanie klawiatury" i skuteczne wprowadzenie żądanej informacji. Jeśli przy wyprowadzaniu łańcucha na ekran za pomocą instrukcji PRINT INKEY$ żaden klawisz nie został naciśnięty – wprowadzany jest łańcuch pusty /niewidoczny/ nie zmieniając jednak poprzedniej zawartości danej pozycji ekranu.

Przykłady

Przykład 1

```
10 REM***PROGRAM WYSWIETLA W LEWYM GORNYM ROGU EKRANU  ***
20 REM***ZNAK AKTUALNIE NACISKANEGO KLAWISZA            ***
30 REM***                                              ***
40 CLS
50 PRINT@1,INKEY$
60 GOTO 50
```

Przykład 2

```
10 PRINT "WPROWADZ 2 ZNAKI W POSTACI:X,Y"
20 X$=INKEY$
30 IF X$=""THEN 20 ELSE PRINT X$;
40 P$=INKEY$
50  IF P$=""THEN 40 ELSE IF P$< >","THEN PRINT "JESZCZE RAZ":GOTO 10 ELSE PRINT P$;
60 Y$=INKEY$
70 IF Y$=""THEN 60 ELSE PRINT Y$
80 PRINT "O'K"
```

W liniach 30, 50, 70 pokazano porównywanie jednoznakowych łańcuchów z łańcuchami pustymi /""/. Pętle w liniach 30, 50 i 70 powtarzane są do skutku, aż do wprowadzenia konkretnych znaków z klawiatury.

FRE(łańcuch)

Wyznacza wielkość aktualnie dostępnego obszaru pamięci przeznaczonej na łańcuchy. Argument "łańcuch" jest argumentem pozornym i może nim być dowolna stała lub zmienna łańcuchowa.

Przykład

```
10 PRINT FRE(XS)"WYPROWADZENIE WIELKOSCI
20                 'OBSZARU PAMIECI DLA LANCUCHOW
```

LEN(łańcuch)

Wyznacza długość podanego w argumencie łańcucha /stała, zmienna lub wyrażenie łańcuchowe/ liczoną w znakach.

Przykład

```
10 X$="ROZNE":Y$="DLUGOSCI":Z$="LANCUCHOW"
20 PRINT,X$,Y$,Z$
30 PRINT "LICZBA ZNAKOW:";
40 PRINT,LEN(X$),LEN(Y$),LEN(Z$)
>RUN
                ROZNE           DLUGOSCI        LANCUCHOW
LICZBA ZNAKOW:   5               8               9
READY
>_
```

Rozdział 6

PRACA W TRYBIE EDYCJI

6. Praca w trybie edycji

Pod pojęciem edycji /inaczej redakcji/ programu w języku BASIC rozumiany jest proces przygotowania linii programu w trakcie jej pierwotnego wprowadzenia lub poprawianie treści linii po wskazaniu przez komputer błędu składniowego czy dla dokonania zmiany struktury logicznej programu np. po zaobserwowaniu błędów w działaniu programu. Proces edycji w komputerze MERITUM-I może się odbywać wyłącznie w obrębie jednej linii. Edycję przeprowadzać można w trybie edycji który wywołuje użytkownik lub automatycznie wywołuje go sam system po napotkaniu błędu w określonej linii programu przechodząc do edycji tej właśnie linii. W trybie edycji można wykonywać szereg zleceń edycji /można je traktować jako podzlecenia zlecenia trybu bezpośredniego: EDIT / oraz używać klawiszy funkcyjnych umożliwiających poprawianie tekstu. Zlecenia i klawisze tworzą efektywny mechanizm umożliwiający sprawne pisanie i poprawę programu. Z trybu bezpośredniego do trybu edycji użytkownik przechodzi przy pomocy zlecenia:

EDIT numer linii

Jeśli dana linia nie istnieje to wystąpi błąd FC. Gdy zamiast numeru linii użyta zostanie kropka /./
EDIT.
to nastąpi przejście do trybu edycji linii ostatnio wprowadzanej, ostatnio zmienianej lub tej w której wystąpił błąd. Po wprowadzeniu zlecenia EDIT /poprzez naciśnięcie (ENTER) / komputer drukuje właściwy numer linii /zatrzymując kursor na pierwszej pozycji, bez wyświetlenia zawartości linii/ i oczekuje na wprowadzenie któregoś ze zleceń edycji lub naciśnięcie któregoś z klawiszy funkcyjnych edycji.
>EDIT 30 (ENTER)
30 –

Poprzez sesję edycji rozumiane są wszelkie operacje którym poddana została dana linia w trybie edycji od wykonania zlecenia EDIT do powtórnego przejścia komputera do trybu bezpośredniego. Zasada pracy edytora polega na przepisaniu treści poprawianej linii do tymczasowego bufora linii w pamięci. Wszelkie poprawki w treści linii wykonywane przy pomocy klawiszy funkcyjnych edycji /p. 6.2/ lub zleceń edycji /p. 6.3/ wprowadzane są do bufora podczas gdy w treści programu występuje wciąż stara postać linii. W trakcie pojedynczej sesji edycji operator użyć może opisanych w p. 6.1 zleceń (klawiszy) sterujących przebiegiem danej sesji.

6.1 Zlecenia sterujące przebiegiem sesji edycji

A

Powoduje skasowanie wszystkich poprawek w treści linii w buforze dokonanych w danej sesji edycji, przywraca pierwotną zawartość bufora /przepisanie wciąż niezmienionej treści linii z programu do bufora/, przesuwa kursor do początku linii. Sesja edycji nie jest zakończona /komputer nie przechodzi do trybu bezpośredniego/. Komputer pozostaje w trybie edycji danej linii.

Q

Powoduje skasowanie wszystkich poprawek dokonanych w bieżącej sesji edycji i zakończenie sesji edycji poprzez wyjście do trybu bezpośredniego. Linia programu zostaje nie zmieniona.

E

Powoduje zakończenie aktualnej sesji edycji /wyjście do trybu bezpośredniego/ przy zaakceptowaniu wszelkich dokonanych zmian poprzez przepisanie zawartości bufora linii w miejsce dotychczasowej zawartości linii programu.

