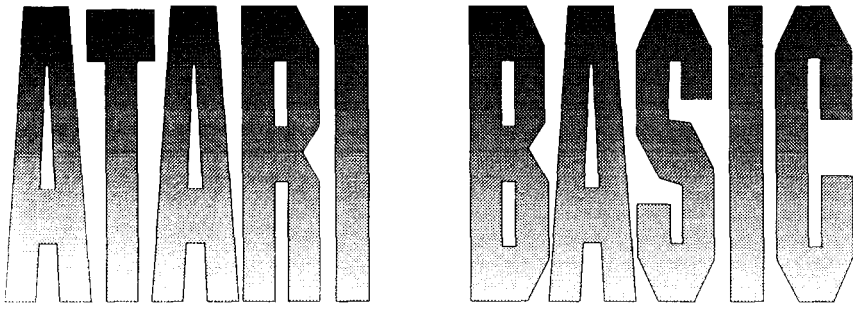
Naczelna
Organizacja
Techniczna
OŚRODEK
DOSKONALENIA
KADR
TECHNICZNYCH
Rady
Stołecznej

ATARI BASIC

JĘZYK PROGRAMOWANIA I OBSŁUGA MIKROKOMPUTERA ATARI

WARSZAWA
1987

Naczelna Organizacja Techniczna
OŚRODEK DOSKONALENIA KADR TECHNICZNYCH
Rady Stołecznej



JĘZYK PROGRAMOWANIA I OBSŁUGA MIKROKOMPUTERA ATARI

Krzysztof Bednarek
Mariusz Giergiel
Wojciech Jedliczka
Tadeusz Kowalek
Andrzej Macioł
Wiesław Migut
Andrzej Turmiński

WARSZAWA 1987

SPIS TREŚCI

Rozdział 1

WPROWADZENIE

W ostatnim okresie nastąpił gwałtowny wzrost popularności mikrokomputerów ATARI. Również w Polsce liczba użytkowników tego mikrokomputera zwiększyła się znacznie.

Zjawisku temu towarzyszy duże zapotrzebowanie na opracowania z zakresu programowania i obsługi. Fakt ten skłonił nas do zebrania naszych wiadomości i doświadczeń, które uzupełnione przekładami z literatury zachodniej stanowią treść tej książki. Wykaz wykorzystanej przez nas literatury został zamieszczony na str. 20.

Naszym podstawowym zamiarem było przygotowanie podręcznika, który mógłby służyć użytkownikom ATARI - w opanowywaniu tajników programowania i obsługi. Stąd też książka ta nie powinna być zbyt trudna dla początkujących, a niektóre jej rozdziały mogą zadowolić bardziej zaawansowanych programistów. Jeżeli cel ten udało nam się osiągnąć w niewielkim chociaż stopniu, możemy czuć się usatysfakcjonowani.. Nie piszemy tej książki dla zawodowych informatyków, ani dla językoznawców, choć w miarę możności będziemy się starać zadowolić (lub przynajmniej nie drażnić) jednych i drugich. Większość dostępnej nam literatury jest napisana w języku angielskim, którego proste i zwięzłe

terminy nie dają się łatwo przetłumaczyć na język polski. Dosłowne tłumaczenie jest bowiem, w niektórych przypadkach nienaturalne, a opisowe - długie i niewygodne. I tak na przykład nie będziemy rozróżniać rodzajów instrukcji, mimo różnych angielskich znaczeń statement i command. Jako uzupełnienie zaprezentowanych w niniejszej książce wiadomości i informacji polecamy lekturę następujących czasopism:

Informatyka, Politechnik, Młody Technik, Problemy, Audio Video, Radioelektronik, Bajtek.

1.1 Zastosowanie mikrokomputerów

Obliczenia matematyczne były potrzebą-matką wynalazku jakim jest mikrokomputer. W miarę usprawniania komputerów pojawiły się inne możliwości ich wykorzystania, co doprowadziło do znacznego przeobrażenia funkcji i wzrostu znaczenia komputerów w codziennym życiu.

Informacje o zastosowaniu mikrokomputerów do obliczeń naukowych i biurowych, oraz do przetwarzania tekstów, znaleźć można w podręczniku A. Macioł, W. Migut, A. Stawowy "BASIC - wprowadzenie do programowania komputerów domowych". Podano tam również krótką charakterystykę gier komputerowych.

Naturalną koleją rzeczy komputery znalazły szerokie zastosowanie w nauczaniu. Ujmując zagadnienie w najszerszym zakresie uważamy, że dzisiaj w polskich warunkach każdy komputer spełnia rolę środka edukacji, choćby przez to, że umożliwia dzieciom, a także dorosłym kontakt z nowoczesnym urządzeniem elektronicznym. Komputer jest bardzo cierpliwym nauczycielem, uczy logiki i konsekwencji myślenia, pobudza wyobraźnię, pokazuje różnorodne możliwości osiągania zróżnicowanych celów badawczych i popularnonaukowych.

ATARI800 XI. ma wszelkie cechy pozwalające na szerokie zastosowanie go w polskich szkołach (tak dzieje się już w Anglii, Holandii, Danii i Francji). Jest to tym bardziej możliwe, że PEWEX bardzo trafnie zdecydował się na wprowadzenie sprzedaży mikrokomputera ATARI.

Niebagatelne znaczenie może mieć też polskie pochodzenie właściciela firmy ATARI Jacka Trzmiela (obecnie Jack Tramiel), który jest rodowitym Warszawiakiem i uchodzi w tej chwili za najwybitniejszą postać w przemyśle mikrokomputerowym.

Nie jest możliwe udzielenie wyczerpującej odpowiedzi na pytanie o zastosowanie mikrokomputera. Warto jednak przytoczyć kilka przykładów mało znanych prób wykorzystania bogatych możliwości ATARI.

- Przetworniki analogowe-cyfrowe umożliwiają przyłączenie 4 czytników (np. termometrów) i rejestrację danych z doświadczeń.
- Komputer nuże spełniać rolę sterowania światłami w teatrze i salach widowiskowych.
- Szczególnie ciekawe są próby wykorzystania w krótkofalarstwie: koder i dekoder alfabetu Morse'a, komputerowy dalekopis RTTY oraz transmisje wizji SSTV.
- Automatyczna sekretarka przyjmująca telefony i wybierająca bez znudzenia wciąż zajęte numery.
- Archiwowanie poufnych dokumentów z możliwością ich odczytania tylko przy pomocy kodu.
- Komputer może być użyty do sterowania pracą centralnego ogrzewania oraz systemów alarmowych w domku jednorodzinnym.

Z polskich publikacji poruszających omawianą przez nas problematykę na wyróżnienie zasługują:

Z. Czech, K. Nałęcki, S. Wołek, "Programowanie w języku BASIC",

W. Iszkowski, M. Maniecki "Programowanie w języku BASIC"

A. Macioł, W. Migut, A. Stawowy "BASIC - wprowadzenie do programowania komputerów domowych"
K. Orlicz "Programowanie w języku konwersacyjnym BASIC"

1.2. Dlaczego ATARI?

Decydując się na zakup mikrokomputera powinniśmy zdawać sobie sprawę z ważnego faktu, że jego koszt stanowi około połowę wszystkich wydatków jakie musimy ponieść chcąc w pełni korzystać z jego możliwości. Tyle samo, a często znacznie więcej kosztuje oprogramowanie. Jest jednak sposób na częściowe ominięcie ogromnych w naszych warunkach wydatków. Możemy bowiem wymieniać programy z innymi użytkownikami tego samego typu komputera. Powinniśmy więc zdecydować się na urządzenie popularne w Polsce. Aktualnie są trzy możliwości:

- rodzina ATARI: 400, 600, 600XL, 800XL, 65XE, 130XE
- Commodore C64
- Sinclair Spectrum albo Spectrum Plus

Z innych popularnych w kraju mikrokomputerów ZX81 nawet w polskich warunkach jest już tylko zabawką. TI99/4A stał się pamiątką błędów marketingowych, a nieudane bliźniaki Schneider i Amstrad utrzymywane są przy życiu za pomocą subwencji rządowych. Nowe modele firmy Commodore, w których wyeliminowano wiele wad poprzedniej generacji, m.in. zastosowano znacznie lepszy interpreter języka BASIC, są niekompatybilne z C64.

Porównania komputerów ATARI 800 XL, C64 i ZX Spectrum 48KB dokonamy na podstawie własnych doświadczeń oraz testów przeprowadzonych przez następujące pisma:

1. Computers and Programs" - II 1984
2. Mein Home-Computer - XII 1983 (pismo to stosuje skalę ocen 0-10)
3. Computers '84,
4. Creative Computing- I 1984

W zasadzie wszystkie trzy komputery można ocenić na tym samym poziomie - żaden z nich nie został skonstruowany do profesjonalnych zastosowań. Istnieją jednak pewne różnice - co wykazuje test na szybkość obliczeń, dokładność numeryczną i losową. Ocena dotyczy języka BASIC - resident. Szybkość obliczeń podobna jest dla 800XL i C64 (znacznie mniejsza dla Spectrum, który jest nieco dokładniejszy od rywali), natomiast ocena generatora liczb losowych wypadła zdecydowanie najgorzej dla C64. Zakres liczbowy jest największy dla ATARI - od 10E - 99 do 10E98 wobec 10E-39 do 10E38 dla Spectrum i C64. Spectrum nie dotrzymuje kroku ani ATARI ani COMMODORE.

Dokonując oceny i porównań komputerów pod względem ich użyteczności do gier telewizyjnych, należy rozpatrzyć dwa aspekty: hardware i software. Szerokie możliwości dźwiękowe cechują ATARI i COMMODORE, najgorsze możliwości posiada SPECTRUM. Wg punktacji dziesięciostopniowej "Mein Home-Computer" ocenia kolejno: kolory, dźwięk, grafikę, ilość programów-gier.

ATAR1 - 10, 10, 10, 10
COMMODORE - 7, 7, 7, 10
SPECTRUM - 5, 4, 8, 9

Dużą wadą SPECTRUM jest brak złącz dla manipulatorów, zaś używanie wyłącznie klawiatury zmniejsza atrakcyjność wielu gier. Istnieje możliwość dokupienia takich złączy w cenie ok. 20-30°l0 wartości komputera, co trzeba wziąć pod uwagę kalkulując cenę przyszłego zakupu. Uwzględniając polskie warunki należy zweryfikować punktację za powszechność oprogramowania albowiem inaczej przedstawia się ona w Polsce. Zdecydowanie pierwsze miejsce zajmuje tu SPECTRUM.

Wysoka pozycja ATARI w ocenie niemieckich czasopism bierze się stąd, że firma ta jako pierwsza w świecie zaczęła produkować mikroprocesorowe gry telewizyjne i do dzisiaj zajmuje czołową pozycję w tej dziedzinie. W produkowanych mikrokomputerach stosuje się wiele własnych patentów, m.in. dwa specjalistyczne mikroprocesory, odpowiedzialne za generowanie obrazu i dźwięku.

Ostatnim kryterium porównawczym będzie zdolność komputerów do przetwarzania tekstów. SPECTRUM zupełnie nie nadaje się do tego rodzaju działania. ATARI 800 XL jest komputerem domowym i jako taki nie nadaje się do zastosowania profesjonalnego. Wyświetla tylko 40 znaków w wierszu co jest poważną wadą. Wprawdzie istnieje możliwość przyłączenia dodatkowego urządzenia likwidującego tę wadę, ale zwiększa to koszty systemu. Zaletą ATARI jest bardzo dobrze zaprojektowana klawiatura i możliwość bezpośredniego użycia liter spoza alfabetu angielskiego. Znacznym ułatwieniem w pisaniu jest krótki dźwięk dochodzący z głośnika w momencie, gdy dana litera jest zaakceptowana przez komputer po naciśnięciu klawisza. Z pewnością 800 XL jest wystarczającym komputerem do przetwarzania tekstów w zakresie potrzebnym w domu lub małym biurze.

COMMODORE 64 również nie wyświetla więcej niż 40 znaków w wierszu. Pełnowymiarowe dokumenty uzyskuje się dzięki oprogramowaniu. C64 posiada porównywalną jakościowo z ATARI klawiaturę i z całą pewnością można go wykorzystać do nieprofesjonalnego przetwarzania tekstów.

Kończąc te rozważania należy wspomnieć o niebagatelnej sprawie jaką jest cena mikrokomputera. Często bywa ona decydującym czynnikiem wpływającym na zakup określonego typu komputera. Aktualne ceny ATARI 800XI. i COMMODORE 64 są identyczne. Tańszy jest SPECTRUM, ale zważywszy mniejszą pamięć, konieczność dokupienia co najmniej jednego interface'u do manipulatora, czyni go najdroższym z omawianych przez nas komputerów.

Niektórzy być może zadają sobie pytanie, czy kupić mikrokomputer dzisiaj, czy poczekać na nowe, lepsze wersje. Oczywiście, że za miesiąc czy rok będą nowe komputery. Jeżeli jednak będziemy czekać tylko dlatego, że następuje szybki rozwój mikroinformatyki, będziemy czekali do końca życia.

Rozdział 2

CO TO JEST MIKROKOMPUTER

2.1. Budowa mikrokomputera.

Komputer jest urządzeniem elektronicznym służącym do przetwarzania danych. Pekiny cykl przetwarzania danych obejmuje wprowadzenie informacji, przetworzenie ich zgodnie z wymaganiami użytkownika oraz wyprowadzenie informacji wynikowych. Cele jakim służy komputer (maszyna cyfrowa) determinującego budowę. Maszyna cyfrowa składa się z:

- urządzeń wejściowych, - jednostki centralnej,
- pamięci zewnętrznej,
- urządzeń dialogowych, - urządzeń wyjściowych.

Wszystkie te elementy wchodzą w skład zarówno "dużych" maszyn cyfrowych jak i mini- i mikrokomputerów.

Urządzenia wejściowe realizują zadanie wprowadzania informacji do komputera. Problem wprowadzenia danych polega na kodowaniu danych czytelnych dla człowieka (zapisanych przy pomocy liter, cyfr, znaków graficznych) w języku maszyny cyfrowej (sygnały elektryczne).

W komputerach profesjonalnych stosuje się do tego celu czytniki taśmy papierowej czytniki kart perforowanych, urządzenia klawiaturowe, czytniki pisma, czytniki rysunków (ang., digitizer) itp. W mikrokomputerach stosowane są do tego celu zasadniczo wyłącznie urządzenia klawiaturowe. Zaletą rozwiązania jest uniknięcie konieczności stosowania tzw. maszynowych nośników danych (karty perforowane, taśma papierowa itp. Znacznym ułatwieniem przy wprowadzaniu informacji, jest stosowanie w mikrokomputerach urządzeń niekonwencjonalnych takich jak manipulatory (ang. Joystick) czy pióra świetlne. Mogą być one wykorzystane do wprowadzania danych, które trudno jest zapisać przy pomocy liter i cyfr (np. rysunki) oraz do dialogu z komputerem. Mikrokomputery mogą być dostosowane do współpracy z klasycznymi urządzeniami wprowadzania danych (np. czytnik taśmy papierowej). Rozwiązania takie stosuje się jednak stosunkowo rzadko.

Jednostka centralna jest podstawową częścią składową każdego komputera. Składa się z jednostki sterującej, arytmometru i pamięci operacyjnej. Cechą odróżniającą mikrokomputery od innych tego typu urządzeń jest fakt, że jednostka sterująca oraz arytmometr wykonane są w postaci jednego układu wysokiej skali integracji tzw. mikroprocesora. Mikroprocesor steruje wszystkimi operacjami realizowanymi przez mikrokomputer oraz dokonuje operacji arytmetycznych. W pamięci mikrokomputera rozróżnia się: pamięci typu RAM (Random Access Memory) i pamięci ROM (ang. Read Only Memory). Pamięć typu RAM realizowana w formie układów scalonych służy do przechowywania informacji aktualnie przetwarzanych przez komputer. Dane w komputerze reprezentowane są przez sygnały elektryczne. Stosuje się binarny sposób kodowania danych. Podstawowy element pamięci (tzw. bit) może przyjmować wartość 1 lub 0. Takie binarne elementy zorganizowane
są w grupy (tzw. bajty). Każdy bit w bajcie ma określoną wagę zgodną z formułą zapisu dwójkowego. Bajt składa się z ośmiu bitów. W każdym bajcie możemy zapisać 256 różnych znaków (liczb, liter, znaków graficznych itp.). Przedstawimy teraz dwa przykłady bajtów zawierających informacje:

1.
11000011
76543210
Zgodnie z regułami zapisu dwójkowego odczytujemy wartości tego bajta jako:
$1 * 2^0 + 1 * 2^1 + 0 \ 2^2 + 0 * 2^3 + 0 * 2^4 + 0 * 2^5 + 1 * 2^6 + 1 * 2^7 = 195$

2.
00110101
76543210
wartość bajta wynosi:
$1 * 2^0 + 0 * 2^1 + 1 \ 2^2 + 0 * 2^3 + 1 * 2^4 + 1 * 2^5 + 0 * 2^6 + 0 * 2^7 = 53$

W obydwu przykładach gwiazdka (*) oznacza znak mnożenia.

W jednym bajcie możemy zapisać liczby z zakresu 0 do 255. Komputer musi jednak pamiętać także większe liczby oraz litery i znaki graficzne.

Do zapisu liczb większych konieczne jest stosowanie większej liczby bajtów (do sześciu). Komputer wyraźnie rozróżnia liczby całkowite od rzeczywistych. Liczby całkowite zapamiętywane są wprost jako ciąg cyfr natomiast liczby rzeczywiste zapamiętywane są w postaci wykładniczej (osobno mantysa i osobno wykładnik). Wszelkie znaki tj. litery, cyfry, znaki graficzne i specjalne są przed "zapamiętaniem" kodowane. Do tego celu służy najczęściej kod ASCII (od American Standard Code of Information Interchange), a w komputerze ATARI jego zmodyfikowana wersja ATASCII. Kod ten przyporządkowuje każdemu znakowi liczbę z przedziału 0 do 255. Dzięki temu do zapisania jednego znaku wystarcza jeden bajt.

Każdy bajt pamięci ma określony adres. Przy pomocy tego adresu mikroprocesor może odszukać zapisane informacje. Ilość bajtów określa pojemność pamięci komputera. Do określenia tej wielkości używa się jednostki "kilobajt" (KB). 1 KB to 2^{10} czyli dokładnie 1024 bajty. Pamięć RAM komputera ATARI 800 XL ma pojemność 64 KB tzn., że zawiera 65536 bajtów. Pojemność pamięci zależy od ilości i rodzaju zastosowanych układów scalonych.

W pamięci typu RAM mikroprocesor może zapisywać informacje oraz odczytywać je. Po wyłączeniu zasilania zawartość pamięci zostaje skasowana (wszystkie bity przyjmują wartość 0). Ponieważ oprogramowanie podstawowe komputera musi być na stałe zapisane w pamięci, część jej obszaru zrealizowana jest w postaci tzw. pamięci ROM. Struktura jej i sposób adresowania jest identyczna jak w pamięci typu RAM. Różnica polega na tym, że zawartość poszczególnych bajtów jest stała. Mikroprocesor nie może zapisywać informacji w pamięci ROM, a jedynie odczytywać je. Wyłączenie zasilania nie powoduje skasowania informacji zawartych w modułach ROM. Z modułów typu ROM i RAM tworzony jest tzw. roboczy obszar pamięci o wymiarach zależnych od możliwości adresowania mikroprocesora. Należy sobie zdawać sprawę, że nie cały roboczy obszar pamięci może być dowolnie wykorzystany przez użytkownika. Część tego obszaru stanowią moduły ROM, a część obszaru RAM zarezerwowana jest dla programów sterujących (system operacyjny). W komputerze ATARI z 64 KB pamięci, dla użytkownika dostępne jest nieco mniej niż 40 KB obszaru roboczego. Struktura pamięci może być zmieniana poprzez dołączenie zewnętrznego modułu ROM (w postaci tzw. cartridge'a) czy wpisywanie innych systemów operacyjnych (np. dyskowych).

Pamięć zewnętrzna mikrokomputera służy do przechowywania programów i zbiorów danych. Bez tych urządzeń nie jest możliwe praktyczne wykorzystanie możliwości komputera. Programy i dane zapisane w pamięci operacyjnej są kasowane po wyłączeniu komputera i ponowne ich wykorzystanie wymagałoby ich wprowadzenia poprzez urządzenia wejścia. O wiele bardziej efektywne jest zapisanie ich w pamięci zewnętrznej, z której można je w dowolnym momencie wprowadzić do pamięci operacyjnej. W mikrokomputerach wykorzystuje się jako urządzenia pamięci zewnętrznej magnetofony oraz stacje dysków. Nośnikiem informacji może być taśma magnetyczna, dysk elastyczny czy dysk sztywny. Opis urządzeń pamięci zewnętrznej znaleźć można w rozdziale 7.

Podstawowym urządzeniem dialogowym mikrokomputera jest klawiatura alfanumeryczna i monitor ekranowy. Klawiatura służy, jak już zostało powiedziane, do wprowadzania danych i programów oraz do przekazywania komputerowi poleceń. Komputer przekazuje informacje o przebiegu pracy przy pomocy komunikatów pojawiających się na ekranie monitora ekranowego. Ułatwieniem w dialogu człowiek - komputer są różnego rodzaju urządzenia pomocnicze (manipulator, mysz, pióro świetlne) oraz wyposażenie klawiatury w klawisze specjalne (do sterowania kursorem, przerywania biegu programów itd.).

Wyniki działania komputera muszą być przedstawione w formie czytelnej dla użytkownika. Zadanie to realizują urządzenia wyjściowe. W podstawowym zestawie mikrokomputera jest nim monitor ekranowy. Ponieważ jednak zapis na monitorze nie jest trwały i nie może stanowić dokumentu w "poważniejszych" zastosowaniach konieczne jest stosowanie drukarek ewentualnie urządzeń rysujących (tzw. ploterów). W pewnych specyficznych zastosowaniach urządzeniem wyjścia może być perforator taśmy papierowej, łącze teletransmisyjne. Do współpracy z mikrokomputerem stosuje się szereg różnego rodzaju drukarek różniących się sposobem przesuwu papieru, rodzajem stosowanych głowic, liczbą dostępnych znaków.

2.2. Oprogramowanie

Oprogramowaniem nazywamy zbiór programów, a więc opisów sposobu działania maszyny zapisanych w formie „czytelnej” dla komputera. Bez oprogramowania komputer nie może zrealizować żadnej z przydzielonych mu funkcji., Programy realizujące funkcje sterowania modułami komputera oraz umożliwiające kontakt z otoczeniem zaliczamy do tzw. oprogramowania podstawowego (systemowego). Programy te zapisane są w modułach pamięci ROM. Włączenie zasilania powoduje uruchomienie programów sterujących.

Użytkownik może wybrać sposób działania tych programów naciskając w momencie włączenia jeden z klawiszy funkcyjnych (START lub OPTION). Zgodnie z przyjętym schematem „zagospodarowana” jest pamięć wewnętrzna maszyny. Program „włącza” do obszaru pamięci roboczej odpowiednie moduły RAM i ROM. Po przetestowaniu komputera wybranym komórkom pamięci przydzielone są odpowiednie wartości. Proces ten nazywa się inicjacją systemu operacyjnego. Program przydziela obszary pamięci, które stanowią "bufory" dla urządzeń zewnętrznych. W buforach zapisywane są dane, które przekazywane są z lub do komputera z takich urządzeń jak klawiatura, monitor, pamięć zewnętrzna, drukarka itp. Istnieją specjalne programy sterujące komunikacją komputera z otoczeniem. Program sterujący określa również podstawowe „zasady” działania komputera (np. liczba linii na ekranie monitora,. kolor liter i tła, miejsce przechowywania zakodowanych znaków graficznych itp.) poprzez nadanie wartości określonym bajtom. Zasady te mogą być zmienione albo przez program sterujący albo przez użytkownika. Po zrealizowaniu wymienionych zasad program sterujący zgłasza gotowość komputera do dialogu z użytkownikiem - na ekranie pojawia się komunikat READY. Dialog ten nadzorowany jest także przez program z grupy oprogramowania podstawowego. W większości mikrokomputerów domowych, także w ATARI zadanie to realizuje interpreter języka BASIC. Interpreter to program, który "tłumaczy" instrukcje języka wyższego poziomu na język maszynowy, "obsługuje" zlecenia użytkownika oraz przydziela dla potrzeb tłumaczonego programu pamięć. Wprowadzona instrukcja może być natychmiast wykonana lub zapamiętana jako element programu w języku BASIC. Instrukcje zapamiętywane nie są tłumaczone na język maszynowy tylko w odpowiedni sposób kodowane. Tłumaczenie instrukcji realizowane jest przez interpreter dopiero w trakcie wykonywania programu. Wydłuża to niestety znacznie czas wykonywania programu. Zaletą interpretera jest jednak możliwość przerywania i wznawiania programu w dowolnym punkcie. Po przerwaniu programu można dokonywać zmian i kontynuować jego działanie. Interpreter steruje również wykorzystaniem pamięci operacyjnej przydzielając zmiennym użytym w programie odpowiednie obszary pamięci.

Interpreter zapisany jest na stałe w pamięci ROM. Pojawienie się komunikatu READY oznacza, że interpreter gotowy jest do pracy. Omówione wcześniej programy sterujące określone są często jako system operacyjny. Jest to podstawowy system operacyjny o ograniczonych możliwościach (np. w zakresie sterowania pamięcią zewnętrzną). ATARI może pracować także pod kontrolą innych systemów operacyjnych. Mogą one być zapisane w pamięci ROM na stałe zainstalowanej w obudowie komputera, dołączane jako dodatkowa pamięć ROM (cartridge) lub wprowadzone z urządzenia pamięci zewnętrznej, najczęściej dysków elastycznych. Obecnie do najnowszych wersji komputera ATARI polecany jest dyskowy system operacyjny DOS 2.5 oraz bardzo rozbudowany system GEM. Rozbudowane systemy operacyjne, poza tym, że realizują wspomniane wcześniej funkcje, pozwalają na sprawne operowanie pamięcią zewnętrzną i dają użytkownikowi duże możliwości w zakresie wykorzystania komputera.

Do pamięci mikrokomputera można wprowadzać w podobny sposób jak zewnętrzne systemy operacyjne, interpretery i kompilatory innych języków programowania (FORTRAN, COBOL, C, FORTH). Kompilatory tym różnią się od interpreterów, że przed wykonaniem programu dokonują pełnego tłumaczenia na język maszynowy. Często stosuje się także programy umożliwiające wygodne wprowadzanie i uruchamianie programów w języku maszynowym (asemblery, monitory, debuggery)

Dużym udogodnieniem szczególnie dla mało zaawansowanych użytkowników jest tzw. oprogramowanie narzędziowe. Szczególnie popularne są systemy zarządzania bazami danych, systemy do przetwarzania tekstów, programy kalkulacyjne (spreadsheet) i systemy graficzne. Do oprogramowania użytkowego mikrokomputerów należy zaliczyć różnego rodzaju gry.

Rozdział 3

PODSTAWY PROGRAMOWANIA W JĘZYKU BASIC

3.1. Obsługa komputera

ATARI przystosowany jest do współpracy z odbiornikiem telewizyjnym, monitorem kolorowym albo monitorem monochromatycznym. W przypadku korzystania z telewizora posiadającego tylko wejście antenowe 75 ohm musimy najpierw dostroić odbieraną częstotliwość do kanału 4 albo 36 w zależności od typu modulatora naszego komputera. Jeżeli natomiast dysponujemy nowocześniejszym telewizorem posiadającym dwa wejścia niezależne dla wizji i fonii, należy wykorzystać tę możliwość, co w pozytywny sposób wpłynie na jakość obrazu. Można jednak i w przypadku zwykłego odbiornika 'I'V poprawić znacznie jakość obrazu w bardzo prosty sposób. Kabel połączeniowy opleciony jest na pierścieniu wewnątrz niewielkiego czarnego pudełeczka. Skuteczność takiego filtru wzrasta jeśli znajduje się on nie przy komputerze, lecz bardzo blisko telewizora. Spróbujmy rozdzielić pudełko i przewinąć kolejno zwoje na pierścieniu tak aby nowy filtr przesunąć ok. 10 cm od wtyku antenowego. Pozostały zdeformowany nieco kabel wróci do normalnej postaci po delikatnym podgrzaniu nad gazem. Cała operacja jest prosta - a efekty często są zaskakujące, zwłaszcza w miejscach szczególnie narażonych na zakłócenia radiowe.

Niezależnie od sposobu podłączenia typu telewizora lub monitora, urządzenie służące do wprowadzania informacji na lampie kineskopu będziemy dalej nazywać ekranem i oznaczać S: (od ang. Screen). Jako odrębne urządzenie wejścia rozróżniać będziemy klawiaturę (K:), pomimo że fizycznie stanowi ona całość z komputerem. Jeżeli natomiast będziemy pisać o edytorze ekranowym (E:), będziemy mieli na myśli równocześnie ekran i klawiaturę, czyli urządzenie we/wy.

Uzupełnimy i zbierzemy oznaczenia wszystkich urządzeń. Tak więc:

S: ekran

K: klawiatura

E: edytor ekranowy

D: stacja dysków

C: magnetofon kasetowy

P: drukarka

R: złącze RS232

Po włączeniu komputera na ekranie pojawia się słowo READY. Oznacza ono gotowość systemu do pracy. Ekran podzielony jest niewidzialną siatką 24 wierszy i 40 kolumn. W rozdziale piątym omówione zostaną tzw. tryby graficzne i możliwości innego podziału ekranu.

Jasny kwadracik nazywany jest znacznikiem lub częściej kursorem. Oznacza on miejsce, w którym zostanie wpisany wprowadzony do komputera znak.

Większość instrukcji języka BASIC wprowadza się używając liter dużych. Dlatego naciśnięcie klawisza oznaczonego literą powoduje wypisanie na ekranie dużej litery. W celu uzyskania małych liter należy zmienić tryb pracy klawiatury. Uzyskuje się to przez naciśnięcie klawisza CAPS. Powrót do trybu normalnego (duże litery) uzyskujemy przez powtórne naciśnięcia klawisza CAPS.

W momencie gdy uruchomiony jest tryb z małymi literami, naciśnięcie klawisza SHIFT powoduje uzyskanie dużych liter. Klawiatura działa wtedy identycznie jak w maszynie do pisania.

W niektórych przypadkach programista chce przedstawić pewne fragmenty tekstu na ekranie w tzw. „Inverse Video”. Służy do tego klawisz◩ (oznakowany logo ATARI w komputerach ATARI 400/800). Po uruchomieniu komputera na ekranie pojawiają się białe litery na błękitnym tle. Po naciśnięciu tego klawisza uzyskujemy błękitne litery na białym tle. Powtórnie naciskając ten klawisz powracamy do normalnego stanu. Poza typowymi znakami alfanumerycznymi i znakami specjalnymi istnieje możliwość uzyskania na ekranie znaków graficznych. Do generowania tych znaków służy klawisz CONTROL, który w połączeniu z innymi klawiszami powoduje wyświetlenie na ekranie określonego znaku. Naciśnięcie CONTROL i CAPS powoduje, że jedynymi dostępnymi znakami są znaki graficzne.

Klawisze	Znak	Kod ATASCII	Znaczenie
ESC/ESC		27	Kod ESC
ESC/BACK SP.		126	Kursor w lewo – powstaje spacja
ESC/TAB		127	Kursor w prawo do TAB
ESC/CTRL -		28	Kursor w górę
ESC/CTRL =		29	Kursor w dół
ESC/CTRL +		30	Kursor w lewo
ESC/CTRL *		31	Kursor w prawo
ESC/CTRL BACK SP.		254	Kasowanie znaku
ESC/CTRL >		255	Wstawienie znaku
ESC/CTRL <		125	Kasowanie ekranu
ESC/CTRL TAB		158	Kasowanie TAB
ESC/CTRL 2		253	Dźwięk
ESC/SHIFT BACK SP.		156	Usunięcie linii
ESC/SHIFT >		157	Wstawienie linii
ESC/SHIFT <		125	Kasowanie ekranu
ESC/SHIFT TAB		159	Ustawienie TAB

TABELA 3.1.

Klawiatura wyposażona jest w klawisze specjalne. Należą do nich klawisze: ◩ **,** **C**APS, SHIFT, o których była już mowa oraz klawisze RETURN, BREAK, TAB i ESC. Naciśnięcie klawisza RETURN powoduje powrót znacznika (kursora) do nowej linii oraz zaakceptowania przez komputer linii poprzedniej. Klawisz BREAK używany jest do przerwania działania programu i powoduje powrót do kontroli działania komputera przez operatora. Klawisz TAB działa identycznie jak w maszynie do pisania; w połączeniu z klawiszem SHIFT powoduje ustawienie pozycji tablicowania, w połączeniu z CONTROL wyłączenie pozycji, a użyty sam powoduje przeskok do następnej pozycji. Równoczesne naciśnięcie CONT'ROL i 1 powoduje zatrzymanie listowania programu i ponowne uruchomienie. CONTROL i 2 wyłącza lub włącza sygnał dźwiękowy. CONTROL i 3 oznacza koniec danych przy wprowadzeniu (EOF).

W trakcie wprowadzania informacji do komputera bardzo pomocna jest możliwość sterowania kursorem. Do bezpośredniego poruszania kursora służą klawisze oznaczone strzałkami w połączeniu z klawiszem CONTROL.

Naciśnięcie klawisza DELET'E powoduje przesunięcie kursora o jedną pozycję w lewo i skasowanie znajdującego się w tym miejscu znaku.

Podamy obecnie kilka procedur niezbędnych do wpisywania danych. SHIFT i INSERT powoduje "wstawienie" jednego wiersza nad kursorem. CONTROL i INSERT - powoduje wstawienie jednej spacji, w miejsce której można dopisać znak. SHIFT i DELETE - kasuje wiersz, w którym znajduje się kursor. CONTROL i DELETE - powoduje wymazanie znaku wskazywanego przez kursor i przesuwa na to miejsce znak w słowie lub wierszu. SHIFT lub CONTROL i CLEAR - kasuje zawartość ekranu. Klawisz ESC nie wywołuje bezpośredniego działania na ekranie. Służy natomiast do wpisywania w tekst programu instrukcji dotyczących kursora lub ekranu. Tabela 3.1 przedstawia wszystkie możliwości.

ATARI posiada grupę 5 klawiszy pomocniczych. Są to: RESET, którego przyciśnięcie powoduje zatrzymanie wszystkich operacji komputera i wyzerowanie zmiennych programu. OPTION, SELECT, START i HELP mogą być używane jako programowalne klawisze i często wykorzystane są w firmowych programach do wyboru wariantów. OPTION wciśnięty przy włączaniu komputera do sieci "wyłącza" BASIC. START używany jest do ładowania programów maszynowych.

3.2. Mikrokomputer jako kalkulator

Każdy mikrokomputer domowy może być wykorzystany w bezpośrednim trybie pracy do *wyk*onywania obliczeń arytmetycznych. Sposób posługiwania się nim podczas takich obliczeń musi być uwidoczniony na ekranie monitora. Dlatego koniecznym jest poprzedzenie każdej operacji instrukcją języka BASIC:

PRINT W momencie gdy komputer jest gotowy do pracy, możemy wprowadzić np. następujący rozkaz:

PRINT 4*6

Znak * jest operatorem mnożenia. Przedstawiony rozkaz możemy przetłumaczyć jako: wyprowadź wynik mnożenia 5 przez 6. Po naciśnięciu klawisza RETURN na ekranie pojawi się wynik: 24 i znak gotowości do dalszej pracy.

W podobny sposób możemy wykonywać dowolne z podstawowych działań a także bardziej złożone operacje. W języku BASIC stosujemy następujące operatory arytmetyczne:

+ dodawanie
- odejmowanie
* mnożenie
/ dzielenie
^ potęgowanie.

Przy operacjach złożonych można stosować nawiasy wyłącznie okrągłe, co nie wyklucza ich wielokrotnego stosowania. Kolejność wykonywania operacji zgodna jest z zasadami arytmetyki tzn. najpierw wykonane jest potęgowanie, potem mnożenie i dzielenie, a następnie dodawanie i odejmowanie, o ile inaczej nie wskazują nawiasy. Należy pamiętać, że zgodnie z konwencją liczby część ułamkową zapisujemy przy pomocy kropki, a nie przecinka dziesiętnego. Dane do obliczeń mogą być dowolnie dużymi czy małymi liczbami (tzn. mogą składać się z dowolnej liczby znaków). Komputer jest w stanie zapamiętać jednak jedynie dziesięciocyfrową mantysę. Jeżeli wprowadzana została liczba składająca się z większej liczby znaków, to pamiętana będzie (i ewentualnie wypisana) w notacji wykładniczej.
Jeżeli napiszemy:

PRINT 12345678899

po naciśnięciu klawisza RETURN na ekranie pojawi się: 1.23456780E + 10

Przedstawiamy teraz kilka przykładów zastosowania mikrokomputera jako kalkulatora:

Przykład 1.

Obliczamy 2 + 6. Wprowadzamy :

PRINT 2 + 6

i naciskamy klawisz RETURN na ekranie pojawia się wynik: 8

Przykład 2.

Obliczamy $(3 + 2)^2 - (2 + 7)^2 * (1 + 2)$. Wprowadzamy:

PRINT (3 +2)^ 2 - ((2 + 7)^ 2) * (1 + 2)

otrzymujemy wynik - 218

Przykład 3.