Komputer reaguje w opisany powyżej sposób na przyciśnięcie klawiszy A, Q, E o ile nie wykonywał fazy wprowadzania tekstu dla któregoś ze zleceń edycji. W przeciwnym wypadku należy najpierw zakończyć fazę wprowadzania tekstu. Działanie zlecenia E jest podobne do akcji wywołanej przyciśnięciem klawisza (ENTER) z tym, że ten ostatni przyciśnięty może być w każdej sytuacji.

Wyjście z trybu edycji możliwe jest więc również poprzez naciśnięcie klawisza (ENTER), co spowoduje:

– wprowadzenie na trwałe do programu wszystkich dokonanych zmian w treści danej linii w buforze,

– wyświetlenie całej zawartości linii bez względu na to w jakiej pozycji linii znajdował się kursor

– przejście do trybu bezpośredniego

L

Wyświetla w całości aktualnie edytowaną linię programu bez względu na pozycję kursora po czym w nowej linii drukuje ponownie numer edytowanej linii umieszczając kursor na pierwszej pozycji i oczekuje na nowe zlecenia edycji. Przy wykonywaniu L komputer nie może znajdować się w fazie wprowadzania tekstu dla któregoś ze zleceń edycji. Zlecenie to umożliwia skontrolowanie aktualnej zawartości linii /np. po wykonaniu wszystkich lub części poprawek/ bez potrzeby wychodzenia z trybu edycji tej linii.

Przykład
Przyjmując następującą pozycję kursora w przykładowej linii:

30 IF X=1–
po naciśnięciu L otrzymamy:
30 IF X=1.5 THEN PRINTX
30 –

Przebieg przykładowej sesji edycji przedstawiono na następnej stronie.

Teksty wyświetlane na ekranie	Przyciskane klawisze – zlecenia sterujące przebiegiem sesji	Uwagi
>EDIT 100	(ENTER)	Rozpoczęcie sesji edycji linii 100 /przejście z trybu bezpośredniego do trybu edycji/.
100 –		
100 PIERWOTNA ZAWARTOSC LINII	L	Po licznych poprawkach listowanie.
100 POPRAWIONA ZAWARTOSC LINII	L	
100 NOWE POPRAWKI	A	Poprawki były niewłaściwe, powrót do pierwotnej postaci linii.
100 –		
100 PIERWOTNA ZAWARTOSC LINII	L	Po licznych poprawkach listowanie.
100 INACZEJ POPRAWIONA LINIA	E lub(ENTER)	Zaakceptowanie zmian, wpisanie nowej treści linii do programu, zakończenie sesji edycji przejście do tryby bezpośredniego.
>–		Komputer znajduje się ponownie w trybie bezpośrednim.

Właściwy proces edycji polegający na wstawianiu, usuwaniu lub poprawie znaków w linii oraz konieczne do tego celu sterowanie pozycją kursora osiąga się przy pomocy klawiszy funkcyjnych edycji i zleceń edycji, których działanie opisano poniżej. Dla zilustrowania realizowanych przez nie funkcji przyjmujemy przykładową linię programu:

30 IF X=1.5 THEN PRINTX /jak z poprzedniego przykładu/

6.2 Klawisze funkcyjne edycji

Kursor może być przesuwany wzdłuż linii przy czym zawartość linii na lewo od aktualnej pozycji kursora jest wyświetlona a na prawo wygaszona. Przesuwaniem kursora sterują klawisze:

(SP) /listwa spacji u dołu klawiatury ang. space bar/ – przesunięcie w prawo o 1 pozycję i wyświetlenie napotkanego znaku

(←) /ang. back space/ – przesunięcie w lewo o 1 pozycję i zgaszenie napotkanego znaku.

Przykład
>EDIT 30 (ENTER)
30 –
Przyciskamy (SP)
30 I–
Przyciskamy (SP)
30 IF–
Przyciskamy (←)
30 I–

Oczywiście cofanie kursora powodując wygaszenie znaku na ekranie nie usuwa go z bufora linii programu w pamięci. Jeśli przed naciśnięciem klawisza (SP) lub (←) zostaną przyciśnięte klawisze specyfikujące liczbę n

n(SP)

n(←)

akcje związane z tymi klawiszami zostaną wykonane n-krotnie /przesunięcie o n pozycji/. Oczywiście sama liczba n nie zostanie wyświetlona na ekranie /gdyż zniszczyłaby wówczas fragment tekstu linii programu/.

Przykład
Kontynuując poprzedni przykład od ostatnio występującej pozycji kursora:

– przyciśnięcie klawisza 7 po czym (SP) spowoduje wyświetlenie:
30 IF X=1.5–

– przyciśnięcie w tej sytuacji 4 po czym (←) spowoduje wyświetlenie:
30 IF X–

Działanie niektórych z opisanych poniżej zleceń polega na wstawianiu tekstu poprawek. Takimi zleceniami są: I, H, X, nC. /patrz poniżej/ Po napisaniu powyższych zleceń /bez końcowego (ENTER)/ komputer oczekuje na wprowadzanie tekstu poprawek, przechodząc do tzw. fazy wprowadzania tekstu. W wypadku zlecenia nC komputer automatycznie zakończy przyjmowanie tekstu kończąc działanie tego zlecenia po wprowadzeniu n znaków zgodnie z podanym parametrem n zlecenia po czym przejdzie do stanu gotowości wykonania funkcji kolejnych zleceń czy klawiszy edycji. W przypadku zleceń I, H, X długość tekstu poprawki nie jest z góry określona tak więc konieczne jest wprowadzenie do komputera znacznika końca tekstu poprawki a tym samym zakończenie fazy wprowadzania. Celem zakończenia fazy wprowadzania tekstu dla zleceń edycji służących do wstawiania poprawek o nieokreślonej w parametrze zlecenia długości /zlecenia X, I, H/ należy nacisnąć równocześnie klawisze (SHIFT) i (↑) Komputer pozostaje wówczas nadal w trybie edycji, a kursor pozostaje na dotychczasowej pozycji. Działanie zleceń X, I, H zakończyć można również poprzez przyciśnięcie (ENTER) ale to jak już wspomniano spowoduje również zakończenie edycji, zaakceptowanie dotychczysowych poprawek i wyjście do trybu bezpośredniego.

6.3 Zlecenia edycji

Podobnie jak parametr "n" w kombinacji n(SP) i n(←) klawiszy funkcyjnych tak też i tekst poniższych zleceń edycji nie jest wyświetlany na ekranie, aby nie zniszczyć zawartości pozycji poprawionej linii programu wskazywanej aktualnie przez kursor. Zlecenia edycji nie są też /jak to było w wypadku zleceń trybu bezpośredniego/ wprowadzane do komputera poprzez przyciśnięcie klawisza (ENTER) Komputer automatycznie podejmuje akcję przypisaną zleceniu po wprowadzeniu jego mnemoniki.

I – "wstaw"

Wstaw tekst do linii programu w buforze począwszy od aktualnej pozycji kursora. Po przyciśnięciu I komputer przechodzi do fazy wprowadzania oczekując na tekst poprawki. Użycie w tej sytuacji klawisza (←) będzie powodowało cofanie kursora oraz usuwanie tekstu z ekranu i z bufora linii. Wyjście z trybu wprowadzania tekstu odbywa się poprzez naciśnięcie (SHIFT) (↑) i komputer przechodzi do oczekiwania na nowe zlecenia edycji.

Przykład
>EDIT30(ENTER)
30 –
Po naciśnięciu 4 i(SP)otrzymamy
30 IF X–
Po naciśnięciu w kolejności klawiszy I 2 5 otrzymamy:
30 IF X25–
Po naciśnięciu (SHIFT)(↑) nastąpi wyjście z trybu wstawiania a po wywołaniu zlecenia L /litera L już nie będzie wyświetlona/ otrzymamy:
30 IF X25=1.5 THEN PRINTX
30–

X – "przejdź do końca linii i wstaw"

Powoduje przesunięcie kursora do końca linii /wyświetlając jej zawartość/ **i przejście do fazy wprowadzania tekstu.** Umożliwia to dopisanie w buforze poprawek na końcu linii. Użycie w fazie wprowadzania tekstu poprawki klawisza (←) cofa kursor gasząc znaki na ekranie oraz usuwając je z zawartości bufora poprawionej linii. Wyjście z fazy wprowadzania tekstu odbywa się poprzez naciśnięcie klawiszy (SHIFT)(↑) po czym komputer przechodzi do oczekiwania na nowe zlecenie edycji.

Przykład /linia 30 ma swą początkową postać/
>EDIT 30
30 –
Po przyciśnięciu klawisza X natychmiast /bez kończącego (ENTER)/ na ekranie pojawi się:
30 IF X=1.5 THEN PRINTX–
i komputer oczekiwał będzie na wprowadzenie tekstu. Przyciśnięcie klawisza (←) oraz klawiszy znaków Y + 2 spowoduje:
30 IF X=1.5 THEN PRINTY+2 –
po czym przyciśnięcie (SHIFT)(↑) kończy wprowadzanie tekstu i komputer oczekuje na dalsze zlecenia edycji.

H – "odetnij i wstaw"

Powoduje usunięcie znaków od aktualnej pozycji kursora do końca linii tak z ekranu jak i z zawartości bufora poprawianej linii oraz przejście do fazy wprowadzania tekstu. Klawisz (–) ma w fazie wprowadzania tekstu działanie analogiczne jak w zleceniu X. Wyjście z fazy wprowadzania tekstu odbywa się poprzez przyciśnięcie klawiszy (SHIFT)(↑) po czym komputer przechodzi do oczekiwania na nowe zlecenia edycji.

Przykład

Załóżmy, że postać linii 30 jest jak na końcu poprzedniego przykładu i chcemy przywrócić jej poprzednią postać. Przyciśnięcie 3(←)po czym H spowoduje usunięcie znaków Y+2 i przejście do fazy wprowadzania tekstu od następującej pozycji kursora:

30 IF X=1.5 THEN PRINT–

Wprowadzenie poprawki /literka zmiennej X/ oraz przyciśnięcie klawiszy(SHIFT)(↑)spowoduje zakończenie tekstu i przejście do oczekiwania na następne zlecenie edycji. Aktualna postać ekranu i zawartości bufora poprawionej linii programu jest przy tym następująca:

30 IF X=1.5 THEN PRINTX–

nD – "usuń n znaków"

Powoduje usunięcie z bufora linii n znaków na prawo od aktualnej pozycji kursora. Usunięte znaki zostają na ekranie umieszczone pomiędzy znakami wykrzyknika celem pokazania fragmentu który został usunięty.

Przykład

Załóżmy następującą pozycję kursora w linii 30

30 IF X=1.5 THEN –

Przy przyciśnięciu w kolejności klawiszy 6 oraz D na ekranie pojawi się:

30 IF X=1.5 THEN !PRINTX!–

a znaki PRINTX zostaną usunięte z zawartości bufora linii tak więc po wykonaniu zlecenia listowania L na ekranie pojawi się:

30 IF X=1.5 THEN

30 –

nC – "zastąp n znaków"

Powoduje usunięcie n znaków /z ekranu i z bufora/ począwszy od aktualnej pozycji kursora i przejście do fazy wprowadzania tekstu w której komputer znajduje się dopóki nie wprowadzi się n nowych znaków. Umożliwia to zastąpienie n znaków tą samą liczbą nowych znaków. Po wprowadzeniu n nowych znaków komputer automatycznie kończy fazę wprowadzania tekstu /bez potrzeby naciskania (SHIFT)(↑)/ i przechodzi do oczekiwania na nowe zlecenie edycji.

Przykład

Po ustawieniu kursora w linii 30 na przedstawioną poniżej pozycję

30 IF –

przyciśnięcie w kolejności 3 C po czym Y=3 spowoduje

30 IF Y=3–

i komputer po wyjściu z fazy wprowadzania tekstu oczekuje na dalsze zlecenia edycji. Np. po naciśnięciu L /listowanie/ otrzymamy:

30 IF Y=3.5 THEN PRINTX

30 –

nS znak – "znukaj znaku"

Powoduje odszukanie n-tego wystąpienia "znaku" w linii na prawo od aktualnej pozycji kursora i przesunięcie kursora do tej pozycji. Jeśli n zostanie pominięte to odszukane będzie pierwsze wystąpienie znaku. Jeśli dany znak nie wystąpi w rozpatrywanym fragmencie linii kursor przesunięty zostanie do końca linii.