Obliczamy -1,2 - (2 - 3)3. Wprowadzamy:

PRINT -1.2-(2-3)^ 3

otrzymujemy - 0.2

Przykład 4.

Obliczamy 2^{63}. Wprowadzamy:

PRINT 2^ 63

otrzymujemy 9.2233721E + 18

3.3. Elementy języka BASIC

Każdy program napisany w języku BASIC zbudowany jest z linii. Linia to jedna lub kilka instrukcji oddzielonych dwukropkiem. Każda linia musi rozpoczynać się od numeru, który jest jej identyfikatorem, a jego wartość decyduje o kolejności wykonywania linii. Oto przykłady różnych linii:

```
5 FOR I = 1 TO N
15 INPUT I: A = B + I:  PRINT C
25 REM LINIA PRZYKLADOWA O DLUGOSCI WIEKSZEJ NIZ 45 ZNAKOW
```

Jak widać wbrew nazwie linia nie musi zawierać się w jednym wierszu na ekranie czy wydruku. Nie może być jednak dłuższa niż 114 znaków. Liczba instrukcji w jednej linii nie jest ograniczona.

Jak wspomniano, o kolejności realizacji linii w trakcie wykonywania programu decyduje jej numer. Linie realizowane są wg wzrastającej kolejności numerów od najmniejszego do największego. Numer musi być całkowitą linią dodatnią (włączając 0) i nie może przekraczać 32 767. Zazwyczaj stosuje się numerowanie z przeskokiem (najczęściej o 10). Dzięki temu możliwe jest dopisanie linii pomiędzy linie już istniejące.

Podczas pisania programu nie sposób ustrzec się błędów. Część z nich wykrywa sam programista i przed zakończeniem danej linii może je usunąć posługując się klawiszami sterującymi kursorem i specjalnymi klawiszami INSERT lub DELETE, a cześć jest wykrywana przez komputer. Interpreter języka BASIC zgłasza wszystkie błędy formalne zaraz po wprowadzeniu linii, w której one wystąpiły. Jeżeli programista próbował wprowadzić linię:

5 PTINT "ALA"

(zamiast PRINT jest PTINT), wówczas na ekranie pojawia się komunikat:

5 ERROR - PTINT "ALA"

oznaczający, że w linii 5 wystąpił błąd. Pierwszy znak " jest wyróżniony ponieważ w momencie jego odczytywania komputer „zorientował się", że wystąpił błąd. Jak widać nie zawsze musi to oznaczać, że błąd popełniono akurat w tym miejscu.

Istnieje kilka możliwości poprawy błędu. Można przepisać całą linię od początku. Poprzedniej linii nie trzeba w żaden sposób kasować. Pojawienie się nowej linii o takim samym numerze jak już istniejąca powoduje skasowanie starej i wprowadzenie na jej miejsce nowej (nawet wtedy gdy linia ta była poprawna). Łatwiej jest jednak poprawić linię posługując się klawiszami sterującymi kursorem oraz klawiszami INSERT i DELETE. Wystarczy powrócić kursorem do miejsca, w którym chcemy nanieść poprawkę, wprowadzić ją i nacisnąć klawisz RETURN. Jeżeli linia dalej jest błędna, to kursor zatrzyma się po komunikacie ERROR, a jeśli jest poprawna to komunikat co prawda pozostaje na ekranie, ale kursor zatrzyma się na jego pierwszym znaku. Można wówczas usunąć zbędny komunikat naciskając klawisze SHIFT i DELETE.

Po wprowadzeniu wszystkich linii program jest gotowy do działania. Często programista chce przeglądnąć tekst programu na ekranie. Konieczność taka pojawia się również wtedy, gdy w trakcie wykonywania programu komputer zgłosił błąd lub sam programista stwierdzi) jego nieprawidłowe działanie. Do "przeglądania" tekstu programu służy instrukcja LIST.

LIST
LIST *numer linii*

LIST *numer linii, numer linii*

Po wprowadzeniu tej instrukcji na ekranie pojawi się linia lub linie z zakresu określonego parametrami. Jeżeli nie występują parametry to wyświetlony zostanie cały tekst programu. Można poprawić dowolne linie posługując się klawiszami sterującymi kursorem. Można także dopisywać lub usuwać linie. Ponieważ kolejność wpisywania linii nie odgrywa żadnej roli (komputer sam ustawia linie wg kolejności numerów) nową linię możemy wprowadzić w dowolnym miejscu. Usunąć linię można poprzez napisanie jej numeru i naciśnięcie klawisza RETURN.

Jeżeli programista uzna, że program nadaje się do pracy, może uruchomić go przy pomocy instrukcji RUN.

Po wprowadzeniu tej instrukcji wykonywane są kolejne linie programu. W programie poprawionym pod względem formalnym mogą występować błędy, których nie wykryli ani programista ani komputer. Mogą to być np. błędy związane z operacjami arytmetycznymi (dzielenie przez zero, pierwiastek z liczby ujemnej itp.). Jeżeli w trakcie wykonywania programu komputer napotka taki błąd - przerywa działanie na błędnej linii i zgłasza komunikat:

ERROR - n AT LINE m

gdzie n oznacza kod błędu (pełny wykaz kodów zawiera dodatek A) a m jest numerem linii, w której błąd wystąpił.

W niektórych przypadkach programista sam chce przerwać działanie programu. Można tego dokonać poprzez naciśnięcie klawisza BREAK. Program zostaje zatrzymany, a na ekranie pojawia się komunikat:

STOPPED AT LINE m

gdzie m to numer linii, która została wykonana jako ostatnia. Można wówczas przeglądać program wypisywać wartości poszczególnych zmiennych, a nawet zmieniać treść programu. Po dokonaniu tych operacji może być kontynuowana praca programu od miejsca, w którym został zatrzymany. Po uruchomienia programu od miejsca zatrzymania służy instrukcja CONT.

Wprowadzenie jej powoduje realizację kolejnych linii od miejsca w którym program został przerwany.

Instrukcja STOP umieszczona w programie działa identycznie jak naciśnięcie klawisza BREAK. Instrukcję STOP wykorzystuje się np. przy testowaniu fragmentów programów. Zatrzymanie wykonania programu na linii z instrukcją STOP świadczy o wykonaniu poprzedzających instrukcji. Wpisane do pamięci komputera programy możemy usunąć z niej stosując instrukcję NEW.

Wprowadzenie tej instrukcji powoduje usunięcie z pamięci komputera wszystkich zapisanych tam linii.

3.4. Stałe i zmienne

Podstawowymi elementami w języku BASIC są, obok instrukcji, zmienne i stałe. Dzielimy je na liczbowe i tekstowe. Inne rodzaje zmiennych będą omówione w dalszych rozdziałach.

Stała liczbowa podczas realizacji programu posiada niezmienną wartość. Zasady tworzenia stałych liczbowych zostały już omówione w poprzednich rozdziałach. Stała tekstowa zwana literałem jest ciągiem znaków ograniczonym dwustronnie symbolem ". Oto przykłady stałych tekstowych:

"ALA"
"1A9///BCD"

Pomiędzy dwa znaki " można wprowadzać dowolne znaki kodu ASCII. Są one traktowane jako tekst lub fragment tekstu. Np. zmienna:

"1985"

jest tylko tekstem i nie można jej używać jako stałej liczbowej. Linia

5 PRINT "1985" + 1

jest błędna, ponieważ nie wolno dodawać stałej tekstowej i liczbowej.

Zmienne reprezentują wydzielone obszary pamięci, których zawartość może być zmieniana bezpośrednio przez programistę lub w wyniku działania programu. Zmienne oznaczamy symbolami, które służą do

identyfikacji zawsze tych samych obszarów pamięci. Symbole te są nazwami zmiennych. W języku BASIC obowiązuje zasada, że nazwy zmiennych muszą rozpoczynać się od litery. Nazwy mogą składać się z liter, cyfr i znaku -. Stosowanie innych znaków jest niedopuszczalne.

Oto przykłady prawidłowych nazw:

A, NAZWA, NUMER 1

Natomiast nazwy 1NUMER, A: - są niedopuszczalne.

Zmienne podobnie jak stałe mogą być liczbowe lub tekstowe. Zmienne liczbowe mogą przyjmować różne wartości wg zasad, które dotyczą również stałych. Zmienna tekstowa to łańcuch znaków.

Po zatrzymaniu programu klawiszem RESET lub zakończeniu działania programu, w którym użyto instrukcji END wszystkie zmienne są wyzerowane tzn. zmienne liczbowe mają wartość 0, a tekstowe zawierają łańcuch spacji (odstępów),

3.5. Funkcje matematyczne

Zajmiemy się teraz funkcjami matematycznymi języka BASIC.

INT(X) - funkcja ta wyznacza część całkowitą liczby rzeczywistej X przez zaokrąglenie w dół. Na przykład instrukcja:

```
5 PRINT INT (-5.6),INT(0),INT(5.6)
```

spowoduje wyświetlenie na monitorze kolejno liczb.
-6 0 5

Poniższy program demonstruje wykorzystanie funkcji IN'T do znalezienia wspólnego podzielnika dwóch liczb naturalnych. Spróbujmy najpierw sprawdzić działanie algorytmu. W tym celu narysujemy trzy kolumny A, B, C i wpiszemy do nich wartości początkowe. Następnie śledząc program wpisujemy w odpowiednie kolumny wartości zmiennych.

PROGRAM „WSPÓLNY PODZIELNIK”

```
5 REM PROGRAM "WSPOLNY PODZIELNIK"
10 PRINT
15 PRINT " PROGRAM WYSZUKUJE PODZIELNIKI"
16 PRINT " DWOCH DANYCH LICZB"
20 PRINT
25 YL1NT " PODAJ PIERWSZA LICZBE " ;
30 INPUT A
35 PRINT " PODAJ DRUGA LICZBE " ;
40 INPUT B
45 IF A<1 OR B<1 THEN 25
50 C=B:B=A-B*INT (A/B):A=C
55 ZF B<>0 THEN 50
60 PRINT
65 PRINT " WSPOLNY PODZIELNIK ";C
70 GOTO 20
```

RND(0) - funkcja ta za każdym kolejnym użyciem przyjmuje wartości liczby pseudolosowej podawanej przez generator liczb pseudolosowych. Funkcja ta jest bezargumentowa i wywołuje się ją podając nazwę z pozornym argumentem. Generowane liczby mają rozkład równomierny i należą do przedziału lewostronnie domkniętego < 0,1)

W celu uzyskania liczb pseudolosowych z dowolnego przedziału (a, b) należy zastosować następujący wzór:

a + (b-a)*RND(0).

Program "Rzut kością" symuluje 100 rzutów kością. Zostało to osiągnięte przez użycie 1001iczb losowych z przedziału od 1 do 6. Zbiór został utworzony przez użycie instrukcji INT(RND(0)) + 1). Program oblicza również średnią i wariancję.

PROGRAM "RZUT KOŚCIĄ"

```
5 REM PROGRAM "RZUT KOSCIA"
10 DIM Q $ (1) : N = 100
15 PRINT
20 PRINT " RZUT KOSCIA"
25 PRINT
30 S=O: SS=0
35 FOR X=1 TO N
40 D=INT (RND (0)*6+1)
45 S=S+D;SS=SS+D*D
50 PRINT D; " ";
55 NEXT X
60 A=S/N
62 PRINT :PRINT
65 PRINT "WARTOSC SREDNIA "; A
70 V=SS/N-A*A
72 PRINT
75 PRINT "WARIANCJA ";V:PRINT
80 INPUT Q$
85 GOTO 15
```

EXP(X)

- funkcja ta oblicza wartość stałej e podniesionej do potęgi x, gdzie e jest równe 2.718281828... Zastanówmy się co się dzieje ciągiem $a_n=(1 +1/n)^n$ gdy n staje się bardzo duże (matematyk powiedziałby, że n zmierza do nieskończoności). Poprzez wykonanie programu widać że ciąg $(1 + 1/n)^n$ zmierza do e = 2.718281828... gdy n zmierza do nieskończoności. Matematycznie zapisuje się to następująco:

$\lim (1 +1/n) = e$

$n \to \infty$

Bardziej ogólnie można powiedzieć, że przedstawiony ciąg a_n... zmierza do e gdy n zmierza do nieskończoności.

PROGRAM "LICZBA E"

```
5 REM PROGRAM "LICZBA E"
10 PRINT " OBLICZANIE WARTOSCI E"
15 PRINT
20 FOR X=1 TO 5
25 N=10^X
30 E=(1+1/N)^N 35 PRINT N,E
40 NEXT X
```

Zwróćmy uwagę, że n nie może być zbyt duże, gdyż interpreter będzie wprowadzał duże niedokładności. Weźmy obliczoną wartość e dla trzech największych x (20 linia programu) i porównajmy z wartością z tablic.

LOG(X) - funkcja LOG oblicza logarytm naturalny danej liczby np.: LOG(25). Możliwość obliczenia logarytmu naturalnego pozwala obliczyć logarytm przy dowolnej podstawie za pomocą następującego wzoru:

$\text{Log}_x(x) = \text{LOG}(x)/\text{LOG}(s)$

CLOG(X) - funkcja logarytmu dziesiętnego. Oblicza logarytm dziesiętny bezpośrednio
SQR(X) - funkcja ta oblicza pierwiastek kwadratowy z liczby x.

ABS(X) - oblicza wartość bezwzględną (moduł) z argumentu

$$\mathbf{ABS(X)}=\begin{cases} x \text{ gdy } x>0 \\ -x \text{ gdy } x<0 \end{cases}$$

SGN(X) - funkcja SGN podaje znak argumentu

$$\mathbf{SGN(X)}=\begin{cases} 1 \text{ gdy } x > 0 \\ 0 \text{ gdy } x = 0 \\ -1 \text{ gdy } x < 0 \end{cases}$$

SIN(X) - funkcja sinus

COS(X) - funkcja cosinus

Argument funkcji **SIN** i **COS** musi być wyrażeniem arytmetycznym określającym kąt w radianach. Wynik jest z przedziału od -1 do 1. Można również używać jako jednostki stopni ale wymaga to uprzedniego zadeklarowania instrukcją **DEG**. Powrót do miary w radianach jest możliwy instrukcją **RAD**.

ATN(X) - funkcja arcus tangens

Argument X funkcji **ATN** jest z przedziału określonego dopuszczalnym zakresem liczb w ATARI BASIC, wynik jest z przedziału od $-\pi/2$ do $\pi/2$

Jeżeli zachodzi konieczność wykorzystania funkcji innych niż podane wyżej, można skorzystać z następujących wzorów:

secans	SEC(X) = 1/COS(X)
cosecans	CSC(X) = 1/SIN(X)
tangens	TAN(X) = SIN(X)/COS(X)
cotangens	CTG(X) = COS(X)/SIN(X)
arcus cosinus	ARCCOS(X) = -ATN(X/SOR(1-X*X)) + π/2
arcus sinus	ARCSIN(X) = ATN(X/SOR(1-X*X))
arcus cotangens	ARCCI'G(X) = ATN(X) + π/2
sinus hiperboliczny	SINH(X) = ESP(X) - EXP(-X))/2
cosinus hiperboliczny	COSH(X) _ (EXP(X)+EXP(-X))/2
tangens hiperboliczny	TANH(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))

3.6. Wprowadzenie i wyprowadzenie danych

Jak wspomniano, jednym ze sposobów nadawania wartości zmiennym jest wprowadzanie danych przez programistę. Umożliwia to instrukcja:

INPUT

Po uruchomieniu programu instrukcją **RUN** klawiatura jest "wyłączona" i wpisywanie jakichkolwiek danych nie jest możliwe. Instrukcja **INPUT** zatrzymuje bieg programu i informuje nas o konieczności wprowadzenia danych.

```
5 INPUT A,B
```

spowoduje wyświetlenie na ekranie znaku zapytania. Oznacza to, że komputer oczekuje na wprowadzenie danej, która będzie podstawiona do zmiennej A. Operator wpisuje na ekranie żądaną wielkość np.

```
? 2
```

i naciska klawisz RETURN. Od tego momentu zmienna A ma wartość 2. Jednocześnie na ekranie pojawi się kolejny znak zapytania oznaczający gotowość do wprowadzenia kolejnej zmiennej. Jeżeli operator napisze

?15

i naciśnie klawisz RETURN to komputer nada zmiennej B wartość 15 i przejdzie do realizacji następnej linii. W podobny sposób nadajemy wartość zmiennym tekstowym. Rozpatrzymy przykład linii:

```
5 INPUT A$
```

gdzie zmienna A jest zmienną tekstową. Komputer wykonując tę linię wyświetli znak zapytania.
W tym momencie programista może wprowadzać tekst, który będzie zawarty w pamięci maszyny jako zmienna A$ np.

? ALA

Po naciśnięciu klawisza RETURN komputer wprowadzi znaki A L A do zmiennej A$ i przejdzie do wykonywania następnej linii.

Pojawienie się znaku zapytania informuje co prawda o gotowości do wprowadzenia danych, nie podaje jednak jaka zmienna ma być podstawiana.

Ponieważ w wersji języka BASIC dla ATARI nie ma możliwości umieszczenia tekstu objaśniającego instrukcję **INPUT**, komunikat dla operatora należy przekazać przy użyciu instrukcji **PRINT** np.

```
5 PRINT "A="; : INPUT A
```

Jedną z najczęściej realizowanych w programach funkcji jest wyprowadzenie informacji na ekran. W ten sposób informuje się operatora o wynikach obliczeń, przekazuje się komunikaty itd. Do wyprowadzenia informacji służy instrukcja:

PRINT

Przy pomocy tej instrukcji można wyprowadzić na ekran stale liczbowe i tekstowe, zmienne oraz wyniki obliczeń. Wynikiem realizacji następującej linii:

```
5 PRINT 2
```

będzie pojawienie się na ekranie cyfry 2

Po wykonaniu linii program przechodzi oczywiście do realizacji linii następnej nie przerywając działania.

Przedstawimy teraz kilka przykładów zastosowań instrukcji **PRINT**:

1.

```
5 PRINT"LITERAL"
```

w wyniku działania tej linii na ekranie pojawia się napis: LITERAL

2.

```
15 PRINT 2*2
```

po wykonaniu tej linii pojawi się wynik obliczeń: 4

3.

```
25 PRINT "2*2=";2*2
```

realizacja tej linii spowoduje pojawienie się tekstu i wyniku obliczeń

2*2 = 4

W ostatnim przykładzie można zauważyć, że w jednej instrukcji **PRINT** można wyprowadzić więcej niż jedną daną. Sposób ich rozdzielania nie jest jednak obojętny. Używając średnika spowodujemy wypisanie rozdzielonych danych (tekstu lub liczb) w jednej linii bez odstępu. Przy użyciu przecinka wprowadzamy tabulację ekranu. Tabulacja polega na podzieleniu ekranu na cztery strefy co 10 znaków każda, w których wypisane są kolejne dane. Np. realizacja linii

5 PRINT "WARSZ";"AWA"

spowoduje pojawienie się napisu
WARSZAWA
Natomiast wykonanie linii

15 PRINT 1,2,3,4

spowoduje wyświetlenie znaków w następującym układzie
1 2 3 4

Znaki rozdzielające mogą znajdować się również na końcu listy danych w instrukcji **PRINT.** Jeżeli na końcu listy będzie znajdował się średnik, to pierwsza dana w następnej instrukcji **PRINT** zostanie wypisana bez odstępu za ostatnią wypisaną daną. Jeżeli na końcu listy będzie przecinek, to pierwsza dana w kolejnej instrukcji **PRINT** pojawi się w pierwszej wolnej strefie. Wykonanie dwóch kolejnych linii:

5 PRINT"ALA MA";
15 PRINT"KOTA"

spowoduje wyświetlenie napisu **ALA MA KOTA**
Natomiast realizacja następujących linii

5 PRINT "WYNIK=", 15 PRINT 2*2

spowoduje wyświetlenie **WYNIK = 4**

Czasami dla większej przejrzystości ekranu korzystne jest rozdzielenie kolejnych wierszy wyświetlonych na ekranie. Można użyć do tego celu "pustej" instrukcji **PRINT.** Wykonanie linii:

25PRINT 15PRINT"ADAM KOWALSKI"
"JAN KOZLOWSKI"

spowoduje wyświetlenie napisów:
ADAM KOWALSKI
JAN KOZLOWSKI
Natomiast realizacja linii:

5 PRINT "ADAM KOWALSKI"
15 PRINT
25 PRINT "JAN KOZLOWSKI"

spowoduje wyświetlenie napisów w następującym układzie:
ADAM KOWALSKI

JAN KOZLOWSKI

Instrukcja **PRINT** służy do przedstawiania wartości zmiennych liczbowych i tekstowych. Rozważmy teraz następujący przykładowy program:

5 INPUT A,B
15 PRINT "A=",A
25 PRINT "B=",B

Po wprowadzeniu wszystkich linii, a następnie RUN realizowana jest pierwsza linia. Na ekranie pojawia się znak zapytania

Operator wprowadza wartość zmiennej A ? 15

Po naciśnięciu klawisza RETURN komputer "pyta" o wartość zmiennej B ? 25

Po wprowadzeniu wartości i naciśnięciu klawisza RETURN komputer podstawia za zmienną B podaną wartość i realizuje kolejno 15 i 25 linię. Po zakończeniu programu na ekranie będą widoczne wprowadzane dane i dane wyprowadzone:

? 15 ? 25 A = 15 B = 25

W podobny sposób można wyprowadzać zmienne tekstowe.

Należy zaznaczyć, że w jednej instrukcji PRINT mogą być wyprowadzone stałe, wyniki obliczeń i wartości zmiennych. Poprawna jest następująca linia:

```
15 PRINT "2*2 ="; 4, "A="; A
```

Jeżeli zmienna A miała wartość 7 to na ekranie pojawi się następujący napis
2*2 = 4 A = 7

W podstawowej w wersji języka BASIC zagwarantowano możliwość emisji wydawnictw na drukarkę. Służy do tego instrukcja

LPRINT

działająca analogicznie jak **PRINT**, z tym że w wyniku jej działania powstaje odpowiedni zapis na drukarce, a nie na monitorze.

W zaawansowanym programowaniu istotne jest projektowanie tzw. "ekranów" czyli odpowiednie rozmieszczenia na ekranie monitora odpowiednich zapisów. W instrukcji PRINT można stosować znaki sterujące kursorem umieszczając je w cudzysłowie, np.:

```
PRINT " ↖ ↓ → → ↓ ↓ ↑ "
```

Działanie tej linii spowoduje zmazanie ekranu, a następnie przesunięcie kursora o jeden wiersz w dół, dwie kolumny w prawo, dwa wiersze w dół i jeden wiersz w górę. Zapisanie znaków sterujących kursorem w instrukcji **PRINT** wymaga naciśnięcia przed każdym z nich klawisza ESC Np. aby uzyskać zapis "↖ " naciskamy klawisz ESC, a następnie SHIFT i CLEAR równocześnie. Do projektowania ekranów można wykorzystać również instrukcję:

POSITION Pozwala ona na dokładne określenie miejsca na ekranie, w którym umieszczony zostanie zapis z następującej instrukcji PRINT. Wykonanie linii:

```
5 POSITION 21, 11: PRINT "ALA"
```

spowoduje wypisanie tekstu ALA w dwunastym wierszu od 22 kolumny począwszy (parametry instrukcji **POSITION** podają ile znaków od lewego marginesu i górnego brzegu ekranu mają być opuszczone).

3.7. Instrukcje podstawienia

Do tej pory przedstawiono w jaki sposób można nadawać zmiennym wartości, a później wyświetlać te wartości na ekranie. Obecnie objaśnimy, w jaki sposób można zmieniać wartość zmiennych w trakcie realizacji programu. Można tego dokonać przy pomocy instrukcji podstawienia **LET**.
Rozważmy następujący przykład:

```
5 LET A = 7
```

Ponieważ po angielsku let oznacza "niech" przedstawioną linię możemy przetłumaczyć "niech A będzie miało wartość 7". Znak "=" nie oznacza w instrukcji podstawienia operatora matematycznego, ani logicznego, a wskazuje jedynie jaka wartość ma być podstawiona za zmienną, oddzielając ją od nazwy zmiennej. Działanie instrukcji podstawienia przedstawiamy w przykładowym programie:

```
5 INPUT A
```

```
15 PRINT "A=";A
25 LET A = 17
35 PRINT"A=";A
```

Po uruchomieniu programu i wprowadzeniu liczby 5 na ekranie pojawi się: ?5

A=5 A=17

W przedstawionym programie linie 15 i 35 są identyczne. Mimo to ich wykonanie powoduje różne skutki., Wykonanie linii 15 powoduje wyświetlenie napisu:

A=5

a wykonanie linii 35 powoduje pojawienie się na ekranie: A=17
Zmianę wartości zmiennej A spowodowała instrukcja podstawienia zapisana w linii 25.

W większości wersji języka BASIC słowo LET w instrukcji podstawienia może być pominięte. O tym, że to instrukcja podstawienia świadczy obecność w linii znaku "=". Po lewej stronie znaku "=" musi zawsze znajdować się nazwa zmiennej, która w wyniku operacji podstawienia przyjmuje nową wartość. Po prawej stronie znaku mogą znajdować się: stałe, zmienne, operacje matematyczne, funkcje, zdania logiczne. Przedstawmy kilka przykładów zastosowania instrukcji podstawienia:

1.

5 A=B

zmienna A przyjmie wartość taką jaką posiada zmienna B,

2.

15 A=2*2

zmienna A przyjmie wartość równą wynikowi operacji zapisanej z prawej strony znaku równości (w tym przypadku 4),

3.

25 A=B+C

zmienna A przyjmie wartość równą sumie wartości zmiennej **B** i C,

4.

35 A=B*2

za zmienną A podstawiona zostanie podwojona wartość zmiennej B,

5.

45 A=(B=0)

za A zostanie podstawione 1 jeśli **B** jest równe zero lub 0 jeśli **B** różne jest od zera. W tym przypadku znak "_" oznacza najpierw symbol operacji podstawienia, a potem symbol w zdaniu logicznym.

6.

55 A=SQR(4)

A przyjmuje wartość równą pierwiastkowi z 4,

7.

65 A=LOG(B)

A przyjmie wartość równą logarytmowi z wartości reprezentowanej przez zmienną B,

8.

```
75 A=A+1
```

wykonanie tej linii spowoduje zwiększenie wartości zmiennej A o jeden tzn. za A zostanie podstawiona poprzednia wartość zmiennej powiększona o 1.

W tej chwili możemy już napisać program, który wykonywał będzie użyteczne zadania. Przedstawimy program obliczający pole i obwód koła. Przedtem jeszcze wprowadzimy następną instrukcję:

REM

Jest to instrukcja pozwalająca na umieszczanie w tekście programu, komentarzy objaśniających znaczenie poszczególnych instrukcji, fragmentów programu czy poszczególnych zmiennych. Po instrukcji **REM** można napisać dowolny tekst, który nie jest analizowany przez komputer, a wyłącznie wyświetlony podczas przeglądania programu. Przykłady zastosowań instrukcji zobaczymy w tekście programu obliczającego obwód i pole koła.

```
10 REM PROGRAM OBLICZA OBWOD I POLE KOLA
20 PRINT "PROMIEN="; : INPUT P
30 REM OBLICZENIE OBWODU
40 D=2*3.14*P
50 PRINT
60 PRINT "OBWOD=";  D
70 REM OBLICZANIE POLA
80 PL=3.14*P^2
90 PRINT
100 PRINT "POLE="; PL
```

Przedstawimy teraz, jak będzie wyglądał ekran po wykonaniu programu. Załóżmy, że obliczamy promień koła o promieniu równym 1

```
RUN
PROMIEN=?1
OBWOD=6.28
POLE= 3.14
```

Raz wprowadzony program pozostaje w pamięci aż do momentu wyłączenia komputera lub wykonania instrukcji **NEW.** Program można więc przeglądnąć posługując się instrukcją **LIST** lub ponownie go wykonać wprowadzając **RUN.** Po wykonaniu obliczeń dla promienia równego 1 możemy je ponowić dla promienia równego np. 2. W tym, celu wprowadzamy **RUN,** a następnie po pojawieniu się "pytania" o wielkość promienia wprowadzamy 2. Po zrealizowaniu programu ekran będzie wyglądał następująco:

```
RUN
PROMIEN=?2
OBWOD= 12.566371
POLE = 12.566371
```

W ten sam sposób można wielokrotnie uruchamiać program i wykonywać obliczenia dla rożnych wartości promienia.
Przedstawimy teraz program obliczający wartość średnią trzech liczb:

```
10 REM OBLICZANIE WARTOSCI SREDNIEJ
20 REM X1, X2, X3 TO KOLEJNE LICZBY
30 INPUT X1, X2, X3
40 REM OBLICZENIE SUMY
50 SUMA=X1+X2+X3
60 REM OBLICZANIE SREDNIEJ
70 SREDNIA=SUMA/3
```

```
80 PRINT "SREDNIA=";SREDNIA
```

Uruchomienie programu i wprowadzenie liczb 3, 4, 5 spowoduje pojawienie się na ekranie:

```
RUN
?3,4,5
SREDNIA=4
```

Program ten można wykonywać wielokrotnie dla różnych wartości danych. W linii 30 znajduje się instrukcja wprowadzająca trzy zmienne. Dane można wprowadzać w jednej linii rozdzielając je przecinkami.

3.8. Sterowanie przebiegiem programu

Przedstawione do tej pory programy realizowane były w ten sposób, że kolejne linie komputer wykonywał wg kolejności ich numerów. Bardzo często zachodzi konieczność zmiany kolejności czy wielokrotnego wykonania tych samych linii.

Podstawową instrukcją sterującą przebiegiem programu jest instrukcja:

GOTO *numer linii*

Po angielsku „go to" oznacza „idź do". W wyniku działania tej instrukcji zmieniona zostanie kolejność realizowania linii. Jeżeli w programie pojawi się linia

```
85 GOTO 55
```

to po jej wykonaniu nie będzie realizowana linia o następnym w kolejności numerze lecz linia 55. Wprowadźmy teraz nową linię do programu obliczającego średnią:

```
10 REM OBLICZANIE WARTOSCI SREDNIEJ
20 REM X1, X2, X3 TO KOLEJNE LICZBY
30 INPUT X1, X2, X3
40 REM OBLICZENIE SUMY
50 SUMA=X1+X2+X3
60 REM OBLICZANIE SREDNIEJ
70 SREDNIA=SUMA/3
80 PRINT "SREDNIA=";SREDNIA
90 REM NOWA LINIA-POWROT DO LINI 30
100 GOTO 30
```

Po wyprowadzeniu wyników obliczeń (linia 80) program nie kończy działania lecz powraca do wprowadzania kolejnych danych. Przypuśćmy, że chcemy obliczyć wartości średnie dla trzech zestawów liczb (4,5,6), (1,2,3) i (2,1,6). Przebieg obliczeń przy pomocy komputera będzie przedstawiał się następująco:

```
RUN
?4,5,6
SREDNIA=5
?1,3,5
SREDNIA=3
?2,1,6
SREDNIA=3
```

W tym momencie program nie przerywa działania, lecz oczekuje na wprowadzenie kolejnych danych. Program zatrzymać można poprzez naciśnięcie klawisza BREAK lub RESET.

A oto jeszcze jeden przykład działania instrukcji **GOTO**:

```
10 REM PROGRAM WYPISUJACY KOLEJNE LICZBY
20 I=0
30 PRINT I;"-";
40 REM ZWIEKSZENIE WARTOSCI o 1
50 I=I+1
60 GOTO 30
```

Po uruchomieniu programu na ekranie będą pojawiały się kolejne liczby naturalne oddzielone znakiem "–"
RUN 0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19

Podobnie jak poprzedni program ten można przerwać naciskając klawisz BREAK lub RESET. Do instrukcji sterujących przebiegiem programu możemy zaliczyć także instrukcję:

GOSUB *numer linii*

Instrukcja ta podobnie jak GOTO zmienia kolejność wykonywania instrukcji. Różnica polega na tym, że komputer zapamiętuje linię, w której występowania instrukcja GOSUB i po natrafieniu na instrukcję

RETURN

powraca do następnej linii po tej, w której występowała instrukcja GOSUB. Oto przykład działania obu tych instrukcji:

```
10 PRINT "LINIA 10"
20 GOSUB 100
30 PRINT "LINIA 30"
40 STOP : REM KONIEC PROGRAMU
100 PRINT "LINIA 100"
110 RETURN

RUN
LINIA 10
LINIA 100
LINIA 30
STOPPED AT LINE 40
```

Język BASIC dopuszcza również stosowanie instrukcji skoku warunkowego:

ON *nazwa zmiennej* GOTO *lista numerów linii*

Wartość zmiennej w instrukcji określa, która linia będzie wykonywana jako następna. Na przykład jeśli wartość zmiennej wynosi 3, to nastąpi skok do linii, której numer jest trzecim w kolejności na liście. W przypadku, gdy wartość zmiennej nie jest całkowita, to zostanie ona zaokrąglona do najbliższej liczby całkowitej. Wartość zmiennej musi być dodatnia (może być równa zero) oraz mniejsza niż 256. W przeciwnym wypadku wystąpi błąd. W przypadku gdy wartość zmiennej jest równa zero lub jest większa niż ilość numerów linii na liście to następną wykonywaną instrukcją będzie znajdująca się bezpośrednio po **ON...GOTO.**

Stosować można również instrukcję:

ON...GOSUB

która różni się od instrukcji: **ON...GOTO** tym, że w wyniku jej działania następuje skok do podprogramu o początkowym numerze linii podanym na liście.

Przykład:

```
5 REM PROGRAM "LICZBY ZESPOLONE"
6 PRINT "PODAJ LICZBY ZESPOLONE"
10 INPUT A1,A2
```

```
20 INPUT B1,B2
25 PRINT "PODAJ DZIALANIE"
26 PRINT "1) +":PRINT"2)-':PRINT"3)*":PRINT"4)/"
30 INPUT K
40 ON K GOSUB 100, 200, 300, 400
50 PRINT A;"+I *(";B;")"
60 GOTO 10
100 REM DODAWANIE
110 A=A1+81
120 B=B1+B2
130 RETURN
200 REM ODEJMOWANIE
210 A=A1-B1
220 B=B1-B2
230 RETURN
300 REM MNOZENIE
310 A=A1*A2-B1*B2
320 B=A1*B2+A2*B1
330 RETURN
400 REM DZIELENIE
410 C=B1*B1+B2*B2
420 A=(A1*B1+A2*82)/C
430 B=(A2*B1-A1*B2)/C
440 RETURN
```

Do instrukcji sterujących przebiegiem programu w zależności od zaistniałej sytuacji można zaliczyć instrukcję:

TRAP *nr linii*

Powoduje ona przejście do wykonania programu od podanego numeru linii w wypadku wystąpienia błędu w trakcie egzekucji programu. Należy przy tym pamiętać, że wystąpienie błędu powoduje skasowanie tej instrukcji i kolejny błąd spowoduje przerwanie egzekucji programu. Instrukcja TRAP może być skasowana w dowolnym punkcie programu poprzez jej powtórne użycie z podaniem numeru linii spoza dopuszczalnego zakresu np.:

TRAP 40000

Użycie instrukcji **TRAP** jest szczególnie polecane w programach, w których wprowadza się wiele danych. Pozwala ona wówczas zabezpieczyć program przed niepożądanym przerwaniem w przypadku błędnego wprowadzenia danych.

Przy złożonej strukturze programu korzystne jest stosowanie graficznych metod projektowania (tzw. schematów blokowych).

3.9. Podejmowanie decyzji

Dotychczas przedstawione programy przykładowe wykonywały się wg niezmiennej, ustalonej przez programistę struktury. Istnieje jednak możliwość zmiany struktury w trakcie realizacji programu w zależności od zaistnienia określonych warunków. Pozwala na to instrukcja:

IF *warunek* THEN *instrukcja*

Rozważmy następujące przykładowe linie:

```
15 IF X<5 THEN X=5
25 PRINT X
```

Linię 15 możemy przetłumaczyć: "Jeśli X jest mniejsze od 5 to za X podstaw 5". Przedstawiony fragment programu może być zrealizowany na dwa sposoby, w zależności od wartości zmiennej X. Jeżeli zmienna X posiada wartość mniejszą od 5 (warunek jest spełniony - zdanie ma wartość logiczną 1) to zostanie wykonana instrukcja podstawienia i X przyjmie wartość 5; następnie wykonana będzie linia 25. Jeżeli zmienna X posiada wartość > 5 (warunek nie jest spełniony - zdanie ma wartość logiczną 0) instrukcja podstawienia znajdująca się za słowem kluczowym **THEN** nie będzie wykonywana. Komputer po stwierdzeniu, że warunek nie jest spełniony przechodzi do realizacji następnej linii. Rozpatrzmy działanie instrukcji **IF...THEN** na przykładzie:

```
15 PRINT "X=";:INPUT X
25 IF X<5 THEN X=5
35 PRINT "X=";X
45 GOTO 15

RUN
X=?-2
X=5
X=?12
X=12
X=?
```

Przerwijmy działanie programu klawiszem BREAK. Na przedstawionym przykładzie widać, że wprowadzanie za zmienną X wartości -2 powoduje zadziałanie (w linii 25) instrukcji podstawienia 5 za zmienną X. Jeśli podstawimy za X liczbę 12 to instrukcja podstawienia w linii 25 nie jest wykonywana.