Przykłady

Przykład 1 /linia 30 ma swą początkową postać/
>EDIT 30
30–
Po naciśnięciu klawiszy 2 S X odnalezione zostanie drugie pojawienie się znaku X i komputer przejdzie do oczekiwania na nowe zlecenie edycji
30 IF X=1.5 THEN PRINT–

Przykład 2
Dla następującego położenia kursora w linii 30:
30 IF X=1.5 –
wykonanie 2 S X spowoduje przesunięcie kursora do końca linii co wskazuje, że "po drodze" komputer nie znalazł drugiego wystąpienia znaku X.
30 IF X=1.5 THEN PRINTX –

nKznak – "usuń aż do znaku"

Powoduje usunięcie zawartości z bufora linii od aktualnej pozycji kursora do n-tego pojawienia się "znaku" i przesunięcie kursora do tej pozycji /bez usunięcia odszukanego znaku/. Usunięte znaki zostają na ekranie umieszczone pomiędzy znakami wykrzyknika.

Przykład /linia 30 ma swą początkową postać/
Dla następującego położenia kursora w linii 30:
30 IF X=1–
Po przyciśnięciu 1 K (SP) nastąpi skasowanie tekstu do pierwszego wystąpienia znaku spacji tzn.
30 IF X=1!.5! –
Po przyciśnięciu w tej sytuacji dwukrotnie L /listowanie/ na ekranie pojawi się:
30 IF X=1 THEN PRINTX
30 –

Rozdział 7

PRAKTYCZNE UWAGI DOTYCZĄCE OPTYMALIZACJI PROGRAMÓW

7. Praktyczne uwagi dotyczące optymalizacji programów.

Celem tworzenia programów oszczędnie wykorzystujących pamięć komputera oraz charakteryzujących się krótkim czasem wykonania użytkownik powinien znać nieco bardziej szczegółowo niektóre mechanizmy wykonywania przez komputer programu w języku BASIC /w naszym przypadku w trybie interpretacyjnym/. Do problemu optymalizacji czasu wykonania programu oraz zajętości pamięci podchodzić należy elastycznie i zawsze należy mieć na uwadze przeznaczenie jakie spełniać winien tworzony program /np. powinien być szybki lub może być wolniejszy/. Najczęściej również postulaty minimalizacji czasu wykonania i zajętości pamięci programu stoją w sprzeczności z zasadami "dobrego programowania" wydłużając czas uruchamiania, późniejszego przerabiania programu itd. /ze względu na użycie tzw. "chwytów programowych" skracających program, czy np. ze względu na minimalizację komentarza i związaną z tym ubogą dokumentacją programu. Widać to wyraźnie właśnie na przykładzie użycia komentarzy /instrukcja REM/. W opisie instrukcji REM podkreślano wagę dobrego, bogatego komentowania programu, tu jednak należy ostrzec iż komentarze zwiększają istotnie pamięć zajętą przez program. Tak więc w przypadku interpretatora języka wyższego rzędu /np. BASIC/ pewnym wyjściem z sytuacji jest bogate komentowanie wersji roboczej programu, uruchamianej i poprawianej. W momencie gdy program jest uruchomiony i przetestowany komentarze mogą być usunięte i w takiej formie program może być eksploatowany. W dokumentacji oprogramowania winna znajdować się jednak wersja z komentarzem odpowiadająca aktualnej wersji eksploatowanej. Zupełnie inaczej przedstawiałaby się sytuacja gdyby BASIC pracował w trybie kompilacyjnym i komentarze nie znajdowałyby się w finalnej, przetłumaczonej postaci programu w pamięci.

Analogiczne uwagi dotyczą nadmiernego używania w programie /celem poprawienia jego czytelności/ spacji, rozbijania instrukcji /takich jak np. IF–THEN–ELSE/ na wiele linii na ekranie /linii dokumentacyjnych/, używania wcięć czy omijania umieszczenia wielu instrukcji w jednej linii /oddzielonych znakami ":"/. Używanie np. wielu instrukcji w jednej linii programu pogarsza znacznie czytelność dokumentacji programu, zwiększając jednak szybkość jego wykonania.

Duże oszczędności pamięci uzyskać można właściwie gospodarując typami zmiennych. Początkowo komputer przypisuje wszystkim niezadeklarowanym zmiennym typ pojedynczej precyzji. Przypisanie, jeśli to tylko możliwe /np. problem zakresu/, zmiennym typu całkowitego /%/ daje spore oszczędności.

Poniższa tabelka charakteryzuje typy zmiennych pod względem zakresu i zajętości pamięci.

Typ zmiennej	Zajętość pamięci	Zakres
całkowita	5 bajtów /2–wartość, 3–nazwa/	–32768 do +32767
pojedynczej precyzji	7 bajtów /4–wartość, 3–nazwa/	–1.701411E+38 do +1.701411E+38
podwójnej precyzji	11 bajtów /8–wartość, 3–nazwa/	–1.7014118345445556E+38 do +1.7014118345445556E+38
łańcuchowa	minimum 6 bajtów /3–nazwa, 3–znacznik stosu łańcucha, 1–każdy znak/	do 255 znaków
tablica	minimum 12 bajtów /3–nazwa, 2–rozmiar, 1–liczba wymiarów, 2–dla każdego wymiaru oraz 2, 3, 4 lub 8 dla każdego elementu zależnie od typu przechowywanych wartości/	

Pewne informacje na podstawie których można również optymalizować zajętość pamięci daje znajomość liczby bajtów wykorzystywanych przez komputer w celach organizacyjnych przy tworzeniu takich struktur programu jak:

– linia programu – 5 bajtów /2–numer linii, 2–wskaźnik linii, 1–znak CR/

– słowo zarezerwowane, operator, znaki specjalne, znaki stałe – 1 bajt

– aktywna pętla FOR–NEXT – 16 bajtów

– aktywne /jeszcze nie nastąpił powrót/ GOSUB – 6 bajtów

– każdy poziom nawiasów w wyrażeniu – 4 bajty + 12 bajtów na chwilowe zapamiętanie wyników pośrednich.

Tak więc użycie np. zamiast struktury GOSUB dwukrotnego GOTO /i x na powrót/ daje również zyski pamięci gdyż GOTO nie wymaga żadnych bajtów organizacyjnych. Minimalizacja liczby nawiasów daje poważne zyski /do 16 bajtów przy opuszczeniu nawiasów wewnątrz których znalazłoby się wyrażenie obliczone przed innymi i wynik pośredni musiałby być zapamiętany/.
Jeśli nie potrzeba aż 50 bajtów /automatycznie zarezerwowanych/ na wszystkie użyte w programie łańcuchy, to poprzez użycie CLEAR można zmniejszyć zakres zarezerwowanego obszaru. Podobna uwaga dotyczy tablic dla których komputer automatycznie rezerwuje 11 elementów dla każdego wymiaru tablicy. Jeśli nie jest to konieczne należy zmniejszyć rozmiary tablicy przez użycie instrukcji DIM.
Dla zmniejszenia zajętości pamięci poprzez ciągłe używanie deklaracji /=, %, $/ można jednorazowo użyć właściwych instrukcji DEF . . . . z których każda zajmie 6 bajtów, co przy dużej liczbie używanych znaków deklaracji staje się opłacalne.
Zmniejszenie czasu wykonywania programu uzyskać można m.in. poprzez:

– usunięcie komentarzy /REM/

– użycie wielu instrukcji w 1 linii programu

– użycie instrukcji POKE dla wyświetlania obrazów graficznych /sześciokrotne przyspieszenie/

– zdefiniowanie w pierwszej kolejności najczęściej używanych zmiennych /znajdą się wówczas na szczycie tablicy zmiennych przeszukiwanej od góry do dołu/.

Duże zyski czasowe uzyskuje się poprzez użycie /jeśli to jest możliwe/ zmiennych zamiast stałych w wyrażeniach. Znacznie szybciej bowiem pobierana jest zmienna niż trwa konwersja stałej na postać zmiennoprzecinkową /arytmetyka zmiennoprzecinkowa jest normalnym trybem przetwarzania wyrażeń/. Zamiast więc użyć w wyrażeniu liczby np. 1.234 można przypisać X=1.234 i użyć w wyrażeniu X. Największe zyski czasowe daje używanie zmiennych całkowitych w pętlach FOR–NEXT.

Lista błędów sygnalizowanych przez komputer

Kod błędu	Mnemonika	Znaczenie /w nawiasie skrócona nazwa ang./
1	NF	/NEXT without FOR/. Instrukcja NEXT użyta bez poprzedzającej ją instrukcji FOR.
2	SN	/Syntax Error/. Błąd składni /np. zła mnemonika instrukcji, niezamknięty nawias itd./
3	RG	/Return without GOSUB/. Instrukcja RETURN użyta bez poprzedzającej ją instrukcji GOSUB.
4	OD	/Out of Data/. Brak danych dla wykonania instrukcji READ /wyczerpana lista z instrukcji DATA/ lub instrukcji INPUT /brak danych na taśmie/.
5	FC	/Illegal Function Call/. Próba niedozwolonego wywołania funkcji /np. pierwiastek z liczby ujemnej/.
6	OV	/Overflow/. Wprowadzana lub wyprowadzana /obliczona/ wartość liczbowa za duża lub za mała.
7	OM	/Out of Memory/. Brak pamięci.
8	UL	/Undefined Line/. Skok lub odwołanie do nieistniejącej linii.
9	BS	/Bad Subscript/. Wskaźnik elementu tablicy /matrycy/ poza zadeklarowanym zakresem.
10	DD	/Double - Dimensioned Array/. Próba ponownego zadeklarowania /poprzez DIM/ uprzednio zadeklarowanej już tablicy.
11	/0	Dzielenie przez 0.
12	ID	/Illegal Direct/. Próba wykonania instrukcji INPUT jako zlecenia w trybie bezpośrednim.

13	TM	/Type Mismatch/. Skojarzenie zmiennej łańcuchowej z wartością niełańcuchową lub odwrotnie.
14	OS	/Out of String Space/. Wyczerpanie się obszaru pamięci zarezerwowanego na łańcuchy.
15	LS	/Long String/. Długość łańcucha przewyższa 255 znaków.
16	ST	/String Formula Too Complex/. Operacja na łańcuchu zbyt skomplikowana. Podziel operację na mniejsze kroki.
17	CN	/Can not Continue/. Brak możliwości dalszego kontynuowania programu /np. wykonanie zlecenia CONT gdy nie ma już żadnego programu do wykonania/.
18	NR	/No RESUME/. Brak RESUME.
19	RW	/RESUME without ERROR/. Instrukcja RESUME użyta bez poprzedzającej ją instrukcji ON ERROR GOTO.
20	UE	/Unprintable Error/. Parametr instrukcji ERROR nie odpowiada żadnemu z kodów z tej tablicy błędów.
21	MO	/Missing Operand/. Brak parametru.
22	FD	/Bad File Data/. Dane wprowadzane z taśmy były niewłaściwe lub wystąpiły w niewłaściwej kolejności.

Mapa pamięci systemu MERITUM-I
/ 16 kilobajtów /
/↑ ↓ kierunek narastania /

Adres dziesiętnie	Adres szesnastkowo	Przeznaczenie
0 12 287	0000 2FFF	EpROM /system operacyjny, rezydujący BASIC/
12 288 14 335	3000 37FF	Zarezerwowane
14 336 15 359	3800 3BFF	Pamięć RAM dla potrzeb wewnętrznych komputera
15 360 16 383	3C00 3FFF	Obszar pamięci roboczej /RAM/ monitora ekranowego
16 384 17 128	4000 42E8	Pamięć RAM dla potrzeb wewnętrznych komputera
17129 32767/16K/	42E9 7FFF/16K/	Tekst programu użytkownika ↓ Zmienne niełańcuchowe ↓ Zmienne tablicowe ↓ Wolna pamięć RAM ↑ Stos ↑ Obszar na łańcuchy ↑ Obszar zarezerwowany na programy pisane w języku wewnętrznym.