Warunkiem w instrukcji **IF...THEN** mogą być różne proste i złożone zdania logiczne. Oto kilka przykładów:

1.

```
15 IF A<>B THEN A=B
```

jeśli A jest różne od B to za A podstaw B,

2.

```
25 IF A=2 THEN PRINT "A=2"
```

jeśli A jest równe 2 to wypisz tekst "A =2",

3.

```
35 IF A THEN GOTO 55
```

jeśli zmienna A jest równa 1 (a właściwie ma wartość logiczną 1) to przejdź do wykonania linii 55,

4.

```
35 IF A=B AND C=D THEN A=D
```

jeśli A jest równe B oraz C jest równe D to za A podstaw D,

5.

45 IF A=B OR B>0 THEN A=SQR(B)

jeśli A jest równe B lub B jest większe od zera to za A podstaw pierwiastek z B. W języku BASIC dopuszczalne są następujące operatory logiczne:

> większy
< mniejszy
= równy
< > różny
AND i
OR lub
NOT nie
< = mniejszy lub równy
> = większy lub równy

Wykorzystując poznane do tej pory instrukcje możemy napisać prosty program sprawdzający umiejętność dodawania w zakresie 20 liczb naturalnych. Do zrealizowania programu potrzebne nam będą jeszcze dwie funkcje: **RND(X)** i **INT(X)**. Działanie pierwszej funkcji polega na generowaniu losowych liczb z przedziału 0 do 1. Każde odwołanie się do tej funkcji powoduje wylosowanie kolejnej liczby losowej z podanego przedziału. Funkcja **INT(X)** oblicza część całkowitą liczby X (np. **INT(1.4)** równy jest 1).

Program - egzaminator będzie losował dwie liczby z przedziału 1 do 10, a następnie "pytał" operatora o wartość sumy tych liczb. Jeśli odpowiedź będzie poprawna, pojawi się potwierdzenie wyniku, jeśli nie, komputer poinformuje operatora o popełnionym błędzie. Oto tekst programu:

```
10 REM PROGRAM - EGZAMINATOR
20 REM LOSOWANIE LICZB
30 X=INT (10*RND (0)+1):Y=INT (10*RND (0)+1)
40 REM PYTANIE KOMPUTERA
50 PRINT X; "+";Y;"=";:INPUT W
60 REM OBLICZENIE POPRAWNEGO WYNIKU
70 S=X+Y
80 REM SPRAWDZANIE POPRAWNOSCI ODPOWIEDZI
90 IF S=W THEN PRINT "WYNIK POPRAWNY":GOTO 30
100 PRINT "BLAD": GOTO 30
```

W linii 30 odbywa się generowanie liczb losowych z przedziału 1 do 10. Funkcja **INT** daje nam część całkowitą z wylosowanej liczby (założyliśmy, że liczby mają być naturalne). Funkcja **RND(0)** daje liczbę losową z przedziału 0 do 1. Ponieważ, jak uczy rachunek prawdopodobieństwa, szansa wylosowania w funkcji **RND** jedynki jest praktycznie zerowa, a funkcje **INT(X)** "obcina" wartość X w dół, konieczne było dodanie do argumentu tej funkcji 1.
Prześledźmy teraz przebieg programu:

```
RUN
7+4=?11
WYNIK POPRAWNY
2+9=?13
BLAD
3+5=?8
WYNIK POPRAWNY
6+3=?
```

Przerwijmy w tym momencie działanie programu. Jak wiadomo, nasz program może działać bez końca. Jeżeli czytelnik uruchomi przedstawiony program na swoim komputerze i otrzyma inne pytanie nie będzie to wcale świadczyć o błędzie. Po prostu funkcja **RND(X)** działa losowo i na innym komputerze daje inny szereg liczb losowych.

Do tej pory używaliśmy instrukcji **IF...THEN** do decydowania o wykonaniu, bądź nie, jednej operacji lub wyboru jednej z dwóch instrukcji. Bardzo często konieczny jest wybór nie jednej lecz całego ciągu instrukcji. Może to być zrealizowane w jednej linii. Rozważmy następującą linię:

```
35 IF A>=0 THEN B=SQR(A):PRINT "B=";B:PRINT
```

Jeśli warunek A>=0 jest spełniony to wykonany zostanie cały ciąg instrukcji, a mianowicie **B=SQR(A):PRINT'B=":PRINT**, a następnie linia 40. Jeśli warunek nie jest spełniony, wymieniony ciąg będzie pominięty i komputer przejdzie do linii następnej.

Przedstawiona możliwość nie zawsze jednak wystarczy. Spróbujmy napisać program obliczający potęgi kolejnych liczb naturalnych z przedziału 1 do 10. Stosując dotychczasowe rozwiązania musielibyśmy 10-krotnie napisać linię obliczającą kolejne wartości potęg. Możemy jednak program znacznie uprościć stosując połączenie instrukcji **IF...THEN** i instrukcji **GOTO**. A tak będzie wyglądała realizacja programu w języku BASIC:

```
10 REM PROGRAM OBLICZAJACY POTEGI
15 REM LICZB NATURALNYCH od 1 do 10
20 I=1
30 PRINT I;""2=";I^2
40 I=I+1
50 IF I<=10 THEN GOTO 30
60 PRINT "KONIEC OBLICZEN"

RUN
1        2=1
2        2=4
3        2=9
4        2=16
5        2=25
6        2=36
7        2=49
8        2=64
9        2=81
10       2=100
KONIEC OBLICZEN
```

W linii 20 programu nadajemy zmiennej I wartość początkową 1. Następnie (linia 30) wyświetlana jest zmienna I i potęga zmiennej I. W linii 40 wartość zmiennej zostaje powiększona o jeden (I jest równe 2). Komputer sprawdza (linia 50) czy aktualna wartość zmiennej I jest mniejsza lub równa 10. Jeśli tak, to komputer przechodzi do realizacji linii 30. Następuje wyświetlenie aktualnej wartości I i kwadratu I (wartość I wynosi w tym momencie 2). Linie 30, 40, 50, powtarzane są tak długo, aż w linii 50 stwierdzone zostanie, że zadany warunek nie jest spełniony. Ponieważ każde powtórzenie linii 40 powoduje zwiększenie wartości zmiennej I, po dziesięciu powtórzeniach zmienna I będzie miała wartość 11 Warunek I < = 10 nie będzie wówczas spełniony i instrukcja **GOTO 30** zostanie pominięta. Komputer zrealizuje linię 60 i zakończy działanie.

3.10 Pętla

W informatyce pętlą nazywamy wielokrotne powtarzanie tych samych instrukcji przy zmieniających się wartościach jednej lub kilku zmiennych. W ostatnio prezentowanym programie pętla realizowana jest przez trzy linie. W linii 20 nadajemy początkową wartość zmiennej I, która stanowi tzw. licznik w pętli, w linii 40 zwiększamy wartość licznika o zadany krok (w tym przypadku o jeden), a w linii 50 badamy, czy wartość licznika przekroczyła zadaną granicę (tutaj 10). Ponieważ pętla występuje w każdym niemal programie napisanym w języku BASIC, wprowadzono dwie instrukcje ułatwiające organizowanie pętli w programie. Są to instrukcje:

FOR ***nazwa zmiennej=wyr num1*** **TO** ***wyr num2*** **STEP** ***wyr num3*** i **NEXT**

Wprowadzenie tych instrukcji do programu obliczającego potęgi liczb naturalnych nie zmieni jego struktury, a uprości tylko jego tekst. Program będzie wyglądał następująco:

```
10 REM PROGRAM OBLICZAJACY POTEGI
15 REM LICZB NATURALNYCH od 1 do 10
20 FOR I=1 TO 10
30 PRINT I;"^2=";I^2
40 NEXT I
50 PRINT "KONIEC OBLICZEN"
```

Linię dwudziestą można rozumieć następująco: "wykonaj następne linie zmieniając wartość zmiennej I od 1 do 10 co 1". W linii nie użyliśmy słowa **STEP**, które określa krok zmian licznika, w takim przypadku komputer automatycznie przyjmie za krok wartość 1. Linia 40 oznacza dosłownie "następne I" i wskazuje komputerowi, że w tym miejscu kończy się pętla i należy przechodząc do następnej wartości licznika I (zwiększonego o jeden krok) powrócić do pierwszej linii po instrukcji **FOR I=1 TO 10**. W trakcie wykonywania instrukcji **NEXT** I komputer sprawdza czy wartość licznika nie przekroczyła zadanej w instrukcji **FOR**... po słowie **TO** wartości.

Jeżeli wartość licznika jest równa zadanej wartości granicznej program wykonuje następną po **NEXT I** linię (w tym przypadku 50).

W jednym programie może występować kilka pętli zarówno jedna za drugą, jak i jedna wewnątrz drugiej. Parametry instrukcji **FOR**... nie muszą być stałymi. Mogą to być również zmienne. Poprzednio pisaliśmy program obliczający średnią dla wielu zestawów trzech liczb. Napiszemy teraz program, który obliczał będzie dla zadanej ilości liczb ich wartość średnią. Załóżmy dodatkowo, że ilość liczb w zestawie może być różna.

Tekst programu realizującego ten schemat jest następujący:

```
10 REM OBLICZANIE SREDNICH
20 REM N-TO ILOSC ZESTAWOW LICZB
30 PRINT "ILOSC ZESTAWOW=";:INPUT N
40 FOR I=1 TO N
50 PRINT :PRINT "ZESTAW ";I
55 PRINT "ILOSC LICZB=";
60 INPUT M
70 REM ZEROWANIE SUMY
80 SUMA=0
90 FOR J=1 TO M
100 REM WPROWADZANIE LICZBY I DODAWANIE
110 REM DO JEJ SUMY OGOLNEJ
120 INPUT X:SUMA=SUMA+X
130 NEXT J
140 REM OBLICZANIE I WPISYWANIE SREDNIEJ
150 SREDNIA=SUMA/M:PRINT "SREDNIA=";SREDNIA
160 REM PRZEJSCIE DO KOLEJNEGO ZESTAWU
170 NEXT I
180 END
```

W linii 30 komputer pyta się o liczbę zestawów danych -N. W linii 40 ustala, że następne linie poczynając od 50 będą realizowane wielokrotnie przy czym za każdym razem wartość zmiennej I będzie o 1 większa poczynając od 1, a kończąc na zadanej w zmiennej N wartości. W linii 60 komputer zostaje poinformowany o ilości danych w kolejnym (I-tym) zestawie, a w linii 80 nadaje początkową wartość ogólnej sumie danych dla danego zestawu.

W rezultacie wykonania linii 90 komputer będzie realizował następne linie zmieniając wartość licznika J od 1 do M. W linii 120 wprowadzane są kolejne dane, a wartość sumy przyrasta o wprowadzoną wielkość. Dopóki wartość licznika nie przekroczy zadanej w zmiennej M granicy, dopóty rezultatem linii 130 będzie powrót do linii 120 i zmiana wartości licznika. Jeśli granica zostanie przekroczona, zostanie obliczona i wypisana wartość średnia (linia 150). Jeśli komputer wykona obliczenia dla wszystkich zestawów (licznik będzie większy od N) to w wyniku działania linii 170 komputer przejdzie do linii 180 i zakończy działanie, w innym przypadku wartość licznika I zostanie zwiększona o 1 i komputer powróci do linii 50. Instrukcja END zawarta w linii 180 oznacza

zakończenie programu i powoduje wyzerowanie wszystkich zmiennych oraz powrót komputera do stanu w jakim znajdował się przed rozpoczęciem wykonania obliczeń.

Przedstawmy przebieg obliczeń programu dla następujących trzech zestawów danych (1,2,3), (0,2) i (2,6,8,9):

```
RUN
ILOSC ZESTAWOW =?3
ZESTAW 1
ILOSC LICZB=?3
X=?1
X=?2
X=?3
SREDNIA=2

ZESTAW 2
ILOSC LICZB=?2
X=?0
X=?2
SREDNIA=1

ZESTAW 3
ILOSC LICZB=?4
X=?2
X=?6
X=?8
X=?9
SREDNIA=6.25
```

Podstawową zaletą naszego programu w stosunku np. do przedstawionego poprzednio programu obliczającego wartość średnią z 3 liczb jest jego uniwersalność. Przy jego pomocy możemy dokonać obliczenia wartości średniej dla praktycznie dowolnej liczby zestawów zawierających różne ilości danych. Uniwersalność taka, która powinna być cechą wszystkich programów, możliwa była do osiągnięcia dzięki zastosowaniu instrukcji FOR...oraz użyciu w niej jako parametrów zmiennych, a nie stałych.

Dokładna analiza ostatniego programu pozwoli zauważyć pewne ogólne prawidłowości dotyczące realizacji pętli w języku BASIC:

1 - dopuszczalne jest umieszczenie wewnątrz pętli następnej pętli z tym, że w każdej z nich musi występować inny licznik,

2 - wewnątrz pętli nie wolno dokonywać zmian wartości licznika,

3 - pętle nie mogą zachodzić na siebie tzn. najpierw musi być zamknięta (instrukcją **NEXT**) pętla wewnętrzna, a dopiero potem pętla zewnętrzna,

4 - wszystkie parametry instrukcji **FOR** mogą być zmiennymi. W języku BASIC dopuszczalne jest używanie jako parametrów instrukcji pętli **FOR** zmiennych czy stałych całkowitych, ułamkowych, dodatnich i ujemnych. Na zakończenie rozważań o pętli przedstawmy prosty program ilustrujący działanie słowa kluczowego **STEP**:

```
10 REM TROJKAT Z GWIAZDEK
20 FOR I=10 TO 1 STEP –1
30 FOR J=1 TO I:PRINT "*";: NEXT J
40 PRINT
50 NEXT Z
60 END
```

```
**********
*********
```

```
********
*******
******
*****
****
***
**
*
```

Wykonanie linii 20 powoduje wielokrotne powtarzanie linii 30, 40 i 50 przy czym za każdym razem licznik I ma wartość o 1 mniejszą. Zmienna I przyjmuje więc kolejno wartość 10,9,8, itd. W linii 30 zapisano całą pętlę drukującą gwiazdki. Ilość powtórzeń w tej pętli zależy od aktualnej wartości zmiennej I**.**

Jeżeli w trakcie wykonania programu wystąpi konieczność przerwania realizacji pętli i przejścia do innych instrukcji można skorzystać z instrukcji:

POP

Powoduje ona wymazanie adresu powrotnego w instrukcjach **FOR, GOSUB i ON GOSUB.** Oto przykład zastosowania tej instrukcji w połączeniu z instrukcją **GOSUB:**

```
10 PRINT "MAM UZYC POP CZY NIE 1/2";
20 INPUT A
30 GOSUB 100
40 PRINT "POWROT DO PODPROGRAMU GLOWNEGO"
50 GOTO 10
100 GOSUB 200
110 PRINT "POWROT DO 1-SZEGO PODPROGRAMU"
120 RETURN
200 PRINT "DRUGI PODPROGRAM"
210 IF A=1 THEN POP
220 RETURN
```

Nadając zmiennej A wartość 1 spowodujemy, że w linii 210 będzie wykonana instrukcja **POP.** Wymaże ona adres powrotny. Instrukcja **RETURN** w linii 120, spowoduje powrót do programu głównego (linie od 10 do 50), a nie pierwszego podprogramu. A oto przykładowy przebieg programu:

```
RUN
MAM UZYC POP CZY NIE 1/2?2
DRUGI PODPROGRAM
POWROT DO 1-SZEGO PODPROGRAMU
POWROT DO PROGRAMU GLOWNEGO MAM UZYC POP CZY NIE 1/2 ?1 DRUGI PODPROGRAM
POWROT DO PROGRAMU GLOWNEGO
```

Z instrukcji **POP** korzysta się stosunkowo rzadko ponieważ zamiast niej można użyć kombinacji instrukcji logicznych i skoku warunkowego. Warto jednak o niej pamiętać ponieważ w programach o złożonej strukturze może ona znacznie uprościć pracę programisty.

3.11. Zmienne indeksowe

Do tej pory zmienne dzieliliśmy na liczbowe i tekstowe. W języku BASIC niezależnie od tego podziału wyróżnić możemy zmienne indeksowe. Nazwa zmiennej indeksowej reprezentuje nie jedną komórkę pamięci,

lecz wiele komórek. Indeks służy do rozróżnienia różnych komórek tej samej zmiennej indeksowej. Zmienną indeksową zapisujemy używając nawiasów okrągłych za nazwą zmiennej np. Z(1). Symbol Z(1) reprezentuje pierwszą komórkę zmiennej indeksowej Z. Zapis Z(2) mimo zastosowania tej samej nazwy zmiennej dotyczy innego obszaru pamięci. Przykładem takiej zmiennej może być tabela zawierająca ilość dzieci w różnych grupach wiekowych. Oznaczamy tę zmienną przez X. Różnym grupom wiekowym przyporządkowujemy indeksy w następujący sposób (tabela I).

Grupa wiekowa	Indeks
0-1 lat	1
1-2 lat	2
2-3 lat	3
3-4 lat	4
4-5 lat	5
5-6 lat	6
powyżej 6 lat	7

TABELA 3.2. Przykład tablicy indeksów

Zapis X(3) reprezentuje wówczas liczbę dzieci w trzeciej grupie wiekowej tj. w wieku od 2 do 3 lat. Zmienne indeksowe są traktowane w programie w szczególny sposób. Wymagają one "zarezerwowania" przez komputer odpowiedniego obszaru pamięci. Dlatego przed ich użyciem należy zgłosić komputerowi, że określona zmienna będzie indeksowana oraz określić zakres pamięci jaki należy jej przydzielić. Służy do tego instrukcja:

DIM ***lista zmiennych indeksowych***

Ponieważ nasza zmienna X posiada 7 indeksów musimy w programie zadeklarować ją w następujący sposób:

```
DIM X(7)
```

Po wykonaniu linii, w której znajdował się będzie taki zapis komputer traktował będzie zmienna X - jako zmienną zawierającą co najmniej 7 indeksowanych komórek. Liczbę 7 w przedstawionym zapisie nazywamy wymiarem zmiennej X. W trakcie programu nie wolno używać w stosunku do zmiennej X indeksów większych niż zadeklarowany wymiar (np. X(10)). Sytuacja taka spowoduje sygnalizację błędu i zatrzymanie wykonania programu. W wielu przypadkach nie można z góry dokładnie określić jaki największy indeks wystąpi w danej zmiennej. Należy wtedy deklarować wymiar zmiennej z pewnym nadmiarem. Nie wolno jednak zapominać, że pamięć komputera jest ograniczona i zadeklarowanie zbyt dużych wymiarów tablic uniemożliwia wykonanie programu.

Wykorzystanie zmiennej indeksowej (tablicy) przedstawiamy na przykładzie programu obliczającego sumę elementów tablicy. Program wprowadza najpierw tablice, a następnie w kolejnej pętli sumuje wszystkie jej elementy. Oto tekst programu:

```
10 REM SUMOWANIE ELEMENTOW TABLICY
20 DIM T (10):REM WYMIAR TABLICY
30 PRINT "LICZBA DANYCH=":INPUT N
40 REM WCZYTYWANIE TABLICY
50 PRINT "I", "T(I)"
60 FOR I=1 TO N
70 PRINT I,:INPUT T:T(I)=T
80 NEXT I: PRINT
90 REM OBLICZANIE SUMY
100 PRINT "I", "T(I)","SUMA"
110 SUMA=0
120 FOR I=1 TO N
130 SUMA=SUMA+T(I)
140 PRINT I, T(I),SUMA
150 NEXT I
```

```
160 PRINT
170 PRINT "SUMA=";SUMA
```

W wersji języka BASIC dla A'TARI niedopuszczalny jest zapis:

```
INPUT T(I)
```

Trudność tę omijamy w sposób przedstawiony w linii 70.

Oczywiście, drukowanie w drugiej części programu pośrednich wartości sumy oraz kolejnych indeksów nie jest konieczne. Przebieg obliczeń dla N równego 5 i kolejnych danych równych 1.3,4,2,7 będzie następujący:

```
RUN
LICZBA DANYCH =?5
 I            T(I)
1             ?1
2             ?3
3             ?4
4             ?2
5             ?7
 I            T(I)       SUMA
1               1        1
2               3        4
3               4        8
4               2        10
5               7        17
SUMA=17
```

Przedstawione zadanie można rozwiązać bez stosowania tablicy w jednej pętli. Kolejne dane wprowadzane byłyby do tej samej zmiennej. W takim przypadku dokonanie jakichkolwiek innych operacji na wprowadzonych danych po zakończeniu pętli nie byłoby możliwe. Dane wprowadzone do tablicy można przekazywać i wykorzystywać tak długo jak jest to potrzebne. Można np. obliczać sumę wybranych elementów tablicy, szukać elementu najmniejszego czy największego bez konieczności ponownego wprowadzania danych.

Rozważmy teraz następujące zadanie:

Do celów statystycznych musimy ustalić liczbę dzieci w poszczególnych grupach wiekowych (tabela 3.3) zamieszkałych w pewnym osiedlu. Pod uwagę bierzemy 20 dzieci. Dysponujemy następującym zestawieniem:

Lp.	Wiek dziecka (grupa wiekowa)	Lp.	Wiek dziecka (grupa wiekowa)
1	1	11	5
2	1	12	1
3	4	13	2
4	7	14	2
5	3	15	3
6	2	16	3
7	1	17	4
8	5	18	4
9	6	19	7
10	6	20	7

TABELA 3.3.

Zadaniem naszym jest obliczenie ilości dzieci w każdej grupie wiekowej, a następnie ustalenie ile dzieci należy do grupy wiekowej od 3 do 6 lat. Zadanie to można zrealizować na komputerze posługując się następującym programem:

```
10 REM SUMOWANIE WG KLUCZA
20 REM DEKLARACJA WYMIARU TABLICY
30 DIM X(7): FOR I=1 TO 7:X(I)=O: NEXT I
40 PRINT "LICZBA DANYCH": INPUT N
45 PRINT "LP", "GRUPA WIEKOWA"
50 FOR I=1 TO N
60 PRINT I,: INPUT J
70 REM GRUPA WIEKOWA JEST INDEKSEM X
80 X(J)=X(J)+1
90 NEXT I
100 PRINT: PRINT "GRUPA WIEKOWA", "LICZBA DZIECI"
110 FOR I=1 TO 7
120 PRINT "        "; I, , X(I)
130 NEXT I
140 Z=X(4)+X(5)+X(6)
150 PRINT: PRINT "ILOSC DZIECI W WIEKU od 3 do 6 LAT=";Z
```

Po ustaleniu liczby danych (linia 40) komputer rozpoczyna w pętli (linia 50) wczytywanie numerów grup wiekowych do jakich należą kolejne dzieci. Numer grupy wiekowej stanowi jednocześnie indeks tabeli X. W linii 80 komputer dodaje "jeden" do odpowiedniej komórki tabeli X. W linii 140 następuje zsumowanie ilości dzieci z 4,5 i 6 grupy wiekowej. W linii 120 zastosowano pomiędzy zmiennymi dwa przecinki. Wartość zmiennej X(I) będzie wypisana w trzeciej strefie (chodzi tu o przejrzysty układ wyników na ekranie). A oto przebieg obliczeń.

```
RUN
LICZBA DANYCH=?20
LP            GRUPA WIEKOWA
1             ?1
2             ?1
3             ?4
4             ?7
5             ?3
6             ?2
7             ?1
8             ?5
9             ?6
10            ?6
11            ?5
12            ?1
13            ?2
14            ?2
15            ?3
16            ?3
17            ?4
18            ?4
19            ?7
20            ?7

GRUPA WIEKOWA      LICZBA DZIECI
      1                      4
      2                      3
      3                      3
      4                      3
```

```
5                    2
6                    2
7                    3
ILOSC DZIECI W WIEKU OD 3 DO 6 LAT = 7
```

Indeksem w tablicy może być stała liczbowa lub zmienna. Może to być także wyrażenie arytmetyczne np.

X(I+1)
X(I+1-J+2)
X(K+4) itp.

Indeks nie może być liczbą ujemną, może być natomiast liczbą ułamkową i zerem. Jeżeli indeks jest liczbą ułamkową, komputer zaokrągla jego wartość do liczby całkowitej np. zapis X(1.2) jest interpretowany jako X(1), a zapis X(1.6) jest interpretowany jako X(2). Użycie nazwy zmiennej zadeklarowanej jako tablica bez nawiasów jest dopuszczalne - komputer traktuje ją wtedy jako inną zmienną np.

```
10 INPUT X:X(I)=I
```

Zmienne X i X(I) reprezentują inne komórki pamięci.

W niektórych przypadkach stosowanie tablic jednoindeksowych nie wystarcza. W języku BASIC ATARI można wprowadzać tablice z dwoma indeksami. Zapis zmiennej dwuindeksowej wygląda następująco:

A*(numer wiersza, numer kolumny)*

Tablice dwuindeksowe (dwuwymiarowe) muszą być zadeklarowane instrukcją **DIM**. Zapis:

```
DIM A(5,5)
```

oznacza zgłoszenie dwuwymiarowej tablicy, która może mieć co najwyżej 5 wierszy i 5 kolumn. Typowym przykładem tabeli dwuwymiarowej jest często stosowana tablica odległości.
Przedstawmy teraz fragment tablicy odległości drogowych (tabela 3.4.):

Warszawa	Biała Podlaska	Białystok	Bielsko-Biała
Warszawa	173	184	371
Biała Podlaska	-	237	484
Białystok	237	-	555
Bielsko Biała	484	555	-

TABELA 3.4.

Jest to tablica "kwadratowa" (liczba wierszy jest równa liczbie kolumn) i symetryczna (pierwszy wiersz odpowiada pierwszej kolumnie, drugi wiersz drugiej itd.). odległość z Warszawy do Białegostoku znajdujemy w pierwszym wierszu i trzeciej kolumnie lub w trzecim wierszu i pierwszej kolumnie. Ten element tablicy ma więc współrzędne 1 i 3 lub 3 i 1. Jeżeli przedstawianą tablicę wprowadzimy do komputera jako zmienną indeksową dwuwymiarową A(I,J) to odległość z Warszawy do Białegostoku znajdziemy jako element oznaczony A(1,3) lub A(3,1).

Przedstawmy teraz przykładowy program obliczający należność za bilety komunikacji samochodowej. Załóżmy, że cena biletu wynika z pomnożenia odległości przez stawkę wynoszącą 2 zł/km.

```
10 REM NALEZNOSC ZA BILET PKS
20 DIM A(5,5): REM A-TABLICA ODLEGLOSCI
30 PRINT "WYMIAR TABLICY=";: INPUT N
35 PRINT "I","J","A(I,J)"
```

```
40 FOR I=1 TO N: FOR J=1 TO N
50 PRINT I,J,: INPUT A:A(I,J)=A
60 NEXT J: NEXT I: PRINT
70 PRINT "KOD PIERWSZEJ MIEJSCOWOSCI": INPUT K
80 PRINT "KOD DRUGIEJ MIEJSCOWOSCI": INPUT L
90 CENA=A(K,L)*2
100 PRINT "CENA BILETU="; CENA
110 GOTO 70

RUN
WYMIAR TABLICY=?4
I       J       A(I,J)
1       1       ? 0
1       2       ?173
1       3       ?184
1       4       ?371
2       1       ?173
2       2       ? 0
2       3       ?237
2       4       ?484
3       1       ?184
3       2       ?237
3       3       ? 0
3       4       ?555
4       1       ?371
4       2       ?484
4       3       ?555
4       4       ?0
KOD PIERWSZEJ MIEJSCOWOSCI=?1
KOD DRUGIEJ MIEJSCOWOSCZ=?2
CENA BILETU=346
KOD PIERWSZEJ MIEJSCOWOSCI=?
```

Program przerwiemy klawiszem BREAK o ile nie żądamy obliczenia innej ceny biletu. Sprawdźmy, czy program działał prawidłowo. Podaliśmy kod pierwszej miejscowości równy 1 i drugiej równy 2. Komputer miał obliczyć cenę biletu z Warszawy do Białej Podlaskiej. Odległość między tymi miastami wynosi 173 km, tak więc bilet będzie kosztował 346 zł. Taką wartość podał komputer. Program jak widać działa prawidłowo.

3.12. Dane stałe

W pierwszej części powyższego programu (linie 35 do 60) wprowadzona jest tablica odległości. Ponieważ odległości między miejscowościami mają charakter danych stałych, nie ma potrzeby wprowadzania ich przy każdym uruchamianiu programu. Dane te można na stale wprowadzać do programu. Jest to możliwe dzięki zastosowaniu dwóch instrukcji **READ** i **DATA**. Po zmodyfikowaniu program będzie wyglądał następująco:

```
10 REM NALEZNOSCI ZA BILET PKS
20 DIM A(5,5)
30 FOR I=1 TO 4: FOR J=1 TO 4
40 REM NADAWANIE ELEMENTOW TABLICY
```

```
50 READ W: A(I,J)=W
60 NEXT J: NEXT I
70 PRINT "KOD PIERWSZEJ MIEJSCOWOSCI=": INPUT K
80 PRINT "KOD DRUGIEJ MIEJSCOWOSCI=": INPUT L
90 CENA=A(K,L)*2
100 PRINT "CENA BILETU="; CENA
110 GOTO 70
120 DATA 0,173,184,371,173,0,237,484,184 130 DATA 237,0,555,371,484,555,0
```

W linii 50 nadawane są wartości kolejnym elementom tablicy A(I,J). Wartości te "odczytywane" są z tablicy zawartej w liniach 120 i 130, po instrukcji DATA. Każde uruchomienie programu powoduje automatyczne odczytanie tych samych wartości.

Instrukcja **READ** występuje często w połączeniu z instrukcją **RESTORE**. Bez użycia tej instrukcji każda linia zawierająca **READ** powoduje odczytanie pierwszej wolnej dotąd nieczytanej wartości w instrukcji **DATA**. Jeżeli chcemy odczytywać powtórnie te same wartości musimy wcześniej użyć instrukcji **RESTORE**. Załóżmy, że mamy dwie tablice jednowymiarowe A i B o wymiarze 5. Do obu tych tablic chcemy wprowadzić te same wartości np. 1,2,3,4 i 5. Napiszmy odpowiedni fragment programu:

```
10 DIM A(5),B(5)
20 FOR I=1 TO 5
30 READ W:A(I)=W
40 NEXT I
50 RESTORE
60 FOR I=1 TO 5
70 READ W:B(I)=W
80 NEXT I
90 DATA 1,2,3,4,5
```

Zawarta w linii 50 instrukcja **RESTORE** spowoduje, że w linii 70 czytane będą te same informacje co w linii 30.

W instrukcji **RESTORE** można umieścić numer linii, w której znajdują się dane przeznaczone do powtórnego czytania. W przedstawionym fragmencie programu byłaby to linia 90. Można więc napisać:

```
50 RESTORE 90
```

ale nie jest to konieczne, ponieważ w tym programie występuje tylko jedna linia z instrukcją **RESTORE**.

Rozdział 4

OPERACJE NA TEKSTACH

4.1. Stałe i zmienne tekstowe

Tekstem nazywać będziemy ciąg znaków np. liter, cyfr lub znaków graficznych. Wszystkie je nazywamy znakami alfanumerycznymi. Stalą tekstową jest ciąg znaków ujętych w cudzysłów. Długością tekstu jest liczba znaków (wraz ze spacjami) znajdujących się w ciągu. Zmienna tekstowa jest obszarem pamięci, któremu nadaliśmy nazwę. Ostatnim znakiem nazwy musi być **"$"**, np. A$, ATARI$. Użycie w programie zmiennej tekstowej wymaga wcześniejszego jej zadeklarowania instrukcją **DIM** lub **COM** np. **DIM A$(7)**. Oznacza to, że tekst zawarty w zmiennej A będzie miał długość co najwyżej 7 znaków.

Istnieje kilka możliwości nadawania wartości zmiennej tekstowej:

1. Przez instrukcję INPUT np.

```
5 DIM A$(5)
15 INPUT A$

RUN
?ABCDE
```

2. Przez instrukcję LET np.

```
5 DIM A$(5)
15 LET A$="ABCDE"
```

3. Instrukcjami READ i DATA np.

```
5 DIM A$(5)
15 READ A$
25 DATA ABCDE
```

Zmienne tekstowe mogą być argumentami instrukcji logicznych np. IF A$ > B$ THEN... albo IF C$="ALA" THEN GOTO... Porównywanie zmiennych tekstowych ma inny charakter niż porównywanie zmiennych liczbowych. Analiza przeprowadzana jest od lewej strony, wartość znaków zależy od ich położenia w alfabecie. I tak np. B jest większe od A, a C jest większe od B. Każdemu znakowi przyporządkowano umownie pewną wartość liczbową. Najczęściej przyjmuje się standard ASCII (ud American Standard Code for Information Interchange). W ATARI stosuje się zmodyfikowaną wersję, która nosi nazwę ATASCII.

4.2. Funkcje tekstowe

Operacje związane z przetwarzaniem tekstów występują często w różnych programach. Znacznym ułatwieniem w tym zakresie są funkcje tekstowe. Są nimi:

CHR$(X) - funkcja ta daje znak kodu ATASCII o numerze podanym przez zmienną X lub stałą liczbową np.

```
15 DIM A$(1)
25 A$=CHR$(66)
35 PRINT A$

RUN
B
```

ASC(A$) - jest funkcją odwrotną dla **CHR$** - daje numer kodu ATASCII dla znaku zawartego w zmiennej tekstowej A$ np.

```
15 DIM A$(1)
25 A$="B"
35 PRINT ASC(A$)

RUN
66
```

LEN(A$) - funkcja ta podaje długość tekstu (liczbę znaków) zmiennej A np.

```
15 DIM A$(5)
25 INPUT A$
35 PRINT LEN(A$)

RUN
?ALA 3
```

VAL(A$) - funkcja zmienia tekst na odpowiadającą mu wartość liczbową. Pierwszym znakiem argumentu **VAL** musi być cyfra.

```
15 DIM A$(5)
25 INPUT A$
35 PRINT 2*VAL(A$)

RUN
?17 34
```

Bez użycia funkcji **VAL** niemożliwe byłoby dokonanie na zmiennej A$ zawierającej ciąg cyfr operacji arytmetycznych ponieważ zawarta w niej liczba jest traktowana przez komputer jako ciąg znaków.

STR$(X) - funkcja zamienia wartość zmiennej lub stałej liczbowej na odpowiadającą jej stałą tekstową np.

```
15 DIM A$(5)
25 INPUT B
35 A$=STR$(B)
45 PRINT LEN(A$)

RUN
?1234 4
```

W stosunku do zmiennej B nie można było zastosować funkcji **LEN**, gdyż jest ona zmienną liczbową. Dzięki zastosowaniu funkcji **STR$** i później **LEN** można pośrednio ustalić liczbę cyfr zmiennej B. Istnieje możliwość wydzielenia fragmentu tekstu. Dokonuje się tego przez określenie pozycji pierwszego i ostatniego znaku szukanego fragmentu np.

```
15 DIM A$(25)
25 A$="ZIMNA WODA"
35 A$(1, 6)="CIEPLA"
45 PRINT A$

RUN
CIEPLAWODA
```

Jeśli podamy tylko pozycję początkową, komputer za pozycję końcową przyjmuje ostatni znak tekstu.

W przypadku tekstów nie możemy mówić o ich sumowaniu w sensie matematycznym, możemy natomiast wykonywać operacje ich łączenia (konkatenacji). Dokonujemy tego w sposób podany w przykładach:

```
A$(LEN(A$)+1)=B$
15 DIM A$(15), B$(15)
25 A$="JAN"
35 B$="KOWALSKI"
45 A$(LEN(A$)+1)=B$
55 PRINT A$

RUN
JAN KOWALSKI
```

4.3. Zastosowania

W celu zilustrowania omawianych zagadnień zaprezentujemy kilka programów. Pierwszy z nich w podanym zdaniu wyszukuje wszystkie czteroliterowe słowa i zastępuje je gwiazdkami. Wskaźnik X porusza się w obrębie podanego zdania, kolejno od pierwszego do ostatniego znaku. Komputer najpierw sprawdza czy pierwszy znak jest spacją, następnie czy dalsze cztery znaki są literami, oraz czy piąty znak jest spacją. Jeżeli okaże się, że wszystkie te warunki są spełnione, to cały czteroliterowy podciąg jest zastępowany gwiazdkami.

PROGRAM "CENZOR"

```
5 REM PROGRAM "CENZOR"
10 DIM A$(100), B$(102), Q$(1)
15 PRINT CHR$(125)
25 PRINT : PRINT "PISZ SWOJ TEKST": PRINT
30 INPUT A$: L=LEN(A$)
35 B$(1)=" ": B$(2)=A$: B$(L+2)=" "
40 FOR X=1 TO L-3
45 IF B$ (X, X)<>" " THEN 75
50 FOR Y=X+1 TO X+4
55 IF B$(Y, Y)=" " THEN 75
60 NEXT Y
65 IF B$(X+5, X+5)<>" " THEN 75
70 B$(X+1, X+4)="****"
75 NEXT X
80 PRINT : PRINT B$: PRINT
85 INPUT Q$
90 GOTO 15
```

Opisane techniki operacji na tekstach mogą być użyte do poszukiwania określonych stów w zdaniach. Program „Poszukiwane słowo" ilustruje to zagadnienie., W podanym zdaniu poszukuje on spacji, gdy ją znajdzie znaczy to, że wszystkie poprzednie znaki tworzą słowo. Następnie jest ono porównywane z poszukiwanym.

PROGRAM "POSZUKIWANE SŁOWO"

```
5 REM PROGRAM "POSZUKIWANE SLOWO"
10 DIM W$(100),  K$(20), Q$(1)
15 PRINT CHR$(125)
20 PRINT : PRINT "POSZUKIWANE SLOWO"
25 PRINT : PRINT "PODAJ SLOWO SZUKANE": INPUT K$
30 PRINT : PRINT "PODAJ ZDANIE ": INPUT W$
35 L=LEN(W$)+1: IF L<2 THEN 15
40 P=1: W$(L)=" "
45 FOR X=1 TO L
50 IF W$(X, X)<>" " THEN 75
55 IF P>=X THEN P=X+1: GOT0 75
60 IF W$(P, X-1)<>K$ THEN P=X+1: GOT0 75
65 PRINT : PRINT "SLOWO SZUKANE WYSTEPUJE W ZDANIU"
```

```
70 GOTO 85
75 NEXT X
80 PRINT : PRINT "SLOWO SZUKANE NIE WYSTEPUJE W ZDANIU"
85 PRINT : PRINT "NACZSNIJ RETURN"
90 INPUT Q$
95 GOTO 15
```

Następny przykład ilustruje technikę poszukiwania słów kluczowych w dowolnie długich zbiorach zawartych w instrukcjach **DATA**. Dodatkowo program ten podaje również ciąg znaków, z którym związane jest słowo kluczowe.

Program został napisany dla hipotetycznego, dużego przedsiębiorstwa informatycznego, którego pracownicy specjalizują się w obsłudze różnych komputerów i w pisaniu programów w różnych językach. Kierownik, który zajmuje się rozdziałem pracy potrzebuje szybkiej i dokładnej informacji, którzy pracownicy firmy mogą rozwiązać podany program.

Nazwisko każdego zatrudnionego umieszczone jest w instrukcji **DATA**, a po nim następują jego kwalifikacje. Aby rozróżnić nazwisko od kwalifikacji, nazwisko poprzedzone jest znakiem (hasz)#.

PROGRAM "BAZA DANYCH"

```
5 REM PROGRAM "BAZA DANYCH"
6 DIM Q$(1), S$(20), W$(20), N$(20)
10 PRINT CHR$(125)
15 PRINT : PRINT "UMIEJETNOSCI PROGRAMOWANIA"
20 RESTORE : N$=" "
25 PRINT : PRINT "UMIEJETNOSCI WYMAGANE "; : INPUT S$
30 PRINT
35 READ W$
40 IF W$="$" THEN 60
45 IF W$(1, 1)="#" THEN N$=W$(2): GOTO 35
50 IF W$=S$ THEN PRINT W$;  " "; N$
55 GOTO 35
60 PRINT : PRINT "NACISNIJ RETURN"
65 INPUT Q$
70 GOTO 10
1000 REM DANE
1010 REM
1015 DATA # KOWALSKI A, FORTRAN, ADA, VAX
1020 DATA # KRUSZYNKA W, CORAL 66,  COBOL, ARGUS, TANDEM
1025 DATA # GRUBY ST., BASIC, COBOL, PDP-11, ARMY SYSTEM
1030 DATA # NOWAK J, ALGOL-68, COBOL,  BURROUGHS
1035 DATA # KOLODZIEJ P, CAL, UNIX, PROLOG, MICRO-COBOL
1040 DATA # DUDZIK K, PASCAL, POLICE SYSTEM
1045 DATA %
```

Ostatni przykład ilustruje możliwości manipulowania tekstem. Słowo zostaje losowo wybrane z podanej listy, a następnie jego litery są przestawiane tak aby stworzyć anagram. Zadaniem jest odgadnięcie pierwotnego słowa. W przypadku trudności należy wpisać "?" i odpowiedź pojawi się natychmiast.

PROGRAM "MANARAG"

```
1 REM PROGRAM "MANARAG"
5 DIM L$(1), W$(15), WW$(15), A$(15), G$(15)
10 PRINT CHR$(125)
15 PRINT :  PRINT "MANARAG"
20 PRINT :  PRINT "ZGADNIJ SLOWO"
25 A$=" ":  G$=" ":  RESTORE
30 N=50:  REM LICZBA SLOW
35 FOR J=1 TO RND (0)*N+1
```

```
40 READ W$
45 NEXT J
50 L=LEN(W$):  WW$=W$
55 FOR J=1 TO L
60 N=INT (1+RND(0)*L)
65 L$=W$(N, N):  IF L$="*" THEN 60
70 W$(N,N)= "*"
75 A$(J)=L$
80 NEXT J
85 IF A$=WW$ THEN 25
90 PRINT : PRINT " "; A$
95 PRINT : PRINT "PISZ SLOWO ZGADYWANE ": INPUT G$
100 IF G$=" " THEN 95
105 IF G$="?" THEN PRINT :  PRINT "ODPOWIEDZ:  "; WW$:  GOTO 120
110 IF WW$<>G$ THEN PRINT "SPROBUJ JESZCZE RAZ":  GOTO 95
115 PRINT : PRINT "DOBRA ROBOTA"
120 PRINT : PRINT "NACISNIJ RETURN" 125 INPUT L$
130 GOTO 10
1000 REM ZBIOR SLOW
1005 REM
1010 DATA DROGA, AUTO, DRZWI, TECZKA, POGODA, GUZIK, IRENA, TYFUS,  FUJARKA,
NARTY, KOMPUTER, RADIO, KLOCEK, GARNEK
1020 DATA BIKINI, NOCNIK, BUTY, TABORET, KWIATEK, BRAT, DZIECKO,  POLSKA,
SZKALNKA, TOR, LIST, PLAKAT, PEPSI, PRZYCISK, KARTKA
1025 DATA DATA,  OKNO,  GRUBY,  ZEGAREK, PAPIEROS, TOR, GAZ, GRA,  KIEROWNICA,
FLIPER, RONDO, AUTOBUS, PRZYSTANEK
1030 DATA GUMA, JACEK, NOS, KLUCZ, PUSZKA, RURA, NAWIAS, METR
```

Rozdział 5

GRAFIKA

ATARI posiada bardzo rozbudowane a przy tym łatwe do wykorzystania możliwości graficzne. Użytkownik ma do dyspozycji zestaw małych i dużych liter, cyfr i znaków specjalnych oraz graficznych. Przejście do znaków graficznych następuje przy wciśniętym klawiszu CONT`ROL. Poniżej przedstawiamy tzw. tryby graficzne od których zależy format tworzonego obrazu na ekranie monitora. A"I'ARI BASIC posiada 16 podstawowych trybów graficznych, z czego trzy służą do pracy w trybie tekstowym, natomiast trzynaście do pracy w trybie graficznym czyli tworzenia obrazów z pojedynczych punktów.

Tabela 5.1. określa poszczególne tryby graficzne:

Tryb graf.	Tekst	Ilość kolumn	Ilość wierszy	Ilość wierszy całego ekranu	Liczba kolorów	Wymagany obszar pamięci	Wymagany obszar pamięci dla całego ekranu
0	tekst	40	16	24	1	992	992
1	tekst	20	20	24	5	674	672
2	tekst	20	10	12	5	424	420
3	graf.	40	20	24	4	434	432
4	graf.	80	40	48	2	694	696
5	graf.	80	40	48	4	1174	1176
6	graf.	160	80	96	2	2174	2184
7	graf.	160	80	96	4	4190	4200
8	graf.	320	160	192	1	8112	8138
9	graf.	80	-	192	1	-	8138
10	graf.	80	-	192	9	-	8138
11	graf.	80	-	192	16	-	8138
12	graf.	40	20	24	5	1154	1152
13	graf.	40	10	12	5	664	660
14	graf.	160	160	192	2	4270	4296
15	graf.	160	160	192	4	8112	8138

TABELA 5.1.

Możliwe są następujące tryby graficzne pochodne od przedstawionych w tabeli 5.1. Są to tryby o numerach:
n + l6 -jak n ale bez okienka tekstowego
n +32 -jak n ale bez kasowania ekranu
n + 16 + 32 - kombinacja n + 16 i n + 32
gdzie: n od 0 do 15 określa tryb przedstawiony w tab. 5.1. Do wyboru graficznego służy w ATARI BASIC instrukcja:

GRAPHICS n

5.1. Kolory

ATARI ma pięć tzw. rejestrów kolorów oznaczonych numerami od 0 do 4. Odpowiednim obszarom ekranu jak tło, ramka, czy tekst przyporządkowane są rejestry określające ich kolor i jasność. Jasność może być określona poprzez liczbę parzystą z przedziału od 0 do 14. Tak więc dysponujemy ośmioma poziomami jasności. Możliwe do uzyskania kolory wraz z ich kodami zostały podane w tabeli 5.2.

Kolor	Kod
szary	0
jasnopomarańczowy	1
pomarańczowy	2
czerwono-pomarańczowy	3
różowy	4
purpurowy	5
purpurowo-niebieski	6
niebieski	7
niebieski	8
jasnoniebieski	9
turkusowy	10
zielono-niebieski	11
zielony	12
żółto-niebieski	13
pomarańczowo-zielony	14
jasnopomarańczowy	15

TABELA 5.2.

Do zmiany koloru i poziomu jasności zapisanych w określonym rejestrze koloru służy instrukcja:

SETCOLOR r, c, b

gdzie:
r - numer rejestru koloru (stała lub zmienna liczbowa z zakresu 0 - 4)
c - kod koloru (patrz tabela 5.2)
b - poziom jasności (parzyste z przedziału 0 - 14)

W trybach graficznych (od 3 do 15) określonemu rejestrowi koloru możemy przypisać wszystkie instrukcje **PLOT** tj. instrukcje **PLOT** będą rysować punkty o kolorach, których atrybut zawiera dany rejestr.

Można tego dokonać przy użyciu instrukcji:

COLOR r

gdzie:
r - numer rejestru (stała lub zmienna liczbowa)

Przy korzystaniu z możliwości graficznych **ATARI** może być użyteczna instrukcja:

LOCATE x, y, zmienna

Instrukcja ta czyta zawartość komórki ekranu o współrzędnych (x, y) i przypisuje ją wyszczególnionej zmiennej. Możliwe jest to po wcześniejszym otwarciu ekranu do czytania lub do czytania i pisania.

UWAGA: otwieranie ekranu kasuje jego zawartość, więc gdy w programie będziemy wypisywać coś na ekranie, a później włączać instrukcję **LOCATE** należy na samym początku otworzyć ekran do czytania za pomocą instrukcji:

OPEN#6, 4, 0, "S: " (O. #6, 4, 0, "S: ")

lub

OPEN#6, 12, 0, "S: "

5.2. Opis trybów graficznych

Opiszemy obecnie bardziej szczegółowo wybrane tryby graficzne i zaprezentujemy szerokie możliwości graficzne jakimi dysponuje ATARI.

TRYB GRAFICZNY 0

Jest normalnym, roboczym trybem graficznym pracy ATARI. Ekran podzielony jest na 24 wiersze i 40 kolumn. Ten tryb jest automatycznie wybierany po włączeniu komputera. W trybie tym ilość kolorów została zredukowana do jednego. Rejestry kolorów mają następujące znaczenie

1 - tekst (tylko poziom jasności)
2 -tło
3 - ramka

TRYBY GRAFICZE 1 i 2

Umożliwiają wyświetlenie tekstu dwa lub cztery razy większego od normalnego. Do wydruku tekstu poza okienkiem służy instrukcja:

PRINT#6; "....." (PR.#6; "....." lub **?#6; ".....")**

Znaczenie rejestrów kolorów jest następujące:

0 - tekst na ekranie
1 - tekst w okienku
2 - tło w okienku
3 - tło ekranu i ramka

TRYBY GRAFICZNE od 3 do 8 i od 12 do 15.

Tryby te służą do tworzenia ekranu graficznego, tzn. takiego na którym możemy umieszczać pojedyncze punkty (o wymiarach wynikających z wybranego trybu graficznego) i rysować linie.

W ATARI BASIC do tworzenia rysunków służą instrukcje **PLOT** i **DRAWTO**. Instrukcja

PLOT x,y (PL. x,y)

służy do umieszczenia na ekranie punktu o współrzędnych x y.
Instrukcja:

DRAWTO x,y (DR. x,y)

gdzie x, y - jak wyżej, rysuje linię od punktu o współrzędnych określonych ostatnią instrukcją **PLOT** lub od punktu, na którym skończyła się linia rysowania instrukcją **DRAWTO** od punktu x, y

```
15 GR.8: COLOR 5
25 PLOT 95, 25
35 DR.255, 25: DR.225, 155: DR.95, 155: DR.95, 25
45 GO TO 45
```

TRYB GRAFICZNY 9

Umożliwia uzyskanie aż szesnastu poziomów jasności jednego koloru. Jest to bardzo użyteczne do tworzenia jednobarwnych trójwymiarowych grafik, gdzie stopniowe cieniowanie może dać bardzo realistyczne efekty. Kolor tła jest określany w rejestrze 4, natomiast jasność punktów określa się instrukcją COLOR.

TRYB GRAFICZNY 10

Tryb 10 daje możliwości uzyskania dziewięciu kolorów, z których każdy może mieć dowolną jasność. Kolor punktów określa się przez użycie instrukcji **COLOR**. W przypadku gdy chcemy wykorzystać więcej niż pięciu kolorów, instrukcja **SETCOLOR** jest niewystarczająca. Wyboru żądanego koloru (C) oraz jasności (B)

dokonuje się bezpośrednio przez wpisanie odpowiednich wartości do komórek pamięci o numerach od 708 do 712 za pomocą instrukcji **POKE**. Liczbę, którą należy wpisać do pamięci, można obliczyć ze wzoru:

KOLOR=KOD KOLORU * 16 + POZIOM JASNOSCI

czyli kolor ustawiamy instrukcją

POKE L, C *16+B

TRYB GRAFICZNY 11

Tryb 11 umożliwia wprowadzenie 16 kolorów ale o jednakowej jasności. Kolor punktów rysowania instrukcją **PLOT** wybiera się instrukcją . Tryb ten jest użyteczny np. do rysowania wykresów i tabel.

5.3. Przykłady korzystania z możliwości graficznych

Drukowanie powiększonych liter. Pierwszy prezentowany przykład programu graficznego ilustruje przedstawienie na ekranie dużych kolorowych liter przy pomocy trybów graficznych 1 lub 2. Dla wyświetlenia informacji w obszarze graficznym ekranu używamy instrukcji **PRINT#6**. W tym przypadku nie musimy otwierać kanału komunikacji **OPEN**. Robi to za nas system operacyjny w chwili gdy jest wykonywana instrukcja **GRAPHICS**.

Podobnie jak w trybie graficznym 0, litery możemy drukować w ściśle określonym miejscu ekranu używając instrukcji **POSITION**, jednakże należy uważać, gdyż zakres zmian współrzędnych x i y jest znacznie ograniczony. Zakres ich zmian podano w tabeli 5.3.

Tryb graficzny	Rozmiar ekranu (bez okienka tekstowego)
0	40 X 24
1	20 X 24 (21)
2	20 X 12 (11)

TABELA 5.3.

Kolory na ekranie zmienia się instrukcją:

SETCOLOR

Przedstawiony poniżej program symuluje grę w POKERA dla dwóch graczy:

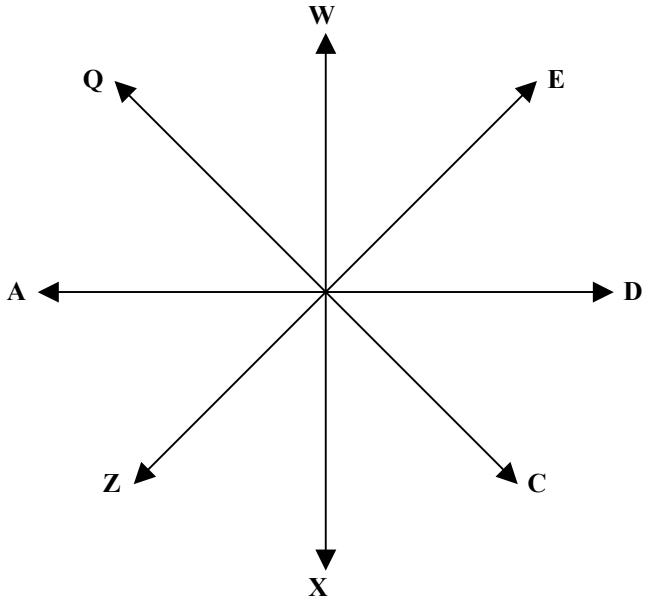
PROGRAM "POKER"

```
5 REM PROGRAM "POKER"
10 OPEN # 1, 4, 0, "K"
15 DIM A$(20), C$(6), N$(10), R$(6)
20 C$="9TJQKA": A$="
25 GRAPHICS 1
30 SETCOLOR 2, 0, 0
35 POSITION 7, 9: PRINT #6; "POKER"
40 FOR J=1 TO 2
45 PRINT "          WPISZ NAZWE GRACZA"
50 INPUT N$: A$(J*10-9, J*10)=N$
55 NEXT J
60 PRINT "          START PO NACISNIECIU SPACE"
65 GET #1, A
70 IF A<>32 THEN 65
75 GRAPHICS 2
```

```
80 FOR J=1 TO 2
85 SETCOLOR 2, O, O: SETCOLOR 0, 4*J+2, 10
90 POSITION 5, 2: PRINT #6; A$(J*10-9, J*10)
95 PRINT CHR$(125)
100 FOR I=1 TO 5: GOSUB 200: NEXT I
105 FOR K=1 TO 2
110 PRINT "          WPISZ NUMERY KART ZAMIENIANYCH"
115 INPUT R$
120 IF LEN(R$)=0 THEN 145
125 I=ASC(R$)-48
130 IF I>0 AND I<6 THEN GOSUB 200
135 IF LEN(R$)>1 THEN R$=R$(2): GOTO 125
140 NEXT K
145 PRINT "NACISNIJ SPACE"
150 GET #1, A
155 IF A<>32 AND J=1 THEN 150
160 NEXT J
165 IF A=32 THEN 80
170 PRINT "KONIEC GRY": END
200 C=INT (RND(0)*6+1)
205 POSITION 2*I+3, 5: PRINT #6; C$(C, C)
210 RETURN
```

Proste rysunki. Jak wspomniano tryby graficzne ATARI umożliwiające użycie ekranu graficznego różnią się rozdzielczością, ilością kolorów. Możemy używać grafiki o dużej rozdzielczości, ale odbywa się to w znacznym stopniu kosztem pamięci dostępnej dla programu np. grafika o rozdzielczości 320x192 punkty zajmuje 8138 bajtów czyli mniej więcej tyle ile interpreter ATARI BASIC.

Przedstawione poniżej programy ilustrują użycie prostych instrukcji do rysowania wykresów. Pierwszy tworzy ciekawy wzór, natomiast drugi umożliwia tworzenie obrazów poprzez rysowanie krzywych pod kontrolą użytkownika. Kierunek rysowania może być wybrany przez naciśnięcie jednego z klawiszy jak na poniższym rysunku:



PROGRAM "RYSUNEK"

```
5 REM PROGRAM "RYSUNEK"
10 GRAPHICS 8+16
15 SETCOLOR 2, 0, 0
20 M=160: P=70: Q=0
25 COLOR 1
30 FOR I=5 TO 1 STEP -1
35 FOR J=0 TO M STEP I
40 PLOT P+J, Q
45 DRAWTO P+M, Q+J
50 DRAWTO P+M-J, Q+M
55 DRAWTO P, Q+M-J
60 DRAWTO P+J, Q
65 NEXT J
70 M=M/2: P=P+M/2: Q=Q+M/2
75 NEXT I
80 GOTO 80
```

PROGRAM "MAŁY MALARZ"

```
5 REM PROGRAM "MALY MALARZ"
10 GRAPHICS 7
15 B=1: T=0
20 XX=159: YY=79: X=80: Y=40
25 SETCOLOR 2, 0, 0
30 SETCOLOR 0, 0, 15
35 COLOR' B
40 OPEN #1, 4, 0, "K"
45 PRINT "NACISNIECIE SPACE ZMIANA KOLORU DRUKU"
50 PRINT "NACISNIECIE TAB OTRZYMASZ WSPOLRZEDNE"
55 PLOT X, Y
60 GET # 1, K
65 PRINT : PRINT : PRINT
70 IF T=0 THEN PRINT "X="; X; "Y="; Y
75 IF X>0 AND (K=65 OR K=81 OR K=90) THEN X=X-1
80 IF X<XX AND (K=67 OR K=68 OR K=69) THEN X=X+1
85 IF Y>0 AND (K=81 OR K=87 OR K=69) THEN Y=Y-1
90 IF Y<YY AND (K=90 OR K=88 OR K=67) THEN Y=Y+1
95 IF K=32 THEN B=5-B: COLOR B
100 IF K=127 THEN T=1-T: K=0: GOTO 65
105 GOTO 55
```

Jedną z zalet ATARI jest możliwość zmian kolorów, co w połączeniu z rysunkami tworzy wrażenie ruchu. Ilustrują to dwa programy:

PROGRAM "SENNY DZIEN"

```
5 REM PROGRAM "SENNY DZIEN"
10 GRAPHICS 7+16
15 DIM A$(4), B$(1), C(4)
20 C(1)=0: C(2)=5: C(3)=12: C(4)=15
25 C=0: A$="0123"
30 FOR J=1 TO 159
35 COLOR C
40 C=C+0.5: IF C>3 THEN C=0
45 PLOT J, 0: DRAWTO 159-J, 95
50 NEXT J
```

```
55 FOR J=0 TO 95
60 COLOR C
65 C=C+0.5: IF C>3 THEN C=0
70 PLOT 0, 95-J: DRAWTO 159, J
75 NEXT J
80 B$=A$(1): A$=A$(2): A$(4)=B$
85 FOR J=1 TO 4
90 SETCOLOR VAL(A$(J, J)), C(J), 8
95 FOR T=1 TO 15: NEXT T
100 NEXT J
105 GOTO 80
```

PROGRAM "RUCH"

```
5 REM PROGRAM "RUCH"
10 GRAPHICS 7+16
15 DIM C(5)
20 FOR X=1 TO 5
25 READ D: C(X)=D
30 SETCOLOR X-1, D, 4
35 NEXT X
40 SETCOLOR 4, 0, 0
45 I=1
50 FOR Y=0 TO 30 STEP 2
55 COLOR I
60 I=I+1:  IF I>3 THEN I=1
65 FOR X=0 TO 1.7 STEP 0.06
70 C=COS(X): S=SIN(X)
75 PLOT 79+Y*C, 40+32*S
80 PLOT 79+Y*C, 40-32*S
85 PLOT 79-Y*C, 40+32*S
90 PLOT 79-Y*C, 40-32*S
95 NEXT X
100 NEXT Y
105 FOR X=1 TO 3
110 FOR Y=0 TO 2
115 SETCOLOR Y, C(X+Y), 4
120 NEXT Y
125 FOR T=1 TO 50: NEXT T
130 GOTO 105
140 DATA 7, 15, 5, 7, 15
```

Rysunki trójwymiarowe. Kolejne przedstawione programy ilustrują technikę tworzenia realistycznych rysunków trójwymiarowych.

Pierwszy z nich rysuje stożek wykorzystując możliwości dziewiątego trybu graficznego do cieniowania jego boków przy użyciu odpowiedniej jasności co daje pewne efekty trójwymiarowe. Następny program pokazuje możliwości rysowania wykresów dwóch funkcji. Ponieważ programy wykonują się dość długo, może się zdarzyć, że ATARI przejdzie do trybu "przyciągania uwagi". Dzieje się tak gdy przez kilka minut nie zostanie nacisinięty żaden klawisz. W tym trybie ekran zmienia kolory. Tryb ten zapobiega wypalaniu się luminoforu jednego koloru w kolorowym kineskopie. Tryb "przyciągania uwagi" może być skasowany przez naciśnięcie dowolnego klawisza z wyjątkiem BREAK.

PROGRAM "TRÓJWYMIAROWY STOŻEK"

```
5 REM PROGRAM "TROJWYMIAROWY STOZEK"
10 GRAPHICS 9
15 SETCOLOR 4, 15, 0
20 X=35: Y=0
```

```
25 FOR I=4.55 TO 4.87 STEP 6.0E-03
30 C=INT(ABS(4.8-I)*60)
35 COLOR 15-C
40 PLOT X, Y
45 DRAWTO X+150*COS (I), Y-150*SIN(I)
50 NEXT I
55 GOTO 55
```

PROGRAM "WYKRES TRÓJWYMIAROWY"

```
5 REM PROGRAM "WYKRES TROJWYMIAROWY"
10 GRAPHICS 8+16
15 SETCOLOR 2, 0, 0
20 COLOR 1
25 DEG: A=30: S=SINA)
30 FOR X=0 TO 127
35 Y=SQR(16129-X*X)
40 MAX= -1E+20
45 FOR Z=-Y TO Y STEP 4
50 V=25*SIN(0.05*(X*X+Z*Z))
55 P=88+V+Z*S
60 IF P<=MAX THEN 75
65 MAX=P
70 PLOT 160+X, 210-P: PLOT 160-X, 210-P
75 NEXT Z
80 NEXT X
85 GOTO 85
```

Generator znaków. W pamięci ATARI wydzielony został pewien fragment na tzw. generator znaków. Zawiera on informacje o wszystkich znakach używanych przez komputer. Pojedynczy znak reprezentowany jest przez zestaw 8 bajtów. Każdy bit mający wartość logiczną "1" reprezentuje sobą jasny punkt w siatce 8x8, w której zawiera się każdy znak. Generator znaków zaczyna się od adresu 57344. Aby znaleźć adres danego znaku należy skorzystać ze wzoru:

adres = 57344 + (k-32)x 8

gdzie k jest kodem ATASCII poszukiwanego znaku. Na przykład adres litery A wynosi:

adres = 57344 + (65-32)x 8 = 57608

Następny program pokazuje powiększone znaki, adresy poszczególnych bajtów i ich wartości dziesiętne.

PROGRAM "POWIĘKSZONE ZNAKI"

```
5 REM PROGRAM "POWIEKSZONE ZNAKI"
10 OPEN #1, 4, 0, "K"
15 A=57344
20 FOR I=A TO A+1024 STEP 8
25 PRINT CHR$(125)
30 FOR J=I TO I+7
35 PRINT J,
40 V=PEEK (J)
45 KK=256
50 FOR K=7 TO 0 STEP -1
55 KK=KK/2
60 IF V<KK THEN PRINT CHR$(32); : GOTO 70
65 PRINT CHR$(160); :  V=V-KK
70 NEXT K
75 PRINT
```

```
80 NEXT J
85 POSITION 5, 12:  PRINT "NACISNIJ SPACE"
90 GET #1, X
95 NEXT I
```

Generowanie własnych znaków. Poniżej przedstawiamy program, który umożliwia tworzenie własnych znaków. Program działa następująco:

Linie 20, 30 ta część programu przypisuje generator znaków do obszaru pamięci zaczynającego się od adresu 32768 i wielkości 1024 bajty (l k).

linia 230 przez użycie instrukcji POKE zmieniamy niektóre ze znaków generatora.

Tak więc program tworzy niejako drugi, poprawiony generator znaków. Aby komputer przełączyć na ten generator należy wykonać instrukcję:

POKE 756, 128

Powrotu do generatora standardowego dokonuje się przez instrukcję:

POKE 756, 224

Wszystkie nowo wpisane znaki będą osiągane przez naciśnięcie CONTROL-A, CONTROL-B itd.

Wartość poszczególnych linii nowo tworzonego znaku należy podawać jako liczby dziesiętne.

PROGRAM "GENERATOR ZNAKÓW'

```
1 REM PROGRAM "GENERATOR ZNAKOW"
10 PRINT CHR$(125): POSITION 10, 10: PRINT "POCZEKAJ 15 SEKUND"
15  PRINT :  PRINT
20 N=57344: M=32768
30 FOR I=0 TO 1023: POKE M+I,  PEEK (N+I):  NEXT I
100 M=33288
110 PRINT: PRINT: PRINT "PODAJ ILOSC DEFINIOWANYCH ZNAKOW=";: INPUT N
145 POSITION 24, 22
200 FOR F=1 TO N
205 PRINT CHR$(125):  PRINT "ZNAK"; F
206 FOR T=4 TO 11:  POSITION 25, T:  PRINT".....": NEXT T
210 FOR Z=0 TO 7
215 POSITION 24, 22: PRINT "              "
220 POSITION 2, 22:  PRINT "PODAJ WARTOSC",  Z+1;  "LIN I";
230 INPUT Q: POKE M, Q:  GOSUB 1000: M=M+1
240 NEXT Z
245 POSITION 2, 22:  PRINT "**** NACISNIJ RETURN****"
250 IF PEEK (764)<>12 THEN 250
255 POKE 764, 255
260 NEXT F
270 END
1000 K=256
1010 POSITION 20, 4+Z: PRINT Q: POSITION 25, 4+Z
1020 FOR P=7 TO 0 STEP -1
1030 K=K/2:  IF Q<K THEN PRINT CHR$(32); :  GOTO 1050
1040 PRINT CHR$(160); : Q=Q-K
1050 NEXT P
1070 RETURN
```

Drugim programem jest generator polskich liter na ATARI. Działa on na tej samej zasadzie co program poprzedni. Aby włączyć polskie litery należy, tak jak poprzednio, przełączyć komputer na nowy generator znaków. Litery ń, ć, ś, ę, ą, ó, ź, ż, ł uzyskamy przez naciśnięcie CONTROL-N, CONTROL-C, CONTROL-S, CONTROL-E itd. Literę "ź' uzyskamy przez naciśnięcie CONTROL -.

Wygenerowane znaki pozostają mimo wykonania instrukcji NEW. Naciśnięcie klawisza RESET przełącza komputer na podstawowy generator znaków, ale nie wymazuje stworzonych polskich liter.

PROGRAM "GENERATOR POLSKICH LITER"

```
5 REM PROGRAM "GEN. POLSKICH ZNAKOW"
10 N=57344:  M=32768
20 FOR I=0 TO 1023:  POKE M+I,  PEEK (N+I):  NEXT I
30 A=M+776:  B=M+520
40 FOR I=0 TO 207:  POKE B+I,  PEEK (A+I):  NEXT I
50 A=M+976:  B=M+512
60 FOR I=0 TO 7: POKE B+I,  PEEK (A+I): NEXT I
70 POKE M+513, 12
71 POKE M+514, 24
72 POKE M+515, 124
73 POKE M+527, 3
74 POKE M+537, 12
75 POKE M+538, 24
76 POKE M+539, 60
77 POKE M+559, 6
78 POKE M+611, 28
79 POKE M+612, 56
80 POKE M+625, 12
81 POKE M+626, 24
82 POKE M+627, 124
83 POKE M+633, 12
84 POKE M+634, 24
85 POKE M+635, 60
86 POKE M+664, 6
87 POKE M+665, 12
88 POKE M+721, 24
89 POKE M+722, 0
90 POKE M+723, 124
95 POKE 756, 128
```

5.4. Kursor

Sterowanie kursorem, zarówno ustawienie jego pozycji, jak i zmiana jego działania, jest istotne nie tylko przy tworzeniu gier, ale i w poważnych programach użytkowych takich jak np. bazy danych, czy programy typu VISI-CALC. Ustawienie kursora na dowolnym miejscu ekranu umożliwia instrukcja:

POSITION x, y

gdzie x, y - współrzędne punktu na ekranie (stałe lub zmienne liczbowe).

Instrukcja ta ustawia kursor w punkcie o współrzędnych x, y. Wykonanie instrukcji **POSITION 0, 0** odpowiada ustawieniu kursora w lewym górnym rogu ekranu. natomiast poprzez zmianę zawartości komórki 755 możemy zmienić działanie kursora. Poprzez wykonanie instrukcji

POKE 755, n

gdzie n jest z przedziału od 0 do 7 można uczynić kursor niewidocznym, bądź spowodować, że wprowadzane znaki będą odwrócone tj. "do góry nogami". Tabela 5.3. demonstruje efekty wynikające ze zmiany wartości n.

Wartość n	kursor	znaki
0	niewidoczny-przeźroczysty	normalne
1	niewidoczny-nieprzeźroczysty	normalne
2	widoczny-przeźroczysty	normalne
3	widoczny-nieprzeźroczysty	normalne
4	niewidoczny-przeźroczysty	odwrócone
5	niewidoczny-nieprzeźroczysty	odwrócone
6	widoczny-przeźroczysty	odwrócone
7	widoczny-nieprzeźroczysty	odwrócone

TABELA 5.4.

5.5. PEEK i POKE w operacjach graficznych

Jak dotąd poznaliśmy dwa sposoby zmiany informacji wyświetlanych na ekranie. Jeden z nich to przesunięcie kursora do wybranej pozycji za pomocą znaków sterujących kursorem, następnie wpisywanie odpowiedniego tekstu. Drugi to użycie instrukcji **POSITION**, a następnie **PRINT**. Istnieje jeszcze jedna możliwość - zmiana informacji wyświetlanych na ekranie przez bezpośrednie wpisanie w pamięć monitora. Możemy tego dokonać poprzez użycie instrukcji:

POKE

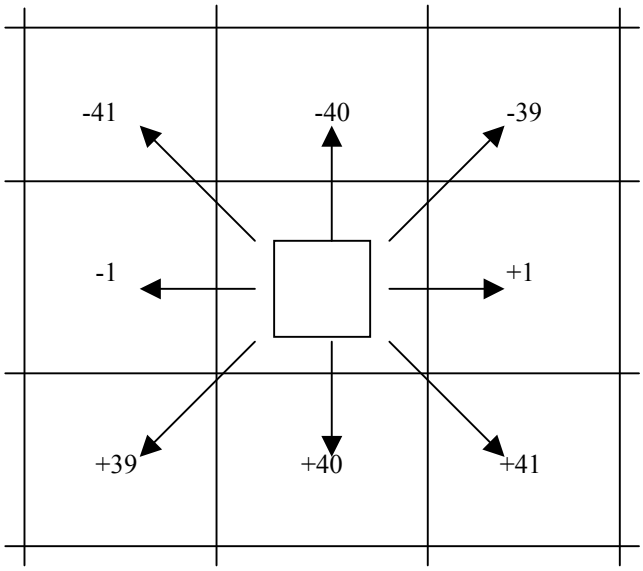
Nic ma ściśle określonych zasad kiedy należy korzystać z instrukcji **PRINT**, a kiedy używać **POKE** do zmiany wyświetlanych informacji, ale za to z dużą szybkością. Dzięki instrukcji **POKE** można niewielkie obiekty przemieszczać po ekranie tak szybko, że daje to wrażenie animacji. Jest to często wykorzystywane do tworzenia gier zręcznościowych.

Przy użyciu funkcji **PEEK** możemy sprawdzić w trakcie wykonywania programu, jaki znak graficzny został przyporządkowany danemu punktowi ekranu.

Jak pamiętamy ekran ATARI podzielony jest na 24 wiersze i 40 kolumn co w sumie daje 960 pozycji. 7awartoSC ekranu jest zapisywana w pamięci RAM wiersz po wierszu. Punkt znajdujący się w lewym górnym rogu ma adres zapisany w komórkach 88 i 89. Adres tego punktu wyprowadzany jest przy pomocy wzoru:
SM=PEEK(88)+256'PEEK(89)
Natomiast adres dowolnego punktu wyznacza się za pomocą:

ADRES=SM+c+40*r

gdzie: c- numer kolumny, r - numer wiersza. zwróćmy uwagę, że kody wpisane do pamięci monitora nie są kodami ATASCII. Noszą one nazwę ATARI PEEK/POKE i są podane w dodatku B.



Rysunek 5.2.

Jeśli znamy adres punktu na ekranie, przesunięcie go do jednej z sąsiednich pozycji nie stwarza żadnych trudności. Przesunięcie w lewo lub w prawo otrzymujemy przez dodanie -1 lub 1 do adresu, natomiast w górę lub w dół przez dodanie -40 lub 40, a przesunięcie pod kątem przez kombinację obu przesunięć.

Jeżeli używamy tej techniki do przesuwania punktu, ważne jest, aby na stare miejsce wpisać spację, chyba że chcemy aby znak był zarówno na starym miejscu jak i na nowym.

Poniżej przedstawiamy przykład gry wykorzystującej nowo poznane techniki tworzenia obrazu. W grach bardzo często konieczne jest wprowadzanie informacji z klawiatury. Można to zrobić używając instrukcji INPUT. Jednak powoduje to zatrzymanie programu do chwili naciśnięcia klawisza RETURN. W wielu grach jest to nie do przyjęcia. Można stosować wówczas inną metodę. W ATARI istnieje wskaźnik o nazwie CH (adres 764), który daje nam informację o ostatnim naciśniętym klawiszu. Testując ten wskaźnik przez funkcję PEEK(764) otrzymamy kod klawisza, który został ostatnio naciśnięty. Pełna lista klawiszy została podana w dodatku C.

Gra "Wążżż" polega na tym, że grający musi sterować wężem jedzącym pożywienie. Gra kończy się w momencie, gdy wąż zacznie jeść sam siebie. Z czasem gdy wąż staje się coraz dłuższy trudniej nim sterować. Aby przesunąć węża zgodnie ze wskazówkami zegara należy nacisnąć klawisz "M", a przeciwnie do ruchu wskazówek zegara klawisz "N".