BASIC - MERITUM

ZESTAWIENIE INSTRUKCJI

Instrukcja	Działanie	Przykłady
I. ZLECENIA		
AUTOnumer linii, krok	Włącza automatyczną numerację linii programu od linii określonej poprzez "numer linii". Kolejne linie numerowane są narastająco liczbami różniącymi się o "krok".	AUTO AUTO 10 AUTO 5,5
CLEAR	Zeruje wszystkie zmienne; zmiennym łańcuchowym przypisuje łańcuchy puste.	CLEAR
CLEARn	Tak samo jak CLEAR ale rezerwuje dodatkowo n bajtów na łańcuchy tekstów.	CLEAR 500 CLEAR MEM/4
CONT	Wznawia dalsze wykonywanie programu zatrzymanego klawiszem BREAK lub po napotkaniu w programie instrukcji STOP.	CONT
DELETEnrlinii-nr linii	Usuwa z programu wszystkie linie od pierwszego podanego "nr linii" do drugiego podanego "nr linii" włącznie.	DELETE 100 DELETE 10-50 DELETE.
EDITnumer linii	Wprowadza komputer w tryb edycji /poprawiania/ linii o podanym "numerze linii"	EDIT 100 EDIT.
LISTnrlinii-nrlinii	Wyświetla na ekranie wszystkie linie programu począwszy od pierszego podanego "nr linii" do drugiego podanego "nr linii" włącznie	LIST 30 LIST 40-60 LIST 40- LIST-95 LIST.
NEW	Usuwa wszystkie linie programu aktualnie przechowywanego w pamięci oraz zeruje wszystkie zmienne /zmiennym łańcuchowym przypisuje łańcuchy puste/	NEW
RUNnrlinii	Rozpoczyna wykonywanie programu od podanego "nr linii" lub jeśli nie jest on wyspecyfikowany od linii o najniższym numerze	RUN RUN 105
SYSTEM	Wprowadza komputer w tryb systemowy w celu między innymi ładowania z kasety programów napisanych w języku wewnętrznym	SYSTEM
TROFF	Wyłącza funkcję śladowania	TROFF
TRON	Załącza funkcję śladowania	TRON

II. INSTRUKCJE WE/WY

PRINTlista elementów	Wyprowadza elementy wymienione w "liście elementów" na ekran. "Lista elementów" może zawierać: stałe, zmienne, łańcuchy lub modyfikatory / , – przecinek, ; – średnik, TAB – TABULACJA/	PRINTAS PRINTX+3
	, – przecinek powoduje przesunięcie kursora do następnego pola w linii / w linii są cztery szesnastoznakowe pola/	PRINT1, 2, 3, 4 PRINT1, 2
	; – powoduje zatrzymanie kursora na aktualnie wyprowadzonej pozycji	PRINT;Y; Z PRINT"ANSWER IS";
PRINT@pozycja, lista elementów	Wyprowadza na ekran elementy wymienione w "liście elementów" z określeniem "pozycji" od której wyprowadzanie ma się rozpocząć	PRINT@540, "ALA" PRINT@N+3, X–3
PRINTTAB (wyrażenie)	Przesuwa kursor w bieżącej linii do pozycji określonej "wyrażeniem"	PRINTTAB(N)N
PRINTUSINGłańcuch; lista elementów	Wprowadza elementy wymienione w "liście elementów" z określeniem formatu wyprowadzania określonego przez "łańcuch"	PRINTUSINGA$;X PRINTUSING "= . =";Y+Z
INPUT"komunikat"; zmienna	Wyprowadza na ekran "komunikat" /jeśli jest wyspecyfikowany/ oraz oczekuje na wprowadzenie danych z klawiatury	INPUT"NAZWISKO";AS INPUTA, B, C, DS
PRINT#–1, lista elementów	Zapisuje wartości zmiennych określonych "listą elementów" na kasetę magnetofonową	PRINT#–1, A, B, C, DS
INPUT#–1, lista elementów	Wczytuje określoną liczbę wartości przechowywanych na kasecie magnetofonowej i przyporządkowuje je do określonych nazw zmiennych wymienionych w "liście elementów".	INPUT#–1, A, B, C, DS
DATAlista elementów	Przechowuje wewnątrz programu dane /wg listy elementów/, które będą dostępne przy użyciu instrukcji READ	DATA 15, 18, 11, 1.2345 DATA"ELA","JAN"
READlista elementów	Czytanie przez komputer wartości określonych instrukcję DATA i przyporządkowanie tych wartości do określonych zmiennych wyszczególnionych w liście elementów	READB, B$, C, D
RESTORE	Powoduje, że następne wykonanie instrukcji READ rozpoczyna się od wczytania pierwszego elementu określonego instrukcją DATA	RESTORE

III.INSTRUKCJE DEFINIUJĄCE TYP ZMIENNEJ

DEFDBL /zakres literowy/	Definiuje zmienne określone "zakresem literowym" jako zmienne podwójnej prezycji	DEFDBLA, J–L, X
DEFINT /zakres literowy/	Definiuje zmienne określone "zakresem literowym" jako zmienne całkowite	DEFINTA, X, Y
DEFSNG /zakres literowy/	Definiuje zmienne określone "zakresem literowym" jako zmienne pojedynczej precyzji	DEFSNGC
DEFSTR /zakres literowy/	Definiuje zmienne określone "zakresem literowym" jako zmienne łańcuchowe	DEFSTRM
DIMnazwa (dim1, ... dimk)	Deklaruje rozmiary tablicy /w każdym wymiarze/ o podanej "nazwie" oraz rezerwuje niezbędną dla tablicy pamięć	DIMA(2, 3) DIMB(X+2), C (J, K) DIMT(3, 3, 5)
CLEARn	Rezerwuje n bajtów pamięci dla zapamiętania łańcuchów	

IV.INSTRUKCJE

LETzmienna=wyrażenie	Przypisuje zmiennej wartość określoną "wyrażeniem". Instrukcja ta jest w tej wersji języka BASIC opcjonalna	LETA$="HANS" LETB1=C1 LETA%=3
END	Kończy wykonywanie programu. Instrukcja ta jest w tej wersji języka BASIC opcjonalna	99END
STOP	Przerywa wykonywanie programu i wyprowadza komunikat BREAK INnumer linii. Wznowienie wykonywania programu następuje po wprowadzeniu zlecenia CONT	105STOP
GOTO"numer linii"	Powoduje wykonanie skoku bezwarunkowego do linii programu określonej "numerem linii"	GOTO3000
GOSUB"numer linii"	Powoduje wykonanie skoku bezwarunkowego do podprogramu rozpoczynającego się w linii określonej "numerem linii"	GOSUB5
RETURN	Kończy podprogram i powoduje wykonanie skoku bezwarunkowego do linii programu znajdującej się po ostatniej wykonanej instrukcji GOSUB	RETURN
ONnGOTOn1, n2, ... nk	Przekazuje sterowanie w programie do tej linii z n1 . . . nk, której pozycja w liście określana jest przez część całkowitą n [INT(n)] ; n – może być wyrażeniem	ONX1+1GOTO5, 100, 120, 30

ONnGOSUBn1, n2, . .nk	Identycznie jak dla instrukcji ON n GOTOn1, n2 . . . nk przy czym numery linii n1 . . . nk wskazują linie początkujące podprogramy n – może być wyrażeniem	ONJGOSUB5, 13, 3000
FORzmienna=wyrażenie 1TOwyrażenie2STEPkrok	Rozpoczyna /otwiera/ pętle typu FOR–NEXT. STEPjest opcjonalny: jeśli ominięto to zmienna będzie zwiększona o 1.	FORI=1TO50STEP3
NEXTzmienna	Kończy /zamyka/ pętlę typu FOR–NEXT	NEXT NEXTI NEXTI, J, K
ERROR(kod)	Powoduje identyczną reakcję komputera jak na błąd o określonym kodzie /symuluje błąd/	ERROR(14)
ON ERROR GOTO numer linii	Jeśli pojawi się błąd w programie /po wystąpieniu tej instrukcji/ to spowoduje on skok do programu jego obsługi rozpoczynającego się w linii określonej "numerem linii"	ON ERROR GOTO 9999
RESUMEn	Kończy program obsługi błędu powodując skok do linii n lub: – do linii zawierającej błąd gdy n=0 lub nieokreślone – do linii następnej po tej która zawierała błąd gdy n=NEXT	RESUME RESUMEO RESUME500 RESUMENEXT
RANDOM	Uruchamia generator liczb pseudolosowych	RANDOM
REM	Powoduje ignorowanie pozostałej linii programu	REAM ALA TO KOT
IFwyrażenieTHENakcja1 ELSEakcja2	Powoduje przetestowanie wyrażenia /logicznego lub relacji/ i przekazanie sterowania w programie do części "akcji 1" gdy jest ono prawdziwe. Wyrażenie jest logicznie prawdziwe jeśli przyjmuje wartość niezerową.	IF A=0 THEN PRINT "ZERO" ELSE PRINT "TO NIE JEST ZERO"

V. INSTRUKCJE SPECJALNĘ

CLS	Zeruje ekran i przesuwa kursor do lewego górnego rogu ekranu	CLS
RESET(X, Y)	Zeruje /gasi/ punkt graficzny o współrzędnych X, Y X – poziom $0 \leqslant X < 128$ Y – pion $0 \leqslant Y < 48$	RESET(b+5, 13)
SET(X, Y)	Wpisuje /zapala/ punkt graficzny o współrzędnych X, Y X – poziom $0 \leqslant X < 128$ Y – pion $0 \leqslant Y < 48$	SET(C+X, 7)

POKEadres, wartość	Zapamiętuje wartość z zakresu 0–255 w pamięci pod adresem określonym w argumencie. Oba argumenty muszą być dziesiętne.	POKE15635, 34 POKE17770, B+C
PEEK(adres)	Odczytuje zawartość pamięci o podanym dziesiętnie w argumencie adresie i przyporządkowuje ją wyrażeniu PEEK	PEEK(15635)
OUTport, wartość	Wysyła bajt danych o określonej w argumencie "wartości" do podanego "portu". Argument "port" musi przyjmować wartość z zakresu 0–255	OUT100, A
ERL	Wyznacza numer linii w której wystąpił błąd i przypisuje go zmiennej o nazwie ERL	ERL
ERR	Wyznacza kod błędu /jeśli wystąpił/ i przypisuje go zmiennej ERR wg zasady ERR = /kod błędu – 1/*2	ERR/2+1
INP(port)	Wczytuje bajt danych z określonego argumentem "portu" i przypisuje jego wartość zmiennej INP	INP(33)
MEM	Wyznacza liczbę wolnych i niechronionych bajtów pamięci i przypisuje ją zmiennej o nazwie MEM	MEM
POINT(X, Y)	Testuje "punkt graficzny" o współrzędnych X, Y. Jeśli punkt jest wyświetlany to wyrażenie POINT (X, Y) przyjmuje wartość 1, gdy zgaszony wartość 0. $0 \leqslant X < 128$ $0 \leqslant Y < 48$	POINT(4, 17)
POS(0)	Przypisuje wyrażeniu POS(0) wartość z zakresu 0–63 określającą bieżącą pozycję kursora w linii ekranu	POS(0)
USR(X)	Wywołuje program /mający charakter procedury/ napisany w języku wewnętrznym procesora Z-80. Parametr X zostanie przekazany do procedury jako liczba całkowita dwubajtowa	USR(0)
VARPTR(zmienna)	Wyznacza wartość adresów w pamięci pod którymi umieszczone są wartości zmiennych o określonych nazwach	VARPTR(B$) VARPTR(X3)

VI. INSTRUKCJE OPERACJI NA ŁAŃCUCHACH

CINT(wyrażenie)	Oblicza największą liczbę całkowitą nie większą od wartości "wyrażenia"
COS(wyrażenie)	Oblicza cosinus "wyrażenia" podanego w radianach
CSNG(wyrażenie)	Wyznacza odpowiednik pojedynczej precyzji z "wyrażenia" będącego liczbą podwójnej precyzji
EXP(wyrażenie)	Podnosi stałą e do potęgi "wyrażenie"
FIX(wyrażenie)	Wyznacza część całkowitą "wyrażenia"
INT(wyrażenie)	Wyznacza największą liczbę całkowitą nie większą od "wyrażenia"
LOG(wyrażenie)	Oblicza logarytm naturalny z wartości "wyrażenia"
RND(0)	Oblicza liczbę pseudolosową z przedziału 0.000001 do 0.999999
RND(wyrażenie)	Oblicza liczbę pseudolosową z przedziału 1 do INT(wyrażenie)
SGN(wyrażenie)	Przyjmuje wartość: −1 gdy wyrażenie jest ujemne 0 gdy wyrażenie jest równe zero +1 gdy wyrażenie jest dodatnie
SIN(wyrażenie)	Oblicza sinus z wartości "wyrażenia" podanego w radianych
SQR(wyrażenie)	Oblicza pierwiastek kwadratowy z wartości "wyrażenia", które musi być nieujemne
TAN(wyrażenie)	Oblicza tangens z wartości "wyrażenia" podanego w radianach Uwaga: "wyrażenie" musi być stałą lub zmienną