PROGRAM "WĄŻŻŻ"

```
10 REM PROGRAM "WAZZZ"
20 M=1000: HS=0
25 DIM S(M),  A(4)
30 A(1)=40: A(2)=1: A(3)=-40: A(4)=-1
35 POKE 755, 0
40 SM=PEEK(88)+256*PEEK(89)
45 SC=0: T=SM+496: H=SM+500: L=5: J=2
50 PRINT CHR$(125)
55 FOR K=SM TO SM+39
60 POKE K, 10: POKE K+880, 10: NEXT K
65 FOR K=SM TO SM+960 STEP 40
70 POKE K, 10: POKE K-1, 10: NEXT K
75 FOR K=T TO H
80 POKE K, 47: NEXT K
85 FOR K=1 TO L
90 S(K)=T+K-I: NEXT K
95 PH=L+I: PT=1
100 POSITION 2, 23: PRINT "WYNIK "; SC;
105 POSITION 14, 23: PRINT "DLUGOSC "; L;
110 POSITZON 29, 23: PRINT "MAX '; HS;
115 K=PEEK(764): POKE 764, 255
120 IF K=35 THEN J=J+1: IF J=5 THEN J=1
125 IF K=37 THEN J=J-1: IF J=0 THEN J=4
130 H=H+A(J)
135 SQ=PEEK(H)
140 ZF SQ=47 OR SQ=10 THEN 220
145 IF SQ=0 THEN 155
150 V=SQ-16: F=F+V: SC=SC+CV: CV=0: L=L+V
155 S(PH)=H: PH=PH+1
160 POKE H, 47
165 IF F<>0 THEN F=F-I: GOTO 175
170 POKE S(PT), 0: PT=PT+1
175 IF PT>M THEN PH=1
180 IF PH>M THEN PH=1
185 IF RND(0) <0.01 THEN CV=9
190 IF CV<10 THEN CV=0: IF PEEK(CP)<>47 THEN POKE CP, 0
195 IF CV<>0 THEN CV=CV-I: POKE CP, INT (CV/10)+16: GOT0 210
200 IF RND (0)<0.8 THEN 210
205 CV=100: CP=SM+940*RND(0):  IF PEEK(CP)<>0 THEN 205
```

```
210 FOR T=1 TO 25: NEXT T
215 GOTO 100
220 IF SC>HS THEN HS=SC
225 POSITION 2, 23: PRINT "WYNIK "; SC;
230 FOR T=1 TO 600: NEXT T
235 GOTO 45
```

5.6. Wykorzystanie manipulatorów

Z rozbudowanymi możliwościami graficznymi ATARI związane są nierozłącznie gry, z grami zaś użycie manipulatorów. Jako manipulatory najczęściej używane są tzw. joysticki lub paddle (brak jest jednoznacznych polskich nazw i dlatego będziemy używać nazw angielskich). Joystick - to manipulator wyposażony w drążek, natomiast paddle w pokrętło. Obydwa mają ponadto klawisz tzw. FIRE nazywany tak gdyż w grach najczęściej realizuje on funkcję strzelania. ATARI serii XL i XE posiada dwa porty do przyłączenia manipulatorów, natomiast w ATARI BASIC istnieją funkcje umożliwiające wygodne pisanie programów korzystających z manipulatorów.

Do obsługi joysticka służą następujące funkcje:

STICK(n)

gdzie: n - numer portu 0 lub 1, do którego podłączony jest manipulator (stała lub zmienna liczbowa). Jako wynik otrzymujemy liczbę zależną od pozycji joysticka tak jak pokazuje poniższy rysunek:

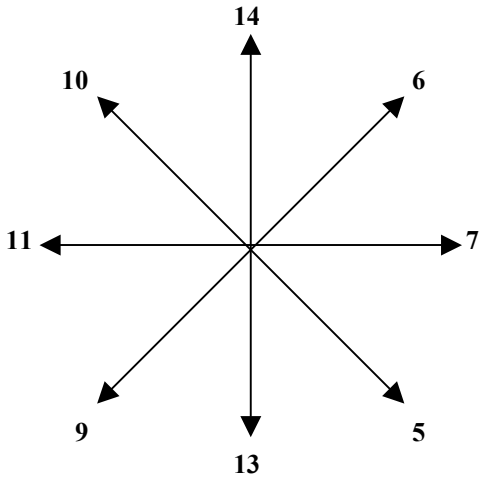
STRIG (n)

gdzie: n - jak poprzednio. Jako wynik uzyskujemy 1 dla klawisza FIRE nie wciśniętego i 0 dla wciśniętego.

Do obsługi manipulatora paddle mamy do dyspozycji funkcję:

PADDLE(n)

gdzie: n -jak w przypadku joysticka. Ale może mieć wartości od 0 do 3 gdyż do jednego portu można podłączyć dwa paddle. Wynik działania tej funkcji to liczba całkowita od 1 do 228 zależna od położenia pokrętła manipulatora. Oraz funkcję:



Rysunek 5.3.

PRTIG (n)

której działanie jest takie samo jak funkcji STRIG dla joysticka, czyli otrzymujemy 1 dla nie wciśniętego klawisza FIRE a 0 dla wciśniętego.

Co zrobić jeżeli nie dysponujemy joystickiem? Istnieje proste wyjście, jeżeli gra napisana jest w BASIC-u. Wystarczy wtedy odszukać miejsca gdzie wywołane są funkcje obsługi joysticka i zastąpić je wg schematu:
zamiast zmienna = STICK(n)
wpisać GOSUB 9000: zmienna= ISTICK zamiast zmienna= STRIG(n)
wpisać GOSUB 9000 zmienna=ISTRIG
i dopisać do programu podany poniżej podprogram "Joystick".

PODPROGRAM "JOYSTICK"

```
9000 REM PROGRAM "JOYSTICK"
9010 ISTICK=PEEK(764)
9020 ISTRIG=1 9030 POKE 764, 255
9040 IF ISTRICK=14 THEN ISTRICK=14: RETURN
9050 IF ISTRICK= 7 THEN ISTRICK= 7: RETURN
9060 IF ISTRICK=15 THEN ISTRICK=13: RETURN
9070 IF ISTRICK= 6 THEN ISTRICK=11: RETURN
9080 IF ISTRICK=33 THEN ISTRICK=15: ISTRIG=0: RETURN
9090 ISTRICK=15: RETURN
```

Poniżej przedstawiamy jedną z zaawansowanych technik wykorzystania możliwości graficznych ATARI. technika ta pozwala na szybkie przełączanie obszaru pamięci ekranu na inne miejsce (ang. page flipping). Daje to możliwość np. animacji obrazu przez szybką zmianę zawartości całego ekranu. Korzysta się tu z rejestru przesunięcia pionowego obrazu (ang. vertical scrolling register). Należy pamiętać, że w przedstawionym programie musi być używany tryb graficzny 0, w przeciwnym razie otrzymamy dziwaczny efekt na ekranie.

PROGRAM "SZYBKA WYMIANA EKRANÓW"

```
1 REM PROGRAM "SZYBKA WYMIANA EKRANOW"
5 GOTO 50
10 DLIST=PEEK(560)+256*PEEK(561): POKE 2, 0
20 L=DLIST+4
25 H=DLIST+S
30 A=ADR(A$): B=INT(A/256): C=A-(256*B)
31 D=ADR(B$): E=INT(D/256): F=D-(256*E)
32 POKE L, C: POKE H, B: FOR I=1 TO 500: NEXT I
33 POKE L, F: POKE H, E: FOR I=1 TO 500: NEXT I: IF PEEK(764)=255 THEN 32
34 GRAPHICS 0: END
50 DIM A$(960),  B$(960)
55 A$=CHR$(3): A$(960)=CHR$(3): A$(2)=A$: REM WYPELNIANIE PIERWSZEGO EKRANU
56 B$=CHR$(4): BS(960)=CHR$(3): A$: REM WYPELNI DRUGIEGO EKRANU
60 GOTO 10
```

Rozdział 6

DŹWIĘK

Mikrokomputer ATARI ma bardzo duże możliwości dźwiękowe. W rozdziale tym zajmiemy się jedynie zagadnieniem możliwości muzycznych tego komputera. Wiadomo, że dochodzenie do perfekcji gry na dowolnym instrumencie trwa latami. Również opanowanie możliwości muzycznych ATARI wymaga czasu.

6.1. Generowanie dźwięku

Dźwięki są otrzymywane za pomocą instrukcji **SOUND**. Posiada ona cztery parametry i działa dopóki nie zostanie wykonany następny **SOUND** lub **END, RUN, NEW**. Postać instrukcji jest następująca:

SOUND c, p, d, v

gdzie:
c - jest numerem generatora (od 0 do 3)
p - jest okresem generowanego dźwięku (od 0 do 255)
d - jest brzmieniem (parzyste liczby od 0 do 14)
v - jest głośnością (0 do 15)

ATARI ma cztery niezależne generatory dźwięku (które czasami nazywane są głosami lub kanałami), tak więc w instrukcji **SOUND** musimy podać informację, który generator ma być uruchomiony. Możemy jednocześnie uruchomić wszystkie generatory w zależności od tego ile głosów chcemy wykorzystać.

Okres generowanego dźwięku określa jego wysokość i jest odwrotnie proporcjonalny do częstotliwości. Dokładna zależność między tymi dwoma wartościami jest następująca: częstotliwość = 31960 (okres+ 1)

Tony niskie		Tony średnie		Tony wysokie	
Dźwięk	Okres (p)	Dźwięk	Okres (p)	Dźwięk	Okres (p)
C	243	C	121	C	60
C#	230	C#	114	C#	57
D	217	D	108	D	53
D#	204	D#	102	D#	50
E	193	E	96	E	47
F	182	F	91	F	45
F#	173	F#	85	F#	42
G	162	G	81	G	40
G#	153	G#	76	G#	37
A	144	A	72	A	35
A#	136	A#	68	A#	33
H	128	H	64	H	31
				C	29

TABELA 6.1.

W tabeli 6.1 przedstawiono okresy (p) odpowiadające poszczególnym dźwiękom oktawy.

Dźwięk w ATARI może mieć różne brzmienie. Dla wartości 10 lub 14 otrzymujemy czyste tony. Dla innych wartości otrzymujemy dźwięki mniej lub bardziej zniekształcone.

Ostatnim parametrem jest głośność dźwięku z zakresu od 0 (wyłączony) do 15 (pełna głośność). Przedstawiony poniżej program wykorzystuje możliwości ATARI do odtworzenia melodii. Podane wartości okresu i czasu trwania dźwięku są zapisane w instrukcji DATA. Przez zmianę tych wartości można otrzymywać własne melodie.

PROGRAM "MELODYJKA"

```
50 REM PROGRAM "MELODYJKA"
51 GRAPHICS 2+16
55 POSITION 1, 5: PRINT #6; "PROGRAM: MELODYJKA"
100 READ A
150 IF A=999 THEN RESTORE:  GOTO 100
310 LET N=10
350 IF A=300 THEN GOTO 690
360 IF A=500 THEN N=0
400 SOUND 0, A, 10, N
690 FOR W=0 TO 50: NEXT W
700 DATA 40, 45, 47, 55, 300, 40, 300, 29, 300, 23
710 DATA 300, 26, 300, 26, 300, 300, 300, 29, 31
720 DATA 35, 300, 29, 300, 31, 300, 35, 300, 40
730 DATA 300, 300, 300, 300, 300, 35, 300, 45
740 DATA 300, 300, 300, 300, 300, 40, 300, 47, 55, 300
750 DATA 300, 300, 300, 300, 45, 48, 53, 3, 300
760 DATA 53, 3, 300, 35, 300, 35, 300, 40, 300, 300
770 DATA 300, 300, 300, 35, 300, 45, 300, 300, 300
780 DATA 300, 300, 40, 300, 47, 55, 300, 300, 300
790 DATA 300, 300, 45, 48, 53, 3, 300, 53, 3, 300
800 DATA 47, 55, 300, 47, 55, 300, 60, 300, 300, 300
810 DATA 300, 999
820 SOUND 0, A, 10, 0: SOUND 1, B, 10, 0
830 SOUND 2, C, 10, 0
850 GO TO 100
900 END
```

Dźwięk użyty w programie może stanowić dodatkową formę zwrócenia uwagi użytkownika na szczególne walory dźwiękowe ATARI. Na przykład podanie złych danych wejściowych do programu może być sygnalizowane jakimś nieprzyjemnym dźwiękiem. Również gry mogą być znacznie uatrakcyjnione przez dobrze dobrane efekty dźwiękowe.

Aby się o tym przekonać spróbujmy zagrać w grę "Ściana" najpierw bez dźwięku, a potem z dźwiękiem.

Zasadą tej gry jest zburzenie ściany składającej się z 240 cegieł przy użyciu piłki odbijanej kijem palantowym. Kij można przesuwać naciskając klawisze:

"M" - ruch w prawo "B" - ruch w lewo

Mamy pięć piłek. Jeśli nie trafimy kijem w piłkę tracimy ją.

PROGRAM "ŚCIANA"

```
5 REM PROGRAM "SCIANA"
10 GRAPHICS 0
15 SETCOLOR 2, 10, 0:  SETCOLOR 1, 10, 15: SETCOLOR 4, 4, 4
20 H=0
25 POS=PEEK(89)*256+PEEK(88)
30 B=5: S=0
35 GOSUB 200
40 GOSUB 300
45 IF Y=1 THEN GOSUB 400
50 IF X>20 THEN GOSUB 500: GOTO 100
60 IF Y+D>39 THEN GOSUB 700
65 IF P=29 THEN GOSUB 800
70 IF P=3 THEN GOSUB 900
75 GOSUB 1000
80 IF T<>240 THEN 40
```

```
85 GOTO 35
100 POSITION 4, 22: PRINT "WYNIK "; S;
105 FOR T=1 TO 500: NEXT T
110 IF B>0 THEN 35
115 POSITION 12, 12: PRINT "NACISNIJ RETURN"
120 IF PEEK(764)<>12 THEN 120
121 POKE 764, 1
125 PRINT CHR$(125) 130 IF S>H THEN H=S
135 GOTO 30
200 PRINT CHR$(125)
205 COLOR 35
210 POKE 755, 1
215 FOR I=2 TO 7
220 FOR J=0 TO 39
225 PLOT J, 1
230 NEXT J
235 NEXT I
240 POSITION 23, 22: PRINT "MAX WYNIK"; H;
245 T=0: Z=15: D=1
250 X=-18: XD=ABS(X)
255 Y=INT(RND(0)*78-39): Y0=ABS(Y)
260 RETURN
300 XX=ABS(X): YY=ABS(Y)
305 POSITION ZZ, 20: PRINT "     "
310 POSITION 2, 20: PRINT "=====";
315 POSITION 4, 22: PRINT "WYNIK"; S;
320 ZZ=Z
325 POKE POS+40*X0+Y0, 0
330 P=PEEK(POS+40*XX+YY)
335 POKE POS+40*XX+YY, 84
340 RETURN
400 PT=35: DR=2: GOSUB 1100
405 RETURN
500 PT=200: DR=50: GOSUB 1100
505 B=B-1
510 POSITION Z, 20: PRINT "===="
515 RETURN
600 PT=100: DR=2: GOSUB 1100
605 D=INT(3-RND(0)*3)
610 RETURN
700 PT=35: DR=2: GOSUB 1100
705 Y=-Y
710 RETURN
800 PT=100: DR=2: GOSUB 1100
805 X=-X
810 D=INT(3-RDN(0)*3)
815 RETURN
900 PT=60: DR=3: GOSUB 1100
905 S=S+1: X=-X
910 RETURN
1000 XO=ABS(XX): YO=ABS(YY): X=X+1: Y=Y+D
1005 K=PEEK(764)
1010 IF (STICK(0)= 7 OR K=37) AND Z<34 THEN Z=Z+2
1015 IF (STICK(0)=11 OR K=21) AND Z>1 THEN Z=Z-2
1020 RETURN
1100 SOUND 0, PT, 14, 15
1105 FOR T=1 TO DR: NEXT
1110 SOUND 0, 0, 0, 0 : RETURN
```

6.2. Techniki specjalne

W ATARI dźwięk jest generowany w specjalnym układzie zwanym "POKEY". Posiada on trzy obszary kontroli dźwięku. Pełne wykorzystanie dźwiękowych możliwości ATARI wymaga znajomości notacji binarnej na tyle dokładnie, aby można bez trudności ustalić lub wyzerować poszczególne bity w odpowiedniej komórce pamięci.

AUDF 1 do 4

Te zmienne, zlokalizowane w komórkach o adresach 53760, 53762, 53764 i 53766 decydują o wysokości tonu. Na przykład instrukcja **POKE 53760,128** jest równoważna instrukcji **SOUND 0, 128, n, n** (n wartość jest tutaj nieistotna).

AUDC 1 do 4

Te zmienne są zlokalizowane w komórkach o adresach 53761, 53763, 53765 i 53767. Ich znaczenie jest bardziej skomplikowane niż AUDF. Przyjrzyjmy się im dokładniej.

Cztery najmniej znaczące bity kontrolują głośność. Cztery bity to w układzie dziesiętnym liczby z zakresu od 0 do 15, czyli dokładnie tyle ile stopni głośności można zapisać w instrukcji **SOUND**.

Znaczeniem piątego bitu zajmiemy się pod koniec rozdziału. Pozostałe trzy bity kontrolują zniekształcenie dźwięku i tu pojawia się pole do eksperymentowania. Jeżeli wszystkie bity posiadają wartość logicznej jedynki (są ustawione) to otrzymamy czysty dźwięk.

Tak więc nadanie wartości zmiennym systemowym AUDF i AUDC jest równoważne instrukcji **SOUND**. Na przykład **POKE 53760, 128** oraz **POKE 53761, 234** jest tożsame z **SOUND 0, 128, 10, 10**.

AUDCTL jest komórką o adresie 53768, której zawartość zmieniamy tylko za pomocą instrukcji **POKE**. Znaczenie poszczególnych bitów jest następujące:
BIT 0 przełącza główny zegar bazowy z 64 kHz na 15 kHz,
BIT 1 włącza filtr górnoprzepustowy dla dźwięku kanału 2, sterowany dźwiękiem kanału 1, BIT 2 włącza filtr górnoprzepustowy dla dźwięku kanału 2, sterowany dźwiękiem kanału 3, BIT 3 powoduje połączenie kanałów dźwiękowych 3 i 4 dając 16 bitową zmienną sterującą, BIT 4 powoduje połączenie kanałów dźwiękowych 1 i 2 dając 16 bitową zmienną sterującą, BIT 5 powoduje przełączenie zegarów kanału 3 na 1, 79 MHz,
BIT 6 powoduje przełączenie zegarów kanału 1 na 1, 79 MHz, BIT 7 przełącza licznik "poly" z 17 na 9 bitów.

Wydaje się, że znaczenie poszczególnych bitów wymaga kilku słów wyjaśnienia.

BIT 0. Bez wdawania się w szczegóły można powiedzieć, że im wyższa jest częstotliwość zegara tym wyższe dźwięki można otrzymać. Zegar bazowy generatorów dźwięku ma częstotliwość 64 kHz. Przełączenie go na częstotliwość 15 kHz pozwala uzyskać niższe częstotliwości.

BIT 1 i 2. Poprzez przepuszczenie dźwięku przez filtry możemy wpływać na brzmienie wytwarzanych dźwięków. W sposób uproszczony można powiedzieć, że taki filtr nie przepuszcza dźwięków o częstotliwościach mniejszych niż częstotliwość kanału sterującego. Z filtrami spotkamy się jeszcze przy opisie syntezatora.

BIT 3 i 4. Aby zilustrować konieczność połączenia dwóch kanałów w celu uzyskania 16 bitowej zmiennej sterującej, wprowadźmy poniższy program i wykonajmy go uważnie słuchając.

```
15 SOUND 0, 0, 0, 0
25 FOR X=55 TO 0 STEP-1
35 SOUND 0, X, 10, 10
45 FOR F=1 TO 55: NEXT F
55 NEXT X
65 SOUND 0, 0, 0, 0
```

Zwróćmy uwagę, że o ile przy niskich częstotliwościach dźwięk wzrasta łagodnie, to przy wysokich częstotliwościach dźwięk wyraźnie przeskakuje (częstotliwość wzrasta skokowo). Zjawisko to można wyeliminować łącząc dwa kanały razem, co daje 16 bitową zmienną sterującą. W ten sposób zmniejszyliśmy najmniejszy skok częstotliwości z 1/256 (jeden bajt) do 1/65536 (dwa bajty).

PROGRAM "POKEY"

```
5 REM PROGRAM "POKEY"
10 SOUND 0, 0, 0, 0
20 POKE 53768, 80
30 POKE 53761, 160
40 POKE 53763, 234
50 FOR X=0 TO 255
60 POKE 53762, X
70 FOR Z=0 TO 255
80 POKE 53760, 2
90 NEXT Z
100 NEXT X
110 SOUND 0, 0, 0, 0
```

Powyższy program ilustruje nasze rozważania.
Linia 10 inicjuje POKEY
Linia 20 ustawia bit 2 w AUDCTL, bit 6 i bit 4
Linia 30 nadaje wartość AUDC 1 i wyłącza pierwszy generator dźwiękowy
Linia 40 ustawia AUDC 2, włącza generator 2, włącza czyste tony oraz ustawia głośność 10.
Linie 50 do 100 - dwie pętle FOR ... NEXT służą do nadawania wartości zmiennym AUDF 1 i AUDF 2.
Wartością zmiennej sterującej generator możemy regulować częstotliwość zgrubnie, natomiast generatorem pierwszym regulujemy częstotliwość dokładnie.

Kontynuujemy tłumaczenie funkcji pozostałych bajtów:
BITY 5 i 6 Działanie ich jest podobne jak działanie bitu 0, z tą różnicą, że bity 5 i 6 zwiększają wysokość generowanego tonu. Bit 5 zwiększa częstotliwość zegara do 1, 79 MHz
w kanale dźwiękowym 3, natomiast bit 6 w kanale 1. Spróbujmy wykonać następujące instrukcje:

SOUND 0, 255, 10, 10 POKE 53764, 64

Instrukcje te zilustrują nam działanie tych dwóch bitów.
BIT 7 Licznik "poly" kontroluje zniekształcenie dźwięku. Im mniejsza pojemność tego licznika to tym częściej powtarzane są zniekształcenia generowanego dźwięku. Aby to zrozumieć spróbujmy wykonać następujące instrukcje:

SOUND 0, 90, 8, 10 POKE 53768, 128

Liczba 8, która jest trzecim parametrem instrukcji **SOUND** ustawia 16 bitowy licznik "poly".
Przedstawiliśmy funkcjonowanie POKEY, a w szczególności zmienne systemowe AUDF 1 do 4, AUDC 1 do 4, AUDCTL. Obecnie zajmiemy się głównym przedmiotem naszych rozważań w tym rozdziale czyli syntezą dźwięku.

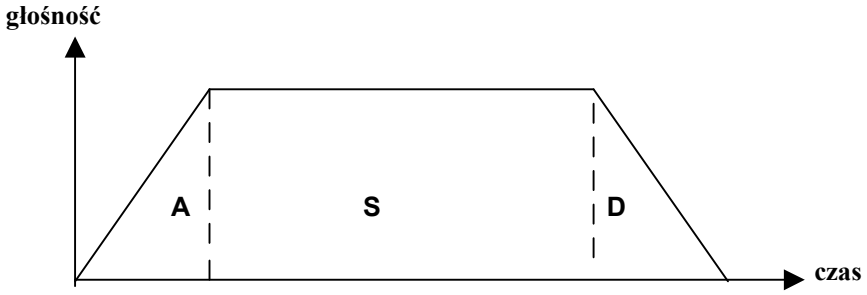
6.3. Wprowadzenie do syntezy dźwięku.

Zanim zajmiemy się ATARI, spójrzmy jak działa syntezator dźwięku, który jest przecież tak różny od tradycyjnych instrumentów muzycznych. Dźwięk oglądany na oscyloskopie ma postać fali. Na przykład dźwięk fletu jest w zasadzie czystą sinusoidą.

Każdy dźwięk instrumentu można scharakteryzować za pomocą czterech parametrów. Są nimi:

- czas nabrzmiewania (ang. Attack, skrót A)
- czas trwania (ang. Sustain, skrót S)
- czas wybrzmiewania (ang. Decay, skrót D)
- powtórzenie dźwięku (ang. Repeat)

Na rysunku 6.2. pokazano trzy pierwsze parametry.



Rysunek 6.2.

Czas nabrzmiewania A jest to czas potrzebny na to aby dźwięk osiągnął pełną głośność. Różnicę w czasach nabrzmiewania dobrze widać na przykładzie dwóch instrumentów muzycznych pianina i skrzypiec.

Dźwięk jest wibracją powietrza wokół nas. Instrumenty strunowe przetwarzają drgania strun na wibrację powietrza. W instrumentach dętych wibruje wdmuchiwane powietrze. Spójrzmy na różnicę w czasie nabrzmiewania dla różnych instrumentów. W pianinie dźwięk powstaje pod wpływem uderzenia młotka w strunę. Dźwięk osiąga swój szczyt i praktycznie w momencie uderzenia, tak więc czas nabrzmiewania jest bardzo krótki. W skrzypcach dźwięk powstaje przez ciągnięcie smyczkiem po strunach, co zwiększa ich drżenie. Z tego powodu czas nabrzmiewania jest dość długi.

Czas wybrzmiewania D jest to czas potrzebny na to, aby dźwięk od maksymalnej głośności zanikł całkowicie.

Jeszcze raz odwołajmy się do przykładu dwóch instrumentów fletu i pianina. Gdy młotek uderzy w strunę, to drga ona do chwili gdy cała energia zawarta w uderzeniu zostanie rozproszona. Tak więc czas wybrzmiewania jest dość długi. We flecie dźwięk kończy się prawie natychmiast gdy muzyk przestaje w niego dmuchać. Czas wybrzmiewania jest więc bardzo krótki.

Czas trwania S to czas maksymalnej głośności dźwięku.

Powtórzenie R jest to parametr mówiący o tym ile razy dana forma dźwięku została powtórzona. Aby wytworzyć potrzebne dźwięki syntezator używa tak zwanego układu kształtowania obwiedni, aby móc kontrolować kształt tworzonej fali dźwiękowej. Poniższy program ilustruje w jaki sposób można wpływać na czas nabrzmiewania dźwięku tworzonego przez ATARI.

```
15 INPUT ATT
25 FOR X=1 TO 15
35 SOUND 0, 120, 10, X
45 FOR Z=1 TO ATT: NEXT Z
55 NEXT X
65 SOUND 0, 0, 0, 0
75 GOTO 15
```

Wczytana w linii 15 zmienna ATT powoduje opóźnienie (linia 45) w czasie wykonywania pętli 25...55 decydującej o narastaniu głośności dźwięku. W ten sposób możemy kontrolować czas nabrzmiewania tworzonego dźwięku. W czasie wykonywania tego programu zauważyć można, że wykonanie programu w języku BASIC jest zbyt wolne. Również głośność dźwięku zwiększa się skokowo, w sposób nieciągły. Oczywistym staje się, że program winien być napisany w kodzie maszynowym, lecz aby zrozumieć do czego dążymy pokażemy dalsze części programu napisane również w języku BASIC. Program tworzący część zanikającą dźwięku jest odwróceniem programu poprzedniego.

```
15 INPUT DEC
25 FOR X=15 TO 0 STEP -1
35 SOUND 0, 120, 10, X
45 FOR Z=1 TO DEC: NEXT Z
55 NEXT X
65 GOTO 15
```

Ten program używa pętli FOR...NEXT z krokiem -1 aby zmniejszyć głośność.

Jeśli w obydwu poprzednich programach wartości zmiennych ATT i DEC są duże, to stwierdzamy, że głośność dźwięku narasta skokowo.

Następnym przykładem jest program, który umożliwia kontrolowanie czasów nabrzmiewania, trwania i wybrzmiewania dźwięku. Najwyższy rząd klawiszy (cyfry) działa podobnie jak klawiatura w fortepianie.

PROGRAM "PIANINO 1"

```
1 REM PROGRAM"PIANINO 1"

5 INPUT ATT
6 INPUT DEC
7 INPUT SUS
8 POKE 731, 2
10 CLOSE #1
20 OPEN #1, 4, 0, "K: "
30 GET #1, A
40 IF A=49 THEN X=243
50 IF A=50 THEN X=217
60 IF A=51 THEN X=193
70 IF A=52 THEN X=182
80 IF A=53 THEN X=162
90 IF A=54 THEN X=144
100 IF A=55 THEN X=128
110 IF A=56 THEN X=121
20 IF A=57 THEN X=108 1
30 IF A=48 THEN X=96
140 GOSUB 1000
150 GOTO 30
1000 FOR R=0 TO 15
1010 SOUND O, X, 10, R
1020 FOR T=0 TO ATT: NEXT T
1030 NEXT R
1040 FOR Y=R TO SUS: NEXT Y
1050 FOR U=15 TO 0 STEP -1
1060 SOUND 0, X, 10, U
1070 FOR I=0 TO DEC: NEXT I
1080 NEXT U
1090 RETURN
```

W programie tym problemy tworzenia dźwięku w BASIC-u ukazały się nam w całej swej okazałości. Linie 10 do 130 służą do obsługi klawiatury. Najpierw otwieramy kanał komunikacji z klawiaturą, a następnie czytamy z niej jeden bajt. Liczba musi być przetworzona na inną odpowiadającą pewnej wysokości dźwięku. Dzieje się to w liniach 40 do 130. Następnie wykonywany jest podprogram generujący odpowiedni dźwięk.. Jest on połączeniem poprzednich przykładów. Pewną modyfikacją programu "Pianino l" jest poniższy program. Jest to w zasadzie najszybszy program muzyczny napisany w BASIC-u.

PROGRAM "PIANINO 2"

```
1 REM PROGRAM "PIANINO 2"
5 PRINT "PODAJ CZAS NABRZMIEWANIA DZWIEKU"; :  INPUT ATT
6 PRINT "PODAJ CZAS TRWANIA DZWIEKU"; :  INPUT SUS
7 PRINT "PODAJ CZAS WYBRZMIEWANIA DZWIEKU"; :  INPUT DEC
8 ATT=ATT/1000000: SUS=SUS/1000000: DEC=DEC/1000000
9 POKE 731, 2
10 CLOSE#1
20 OPEN , #1, 4, 0, "K: "
30 GET #1, A
```

```
40 A=A-47
50 ON A GOSUB 200, 210, 220, 230, 240, 250, 260, 270, 280, 290
100 GOSUB 1000
110 GOTO 30 200 X=96: RETURN
210 X=243: RETURN
220 X=217: RETURN
230 X=193: RETURN
240 X=182: RETURN
250 X=162: RETURN
260 X=144: RETURN
270 X=128: RETURN
280 X=121: RETURN
290 X=108: RETURN
1000 FOR R=0 TO 10
1010 SOUND O, X, 10, R
1020 FOR T=0 TO ATT STEP 1E-06: NEXT T
1030 NEXT R
1040 FOR Y=0 TO SUS STEP 1E-06: NEXT Y
1050 FOR U=10 TO 0 STEP -1
1060 SOUND O, X, 10, U
1070 FOR I=0 TO DEC STEP 1E-06: NEXT I
1080 NEXT U
1090 RETURN
```

Między tymi dwoma programami istnieją dwie podstawowe różnice. Pierwsza to użycie instrukcji ON...GOSUB zamiast ciągu porównań IF ...THEN. Jeśli wśród czytelników jest jakiś matematyczny geniusz, który potrafi napisać wzór, który określa zależność między liczbą przeczytaną z klawiatury, a drugim parametrem (wysokość dźwięku instrukcji SOUND), to prosimy nas o tym zawiadomić Następną różnicą jest zmniejszenie kroku pętli FOR...NEXT do jednej milionowej, co daje bardziej płynny dźwięk, jednak nadal nie jest on do przyjęcia w poważnych zastosowaniach. Jak widać nie ma innych możliwości, tylko musimy napisać podprogram w kodzie maszynowym. Wszystkie trzy parametry wprowadzone do programu muszą mieć wartości powyżej 50. Dlaczego? O tym piszemy poniżej.

PROGRAM "PROSTY INSTRUMENT MUZYCZNY"

```
3 REM PROGRAM
4 REM "PROSTY INSTRUMENT MUZYCZNY"
5 FOR C=1 TO 74: READ Z
6 POKE 1535+C, Z
7 NEXT C
10 PRINT "PODAJ CZAS NABRZMIEWANIA DZWIEKU"; : INPUT ATT
11 PRINT "PODAJ CZAS TRWANIA DZWIEKU"; : INPUT SUS
12 PRINT "PODAJ CZAS WYBRZMIEWANIA DZWIEKU"; : INPUT DEC
15 POKE 731, 2
20 CLOSE #1
25 OPEN #1, 4, 0, "K"
30 GET#1, A
40 A=A-47
50 ON A GOSUB 300, 310, 320, 330, 340, 350, 360, 370,  380, 390
100 Q=USR(1536, X, ATT, SUS, DEC)
110 GOTO 30
200 DATA 104, 104, 104, 141, 0, 210, 104, 133, 204, 104, 104, 104, 133, 205, 104, 104, 133, 206
210 DATA 169, 160, 141, 1, 210, 166, 204, 32, 65, 6, 24, 105, 1, 201, 176,  208, 241
220 DATA 169, 14, 166, 205, 32, 65, 6, 56, 233, 1, 208, 246
230 DATA 169, 175, 141, 1, 210, 166, 206, 32, 65, 6, 56,  233, 1, 201, 159,  208, 241, 96
240 DATA 160, 19, 136, 208, 253, 202, 208, 248, 96
300 X=96: RETURN
310 X=243: RETURN
```

```
320 X=217: RETURN
330 X=193: RETURN
340 X=182: RETURN
350 X=162: RETURN
360 X=144 : RETURN
370 X=128: RETURN
380 X=121: RETURN
390 X=108: RETURN
```

Program wymaga pewnych wyjaśnień. Linie 5 do 7 wczytują program w kodzie maszynowym zawartym w liniach 200 do 240. Poszczególne liczby zawarte w instrukcjach DATA są wpisywane bezpośrednio do pamięci przez użycie instrukcji POKE. Liczby te wpisuje się do komórek pamięci od adresu 1536 (6 strona pamięci). Linie 10 do 12 wczytują wartości czasów nabrzmiewania, trwania oraz wybrzmiewania dźwięku. Linie 20 do 50 i 300 do 390 sprawdzają, który klawisz został naciśnięty i przetwarzają jego kod na liczbę odpowiadającą żądanemu dźwiękowi. Liczba ta jest parametrem X w podprogramie napisanym w kodzie maszynowym. Linia 100 jest wywołaniem podprogramu.

Równocześnie przekazywane są wszystkie parametry charakteryzujące dźwięk, a więc X, ATT, DEC, SUS. Znaczenia ich podane w programie. Podany jest również adres początku podprogramu, a więc 1536. Linie 200 do 240 zawierają podprogram, który przyjmuje parametry, określa głośność oraz wpisuje X do rejestrów dźwięku. Następnie jest realizowana funkcja kształtowania obwiedni dźwięku, a dopiero później następuje skok do BASIC-u i czytany jest następny klawisz. Jest to pewnym ograniczeniem stworzonego instrumentu.

Pozostało jeszcze wyjaśnić dlaczego parametry ATT, DEC i SUS muszą być większe od 50. Otóż jest to najkrótszy czas w jakim ATARI może realizować funkcje nabrzmiewania, trwania i wybrzmiewania dźwięku. Wynosi on 5 ms.

6.4. Filtry

Na początku tego rozdziału znajduje się opis zmiennej systemowej AUDCTL. Wiemy, że bity 1 i 2 tej zmiennej włączają filtry górnoprzepustowe w kanałach 1 i 2 sterowane dźwiękiem kanałów 3 i 4. Najpierw wpiszmy i posłuchajmy następującego programu.

```
15 SOUND 0, 0, 0, 0
25 POKE 53768, 4
35 POKE 53761, 170
45 POKE 53765, 170
55 POKE 53760, 254
65 POKE 53764, 127
70 GOTO 75
```

Przed uruchomieniem tego programu wpiszmy najpierw w trybie bezpośrednim:

POKE 53761, 170: POKE 53760, 254

posłuchajmy dźwięku i w tym momencie kiedy on nadal trwa wpiszmy:

```
POKE 53765, 170: POKE 53764, 127
```

usłyszymy dwa dźwięki, między którymi jest różnica oktawy. Program, który teraz uruchomimy różni się od powyższych instrukcji tym, że w linii 25 włączony zostaje filtr.

Spójrzmy dokładnie na ten program:
Linia 15 inicjuje POKEY
Linie 35 i 45 kontrolują zmienne AUDC 1 i AUDC 3 dając czysty dźwięk i głośność 10. Linie 55 i 65 wpisują do AUDF 1 i AUDF 3 dwa dźwięki oddalone o oktawę.
Linia 75 jest nieskończoną pętlą umożliwiającą trwanie dźwięku. Linia 2 włącza filtr górnoprzepustowy.

PROGRAM "FILTR"

```
1 REM PROGRAM "FILTR"
5 DIM A$(1)
10 SOUND 0, 0, 0, 0
20 PRINT "PODAJ WARTOSC-"
30 PRINT "5 DLA FILTRU 15 kHz"
50 PRINT "68 DLA FILTRU 1, 79 MH2"
70 INPUT A
80 POKE 53768, A
90 POKE 53761, 170
100 POKE 53765, 170
110 PRINT "PODAJ WYSOKOSC DLA KANALU 1"
120 INPUT B
130 PRINT "PODAJ WARTOSC DLA ZEGARA"
135 POKE 53760,  B
140 INPUT C
150 POKE 53764, C
170 PRINT "NACISNIJ RETURN"
180 INPUT A$
190 SOUND 0, 0, 0, 0
200 GOTO 10
```

Powyższy program posłuży nam do eksperymentów.
Linie 20-70 podane wartości są tylko naszymi sugestiami i jeśli bit 2 jest włączony to może być wpisana dowolna inna wartość.
Linie 90 i 100 określamy tutaj, że tworzymy tylko czyste dźwięki o głośności 10. Można wpisać dowolne wartości, takie aby bit 4 był włączony.
Linie 110-160 oktawy tworzą najbardziej interesujące dźwięki. Spróbujmy innych kombinacji.
Interesującym eksperymentem jest użycie dwóch filtrów jednocześnie. Spróbujmy wykonać program:

```
15 SOUND 0, 0, 0, 0
25 POKE 53768, 6
35 POKE 53761, 170
45 POKE 53763, 170
55 POKE 53765, 170
65 POKE 53767, 170
75 POKE 53760, 243
85 POKE 53762, 60
95 POKE 53764, 121
115 POKE 53766, 29
125 GOTO 125
```

Wartości zawarte w liniach 75 do 115 określają dźwięki C w kolejnych oktawach.

Jedną z podstawowych zasad syntezy dźwięku jest eksperymentowanie. Filtry syntezatora są tymi jego elementami, które się do tego celu nadają najbardziej.

6.5. Głośność

Wspomnieliśmy już wcześniej o bicie 4 zmiennych AUDC 1 do 4, który nosi nazwę bitu dwustanowej regulacji głośności.

Każdy głośnik wytwarza dźwięk przez drgania membrany. Częstotliwość tych drgań decyduje o wysokości dźwięku, a amplituda o jego głośności. Bit dwustronnej regulacji głośności umożliwia poruszanie membraną głośnika w przód i w tył.

Spróbujmy wykonać instrukcję:

POKE 53761, 31

Słyszymy pojedynczy trzask w momencie gdy membrana głośnika porusza się do przodu. Wykonując tę instrukcję ustawiliśmy bit 4 oraz nastawiliśmy głośność na 15. Spróbujmy wykonać tę instrukcję jeszcze raz. I cóż, teraz nic nie było słychać. Jest to oczywiste, gdyż wykonując tę instrukcję wysuwamy membranę głośnika, która jest już maksymalnie wysunięta. Aby ją wsunąć z powrotem wykonujemy teraz instrukcję:

POKE 53731, 16

I znów było słychać pojedynczy trzask. Teraz spróbujmy wytworzyć jakiś dźwięk wykorzystując bit dwustanowej regulacji głośności.

```
15 SOUND 0, 0, 0, 0
25 PRINT "PODAJ WARTOSC OPOZNIENIA OD 2 DO 100"
35 INPUT A
45 POKE 53761, 31
55 POKE 53761, 16
65 FOR X=1 TO A: NEXT X
75 GOTO 45
```

Zwróćmy uwagę, że nie używaliśmy tutaj zmiennych AUDF 1 do 4, a jednak wygenerowaliśmy pewien dźwięk, którego częstotliwość możemy zwiększyć wyłączając ekran przez dopisanie do programu instrukcji:

38 POKE 559, 0

Zauważmy, że jeśli podamy dużą wartość opóźnienia to daje się słyszeć pojedyncze przełączenia głośnika.

Pewne formy dźwięku mogą być tworzone poprzez wpływanie na głośność poszczególnych przełączeń głośnika. Typowym przykładem może być odtworzenie głosu trąbki. Poszczególne głośności powinny wówczas wynosić:
9, 10, 11, 10, 9, 8, 7, 6, 5, 7, 8.
Jeżeli taką sekwencję powtórzymy dość szybko, to powstały dźwięk przypomina głos trąbki. Spróbujmy to zrobić w BASIC-u:

```
5 POKE 559, 0
15 FOR X=1 TO 12
25 READ A
35 POKE 53761, A
45 POKE 53761, 16
55 NEXT X
65 RESTORE
75 GOTO 15
85 DATA 25, 26, 27, 26, 25, 24, 23, 22, 21, 22, 23, 24
```

W instrukcji **DATA** mamy ciąg liczb, które odpowiednio ustawiają jednocześnie głośność i bit dwustanowej regulacji głośności. Liczby te powstają przez dodanie do każdego wyrazu ciągu 9, 10, 11, 10, 9, 8 ... itd. szesnastu (szesnaście ustawia bit 4).

W tym miejscu ponownie pojawia się problem zbyt małej szybkości programów napisanych w BASIC-u. Czy nie wydaje się, że otrzymany dźwięk przypomina głos trąbki tyle, że odtwarzany trochę wolniej'?

Bit dwustanowej regulacji głośności umożliwia odwzorowanie dźwięku różnych instrumentów. Odwzorowanie to nie będzie zbyt dokładne gdyż mamy tylko 15 stopni regulacji głośności. Ponieważ szesnastostopniowa regulacja jest cechą POKEY, więc teoretycznie możliwe jest otrzymanie ośmiobitowej regulacji głośności dającej zakres zmian od 0 do 255. Umożliwiałoby to znacznie bardziej realistyczne odwzorowanie dźwięku różnych instrumentów.

6.6. Dudnienia

Spróbuj wykonać instrukcję:

```
SOUND 0, 92, 10, 10
SOUND 2, 91, 10, 10
```

Dziwny efekt, który słyszymy jest rezultatem nakładania się dwóch fal o niewiele różniącej się częstotliwości. W wyniku dodania się dwóch takich fal powstaje trzecia o bardzo małej częstotliwości, która wpływa na głośność dwóch pozostałych. W fizyce zjawisko to nosi nazwę dudnień.

Wykorzystując dudnienia można osiągnąć ciekawe efekty. Spróbujmy na przykład:

```
SOUND 0, 92, 12, 10
SOUND 0, 91, 12, 10
```

Możemy jeszcze dodać filtr g6rnoprzepustowy instrukcją:

```
POKE 53768, 4
```

Inny efekt otrzymujemy wykonując:

```
POKE 53768, 1
POKE 53768, 5
POKE 53768, 32
POKE 53768, 36
POKE 53768, 64
POKE 53786, 68
POKE 53768, 96
POKE 53768, 100
POKE 53768, 101
POKE 53"768, 104
POKE 53768, 105
```

I jeszcze raz okazuje się, że eksperymentowanie jest konieczne. POWODZENIA!

6.7. Syntezator dźwięku

Przedstawimy poniżej największy i najbardziej złożony program ze wszystkich przedstawionych w tej książce. Treść programu może być dowolnie modyfikowana przez bardziej doświadczonych użytkowników ATARI.

Program został podzielony na kilka sekcji, aby stał się bardziej zrozumiały. Działanie programu można przerwać i wrócić do początku przez naciśnięcie SPACE.

Jeżeli w którymkolwiek momencie zachodzi potrzeba wybrania między tak (T) lub nie (N) nie wymaga to naciśnięcia klawisza RETURN. Dźwięki pianina można uzyskać przez wprowadzenie do części ADSR liczb 700, 300, 700, N, natomiast dźwięk harfy przez podanie 700, 300, 700, T, N, T, N.

PROGRAM "SYNTEZATOR DŹWIĘKU"

```
1 REM PROGRAM "SYNTEZATOR DZWIEKU"
 2 REM *** BLOK GLOWNY***
5 MEN=PEEK(559)
6 POKE 731, 1
10 FOR LOAD=1 TO 74: READ PR
20 POKE 1535+LOAD, PR
30 NEXT LOAD
40 POKE 755, 0
50 SETCOLOR 2, 0, 0
60 FOR X=0 TO 3
70 SOUND X, 0, 0, 0
80 NEXT X
85 POKE 559, MEN: FLAG=0: FLAG1=0
90 PRINT CHR$(125)
```

```
100 FOR X=1 TO 5:  PRINT: NEXT X
110 PRINT "WYKAZ DOSTEPNYCH OPCJI"
120 PRINT : PRINT
130 PRINT : PRINT "1 WYBOR OBWIEDNI DZWIEKU"
140 PRINT : PRINT "2 WYBOR FILTROW"
150 PRINT : PRINT "3 WYBOR RODZAJU MODUKACJI"
160 PRINT : PRINT "4 WYBOR RODZAJU ZNIEKSZTALCEN" 200 GOSUB 2000
210 KEY=KEY-48
220 IF KEY<1 OR KEY> 4 THEN GOTO 90
230 ON KEY GOTO 300, 700, 900, 1100
240 REM ***BLOK ADSR***
300 CON=0
310 PRINT CHR$(125)
320 FOR X=1 TO 5: PRINT : NEXT X
330 PRINT " A.D.S.R.MENU"
340 PRINT : PRINT "PODAJ CZAS NABRZMIEWANIA GZWIEKU"
350 INPUT ATT
360 PRINT : PRINT "PODAJ CZAS TRWANIA DZWIEKU"
370 INPUT SUS
380 PRINT : PRINT : PODAJ CZAS WYBRZMIEWANIA DZWIEKU"
390 INPUT DEC
400 PRINT : PRINT "KONTROLA DZWIEKU (T/N)",
420 GOSUB 2000
440 IF KEY=78 THEN GOTO 580
450 PRINT CHR$(125)
460 FOR X=1 TO 5: PRINT : NEXT X
470 PRINT "ZEGAR 15 kHz (T/N)";
480 GOSUB 2000
490 IF KEY=84 THEN CON=CON+1
500 PRINT : PRINT "FILTR GORNOPRZEPUSTOWY (T/N)";
510 GOSUB 2000
520 IF KEY=84 THEN CON-CON+4
540 PRINT : PRINT "ZEGAR 1, 79 MHz(T/N)",
550 GOSUB 2000
560 IF KEY=84 THEN CON=CON+64
570 POKE 53768, CON
580 IF FLAG=1 OR FLAG1=1 THEN GOTO 590
585 GOSUB 2000
590 IF KEY=32 OR PTRIG(0)=0 THEN GOTO 60
595 ZF FLAG1=1 THEN GOSUB 6500: GOT0 620
600 IF FLAG=1 THEN GOSUB 6000
610 GOSUB 3000
620 Q=USR(1536, KEY, ATT, SUS, DEC)
630 GOTO 580
640 REM ***BLOK FILTROW***
700 OCT1=0
710 PRINT CHR$(125)
720 FOR X=1 TO SPRINT : NEXT X
730 PRINT "WYBOR RODZAJU FILTRU"
740 PRINT : PRINT : PRINT "OKTAWY (T/N); "
750 GOSUB 2000
770 IF KEY=84 THEN GOTO 800
780 PRINT : PRINT "CZESTOTLIWOSC ZEGARA";
790 INPUT OCT1
800 POKE 53768, 4
810 IF FLAG=1 OR FLAG1=1 THEN GOTO 820
815 GOSUB 2000
820 IF KEY=32 OR PTRIG(0)=0 GOTO 60
```

```
825 IF FLAG1=1 THEN GOSUB 6500:  GOTO 850
830 IF FLAG=1 THEN GOSUB 6000
840 GOSUB 3000
850 IF OCT1<>0 THEN OCT=0CT1
855 POKE 53768, 4
860 POKE 53761, 170: POKE 53765, 170
870 POKE 53760, KEY: POKE 53764, OCT 880 GOTO 810
890 GOSUB 6500: GOT0 850
895 REM***BLOK MODULACJI*** 900 TONE=0
910 PRINT CHR$(125)
920 FOR X=1 TO 5 : PRINT : NEXT X
930 PRINT " RODZAJ MODULACJI"
940 PRINT : PRINT : PRINT "FILTR WLACZONY (T/N); "
950 GOSUB 2000
960 IF KEY=84 THEN POKE 53768, 4
970 PRINT : PRINT "CZYSTY DZWIEK WYLACZONY (T/N)"
980 GOSUB 2000
990 IF KEY=B4 THEN TONE=1
1000 IF FLAG=1 OR FLAG1=1 THEN GOTO 1010
1005 GOSUB 2000
1010 IF KEY=32 OR PTRIG(0)=0 THEN GOTO 60
1015 IF FLAG1=1 THEN GOSUB 6500: GOT0 1035
1020 IF FLAG=1 THEN GOSUB 6000
1030 GOSUB 3000
1035 ZF TONE=1 THEN GOTO 1070
1040 POKE 53761, 170: POKE 53765, 170
1050 POKE 53760, KEY: POKE 53764, KEY-1
1060 GOTO 1000
1070 SOUND O, KEY, 12, 10
1080 SOUND 2, KEY-1, 12, 10
1090 GOTO 1000
1095 REM***BLOK ZNIEKSZTALCEN***
1100 CON=0
1110 DIST=170
1120 DIST1=170
1130 PRINT CHR$(125)
1140 FOR X=1 TO 5 : PRINT : NEXT X
1150 PRINT "WARTOSC ROZKLADU"
1160 PRINT : PRINT "WARTOSC (1 TO 5)"
1170 INPUT DIST
1180 IF DIST=1 THEN DIST=10
1181 IF DIST=2 THEN DIST=42
1182 IF DIST=3 THEN DIST=74
1183 IF DIST=4 THEN DIST=138
1184 IF DIST=5 THEN DIST=202
1190 PRINT : PRINT "9 BITOWY LICZNIK poly (T/N)"
1200 GOSUB 2000
1210 IF KEY=84 THEN CON=CON+128
1220 PRINT : PRINT "FILTR (T/N)";
1230 GOSUB 2000
1240 ZF KEY=78 THEN GOTO 1400
1250 PRINT CHR$(125)
1260 FOR X=1 TO 5: PRINT : NEXT X
1270 PRINT "PRZELACZANY KANAL ZNIEKSZTALCEN (T/N)";
1280 GOSUB 2000
1290 IF KEY=84 THEN DIST1=DIST
1300 PRINT : PRINT "OBYDWA KANALY ZNIEKSZTALCONE (T/N)";
1310 GOSUB 2000
```

```
1320 IF KEY=78 AND DIST1<>170 THEN DIST=170
1330 IF FLAG=1 OR FLAG1=1 THEN GOTO 1340
1335 GOSUB 2000
1340 IF KEY=32 OR PTRIG(0)=0 THEN GOTO 60
1345 IF FI, AG1=1 THEN GOSUB 6500: GOT0 1370
1350 IF FLAG=1 THEN GOSUB 6000
1360 GOSUB 3000
1370 POKE 53761, DIST: POKE 53765, DIST1
1380 POKE 53760, KEY: POKE 53764, OCT
1390 GOTO 1330
1.400 IF FLAG=1 OR FLAG1=1 THEN GOTO 1410
1405 GOSUB 2000
1410 IF KEY=32 OR PTRIG(0)=0 THEN GOTO 60
1415 IF FLAG1=1 THEN GOSUB 6500: GOTO 1440
1420 IF FLAG=1 THEN GOSUB 6000
1430 GOSUB 3000
1440 POKE 53761, DIST: POKE 53760, KEY
1450 GOTO 1400
7500 REM***CZYTANIE KLAWIATURY***
2000 CLOSE #1.
2010 OPEN #1, 4, 0, "K: "
2020 GET #1, KEY
2030 IF KEY<>4"7 THEN GOTO 2050
2040 GOSUB 5000
2050 RETURN 3000 KEY=KEY-47
3005 POKE 559, 0
3010 ON KEY GOSUB 3030, 3040, 3050, 3060, 3070, 3080, 3090, 4000,  4010, 4020
3020 RETURN
3030 KEY=96: OCT=60: RETURN
3040 KEY=243: OCT=121: RETURN
3050 KEY~=217: OCT=108: RETURN
3060 KEY=193: OCT=96: RETURN
3070 KEY=182: OCT=91: RETURN
3080 KEY=162: OCT=81: RETURN
3090 KEY=144: OCT=72: RETURN
4000 KEY---128: OCT=64: RETURN
4010 KEY=121.: OCT=60: RETURN
4020 KEY=108: OCT=53: RETURN
4030 REM***BLOK DZWIEKOW PRZYPADKOWYCH***
5000 PRINT CHR$(125)
5005 FLAG=0 5006 FLAG1=0
5010 FOR X=1 TO SPRINT : NEXT X
5020 PRINT "GENERATOR DZWIEKOW PRZYPADKOWYCH"
5025 PRINT : PRINT : PRINT "UZYCIE PADDLES (T/N)";
5026 GOSUB 2000
5027 IF KEY=84 THEN GOTO 5060
5030 PRINT : PRINT : PRINT "PREDKOSC (2 DO 500)";
5040 INPUT SPEED
5050 FLAG=1: GOT0 5070
5070 RETURN
5969 FLAG1=1
6000 FOR HOLD=1 TO SPEED: NEXT HOLD
6005 POKE 559, 0
6010 KEY=INT(RND(1)*9)+48
6015 IF PEEK (764)=33 THEN FLAG=0
6020 RETURN
6500 KEY=PADDLE(0)
6505 POKE 559, 0
```

```
6520 RETURN
7000 DATA 104, 104, 104, 141, 0, 210, 104, 133, 204, 104, 104, 4,  104, 133, 205, 104, 104, 133,
206
7010 DATA 169, 160, 141, 1, 210, 166, 204, 32, 65, 6, 24, 105, 1,  201, 176, 208, 241
7020 DATA 169, 14, 166, 205, 32, 65, 6, 56, 233, 1, 208, 246
7030 DATA 169, 175, 141, 1, 210, 166, 206, 32, 65, 6, 56, 233, 1,  201, 159, 208, 241, 96
7040 DATA 160, 19, 136, 208, 253, 202, 208, 248, 96
```

Rozdział 7

PAMIĘĆ ZEWNĘTRZNA

W ATARI występuje pamięć zewnętrzna w trzech postaciach. Pierwszą z nich są pamięci stałe typu ROM. Są to niewielkie pudełeczka (ang. cartridge) o wymiarach 80x60x20 mm zawierające program zapisany przez producenta. Użytkownik nie ma możliwości wymazania lub zmiany zapisu. Drugą i trzecią formą pamięci zewnętrznych są stacje dysków i magnetofony.

7.1. Stacja dysków

Jest to urządzenie pozwalające w pełni wykorzystać konstrukcję i system operacyjny komputera. Istnieje możliwość podłączenia do komputera 4 stacji dysków, przy czym 2 wydają się ilością optymalną. Stosowanie dysków rozszerza znacznie możliwość wykorzystania komputera, zwiększa praktyczną szybkość zapisu i odczytu niemal stukrotnie. Umożliwia także wykonywanie operacji niemożliwych do realizacji w przypadku magnetofonu jak np. samoczynne wyszukiwanie danych.

Przy zastosowaniu stacji dysków zapis i odczyt programów w BASIC'u dokonuje się przy użyciu instrukcji:

SAVE"D: ***nazwa, identyfikator"***

oraz

LOAD"D: ***nazwa, identyfikator"***

Analogiczne zapisy ma para LIST i ENTER. Zamiast LOAD można stosować RUN, co pozwala na ładowanie i samoczynne wykonywanie programu.

Najczęściej stosowane identyfikatory to:

BIN - program w języku maszynowym
ASM - program w asemblerze
LST - program w BASIC'u zapisany instrukcją LIST
SAV - program w BASIC'u zapisany instrukcją SAVE
TXT - zbiór tekstu
DAT - zbiór danych

Rzeczą niezmiernie ważną jest wzajemna odpowiedzialność instrukcji zapisu i ładowania np. jeżeli program został zapisany przy użyciu SAVE odczytanie może nastąpić tylko po LOAD. Skąd i po co tyle możliwości?

Spośród instrukcji służących do zapisu i odczytu programu:

SAVE - działa szybciej i zapis zajmuje mniej miejsca ponieważ kodowane są całe instrukcje programu.

LIST koduje każdy znak osobno, i w ten sposób np. słowo READ wymaga użycia czterech znaków kodu, a nie jednego jak przy SAVE.

Przy odczycie LOAD z pamięci RAM zostaje wymazany znajdujący się tam program. Nie jest wymagane zastosowanie instrukcji NEW.

Natomiast ENTER wprowadza program nie kasując zawartości RAM-u. Instrukcja ta pozwala na wprowadzenie do pamięci programu nowych linii BASICu zawierających dane uzupełniające te programy lub programy pomocnicze. Istnieje wiele dyskowych systemów operacyjnych dla ATARI. Najnowszy DOS 2.5, który jest firmowany przez ATARI wydaje się być najlepszy. Jest on dziełem Optimized Systems Software - jednej z najlepszych firm piszących oprogramowanie dla ATARI.

Opis wszystkich możliwości DOS-a 2.5 przekracza ramy tego opracowania. Zakończmy więc nasze rozważania stwierdzeniem: Komputer bez oprogramowania to złom, ale komputer bez stacji dysków to połowa komputera.

7.2. Magnetofon

Jest najtańszym i najbardziej rozpowszechnionym typem pamięci zewnętrznej. ATARI nie może współpracować ze zwykłym magnetofonem, lecz tylko z dedykowanym tj. o specjalnej konstrukcji. Podnosi to nieco koszty systemu, zapewniając jednak dużą skuteczność zapisu m.in. dzięki zapisowi fazowemu oraz bieżącej weryfikacji.

Zapisu programu na kasetę dokonujemy instrukcjami:

```
SAVE "C: "
LIST "C: "
CSAVE
```

odczyt następuje po:

```
LOAD"C: "
ENTER"C: "
CLOAD
```

Różnice pomiędzy instrukcjami zostały opisane w części 7.1, dotyczącej stacji dysków. Przy wczytywaniu programów firmowych stosuje się jeszcze inną metodę - należy wcisnąć klawisz START równocześnie z włączeniem do sieci komputera. Po sygnale "beep" wciskamy klawisz PLAY magnetofonu, a następnie RETURN, w celu uruchomienia magnetofonu. Magnetofon nie pozwala odszukiwać programów wg nazw. Program "Poszukiwacz" likwiduje tę niedogodność.

1. Wpisując program należy w linii 90 POSITION pisać jako POS.
2. Na początku każdej taśmy należy wpisać program instrukcją LISI'"C: "
3. Skasować RAM instrukcją NEW
4. Wprowadzić bez numeru linii

```
OPEN#1, 8, 0, "C: ": ?#1; "nazwa": CLOSE#1
```

gdzie nazwą jest tytuł programu, który chcemy zapisać na taśmie.

5. Wcisnąć REC i PLAY w magnetofonie i dwa razy RETURN
6. Po pojawieniu się READY można rozpocząć wpisywanie programu instrukcją LIST"C:". Kroki 3 do 6 pozostawiamy dla każdego programu.

Nazwa programu jest limitowana przez wymiar A$ - w przykładzie 100 znaków. Aby znaleźć program należy wpisać do pamięci program „Poszukiwacz” po jego uruchomieniu instrukcją RUN podać szukaną nazwę.

PROGRAM "POSZUKIWACZ"

```
10 REM PROGRAM "POSZUKIWACZ"
20 DIM A$ (100),  B$(256)
30 ? "PODAJ NAZWE PROGRAMU",  INPUT A$
40 FOR I=1 TO 1.0E+97
50 OPEN #1, 4, 0, "C: "
60 TRAP 100; INPUT #1, B$
70 IF B$=A$ THEN 90
80 CLOSE #1: POKE 764,  33: NEXT I
90 ? " ': POSITION 2, 4: ? "NEW": ? : ? : ? "ENTER": CHR$ (34);  "C: "; CHR$(34): ? : ? : ?
"POKE 842, 12";  POSITION 2, 0: POKE 842, 1
100 ? "BLAD LADOWANIA - SPROBUJ JESZCZE RAZ"
```

Proponujemy przyjąć pewną zasadę dotyczącą zapisu na taśmie i ustawienia licznika magnetofonu. Zawsze przyjmujemy "zero" licznika jako fizyczny początek taśmy w kasecie, tj. początek rozbiegówki". Następnie przewijamy taśmę do stanu licznika 003 i rozpoczynamy zapis. Daje to dwie korzyści pod warunkiem, że umowa stanie się konsekwentnie przestrzeganym standardem. Po pierwsze mamy pewność, że nagrywanie rozpoczynamy na taśmie pokrytej nośnikiem magnetycznym, a nie na "rozbiegówce". Po drugie opis kasety np. 003-090 PROGRAM 1 pozwala zorientować się w jaki sposób traktujemy "rozbiegówkę" i że stosujemy "standard trójki". Również odstępy między kolejnymi programami na taśmie byłoby łatwiej ustalić na trzy jednostki licznika.

Jeżeli wskazania te upowszechnią się byłoby to dużym ułatwieniem przy wymianie programów zapisanych na kasetach.

Na koniec jeszcze jedna uwaga - kasetę, której chcemy użyć do zapisu programów, a która była wcześniej nagrana, należy uprzednio skasować. Ponadto zalecamy używanie zwykłych żelazowych kaset dobrej jakości.

Na koniec przedstawiamy program "Turbogenerator" pozwalający zwiększyć szybkość transmisji do i z magnetofonu o ok. 50%. Program ten zajmuje obszar pamięci przeznaczony dla DOS-a i nie może pracować z nim równocześnie. Program został napisany w języku BASIC, ale jego użycie wymaga stworzenia tzw. "boot tape". Będziemy używać angielskiego terminu, ponieważ nie istnieje odpowiednie słowo w języku polskim, a tłumaczenie opisowe jest bardzo niewygodne.

Boot tape (analogicznie boot disc) określa taśmę, na której został zapisany program w wersji maszynowej i jego wczytywanie następuje poprzez wciśnięcie klawisza funkcyjnego START z równoczesnym włączeniem komputera do sieci.

Po przepisaniu programu '"Turbogenerator" wpisujemy go na kasetę instr. CSAVE. Teraz możemy już zrobić boot tape - instr. RUN. Po usłyszeniu podwójnego sygnału "beep" wkładamy nową kasetę do magnetofonu i wciskamy REC i PLAY oraz RETURN. Po zatrzymaniu się magnetofonu i pojawieniu się napisu READY mamy gotową taśmę boot z programem 'Turbogenerator".

Aby wykorzystać "Turbogenerator" musimy wersję boot załadować do komputera przed wpisaniem swojego programu (naciśnięty klawisz START z chwilą włączenia komputera). Program ten jest odporny na RESET i może współpracować z instrukcjami CSAVE, SAVE i LIST oraz CLOAD, LOAD i ENTER.

PROGRAM "TURBOGENERATOR"

```
10 REM PROGRAM "TURBO-GENERATOR"
20 POKE 752, 1: POKE 82, 2: ? CHR$(125)
30 POSITION 7, 5: PRINT "PROSZE CZEKAC ..."
100 FOR A=1536 TO 1612: READ B: POKE A, B: NEXT A
101 DATA 104, 162, 16, 169, 3, 157, 66, 3, 169, 8, 157, 74, 3, 169, 128, 157, 75, 3, 169,
74, 157, 68, 3, 169, 6
102 DATA 157, 69, 3, 32, 86, 228, 48, 40, 169, 11, 157, 66, 3, 169, 0, 157, 68, 3, 169, 32,
157, 69, 3, 169, 0
103 DATA 157, 72, 3, 169, 2, 157, 73, 3, 32, 86, 228, 48, 10, 1, 69, 12, 157, 66, 3, 32,
86, 228, 48, 0, 96, 67
104 DATA 58, 155
110 FOR A=8192 TO 8443:  READ B: POKE A, B: NEXT A
111 DATA 0, 4, 0, 7, 37, 7, 169, 60, 141, 2, 211, 169, 227, 141, 231, 2, 133, 14, 169, 8,
141, 232, 2, 133, 15
112. DATA 173, 254, 1.91, 133, 10, 173, 255, 191, 133, 11, 24, 96, 160, 0, 185, 26, 3, 201,
0, 240, 9, 200, 200, 200, 192
113 DATA 34, 208, 242, 56, 96, 169, 67, 153, 26, 3, 200, 169, 73, 153, 26, 3, 200, 169, 7,
153, 26, 3, 96, 99, 7
114 DA'I'A 27.3, 7, 121, 253, 199, 7, 203, 253, 228; 252, 76, 89, "I, 0, 169, 67, 141, 238,
2, 169, 4, 141, 239, 2, 96
115 DATA 165, 43, 133, 62, 165, 42, 41, 12, 201, 4, 240, 5, 20, 1, 8, 240, 14, 96, 76, 247,
252, 160, 80, 198, 17, 169
116 DA'I'A 0, 141, 137, 2, 96, 169, 128, 141, 137, 2, 169, 2, 32, 252, 253, 48, 238, 1.69,
G7, 141, 4, 210, 169, 4, 141
117 DATA 6, 210, 169, 96, 141, 0, 3, 32, 104, 228, 169, 52, 14, 7, 2, 211, 166, 98, 188,
143, 254, 189, 141, 254, 170, 169
118 DATA 3, 32, 92, 228, 169, 255, 141, 42, 2, 165, 17, 240, 1, 88, 173, 42, 2, 208, 247,
169, 0, 133, 61, 160, 1, 96
119 DATA 166, 61, 157, 0, 4, 230, 61, 160, 1, 224, 127, 240, 1, 96, 169, 252, 32, 13, 8,
169, 0, 133, 61, 96, 173
120 DATA 137, 2, 48, 8, 160, 1, 169, 60, 141, 2, 211, 96, 166, 61, 240, 10, 142, 127, 4,
169, 250, 32, 13, 8, 48
121 DATA 236, 162
122 FOR A=8444 TO 8674: READ B: POKE A, B: NEXT A
123 DATA 127, 169, 0, 157, 0, 4, 202, 16, 250, 169, 254, 32, 1, 3, 8, 76, 231, 7, 141, 255,
3, 169, 85, 141, 253, 3
```

```
124 DATA 141, 254, 3, 169, 87, 32, 30, 8, 96, 141, 2, 3, 169, 0, 141, 9, 3, 169, 131, 141,
8, 3, 169, 3, 141
125 DATA 5, 3, 169, 253, 141, 4, 3, 169, 96, 141, 0, 3, 169, 0, 141, 1, 3, 169, 35, 141, 6,
3, 173, 2, 3
126 DATA 160, 64, 201, 82, 240, 2, 160, 128, 140, 3, 3, 165, 6, 2, 141, 11, 3, 32, 91, 8,
96, 169, 1, 133, 66, 173
127 DATA 1, 3, 72, 173, 71, 2, 240, 26, 162, 8, 32, 189, 201, 2, 40, 19, 138, 72, 32, 5,
216, 104, 170, 144, 242, 169
128 DATA 0, 141, 72, 2, 141, 255, 209, 240, 3, 32, 148, 8, 104, 141, 1, 3, 169, 0, 133, 66,
140, 3, 3, 172, 3,
129 DATA 3, 96, 186, 142, 24, 3, 169, 1, 133, 66, 173, 0, 3, 20, 1, 96, 208, 3, 76, 172, 8,
76, 131, 233, 76, 212
130 DATA 235, 173, 3, 3, 16, 248, 169, 67, 141, 4, 210, 169, 4, 141, 6, 210, 32, 23, 236,
166, 98, 188, 21, 238, 173
131 DATA 11, 3, 48, 3, 188, 17, 238, 162, 0, 32, 226, 237, 169, 52, 141, 2, 211, 173, 23,
3, 208, 251, 32, 135, 235
132 DATA 32, 136, 234, 76, 4, 236
140 POSITION 2, 5: PRINT "W celu wygenerowania programu 'TURBO'
141 PRINT "nalezy wlozyć kasete do magnetofonu"
142 PRINT "wcisnac klawisze <PLAY> i, <RECORD>,  "
143 PRINT "po czym nacisnac dowolny klawisz"
144 PRINT "komputera (z wyjatkiem BREAK)."
200 A=USR(1536)
210 PRINT CHR$(125)
220 POSITION 2, 4: PRINT "Sposob uzycia -": PRINT : PRINT
224 PRINT "Przed rozpoczeciem programowania"
225 PRINT "nalezy wczytac program TURBO.": PRINT
226 PRINT "Komendy CSAVE/LIST/PRINT/PUT beda"
228 PRINT "szybciej wprowadzaly informacje"
229 PRINT "na magnetofon,  ktore nastepnie beda"
230 PRINT "czytane ze zwiekszona predkoscia."
235 POKE 752, 0: NEW
```

Jedynym sposobem sprawdzenia czy 'Turbogenerator' został prawidłowo wczytany jest sprawdzenie komórki 2000

PRINT PEEK (2000)

Odpowiedź komputera:
1 -TURBO
0 - bez TURBO

Rozdział 8

BLOKI STERUJĄCE OPERACJAMI WEJŚCIA/WYJŚCIA

Operacje we/wy są sterowane poprzez bloki sterujące IOCB (ang. Input Output Control Block). IOCB jest to obszar pamięci, w którym znajduje się lista danych określająca typ i sposób przeprowadzenia operacji we/wy; zawiera typ operacji, długość bufora, adres bufora i dwie zmienne sterujące. ATARI BASIC przewiduje maksymalnie 0 IOCB, z których:

IOCB#0 jest używany dla operacji we/wy z edytorem ekranowym (E:),

ICOB#6 jest używany dla operacji we/wy z ekranem (S:),

IOCB#7 jest używany dla operacji we/wy przy wykorzystaniu instrukcji LPRINT, LOAD, SAVE czyli komunikacji z drukarką, magnetofonem i stacją dysków.

IOCB#1 do #5 mogą być używane dowolnie zaś IOCB#0 nie możne być nigdy otwierany i zamykany z programu BASIC. IOCB#6 może być wykorzystywany jeżeli w programie nie korzystamy z trybu graficznego "#6", zaś IOBC#7 może być wykorzystywany jeśli nie używamy instrukcji LPRINT, SAVE, LIST, LOAD lub ENTER.

Każda instrukcja we/wy musi zawierać numer bloku IOCB (numer kanału) dla danej operacji . W BASIC'u występują następujące instrukcje:

OPEN i **CLOSE**
INPUT i **PRINT**
PUT i **GET**
NOTE i **POINT**
STATUS

Omówimy kolejno każdą z nich.

8.1. Użycie instrukcji OPEN i CLOSE

format:

OPEN# ***nr kanału, kod operacji, wyrażenie pomocnicze, opis zbioru***

przykład: OPEN# 2 , 8 , 0 , "D: ATARI800. BAS"

Instrukcja OPEN dołącza dany blok IOCB do odpowiadającego podprogramu sterującego urządzeniem (ang. device handler), przesyła podane parametry do podprogramu sterującego urządzeniem, oraz inicjuje zmienne sterujące głównego programu obsługi we/wy. Parametry instrukcji OPEN są definiowane następująco:

- stały znak, który musi być wprowadzony przez użytkownika,

nr kanału - liczba z przedziału od 1 do 7, która zostaje przyporządkowana zbiorowi lub urządzeniu, jest to więc numer bloku IOCB

kod operacji - liczba, która określa typ operacji:

4 - operacja wejścia - znacznik zbioru ustawiony jest na początek zbioru,

6 - operacja wejścia listy zbiorów na dyskietce,

8 - operacja wyjścia - znacznik zbioru ustawiony jest na początku zbioru,

9 - operacja dołączania - znacznik zbioru ustawiony jest na końcu zbioru, kod 9 pozwala po instrukcji INPUT na natychmiastowe wprowadzenie znaku z klawiatury bez naciskania klawisza RETURN.

12 - operacje wyjścia lub wejścia - znacznik ustawiony jest na początku zbioru, nie jest możliwe dopisywanie do zbioru,

13 - operacja wejścia lub wyjścia - znacznik zbioru ustawiony jest na końcu zbioru, jest możliwe dopisywanie do zbioru.

wyrażenie pomocnicze - kod zależy od typu urządzenia. Np. 83 na tej pozycji powoduje stronicowanie pisania przez drukarkę ATARI 820, w pozostałych przypadkach jest zwykle równy zeru.

opis zbioru - wyrażenie definiuje urządzenie, jego numer oraz ewentualnie nazwę zbioru przyporządkowaną temu urządzeniu.

W przykładzie powyższym otwarto kanał numer 2jako wyjście dla zbioru w stacji dysków numer 1 (o nazwie A'I'ARI 800.BAS). Jeśli na dyskietce w stacji dysków numer 1 nie ma zbioru o takiej nazwie w DOS (ang. Disk Operating System) tworzy zbiór o tej samej nazwie. Jeśli taki zbiór istnieje, to instrukcja OPEN niszczy go i tworzy nowy o tej samej nazwie. Jeśli był już wcześniej otwarty, to na ekranie pojawia się błąd nr 129.

format:

CLOSE# ***nr kanału***

przykład: CLOSE#2

Instrukcja CLOSE zamyka kanał, który był uprzednio otwarty. Liczba po znaku # (znak ten musi występować) określa numer kanału. Użycie instrukcji CLOSE dla zamkniętego kanału nie powoduje żadnego błędu.

Często stosuje się w celu uniknięcia błędu najpierw użycie CLOSE, później zaś OPEN w tej samej linii programu, np.:

```
255 CLOSE#1:OPEN#1,4,0,"K:"
```

Uwaga: Instrukcja BASIC'u **END** zamyka wszystkie otwarte kanały z wyjątkiem IOCB#0.