VIII. TRYB EDYCJI ZLECENIA OPERATORSKIE

(ENTER)	Kończy edycję linii i powoduje powrót do trybu bezpośredniego
(SHIFT)(↑)	Kończy wykonywanie zlecenia edycyjnego – komputer pozostaje w trybie edycji. Zlecenie to nosi nazwę sekwencji ESCape

nSPACE	Przesuwa kursor o n pozycji w prawo
n←	Przesuwa kursor o n pozycji w lewo
L	Wyświetla pozostałą część edytowanej linii oraz ustawia kursor na pierwszej pozycji tej linii
X	Wyświetla pozostałą część edytowanej linii, ustawia kursor na końcu linii oraz rozpoczyna tryb wstawiania dodatkowych znaków
I	Rozpoczyna tryb wstawiania znaków, które są naciskane kolejno. Dla wyjścia z tego trybu -- naciśnij sekwencję ESCape
A	Likwiduje wprowadzone w danej linii zmiany i ustawia kursor na pierwszej pozycji tej linii
E	Kończy edycję, zapamiętuje zmienną linię i powoduje powrót do trybu bezpośredniego
Q	Kończy edycję, kasuje wszystkie wprowadzone w linii zmiany oraz powoduje powrót do trybu bezpośredniego
H	Kasuje linię od pozycji kursora do końca oraz rozpoczyna tryb wstawiania. Dla wyjścia z tego trybu -- naciśnij sekwencję ESCape
nD	Kasuje n znaków od bieżącej pozycji kursora
nC	Pozwala na zmianę n kolejnych znaków
nSc	Przesuwa kursor do miejsca w którym znak c pojawi się n-ty raz licząc od bieżącej pozycji kursora
nKc	Kasuje wszystkie znaki od bieżącej pozycji kursora do miejsca w którym znak c pojawi się n-ty raz

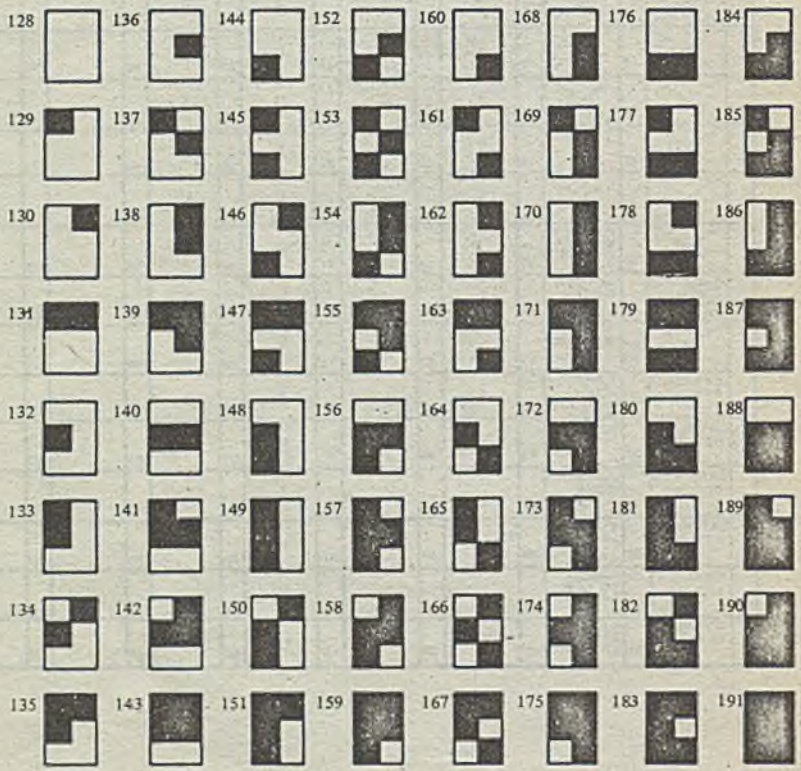
Kody sterujące, kody znaków ASCII i kody znaków graficznych

Tablica kodów sterujących i znaków ASCII

0		16		32	SPACJA	48	0	64	@	80	P	96		112	
1		17		33	!	49	1	65	A	81	Q	97		113	
2		18		34	"	50	2	66	B	82	R	98		114	
3		19		35	#	51	3	67	C	83	S	99		115	
4		20		36	$	52	4	68	D	84	T	100		116	
5		21		37	%	53	5	69	E	85	U	101		117	
6		22		38	&	54	6	70	F	86	V	102		118	
7		23	WŁĄCZ. FORMAT 32 ZNAKI W LINII	39	'	55	7	71	G	87	W	103		119	
8	BACK SPACE I KASUJ ZNAK	24	KURSOR W LEWO ←	40	(	56	8	72	H	88	X	104		120	
9		25	KURSOR W PRAWO →	41	)	57	9	73	I	89	Y	105		121	
10	CARRIAGE RETURN	26	KURSOR W DÓŁ LINE FEED	42	*	58	:	74	J	90	Z	106		122	
11	CARRIAGE RETURN	27	KURSOR W GÓRE ↑	43	+	59	;	75	K	91	↑ LUB [	107		123	
12	CARRIAGE RETURN	28	KURSOR HOME	44	,	60	<	76	L	92		108		124	
13	CARRIAGE RETURN	29	KURSOR DO POCZĄTKU LINII	45	—	61	=	77	M	93		109		125	
14	WŁĄCZ. KURSOR	30	KASUJ DO KOŃCA LINII	46	.	62	>	78	N	94		110		126	
15	WYŁĄCZ. KURSOR	31	KASUJ DO KOŃCA LINII	47	/	63	?	79	O	95		111		127	

Kody 96 – 127 zarezerwowane na małe litery – w tej wersji MERITUM nie są wykorzystywane.

Tablica kodów znaków graficznych: 128 – 191



ZAKŁADY URZĄDZEŃ KOMPUTEROWYCH
M E R A E L Z A B
ul. Kruczkowskiego 39
41-808 Z A B R Z E

BASIC - MERITUM

PODRĘCZNIK PROGRAMOWANIA I UŻYTKOWANIA

44990001