8.2. Użycie instrukcji INPUT i PRINT

format: INPUT [*# nr kanału zmienna [,zmienna...]*
przykład:

```
INPUT#2 ; x, y
INPUT#2;N$
INPUT A
```

Instrukcja ta jest używana dla żądania danych (zarówno numerycznych jak i tekstowych) z wyszczególnionego kanału. Jeśli używa się jej bez podania numeru IOCB dane są żądane tylko z klawiatury. Instrukcja INPUT używa tzw. rekordu we/wy czyli określonej porcji danych, na końcu której występuje znak końca rekordu EOF (ang. End of Line) o kodzie 155 (dziesiętnie). Rozmiar rekordu jest ściśle określony dla danego typu danych. Dla zmiennych arytmetycznych jest on zgodny z formatem tych zmiennych, zaś dla zmiennych tekstowych jest zgodny z długością tekstu przy czym dla instrukcji INPUT długość rekordu nie może przekraczać 110 znaków. Przeczytany rekord (rekordy) za pomocą instrukcji INPUT zostaje podstawiony do zmiennej (ewentualnie zmiennych)

format:

PRINT *"wyrażenie1"; wyrażenie2*

lub

PRINT# *numer kanału; wyrażenie 1(, wyrażenie2)*

przykłady:

```
PRINT#2; X, Y
PRINT#2; A$
PRINT"X=";X
```

Instrukcja ta wpisuje dane (zarówno tekstowe jak i numeryczne - zmienne lub stałe) do urządzenia już otwartego. Jeśli żaden numer kanału nie jest wyszczególniony to komputer samoczynnie przyjmuje edytor ekranowy. Jeżeli dane są kierowane do urządzenia nie otwartego powstanie błąd 133.

Instrukcja PRINT używa również rekordów bez ograniczeń ich długości. Po wysłaniu ciągu danych za pomocą instrukcji PRINT na końcu automatycznie zostaje umieszczony znak EOL.

8.3. Użycie instrukcji PUT i GET

format:

PUT# ***nr kanału, wyrażenie pomocnicze***

przykład:

PUT#6 , ASC ("A")

Instrukcja PUT wysyła pojedynczy bajt (o wartości z przedziału od 0 do 255) do urządzenia wyszczególnionego przez numer bloku IOCB.

format:

GET# *nr kanału, zmienna*

przykład:

GET#2 , X

Instrukcja GET czyta pojedynczy bajt z urządzenia wyszczególnionego przez numer bloku IOCB. Wartość bajta podstawia do zmiennej. Zwróćmy uwagę na różnice pomiędzy instrukcjami PRINT i PUT. PRINT po wysłaniu ciągu danych umieszcza znak końca rekordu EOL, którego instrukcja INPUT używa później do rozpoznania długości rekordu. Zbiór utworzony przy pomocy instrukcji PUT wygląda jak jeden długi rekord, chyba, że użytkownik umieści znak EOL (o kodzie 155) przy pomocy instrukcji PUT na przykład w celu przeznaczenia tego zbioru do czytania instrukcji INPUT.

8.4. Użycie instrukcji NOTE i POINT

Jeżeli zbiory są tworzone w sposób sekwencyjny, to znaczy, że przy otwarciu nowego zbioru (za pomocą OPEN) chwilowy znacznik zbioru CFP (ang. Current File Pointer) jest ustawiony na początek zbioru i każde wpisywanie danych do zbioru (rekordu albo bajta) przesuwa znacznik na następny bajt po wpisaniu porcji danych. Przy czytaniu danych ze zbioru (za pomocą instrukcji INPUT albo GET) proces odbywa się również sekwencyjnie.

Jeżeli utworzony zbiór zawiera 200 rekordów, a interesują nas w danej chwili dane zawarte w rekordach: 25, 120 i 190 to proces sekwencyjnego czytania będzie trwał długo, co przy częstym korzystaniu z tego zbioru staje się bardzo nieekonomiczne.

Sekwencyjna metoda czytania polega na tym że czytane są rekordy (bajty) w pętli od początku zbioru z opuszczeniem wszystkich poprzednich danych, aż do otrzymania żądanych. Do bezpośredniego, a więc szybszego, dostępu do danych w zbiorze służą instrukcje:

NOTE i POINT

format:

NOTE# ***nr kanału, zmienna num1, zmienna num2***

przykład:

NOTE#2 , A, B

Instrukcja ta jest używana do bieżącego określania wartości znacznika CFP dla danego kanału. Pierwsza zmienna tzw. zmienna sektorowa wyraża numer sektora dyskietki zajmowanego przez zbiór. Przyjmuje wartości od 1 do 719 (DOS2) albo 1010 (DOS2.5).

Druga zmienna, tzw. zmienna znakowa, określa położenie znacznika CFP w danym sektorze. Przyjmuje wartości od 0 do 125.

format:

POINT# ***numer kanału, zm. num1, zm num2***

przykład:

POINT#2 , X, Y

Instrukcja POINT jest odpowiednikiem instrukcji NOTE przy odczytywaniu rekordów w trybie bezpośredniego dostępu. Użycie jej przesuwa znacznik CFP w miejsce określone poprzez jego parametry, które są definiowane jak w instrukcji NOTE.

Instrukcji POINT można używać do wpisywania do zbioru w trybie bezpośredniego dostępu lecz nieumiejętne jej zastosowanie grozi zniszczeniem ważnych danych. Nie zalecamy takiego stosowania instrukcji, a w razie zaistnienia takiej konieczności radzimy wypróbowanie działania programu na specjalnie do tego celu stworzonym zbiorze testowym.

Próba ustawienia znacznika zbioru poza otwartym zbiorem powoduje powstanie błędu o numerze 170 lub 171.

Przy tworzeniu zbioru o dostępie bezpośrednim należy założyć zbiór pomocniczy zawierający pary obydwu zmiennych, które są adresami rekordów na dyskietce.

Przykładem wykorzystania bezpośredniego dostępu do zbioru jest program „Demonstracja działania instrukcji NOTE/POINT". Program składa się z dwóch części. W pierwszej części (linie 10 do 260) z klawiatury wprowadzone są poszczególne rekordy, które program umieszcza w zbiorze "D:DATFIL.DAT.", tworząc zbiór pomocniczy "D:POINTS.DAT' zawierający znaczniki zbioru danych. W części drugiej (linie 300 do 500) komputer pyta się o numer żądanego rekordu. Korzystając z wcześniej wczytanej tablicy znaczników komputer pobiera z dysku żądany rekord i wypisuje go na ekranie. Ta część programu uruchamiana jest instrukcją GOTO 300.

PROGRAM "DEMONSTRACJA DZIAŁANIA INSTRUKCJI NOTE/POINT"

```
10 REM PROGRAM "DEMONSTRACJA DZIALANIA
20 REM INSTRUKCJI NOTE/POINT
30 DIM A$(40):LICZNIK=0
40 OPEN #1,8,0,"D:DATFIL.DAT"
50 OPEN #2,8,0,"D:POINTS.DAT"
60 REM WPROWADZIC NALEZY 40 LUB MNIEJ
70 REM ZNAKOW
75 PRINT CHR$(125)
80 INPUT A$
90 REM JESLI TYLKO <RETURN> TO STOP
100 IF LEN(A$)=0 THEN 250
110 NOTE #1,X,Y
120 PRINT #1;A$
130 LICZNIK=LICZNIK+1: IF LICZNIK=100 THEN 250
140 REM PRZECHOWAJ ZNACZNIK ZBIORU
150 PRINT #2;X:PRINT #2,Y
160 REM WYSWIETL PARAMETRY X ORAZ Y
170 PRINT "X=";X,"Y=";Y
180 REM TERAZ WEZ NASTEPNY REKORD
190 GOTO 80
230 REM KONIEC DANYCH: WPISZ -1,
```

```
240 REM NA PRZYKLAD, DLA IDENTYFIKACJI
250 PRINT #2,-1
260 END
300 REM TU ZACZYNA SIE PROGRAM
310 REM DO ODCZYTYWANIA DANYCH
320 CLR :DIM A$(40), X(100), Y(100)
330 OSTATNI =0
340 OPEN #1,4,0,"D:DATFIL.DAT"
350 OPEN #2,4,0,"D:POINTS.DAT"
360 REM CZYTAJ ZNACZNIKI DO TABLIC
370 INPUT #2,P:IF P<>-1 THEN INPUT #2,Q:OSTATNI=OSTATNI +1:X(OSTATNI)=P :
Y(OSTATNI)=Q :GOTO 370
375 PRINT CHR$(125)
380 REM CZYTAJ NUMER REKORDU
390 TRAP 500
400 PRINT "WPISZ NUMER REKORDU"
410 INPUT R
420 IF R<1 THEN 480:REM KONIEC PROGRAMU
430 IF R>STATNI THEN PRINT "REKORD NIE ISTNIEJE W ZBIORZE": GOTO 400
440 REM TU NASTEPUJE SWOBODNY
450 REM DOSTEP DO REKORDU
460 POINT #1,X(R),Y(R)
470 INPUT #1,A$:PRINT A$:GOTO 380
480 PRINT "KONIEC WYKONYWANIA PROGRAMU"
490 END
500 TRAP 40000:GOT0 390:REM JESLI WYSTAPI BLAD TO PROBUJ CZYTAC DALEJ
```

8.5 Użycie instrukcji STATUS

format:

STATUS# ***nr kanału, zm num***

przykład:

```
STATUS#5,ERR: PRINT ERR
```

Instrukcja STATUS jest używana do określania warunków (stanu) zbioru określonego przez podanie numeru kanału. Stosuje się ją głównie do określania rodzaju błędu, który wystąpiłby przy próbie dostępu do zbioru. Pod zmienną arytmetyczną występującą w formacie instrukcji (w przykładzie ERR) zostaje podstawiona liczba odpowiadająca stanowi kanału. Interpretacja kodów STATUS jest identyczna ze znaczeniem błędów - dodatek A. Ponadto kod 1 oznacza "wszystko w porządku".

Rozdział 9

PAMIĘĆ WEWNĘTRZNA

Podziału pamięci komputerów można dokonać według wielu kryteriów. Przedstawimy trzy z nich, ograniczając klasyfikację do tych jej rodzajów, które występują w mikrokomputerach ATARI.

1. Pamięć operacyjna (ang. operational memory lub main memory).

Niekiedy pamięć ta jest nazywana pamięcią główną, ze względu na to, że w niej przechowywane są aktualnie wykonywane programy i dane potrzebne do ich wykonywania. Z pamięcią operacyjną mikroprocesor komunikuje się bezpośrednio. Jest ona wykonana jako pamięć półprzewodnikowa.

2. pamięć sterująca (ang. control memory).

Częściej zwana pamięcią systemową, w której znajdują się programy zarządzające pracą mikrokomputera i współpracą ze standardowymi urządzeniami zewnętrznymi.

3. Pamięć zewnętrzna (ang. external memory).

Jest to pamięć o bardzo dużej pojemności, służąca do przechowywania dużych zbiorów danych. Komunikacja mikroprocesora z tą pamięcią jest możliwa tylko poprzez pamięć operacyjną. Do pamięci operacyjnej są przesyłane, przez kanał IOCB, bloki informacji przechowywane w pamięci zewnętrznej. Jest to najczęściej pamięć taśmowa lub dyskowa, a więc najwolniej działająca lecz o największej pojemności.

Innej klasyfikacji pamięci można dokonać na podstawie właściwości, od których zależy wybór danego rodzaju pamięci dla zastosowania w komputerze.

1. pamięć stała (ang. read only memory - ROM).

Jest to pamięć, do której zapisano jednokrotnie informację (na zlecenie producenta komputera) i jej zawartość nic może być zmieniona przez użytkownika. Użytkownik maże z niej korzystać dowolnie odczytując wielokrotnie zapisane w niej informacje.

2. Pamięć o zmiennej zawartości (ang. volatile memory).

Pamięć o zmiennej zawartości można wielokrotnie zapisywać i odczytywać. 7_ tego typu pamięci wykonana jest pamięć operacyjna. W rozpatrywanym typie mikrokomputerów pamięć ta jest pamięcią ulotną tzn., że po zaniku napięcia zawartość pamięci jest tracona bezpowrotnie. Zawartość tej pamięci nawet przy włączonym zasilaniu mogłaby być utracona po pewnym czasie gdyby nic zachodził proces regeneracji, który wykonywany jest automatycznie podczas cyklu odświeżania pamięci (ang. refresh cycle).

Jeszcze inną klasyfikację można przeprowadzić w oparciu o sposób organizacji wybierania adresów pamięci.

W mikrokomputerze ATARI zastosowano pamięć dynamiczną o swobodnym dostępie (ang. random acces memory - RAM), w której czas dostępu nie zależy od adresu (numeru) komórki. Wyjaśnimy teraz użyte wcześniej terminy związane z pamięciami.

Pojemność pamięci jest parametrem odnoszącym się do ilości przechowywanej informacji i określa się ją przez podanie liczby komórek pamięci i liczby elementów w komórce. Najczęściej liczbę komórek zastępuje się liczbą słów (lub bajtów czyli słów ośmioelementowych), a liczbę elementów - liczbą bitów w słowie. 1 tak na przykład mówimy o pamięci o pojemności 64 kbajtów (czyt. kilobajtów). Oznacza to, że pojedyncza komórka zawiera osiem elementów - bitów (bit jest najmniejszą ilością informacji cyfrowej mogącą przyjąć dwie umowne wartości logiczne zero lub jeden). Liczba komórek tej pamięci wynosi 64 jednostki nazywane bajtami . Dla pamięci o pojemności większej od 1024 wprowadzono jednostkę kilosłowo (1 kilosłowo = 1024 słowa, w przypadku słów 8-bitowych 1 lkbajt=1024 bajty). Ostatecznie więc 64 kbajty to pojemność 64*1024 słów ośmiobitowych czyli 65 536 komórek ośmioelementowych.

Szybkość działania pamięci można określić w różny sposób. Jeden z nich polega na podaniu tzw. czasu dostępu. Czas dostępu jest to czas jaki upływa pomiędzy zażądaniem odczytu z pamięci a ukazaniem się zawartości zaadresowanej komórki na wyjściu pamięci. **W** mikrokomputerach ATARI użyto pamięci

dynamicznej o czasie dostępu równym 150 ns (nanosekund). Oznacza to, że gdyby proces odczytu był zależny jedynie od czasu dostępu to w czasie 1 sekundy można by przeczytać nieco ponad 6 600 000 komórek.

9.1. Mapa pamięci

Przystąpimy teraz do opisu pamięci mikrokomputera ATARI 800XL. Pamięć jego składa się z pamięci stałej ROM - 24 kbajty, oraz pamięci operacyjnej RAM - 64 kbajty.
Pamięć stałą podzielić można na dwa duże obszary:
1. Obszar, w którym przechowywany jest system operacyjny (ang. operating system - OS)
2. Obszar, w którym przechowywany jest interpreter BASIC'u.

Każdy z tych obszarów można podzielić na wiele mniejszych, w których są przechowywane specyficzne podprogramy czy dane i które można wykorzystać w swoim programie użytkowym. Zainteresowanych takim niestandardowym wykorzystaniem podprogramów systemowych odsyłamy do szczegółowych opisów w pracach "Mapping of ATARI" oraz "Operating System User's Manual" dostępnych w angielskiej wersji językowej.

Pamięć mikrokomputera ATARI 800XL jest dekodowana w całym jego obszarze adresowym 64 kbajtów. Obszar adresowy jest narzucony przez typ zastosowanego mikroprocesora - 6502. Nie jest możliwe dalsze rozszerzanie pamięci bez ingerowania w strukturę mikrokomputera. Jednakże struktura istniejącego systemu oraz mechanizmy utworzone przez jego projektantów stwarzają szerokie możliwości zmiany przydziału pamięci dla poszczególnych bloków.

Dla przykładu naciskając klawisz funkcyjny OPTION a następnie włączając zasilanie powodujemy tzw. wyłączanie BASIC'u czyli odłączenie tej części pamięci stałej, w której umieszczony jest interpreter BASIC i podłączenie na jego miejsce dodatkowej pamięci RAM. Stosowane jest to zwykle przy wczytywaniu większych gier lub w przypadku korzystania z innego języka programowania.

Dla podkreślenia dużej elastyczności architektury mikrokomputera ATARI w porównaniu z innymi jemu podobnymi zaznaczymy, że cały system operacyjny ATARI 800 XL można wymienić na inny. Stwarza to możliwość stworzenia własnego mikrokomputera w oparciu o hardware mikrokomputera ATARI 800XL (!), jak również wykorzystanie jego układu elektronicznego do specjalistycznego zastosowania działającego pod kontrolą systemu operacyjnego wczytywanego np. z dyskietki bez zmian układowych.
Pamięć mikrokomputera ATARI 800XL można podzielić na 4 główne obszary:
1. obszar RAM-u,
3. obszar dla cartridge'a,
3. obszar we/wy,
4. obszar rezydującego systemu operacyjnego tzw. OS ROM.

Poniżej podamy najbardziej ogólny podział pamięci wraz z adresami (w postaci heksadecymalnej), jej poszczególnych części.

0000- 1FFF = RAM (minimum pamięci RAM wymagane do działania komputera)
2000 - 7FFF = obszar rozszerzenia RAM-u
8000 - 9FFF = cartridge B lub RAM
A000 - BFFF = BASIC, cartridge A lub RAM
C000 - CFFF = podprogram OS RAM lub RAM
D000 - D7FF = obszar dekodowania we/wy
D800 - DFFF = obszar procedur operacji zmiennoprzecinkowych
E000 - FFFF = rezydujący system operacyjny OS ROM

Na następnych stronach podzielimy wyżej wymienione obszary na pomniejsze części oraz opiszemy ich zastosowanie. W tym celu wprowadzimy pojęcie strony pamięci. Stroną pamięci nazywamy obszar 256 bajtów, który rozpoczyna się zawsze od adresu, w którym młodszy bajt jest równy zeru. Na przykład 4 strona pamięci rozpoczyna się od adresu 0400 i rozciąga się do 04FF (heksadecymalnie). Starzy bajt wyraża więc bezpośrednio numer strony pamięci.

Obszar RAM-u

Rozmiar tego obszaru jest zależny od aktualnego trybu pracy określonego przez system operacyjny bądź przez wykonywany program. Można podzielić go na następujące mniejsze obszary:

strona 0 = obszar komórek użytkowych i obszar niezbędny dla trybu adresowania strony zerowej mikroprocesora 6502
strona 1 = obszar stosu mikroprocesora 6502
strony 2-4 = obszar komórek użytkowych systemu operacyjnego i jego obszar pracy
strony 5-6 = obszar programów użytkowych
strony 7-XX = obszar wczytywanego oprogramowania (ang. bootable software) i wolnego RAM-u
strony XX – do ostatniego adresu aktualnie podłączonego RAM-u = obszar ekranu i programu zarządzającego wyświetlaniem informacji (ang. display list)

XX zależy od trybu pracy ekranu tzw. trybu graficznego.
Dalej opiszemy poszczególne podobszary i ich wykorzystanie.

Strona 0

Mikroprocesor 6502 posiada specyficzny tryb adresowania, w którym strona zerowa ma specjalne znaczenie. Odwołując się do adresów zawartych na stronie zerowej (0000-00FF) mikroprocesor działa szybciej. Ten sposób adresowania wymaga mniej rozkazów oraz jest to jedyny mechanizm dla tzw. adresowania pośredniego. Dla systemu operacyjnego zastrzeżone są komórki o adresach od 00 do 7F. Procedury operacji zmiennoprzecinkowych, jeśli są używane, wymagają użycia komórek o adresach 00D4 do 00FF.

Możliwe jest wykorzystanie przez użytkownika wszystkich komórek strony zerowej pod warunkiem, że zostały wyeliminowane kolizje z systemem operacyjnym, a przede wszystkim zabronione wszystkie przerwania dla systemu operacyjnego.

Strona 1

Strona 1 jest zarezerwowana dla tzw. stosu hardware'owego mikroprocesora 6502. Rozkazy JSR, PHA, PHP i wszystkie przerwania powodują wpisywanie określonych informacji na tę stronę, zaś rozkazy RTS, PLA, PLP i RTI powodują czytanie informacji z tego obszaru.

Strony 2-6

Lokacje pamięci 0200 - 047E są zmiennymi pracy systemu operacyjnego, jego tablicami oraz buforami danych. Część tego obszaru może być wykorzystywana przez użytkownika pod warunkiem braku kolizji z systemem operacyjnym. Na przykład bufory magnetofonu i drukarki mogą być wykorzystane jeżeli wiadomo, że podczas wykonywania programu nie wystąpi żadna transmisja między komputerem a tymi urządzeniami.

Lokacje pamięci 0480 - 06FF są przeznaczone do wykorzystania przez użytkownika z wyjątkiem sytuacji gdy używane są procedury zmiennoprzecinkowe, które używają lokacji 057E -05FF.

Strony 7 i dalsze (obszar wczytywanego oprogramowania)

Strona 7 jest początkiem tzw. boot regionu. Jeśli po włączeniu zasilania następuje proces wczytywania z dyskietki lub kasety magnetofonowej, to właśnie w to miejsce pamięci wprowadzane są czytane dane. Oczywiście dzieje się tak w przypadku korzystania przez wczytywany software ze standardowych procedur. W przypadku gdy wczytywany program umieści swoją procedurę dalszego czytania (wczytaną standardowo na stronę 7) i odda jej sterowanie, dalsze wczytywanie może się dokonywać w dowolny obszar RAM-u jednak poniżej tzw. obszaru display list.

Obszar ekranu i tzw. display list

Display list są to dane i specjalny program dla układu zarządzającego generacją obrazu na ekranie monitora. Są one umieszczone wraz z treścią ekranu w ostatnim obszarze RAM-u aktualnie podłączonego do systemu. Ich rozmiar zależny jest od trybu pracy wyświetlania ekranu (trybu grafiki). Obszar leżący poniżej tych danych, a powyżej ostatnio wybieranej komórki pamięci nazywany jest obszarem wolnego RAM-u.

Obszar cartridge A i B

Obszary od 8000 do 9FFF oraz od A000 do BFFF tworzą dwa 8 kbajtowe obszary przeznaczone jako obszar na dodatkowe oprogramowanie na tzw. wkładkach, podłączonych do mikrokomputera przez specjalne gniazdo. *Przy* czym drugi z tych obszarów A000-BFFF jest standardowo przeznaczony dla interpretera BASIC-u jeśli jest wykorzystywany. W zależności od pojemności pamięci zawartej we wkładce jeden lub dwa te obszary są zajmowane. Gdy zajmowany jest tylko jeden z dwóch obszarów (ajent to determinowane konstrukcją wkładki), do drugiego obszaru może być podłączona pamięć RAM. W przypadku nie korzystania z interpretera BASIC-u i z wkładki (cartridge'a) do obydwu obszarów podłączona jest pamięć RAM. Od konstrukcji wkładki oraz programu w niej zawartego zależy tryb pracy mikrokomputera, np. jest czy nie jest możliwa współpraca ze

stacją dysków. Umieszczony w ten sposób program może w całości przejąć kontrolę nad systemem. Tego typu wkładki nazywane są diagnostycznymi, przejmują one kontrolę nad mikrokomputerem jeszcze przed inicjacją systemu operacyjnego.

Analiza sygnałów doprowadzonych do gniazd wkładki pozwala snuć domysły dotyczące innych możliwości wykorzystania tego złącza, jednakże wkładki diagnostyczne są zastrzeżone dla serwisów ATARI i nie są powszechnie dostępne.

Obszar dekodowania we/wy

Mikroprocesor 6502 podejmuje operacje we/wy poprzez adresowanie układów we/wy jako pamięci. Związane jest to nieodłącznie z jego architekturą. Dlatego został wyodrębniony obszar dekodowania we/wy, w którym wybranie adresu nie powoduje dostępu do któregokolwiek układu pamięci, natomiast generowany jest sygnał wybierający układ we/wy zależny od wybranego adresu.

Pod tymi adresami znajdują się rejestry układów we/wy, niektóre tylko do czytania, inne tylko do pisania, jeszcze inne zezwalające na obydwa typy operacji.

W obszarze D000-D7FF wybranie każdego adresu powoduje wygenerowanie sygnału wybierającego układy we/wy. Jednakże tylko niektóre adresy mają swój układowy odpowiednik. Poniżej wymieniamy wszystkie z nich.

D000-D01F = rejestry układu GTIA
D200-D21F = rejestry układu POKEY
D300-D31F = rejestry układu PIA
D400 - D41F = rejestry układu ANTIC

9.2. Ważniejsze komórki pamięci

Adres (dec.)	ADRESY PAMIĘCI
14, 15	zawiera najwyższy adres jaki może być użyty przez tekst programu lub zmienne
88, 89	zawiera najniższy adres pamięci ekranu. Pod adresem wskazanym przez zawartość tych komórek znajduje się kod znaku wyświetlanego w lewym górnym rogu ekranu
128, 129	wskaźnik końca pamięci dla BASIC
144, 145	wskaźnik początku pamięci dla BASIC
741, 742	zawiera najwyższy adres wolnej pamięci
743, 744	zawiera najniższy adres wolnej pamięci
ADRESY EKRANOWE	
82	zawiera ilość kolumn tworzących lewy margines w trybie graficznym 0. Standardowa wartość wynosi 2.
83	liczba 39 - PEEK(83) określa ilość kolumn tworzących prawy margines w trybie graf. 0. Standardowa wartość wynosi 39.
90	zawartość określa wiersz, od którego rozpoczyna rysowanie DRAWTO lub XIO18
91, 92	zawartość określa kolumnę, od której rozpocznie rysowanie DRAWTO lub XIO18
93	zawiera kod znaku, na pozycji którego aktualnie znajduje się kursor
94, 95	zawiera pozycję kursora
96	zawartość określa wiersz, na którym kończy działanie DRAWTO lub XI018
97, 98	określa kolumnę, na której kończy działanie DRAWTO lub XI018
201	zawartość określa ilość kolumn o jaką jest przesuwany kursor w wyniku działania klawisza funkcyjnego TAB
622	255 - powoduje powolne listowanie programu, 0 – nominalnie
656	zawartość określa wiersz, w którym znajduje się kursor w okienku tekstowym. Przybiera wartość od 0 do 3. Zero oznacza górny rząd okienka tekstowego.
657, 658	zawartość określa kolumnę, w której znajduje się kursor w okienku tekstowym
752	zawartość 0 oznacza kursor widoczny, każda inna niewidoczny. Określa działanie kursora, patrz rozdział 5.4.
755	zawartość 2 oznacza kursor przeźroczysty, inna nieprzeźroczysty lub dający znaki odwrócone, patrz rozdział 5.4.
756	zawartość 204 lub 224 określa, który zbiór znaków jest wykorzystywany w trybach

	graficznych 1 i 2
763	zawiera kod ATASCII ostatniego czytanego lub pisanego na ekranie znaku
765	zawiera informację o danych wykorzystywanych przez graficzne instrukcje XI018
708 - 712	rejestry koloru od 0 do 4
560, 561	zawiera adres "display list"
710	Wartość 192 powoduje że wyświetlane są zielone litery na ciemnym tle, 148 - standardowo.
BUFOR MAGNETOFONU	
61	zawiera wskaźnik następnej lokacji w buforze magnetofonu. Zawartość 0 oznacza bufor pełny
63	zawiera 0 jeśli został stwierdzony koniec zbioru
649	zawiera 0 przy operacji czytania, 128 przy zapisie
650	zawartość określa wielkość bufora w bajtach (od 0 do 128)
1021-1151	obszar używany jako bufor magnetofonu
54018	gdy zawiera 52 włączony jest silnik magnetofonu, gdy zawiera 60 silnik jest wyłączony.
ADRESY DRUKARKI	
29	zawartość określa aktualną pozycję w buforze drukarki
30	zawartość określa wielkość bufora drukarki,
40	oznacza tryb normalny
960-999	bufor drukarki.
ADRESY KLAWIATURY	
16	zawartość 255 oznacza wyłączenie klawiatury
17	jeżeli zawiera 0 oznacza wciśnięcie klawisza BREAK
621	255 – powoduje wyłączenie klawiatury, 0 – stan normalny
694	zawartość 0 oznacza pisanie znaków normalnych, zawartość 128 oznacza pisanie znaków inwersyjnych
702	zawiera: 64 - normalne znaki, 0 - znaki z CAPS, 128 - znaki z CONTROL
731	zawartość 0 oznacza włączony dźwięk towarzyszący wciśnięciu klawisza, zawartość 1 wyłączony
732	17 - oznacza wciśnięcie klawisza HELP 0 - wyłączenie HELP
764	zawiera kod ostatniego naciśniętego klawisza. Jeżeli żaden klawisz nie był wciśnięty zawiera 255.
53279	sprawdzenie zawartości tej komórki daje informacje o wciśniętych klawiszach funkcyjnych: 0 – OPTION, SELECT, START; 1 – OPTION, SELECT 2 – OPTION, START; 3 – OPTION; 4 – SELECT, START 5 – SELECT; 6 – START; 7 – żaden
RÓŻNE	
65	zawartość 0 oznacza, że podczas transmisji z dysku i z magnetofonu dźwięk będzie wyciszany, inna zawartość powoduje normalne działanie
77	77 zawartość 128 oznacza tryb "przyciągania uwagi", 0 - oznacza normalny tryb pracy ekranu
186, 187	zawiera numer linii na której zostało przerwane wykonanie programu w BASIC'u zawiera numer błędu, jeśli wystąpił on w BASIC'u
195	zawiera adres powrotu do programu w BASIC'u z podprogramu w języku maszynowym wywołanego przez USR
212, 213	określa czy argumenty funkcji trygonometrycznych będą podawane w stopniach czy radianach:
251	6 - oznacza stopnie 0 - oznacza radiany

Rozdział 10

WSTĘP DO PROGRAMOWANIA W JĘZYKU MASZYNOWYM

Co to jest język maszynowy?

Język maszynowy, nazywany inaczej językiem wewnętrznym jest to ciąg liczb, które są wartościami kodów rozkazów mikroprocesora, wraz z argumentami tych rozkazów. Umieszczając te liczby w pamięci komputera tworzymy program w języku maszynowym, który następnie możemy uruchomić podając komputerowi, w którym miejscu pamięci znajduje się początek tego programu.

Programowanie w języku maszynowym ma swoje wady i zalety. Do zalet należy zaliczyć:

- dużo większą szybkość wykonywanych operacji w porównaniu z programem napisanym na przykład w języku BASIC,
- istnieją operacje, które mogą być wykonywane jedynie z poziomu języka wewnętrznego. Wady programowania w języku maszynowym to:
- proces tworzenia programu jest dużo wolniejszy od układania programu w języku wyższego poziomu (ze względu na to, że dla każdego rozkazu należy szukać kodu w odpowiedniej tablicy),
- proces poprawiania błędów jest długi i wymaga poświęcenia długiego okresu czasu na znalezienie choćby jednej pomyłki powstałej przy pisaniu programu.

W celu zminimalizowania wad języka wewnętrznego w porównaniu do języków wysokiego poziomu stworzono wiele narzędzi programowych ułatwiających pisanie programów maszynowych i udoskonalających języki wysokiego poziomu.

W celu zmniejszenia pierwszej z wad stworzono tzw. kompilator języków wysokiego poziomu (np. kompilator BASIC-u), które kilkudziesięciokrotnie przyspieszają działanie programów napisanych w tych językach, ale nie dają programów optymalnych. Wprawdzie stworzono języki wysokiego poziomu jak np. FORTH czy JĘZYK C, które posiadają mechanizmy porównywalne z możliwościami języka maszynowego, ale po pierwsze ich kompilatory nie są powszechnie dostępne, a po drugie w przypadku gdy zależy nam na napisaniu programu optymalnego pod względem czasu wykonania nie ma innej możliwości niż programowanie w języku wewnętrznym. Przykładem programu, który musi być optymalny pod względem czasu wykonywania jest program obsługi szybkiego urządzenia zewnętrznego np. stacji dysków czy szybkiego łącza komputerowego.

W celu przyspieszenia procesu programowania w języku maszynowym oraz zmniejszenia liczby błędów programowania powstał język symboliczny zwany językiem asemblera.

Asembler to program, który tłumaczy nazwy symboliczne do postaci kodów liczbowych języka wewnętrznego. Proces tłumaczenia jest translacją lub asemblacją. Program podawany asemblerowi do tłumaczenia nazywa się programem źródłowym, zaś utworzony na jego miejsce program maszynowy nazywa się programem wynikowym. Asembler daje programiście możliwości przy tworzeniu programu źródłowego:

- kody rozkazów mikroprocesora są zastąpione nazwami symbolicznymi tzw. nazwami mnemonicznymi lub krótko mnemonikami. Nazwy symboliczne są tak dobrane, aby kojarzyły się z określonym rozkazem np. LDA oznacza „załaduj akumulator” od słów angielskich "Load Accumulator",
- argumenty rozkazów mogą być również zastępowane nazwami. które w postaci asemblacji zostaną zastąpione odpowiednimi wartościami,
- istnieje możliwość nadawania określonym rozkazom etykiet (nazw) i odwoływania się do nich co zwalnia programistę od pamiętania adresów i wyliczania wartości skoków,
- daje również możliwość używania instrukcji nie tłumaczonych na język wewnętrzny określających początek programu w pamięci komputera, rezerwujących określony obszar w celu późniejszego wykorzystania, itp.

Powstały również programy zwane makroasemblerami dające jeszcze większe możliwości pisania programów tłumaczonych bezpośrednio na język maszynowy.

W celu szybszego poprawiania błędów i sprawdzania poprawności napisanych programów maszynowych stworzono również wiele programów jak np. disasembler (działający odwrotnie od asemblera tłumaczy program maszynowy zastępując kody rozkazów ich odpowiednimi nazwami mnemonicznymi), debugger (stwarzający możliwość śledzenia wykonywanego programu maszynowego) i inne.

Wszystkie wymienione programy ułatwiające pracę programiście są dostępne dla komputera ATARI, lecz co ma zrobić Czytelnik nie dysponujący żadnym z nich?

Po prostu programować bezpośrednio w języku maszynowym w sposób, który postaramy się umówić dokładniej poniżej.

Po pierwsze - należy przyswoić sobie nazwy mnemoniczne rozkazów mikroprocesora wraz ze sposobem ich działania. Nazwy te są podane w rozdziale 11 wraz z prostym symbolicznym określeniem ich działania oraz trybami adresowania.

Ktoś może zapytać, po co się uczyć nazw mnemonicznych skoro i tak nie posiada asemblera. Dlatego, że przed wpisaniem programu do pamięci należy go wcześniej napisać na kartce papieru, a używając nazw symbolicznych program będzie przejrzysty i łatwy do sprawdzenia i analizowania. Pamiętajmy, że nawet doskonałemu programiście programującemu w języku wewnętrznym bardzo rzadko udaje się napisać program poprawnie działający od początku. Często program zaczyna działać po kilkukrotnych próbach uruchomienia i odnalezieniu wielu pomyłek i błędów. Poza tym ucząc się nazw mnemonicznych nic nie tracimy bo na pewno kiedyś będziemy mieli również asembler czy disasembler.

Aby wyeliminować błędy i popełniać jak najmniej pomyłek należy poznać rejestry wewnętrzne mikroprocesora i ich przeznaczenie oraz sposób działania rozkazów mikroprocesora.

Większość mikrokomputerów ATARI zbudowana jest w oparciu o mikroprocesor 6502, który obecnie omówimy.

Mikroprocesor 65(ł2 posiada (i r2jestrów wewnętrznych dostępnych dla programisty. Są tu:

A- akumulator, w którym wykonywane są prawie wszystkie operacje arytmetyczne (dodawanie, odejmowanie), logiczne (iloczyn logiczny, suma log. suma modulo 2, sprawdzenie wartości określonych bitów itp.). Z akumulatora można przesyłać dane do/z innych rejestrów (nie wszystkich) do pamięci. Dokładny opis rodzajów operacji z A umieszczono w tabeli 2.

X - rejestr indeksowy X, którego przeznaczeniem jest wykorzystanie go przy trybach adresowania indeksowego oraz może być wykorzystany jako miejsce chwilowego przechowywania danych, licznik pętli, jak również przy przesłaniach. Dokładny opis - tab. 2.

Uwaga: rejestr X jest jedyną możliwością komunikacji z rejestrem S. Y - rejestr indeksowy Y, opis jak w przypadku rejestru X.

S - rejestr zwany wskaźnikiem stosu. Jest on rejestrem 8 bitowym. Wskazuje w którym miejscu na strunie pierwszej pamięci (komórki 256 do 511) aktualnie znajduje się szczyt stosu. Na stos odkładane są adresy powrotu przy skokach do podprogramów, przechowywane rejestry podczas wystąpienia przerwań oraz mogą być przechowywane dane przez program. Przy każdej z operacji ze stosem rejestr S jest automatycznie aktualizowany w sposób podany w tabeli 2. Wskaźnik stosu może być zmieniony przez program poprzez przesłanie nowej wartości z rejestru X.

1' - rejestr znaczników procesora, tzw. rejestr flagowy. Poszczególne bity rejestru F mają określone znaczenie i są modyfikowane w wyniku niektórych operacji na rejestrach mikroprocesora. W tabeli 2 podano przy każdym rozkazie czy i w jaki sposób dany rozkaz modyfikuje rejestr flagowy. Często rejestr znaczników oznaczany jest literą P zamiast F.

Wszystkie wyżej wymienione rejestry są rejestrami 8 bitowymi. W mikroprocesorze 6502 jest jeszcze jeden rejestr 16 bitowy:

PC - rejestr zwany licznikiem programu. Przechowywany jest w nim adres komórki pamięci, w której znajduje się bieżący rozkaz lub dana. Służy on mikroprocesorowi do pobierania kodów rozkazów lub przesłań danych na drodze procesor-pamięć.

Po uważnym przeczytaniu powyższych wiadomości oraz przeanalizowaniu informacji zawartych w tabeli 2 posiadamy już podstawowe wiadomości potrzebne do programowania w języku maszynowym. Dla dokładnego zaznajomienia się z zasadami działania mikroprocesora, jego rozkazów oraz technik programowania polecamy zainteresowanym książkę w języku angielskim: R. ZAKS "Programming the 6502". Jednakże podane przez nas wiadomości w zupełności wystarczają do stworzenia własnych, pierwszych programów, a analizując przykłady programów napisanych w języku maszynowym (taki przykład znajduje się w rozdziale) Czytelnik zdobędzie równie dobrą znajomość programowania jak po przeczytaniu dość grubej lektury książki Zaks'a.

Następnym krokiem zbliżającym nas do chwili uruchomienia własnego programu napisanego w języku wewnętrznym jest poznanie sposobów wprowadzania kodów programu do pamięci komputera i sposób uruchamiania go.

Ponieważ dysponujemy językiem programowania BASIC, pokażemy w jaki sposób przy jego pomocy stworzyć program maszynowy i uruchomić go. Można tego dokonać na kilka sposobów. Jednym z nich jest instrukcja POKE. Dla przypomnienia:

Instrukcja POKE adres, wyrażenie umieszcza pod wskazany adres (który również może być podany jako wyrażenie) wartość wyrażenia po przecinku. Wartość adresu i wyrażenie po przecinku muszą być liczbami dziesiętnymi, dla adresu z przedziału od zera do 65535 zaś dla wyrażenia po przecinku z przedziału od zera do 255 (konsekwencja wielkości komórki- 8 bitów). W najprostszym przypadku instrukcja POKE 8125,17 umieszcza w komórce o adresie 8125 liczbę 17.

Krótki program poniżej wpisze nam program maszynowy zawarty w instrukcjach DATA do pamięci począwszy od adresu 1536.

```
5 FOR I=1536 TO 1543
15 READ X
25 POKE I,X
35 NEXT I
45 DATA 104,173,198,2,141,200,2,96
```

W celu uruchomienia programu maszynowego, wprowadzonego przez wykonanie powyższego programu (RUN), który powoduje zlikwidowanie ramki ekranu należy użyć funkcji USR.
Należy wpisać z klawiatury:

```
I = USR(1536)
```

i nacisnąć klawisz RETURN.

Przy poprawnym wprowadzeniu danych efekt jest widoczny. Radzimy teraz korzystając z tabeli 10.1 lub 10.2 napisać program maszynowy zawarty w instrukcji DATA w postaci mnemonicznej (w tabeli 10.2 jest podane ile bajtów zajmuje dany rozkaz -jeden bajt należy odliczyć dla kodu rozkazu zaś o ile coś pozostanie to po kodzie występują argumenty czyli adres lub dane) i przeanalizować go. Czy można ten sam efekt otrzymać używając tylko programu napisanego w języku BASIC?

Aby zrozumieć występowanie kodu 104 na początku naszego programu należy wiedzieć jak działa instrukcja USR.
W najprostszej postaci USR (adres), gdzie adres podaje się w postaci liczby dziesiętnej.
np.

```
I=USR(1536)
```

lub

```
PRINT USR(1536)
```

Jej działanie przedstawia się następująco:
1. Na stos mikroprocesora jest wysyłany adres powrotny do BASIC-a
2. na stos mikroprocesora odkładane są automatycznie parametry występujące po przecinku po adresie, wewnątrz nawiasów instrukcji USR, gdy ich nie ma to krok jest pomijany,
3. na stos odkładana jest liczba z przedziału od zera do 255 równa ilości parametrów podanych po adresie, gdy ich nie ma to odkładane jest 0 np. I=USR(1536),18,19,20.
4. wykonywany jest skok do podprogramu po podanym adresie. Skok ten jest niejako rozkazem JSR mikroprocesora, więc nasz program maszynowy musi kończyć się rozkazem RTS (kod 96), aby po jego wykonaniu sterowanie zostało oddane z powrotem do BASIC-u.

W naszym przypadku przy braku parametrów na stosie mikroprocesora został odłożony najpierw adres powrotu do BASIC-u zaś później liczba zero mówiąca o braku przesłanych parametrów. Aby po wykonaniu programu maszynowego mikroprocesor wrócił sterowanie do BASIC-a przed końcem programu maszynowego należy pobrać ze stosu liczbę przesłanych parametrów (rozkazem RTS). Najlepiej jest to zrobić na samym początku programu maszynowego, aby o tym nie zapomnieć i zwiększyć tym samym ilość miejsca na stosie. Liczbę tę pobiera się do akumulatora rozkazem PLA (kod 1(>4). W przypadku przesyłania do programu maszynowego parametrów instrukcją PRINT USR(adres, parametr, parametr, ...) lub ALFA= USR(adres, parametr,...) należy od razu pobrać ze stosu ilość przesyłanych parametrów (PLA). Następnie sprawdzić w akumulatorze ile ich jest (nie wiemy z góry ile parametrów prześlemy do programu - jeden program maszynowy może być wywoływany z kilku miejsc w programie BASIC z różną ilością parametrów), a następnie pobrać wszystkie parametry ze stosu pozostawiając tylko adres powrotu do BASIC-a. Parametry są wysyłane na stos jako dwa bajty - należy dwukrotnie użyć rozkazu PLA. Po każdym PLA w akumulatorze mamy ściągniętą część parametru (bajt) i należy ją wysłać do rejestru lub pamięci w celu przechowania aby następne PLA nie zniszczyło jej.

Uwaga: Każde przesłanie przez mikroprocesor liczby dwubajtowej (więc też ww. parametrów) odbywa się w kolejności: młodszy bajt - LSB potem starszy - MSB !!!

I tak traktuje mikroprocesor dwie sąsiednie komórki w pamięci - w komórce o mniejszym adresie ma być LSB, zaś o większym MSB. Dlatego w naszym programie maszynowym kody 173, 198, 2 odpowiadają rozkazowi:
173 - kod rozkazu LDA w trybie adresowania absolutnego 198 – LSB 2 - MSB
Utworzony z bajtów adresowych adres wynosi: MSB x 256 + LSB czyli 71U.

W związku z instrukcją USR pozostało tylko wyjaśnić różnicę zapisu i działania pomiędzy U=USR(...) a PRINT USR(...).

Pierwszy z nich pod nazwę zamiennej numerycznej (w tym przypadku U) podstawia pewne dane, drugi zaś powoduje wyświetlenie tych danych na ekranie. Co to są za dane i czy są one potrzebne do działania?

Tak jak można przesyłać parametry do programu maszynowego lub ich nie wysyłać tak też program maszynowy może wysyłać jeden parametr dwubajtowy do BASIC-u. Wartość podstawiana do zmiennej numerycznej lub wyświetlana na ekranie jest pobrana przy powrocie do BASIC'u z komórek 212 i 213 - dziesiętnie, niezależnie od tego czy program maszynowy w nich coś umieścił czy nie. W dalszym ciągu programu BASIC'u po instrukcji USR możemy ignorować tę zmienną czy liczbę na ekranie, jeśli nie chcieliśmy nic przesłać do BASIC'u tą drogą. Dlatego napisaliśmy tę drogę, gdyż program -maszynowy może umieszczać w określonym miejscu pamięci potrzebne parametry, które później z BASIC-u możemy odczytać za pomocą instrukcji PEEK. Podobnie można przekazywać parametry do programu maszynowego za pomocą instrukcji POKE w określony obszar pamięci niekoniecznie korzystając z parametrów instrukcji USR.

Inną metodą umieszczania w pamięci parametrów napisanych w języku wewnętrznym jest użycie zmiennych tekstowych.
Przykład:

```
5 DIM A$(8)
15 READ A$
25 DATA (tu musi być wpisane 8 znaków odpowiadających kodom programu maszynowego)
35 X=USR(ADR(A$))
```

Program maszynowy zapisany w postaci kodów ATASCII jest podstawiany do zmiennej tekstowej, zaś funkcja ADR szuka adresu początku wartości tej zmiennej w pamięci, a więc adresu początku naszego programu maszynowego. Zamiast linii o numerach 15 i 25 można oczywiście napisać: 15A$ _" tu to samo co w DATA powyżej " i jest również poprawne.
Działanie funkcji **ADR**:
format:

zmienna numeryczna = **ADR**(*zmienna tekstowa lub tekst*)

Funkcja ta podstawia do zmiennej numerycznej adres pierwszego znaku tekstu w pamięci opisanego przez podanie nazwy zmiennej tekstowej lub tekstu w nawiasie.

Omówimy teraz pokrótce różnice wynikające z wyboru metody wprowadzania programu w języku wewnętrznym. Wprowadzając kody programu za pomocą instrukcji POKE musimy określić, w które miejsce w pamięci należy wprowadzić program. Stwarza to pewne niebezpieczeństwa. W przypadku gdy wprowadzimy nasz program w miejsce już zajęte (np. przez komórki robocze systemu operacyjnego, bufor któregoś z urządzeń zewnętrznych, program BASIC-u itp.) możemy stracić możliwość komunikacji komputera z urządzeniami zewnętrznymi, zniszczyć część programu BASIC-u, z którego dokonujemy wprowadzenia kodów lub w najgorszym przypadku unieruchomić działanie komputera. Po unieruchomieniu działania komputera często nie pozostaje inna możliwość jak tylko wyłączyć i ponownie włączyć go tracąc bezpowrotnie zawartość pamięci operacyjnej. Dlatego korzystając z wprowadzania kodów programu przy użyciu instrukcji POKE należy znać miejsce w pamięci komputera przeznaczone do wprowadzania programów maszynowych. Użycie instrukcji POKE do tego celu jest prostsze i szybsze, natomiast wykorzystanie zmiennych tekstowych i funkcji ADR jest utrudnione ze względu na konieczność wcześniejszego przełożenia kodów programu maszynowego na odpowiadające im znaki. Każdy kod programu jest traktowany jako kod ATASCII. W tym sposobie nie ma żadnych niebezpieczeństw, gdyż interpreter BASIC-u sam umieszcza w pamięci program maszynowy zapisany z postaci zmiennej tekstowej.

Wybór sposobu wprowadzania programu maszynowego należy do użytkownika, który jako udogodnienie może wcześniej stworzyć program w języku BASIC umieszczając kody wprowadzane z klawiatury w instrukcjach DATA lub program przekształcający kody na odpowiadające im znaki.

Poniżej podajemy obszary pamięci, w których można umieszczać program maszynowy za pomocą instrukcji POKE.

Strona 6 pamięci - komórki o adresach od 1536 do 1791. Jest to obszar o długości 256 bajtów przeznaczony właśnie do umieszczania w nim programów w języku wewnętrznym. W przypadku większych programów należy wybrać inny.

W rozdziale 9 opisano podział pamięci komputera ATARI 800XL. Czytając go dokładnie można wywnioskować z opisu gdzie jeszcze można wprowadzić programy maszynowe, wykorzystując np. nieużywane bufory urządzeń we/wy.

Istnieją komórki pamięci, na które warto zwrócić szczególną uwagę.

tzw. **MEMTOP** (741,742) - wskaźnik końca wolnej pamięci RAM. Jest to komórka ustawiana przez system operacyjny przy włączeniu zasilania i modyfikowana przy zmianie grafiki, przy przyciśnięciu klawisza RESET oraz gdy kanał dla ekranu zostanie otworzony.

tzw. **APPMHI**(14,15) - wskaźnik końca używanej pamięci RAM.

tzw. **MEMLO**(743,744) - wskaźnik początku wolnej pamięci RAM.

Nasz program maszynowy, który jest dłuższy niż 256 bajtów możemy umieścić więc w obszarze leżącym pomiędzy początkiem, a końcem wolnej pamięci RAM. Rozmiary tego obszaru należy odczytać po wprowadzeniu do pamięci całego programu BASIC-u wraz z instrukcjami DATA, po czym obliczyć ile zajmuje miejsca program maszynowy i na tej podstawie zaktualizować zawartość odpowiednich wyżej wymienionych komórek. Później wystarczy już tylko wpisać do programu BASIC-a adres miejsca pamięci, gdzie ma być załadowany program maszynowy i wprowadzić kody do pamięci. Aktualizacja komórek APPMHI i MEMLO ma na celu zabezpieczenie programu maszynowego przed zniszczeniem.

Aby zrozumieć w jaki sposób działają rozkazy procesora należy poznać tryb adresowania mikroprocesora 6502.

Adresowanie natychmiastowe

W tym trybie adresowania rozkaz procesora zawiera 2 bajty. Po bajcie kodu rozkazu następuje tzw. operand, który w tym przypadku zawiera bezpośrednie dane, na których rozkaz będzie działał.

Adresowanie absolutne

W tym trybie rozkaz zawiera 3 bajty. Drugi bajt zawiera mniej znaczący bajt adresu LSB, trzeci zawiera bardziej znaczący bajt adresu MSB. Adres jest to efektywny adres komórki pamięci na której rozkaz będzie działał. Ten tryb adresowania pozwala na dostęp do każdej z 65536 komórek pamięci (64 kbajty).

Adresowanie strony zerowej

Rozkaz 2 bajtowy. Operand zawiera adres komórki pamięci na stronie zerowej pamięci. Choć adresowanie absolutne umożliwia również dostęp do strony zerowej, to ten tryb adresowania jest szybszy i rozkaz zajmuje tylko dwa bajty.

Adresowanie akumulatora

Rozkaz 1 bajtowy, powodujący określoną operację na akumulatorze. Adresowanie wewnętrzne

Rozkaz 1 bajtowy, powodujący określone działanie wewnątrz mikroprocesora związane z jego rejestrami.

Adresowanie absolutne, indeksowe

Rozkaz 3 bajtowy. Ten tryb adresowania używany jest z wykorzystaniem któregoś z rejestrów indeksowych X lub Y. Efektywny adres komórki pamięci jest obliczany na podstawie adresu zawartego w drugim i trzecim bajcie rozkazu do którego zostaje dodana zawartość rejestru indeksowego. Ponieważ rejestr indeksowy jest 8 bitowy przy dodawaniu jest traktowany jak rejestr 16 bitowy, którego starszy bajt MSB jest równy zero.

Adresowanie strony zerowej, indeksowe

Rozkaz 2 bajtowy. Ten tryb adresowania jest używany z wykorzystaniem rejestru X lub Y. Efektywny adres komórki na stronie zerowej pamięci jest obliczany na podstawie adresu zawartego w drugim bajcie rozkazu, do którego dodawana jest zawartość rejestru indeksowego. Jeśli w wyniku dodania tych dwu wartości otrzymany zostanie adres wykraczający poza stronę zerową to automatycznie przeniesienie jest ignorowane (w ogóle nie występuje).

Adresowanie względne

Ten tryb adresowania występuje tylko odnośnie tzw. rozkazów skoków warunkowych, które są rozkazami 2 bajtowymi. Operand nazywa się przesunięciem i jest traktowany jako liczba w kodzie uzupełnieniowym do dwóch (U2) czyli z przedziału od - 128 do + 127. Po sprawdzeniu odpowiedniego warunku jeśli wynik sprawdzenia jest negatywny to rozkaz ten jest opuszczany i wykonywany jest rozkaz następny. Jeśli wynik

sprawdzenia jest pozytywny to wartość przesunięcia jest dodawana do zawartości licznika programu (rejestru PC) w chwili gdy wskazuje on na następny rozkaz po rozkazie skoku warunkowego. Tym samym zostaje wykonany skok do rozkazu wskazywanego przez zmodyfikowany rejestr PC.

Adresowanie pośrednie

Ten tryb adresowania występuje tylko dla rozkazu skoku bezwarunkowego JMP(...), który jest rozkazem 3 bajtowym. Operandem zawartym w 2 i 3 bajcie rozkazu jest adres komórki pamięci, w której znajduje się mniej znaczący bajt efektywnego adresu L,SB, zaś MSB znajduje się w komórce pamięci o adresie o 1 większym od operandu.

Adresowanie indeksowe, pośrednie

Ten tryb adresowania (zapisywany jako (pośredni), X) jest używany z wykorzystaniem rejestru indeksowego X i odnosi się do strony zerowej pamięci. Rozkaz 2 bajtowy. Operand zawarty w drugim bajcie rozkazu jest dodawany do zawartości rejestru X z pominięciem przeniesienia poza stronę zerową pamięci. W wyniku dodania otrzymywany jest adres komórki pamięci na stronie zerowej, w której znajduje się L.SB efektywnego adresu. W następnej komórce pamięci znajduje się MSB.

Adresowanie pośrednie, indeksowe

Ten tryb adresowania (zapisywany jako (pośredni), Y) jest używany z wykorzystaniem rejestru indeksowego Y i odnosi się do strony zerowej pamięci. Rozkaz 2 bajtowy. Operand jest adresem komórki pamięci na stronie zerowej. Zawartość tej komórki pamięci jest dodawana do zawartości rejestru indeksowego Y, w wyniku czego otrzymywany jest mniej znaczący bajt efektywnego adresu. Wartość przeniesienia otrzymana z dodawania jest dodana do wartości następnej komórki pamięci strony zerowej co daje w wyniku bardziej znaczący bajt efektywnego adresu.

Kod uzupełnieniowy do dwóch - U2

Sposób przedstawienia liczb całkowitych w kodzie U2 jest następujący:

- najstarszy bit słowa jest bitem znakowym, gdy równa się zero to liczba jest dodatnia (z liczbą zero włącznie), zaś gdy równa się 1 to liczba jest ujemna,
- wartość liczby zapisana jest na pozostałych bitach.

Aby otrzymać liczbę ujemną równą co do wartości bezwzględnej danej liczbie należy przeprowadzić następujące operacje:

1. Wszystkie bity danej liczby dodatniej zamienić na przeciwne (1 na 0 i odwrotnie).
2. Do tak powstałej liczby dodać + I. W wyniku otrzymamy liczbę ujemną w kodzie U2.

Powyższy algorytm ilustruje przykład:

Reprezentacja liczby -5 w kodzie U2

5 = 00000101

1. zmiana bitów 11111010

2. dodanie + 1 I

wynik = 11111011 jest to -5 w kodzie U2 bit znakowy

W celu sprawdzenia można wykonać następujące działania: -5 + 5, co musi dać w wyniku 0.

5 = 00000101

-5 = 11111011

wynik: 100000000

Przeniesienie poza ósmy bit jest ignorowane, więc w wyniku otrzymano 0.

Po lekturze tego rozdziału Czytelnik będzie mógł samodzielnie pisać programy w języku wewnętrznym. Pozostało nam jeszcze podać kilka praktycznych uwag pomocniczych przy układaniu _i tworzeniu programów maszynowych.

1. Przed przystąpieniem do pisania programu w języku wewnętrznym należy stworzyć algorytm działania programu. Polecamy metodę graficzną zapisu algorytmu w postaci bloków i ich połączeń.

2. Na podstawie stworzonego algorytmu należy napisać program maszynowy wykorzystując nazwy mnemoniczne rozkazów mikroprocesora.

3. za pomocą tabeli 1 i 2 zapisać program maszynowy w postaci liczbowej.

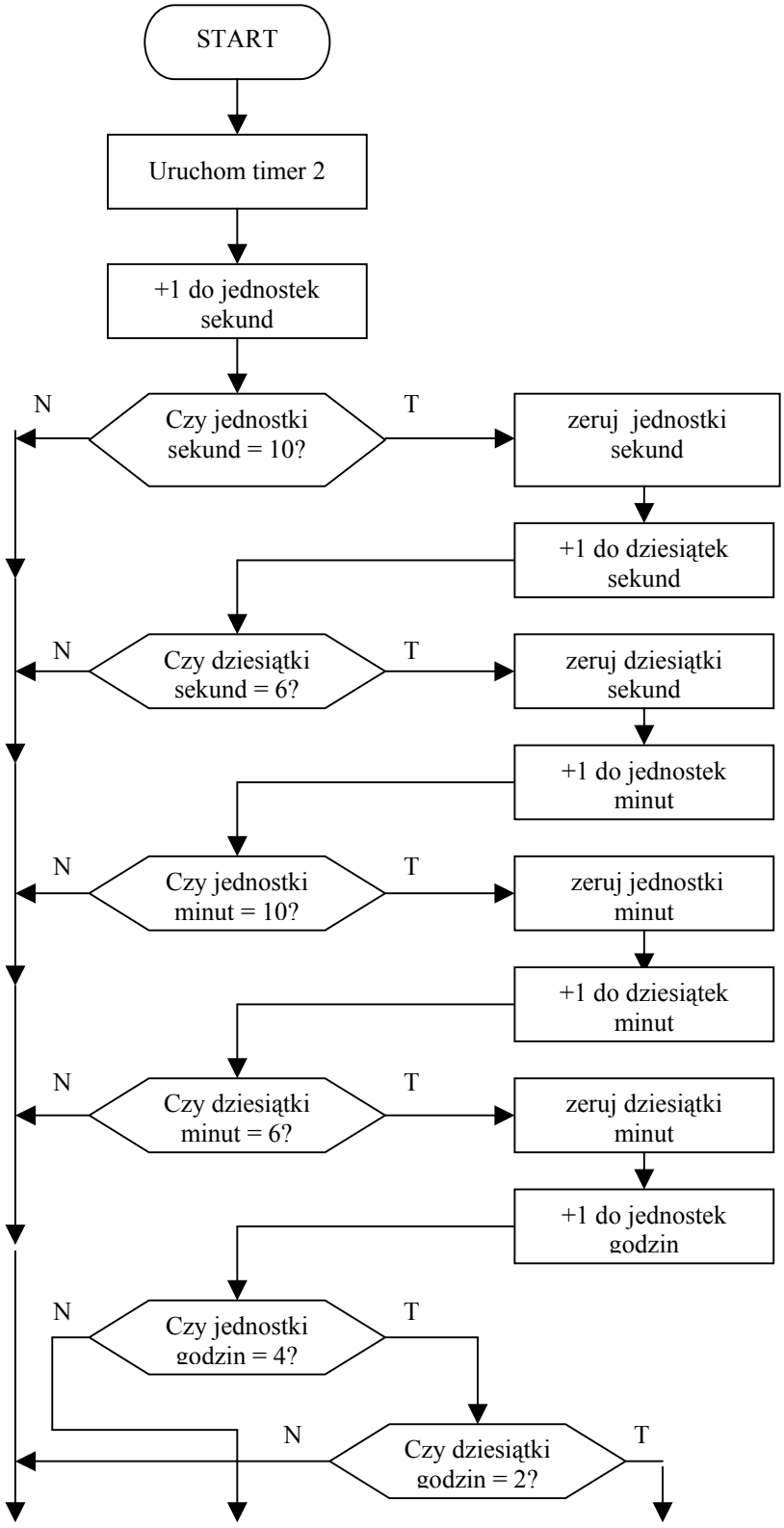
4. W zależności od wybranego sposobu wprowadzania programu maszynowego do pamięci komputera należy umieścić liczby odpowiadające programowi w instrukcji DATA lub znaki odpowiadające tym liczbom podstawić do zmiennej tekstowej.

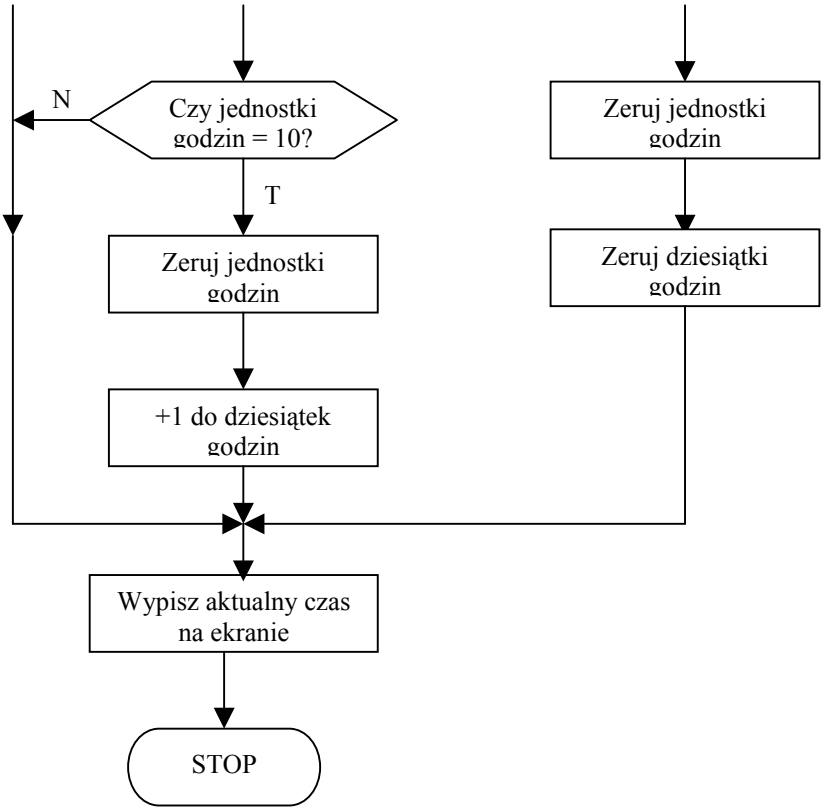
Później czeka nas już tylko proces wprowadzania programu do pamięci i jego poprawiania. Poniżej przedstawiamy proces tworzenia programu zegara czasu rzeczywistego w oparciu o powyższe uwagi. Program ten ma za zadanie w prawym dolnym rogu ekranu wyświetlać czas niezależnie od poczynań użytkownika. Ten

sam program można napisać w języku BASIC, lecz wówczas użytkownik nie nu5głby użytkować komputera, który cały czas byłby zajęty obsługą zegara. Dlatego program ten będzie napisany w języku wewnętrznym.

Naciśnięcie klawisza RESET oraz wyłączenie komputera kasuje ten program. Komunikacja z drukarką, magnetofonem i stacją dysków wstrzymuje zegar. Program napisany w języku BASIC:, służy do wprowadzania programu maszynowego do pamięci. Będzie ponadto pobierał od operatora aktualny czas początkowy i po uruchomieniu zegara samoczynnie się skasuje.

Krok 1.:Stworzenie algorytmu i jego zapis w postaci graficznej:





Krok 2. Napisanie programu w języku wewnętrznym:

```
1 ;****************************************
2 ;** PROGRAM "ZEGAR CZASU **
3 ;**        RZECZYWISTEGO"       **
4 ;** wydruk zrodlowy dla asemblera **
5 ;************************************
6 ;
7 ;
8 ; poczatek programu
9 ;
10 ;
11 *= $0600
12 PROG       PLA                    ; wymagane dla instrukcji USR
13 ;
14 ;ustawienie adresu obslugi
15 ;przerwania od timera 2
16 ;
17            LDA #$OB
18            STA $0228              ;LSB
19            LDA #$06
20            STA $0229              ;MSB
21 ;
22 ;tu jest wejscie dla przerwania
23 ;ktore pojawia sie co 1 sekunde
24 ;          <----
25            LDA #$32
26            STA $021A              ;restart timera
27            INC $06CF              ; +1 do jednostek sekund
```

```
28              LDA $06CF
29              CMP #$OA
30              BNE SKOK1           ;skok jeśli mniej niz 10
31              LDA #$00
32              STA  $06CF          ;zeruj jednostki sekund
33              INC   $06CE         ; +1 do dziesiatek sekund
34              LDA   $06CE
35              CMP   #$06
36 SKOK1        BNE SKOK2           ;skok jeśli mniej niz 6
37              LDA   :$00
38              STA   $06CE         ;zeruj dziesiatki sekund
39              INC   $06CC         ;+1 do jednostek minut
40              LDA   $06CC
41 CMP #$OA
 42 SKOK2       BNE  SKOK3          ;skok gdy < 10
 43             LDA   #$00
 44             STA   $06CC         ;zeruj jednostki minut
 45             INC   $06CB         ; +1 do dziesiatek minut
 46             LDA   $06CB
 47             CMP   #$06
 48 SKOK3       BNE   SKOK4         ;skok gdy <6
 49             LDA   #$00
 50             STA   $06CB         ;zeruj dziesiatki minut
 51             INC   $06C9         ;+1 do jednostek godzin
 52             LDA   $06C9
 53             CMP   #$04
 54             BNE   SKOK5         ;skok gdy rozne od 4
 55             LDA   $06C8         ;gdy =4 to sprawdz
 56             CMP   #$02          ;dziesiatki godzin
 57 SKOK4       BNE SKOK6           ;skok gdy dziesiatki <2
 58             LDA   #$00
 59             STA   $06C8         ;zeruj dziesiatki godzin
 60             STA   $06C9         ;oraz jednostki godzin
 61             BEQ   SKOK7         ;skok bezwarunkowy
 62 SKOK5       CMP   #$OA          ;czy jednostki godzin =10?
 63 SKOK6       BNE   SKOK7         ;nie, to skok
 64             LDA   #$00
 65             STA   $06C9         ;zeruj jednostki godzin
 66             INC   $0608         ;+1 do dziesiatek godzin
 67;
 68 ; w miejsce NOP mozna     wprowadzic
 69 ; RTS z poziomu BASIC'a powodujac
 70 ; nie wyswietlenie zegara na ekranie
 71 ;
 72 SKOK7       NOP
 73 ;
 74 ;w tym miejscu zaczyna sie
 75 ;procedura wypisywania aktualnego
 76 ;czasu na ekranie
 77 ;
 78             CLA                 ;]
 79             LDA   $0230         ;]
 80             ADC   #$AF          ;]w komorkach $D5 i $D6
 81             STA   $D5           ;]ustaw adres pozycji
 82             LDA   $0231         ;]zegara na ekranie
 83             ADC   #$03          ;]
 84             STA   $D6                   ;]
125
```

```
85 ;
86 ;tu nastepuje przepisanie zawartosci
87 ;aktualnych licznikow na ekran
88 ;
89              LDA    x$00
90 BACK        LDA    $06C8,Y
91              BNE    JUMP
92              CPY    #$00
93              BNE    JUMP
94              BEQ    EXIT          ;nie pisz 0 dla dzies. godz.
95 JUMP         ORA    x$10
96 EXIT         STA    ($D5},Y
97              INY
98              CPY    #$08
99              BNE    BACK
100             RTS
101             .END
```

Kroki 3 i 4 razem:.

Zapis programu maszynowego w instrukcjach DATA programu BASIC'u przy wykorzystaniu tabeli 1 oraz dopisanie potrzebnych linii programu BASIC'u dla uruchomienia działania zegara.

```
5 REM PROGRAM "ZEGAR CZASU RZECZYWISTEGO"
10 RESTORE 38:  I=0:  DS=1736
11 REM LADOWANIE PROGRAMU MASZYNOWEGO DO PAMIECI
12 READ X: IF X=-1 THEN 16
14 POKE 1536+Z, X: I=I+1:  GOTO 12
15 REM POBRANIE CZASU POCZATKOWEGO
16 ? CHR$(125);"Podaj:  godz., min., sek": ? : ? "Nacisnij RETURN dla startu zegara ": POSITION 10,7
18 INPUT G, M, S
20 POKE DS,INT(G/10): POKE DS+1, G-INT(G/10)*10
22 POKE DS+3,INT (M/10): POKE DS+4,M-INT(M/10)*10
24 POKE DS+6,INT (S/10): POKE DS+7,S-INT(S/10)*10
 26 POKE DS+2,I0: POKE DS+5,10
27 REM URUCHOMIENIE ZEGARA
28 X=USR(1536)
30 ? : ? "Uwaga!": ?
32 ?"Program do obslugi zegara": ? "czasu rzeczywistego zajmuje obszar": ? "od 1536 ($600) do 1689 ($699)"
34 ? "oraz korzysta z lokacji" : ? "od 1736 ($6CB) do 1743 ($6CF)": ? : ? " W celu zatrzymania wyswietlania";
36 ? "czasu": ? "wykonaj:  POKE 1651, 96 zaś na powrot": ? "POKE 1651,234":  NEW
38 DATA 104, 169, 11, 141, 40, 2, 169, 6, 141, 41, 2
40 DATA 169, 50, 141, 26, 2, 238, 207, 6, 173, 207, 6
42 DATA 201, 10, 208, 13, 169, 0, 141, 207, 6, 238, 206, 6
44 DATA 173, 206, 6, 201, 6, 208, 13, 169, 0, 141, 206, 6
46 DATA 238, 204, 6, 173, 204, 6, 201, 10, 208, 13
48 DATA 169, 0, 141, 204, 6, 238, 203, 6, 173, 203, 6
50 DATA 201, 6, 208, 20, 169, 0, 141, 203, 6, 238, 201, 6, 173,201, 6, 201, 4
52 DATA 208, 17, 173, 200, 6, 201, 2, 208, 12, 169, 0, 141, 20,0, 6, 141, 201, 6
54 DATA 240, 12, 201, 10, 208, 8, 169, 0, 141, 201, 6, 238, 20, 0, 6, 234
56 DATA 24, 173, 48, 2, 105, 175, 133, 213, 173, 49, 2
58 DATA 105, 3, 133, 214, 160, 0, 185, 200, 6, 208, 6
60 DATA 192, 0, 208, 2, 240, 2, 9, 16, 145, 213, 200, 192, 8, 2, 98, 236, 96, -1
```

10.1 Lista rozkazów mikroprocesora 6502

LISTA ROZKAZOW MIKROPROCESORA 6502

Kod dziesiętny	Kod hexadecymalny	Mnemonika wraz z trybem adresowania
0	00	BRK -wewnętrzny
1	01	ORA -(pośredni, X)
5	05	ORA - strony zerowej
6	06	ASL - strony zerowej
8	08	PHP - wewnętrzny
9	09	ORA - natychmiastowy
10	0A	ASL -akumulator
13	0D	ORA - absolutny
14	0E	ASL - absolutny
16	10	BPL - względny
17	11	ORA - (pośredni),Y
21	15	ORA -strony zerowej, X
22	16	ASL - strony zerowej, X
24	18	CLC - wewnętrzny
25	19	ORA - absolutny, Y
29	1D	ORA - absolutny, X
30	1E	ASL - absolutny, X
32	20	JSR - absolutny
33	21	AND-(pośredni, X)
36	24	BIT -strony zerowej
37	25	AND - strony zerowej
38	26	ROL - strony zerowej
40	28	PLP - wewnętrzny
41	29	AND - natychmiastowy
42	2A	ROL - akumulator
44	2C	BIT-absolutny
45	2D	AND - absolutny
46	2E	ROL - absolutny
48	30	BMI - względny
49	31	AND-(pośredni),Y
53	35	AND - strony zerowej, X
54	36	ROL - strony zerowej, X
56	38	SEC - wewnętrzny
57	39	AND - absolutny, Y
61	3D	AND - absolutny, X
62	3E	ROL - absolutny, X
64	40	RTI - wewnętrzny
65	41	EOR - (pośredni, X)
69	45	EOR - strony zerowej
70	46	LSR - strony zerowej
72	48	PHA - wewnętrzny
73	49	EOR - natychmiastowy
74	4A	LSR - akumulator
76	4C	JMP - absolutny
77	4D	EOR - absolutny
78	4E	LSR - absolutny
80	50	BVC - względny
81	51	EOR -(pośredni),Y
85	55	EOR - strony zerowej, X

86	56	LSR - strony zerowej, X
88	58	CLI - wewnętrzny
89	59	EOR - absolutny, Y
93	5D	EOR - absolutny, X
94	5E	LSR - absolutny, X
96	60	RTS - wewnętrzny
97	61	ADC-(pośredni, X)
101	65	ADC - strony zerowej
102	66	ROR - strony zerowej
104	68	PLA - wewnętrzny
105	69	ADC - natychmiastowy
106	6A	ROR - akumulator
108	6C	JMP - pośredni
109	6D	ADC - absolutny
110	6E	ROR - absolutny
112	70	BVS - względny
113	71	ADC - (pośredni),Y
117	75	ADC - strony zerowej, X
118	76	ROR - strony zerowej, X
120	78	SEI - wewnętrzny
121	79	ADC - absolutny, Y
125	7D	ADC - absolutny, X
126	7E	ROR - absolutny, X
129	81	STA - (pośredni, X)
132	84	STY - strony zerowej
133	85	STA - strony zerowej
134	86	STX - strony zerowej
136	88	DEY – wewnętrzny
138	8A	TXA – wewnętrzny
140	8C	STY – absolutny
141	8D	STA – absolutny
142	8E	STX – absolutny
144	90	BCC – względny
145	91	STA - (pośredni),Y
148	94	STY - strony zerowej, Y
149	95	STA - strony zerowej, X
150	96	STX - strony zerowej, Y
152	98	TYA – wewnętrzny
153	99	STA - absolutny, Y
154	9A	TXS – wewnętrzny
157	9D	STA - absolutny, X
160	A0	LDY – natychmiastowy
161	A1	LDA-(pośredni),X
162	A2	LDX – natychmiastowy
164	A4	LDY - strony zerowej
165	A5	LDA - strony zerowej
166	A6	LDX - strony zerowej
168	A8	TAY – wewnętrzny
169	A9	LDA – natychmiastowy
170	AA	TAX - wewnętrzny
172	AC	LDY- absolutny
173	AD	LDA - absolutny
174	AE	LDX - absolutny
176	B0	BCS - względny
177	B1	LDA-(pośredni),Y
180	B4	LDY - strony zerowej, X
181	B5	LDA - strony zerowej, X

182	B6	LDX - strony zerowej, Y
184	B8	CLV - wewnętrzny
185	B9	LDA - absolutny, Y
186	BA	TSX - wewnętrzny
188	BC	LDY - absolutny, X
189	BD	LDA - absolutny, X
190	BE	LDX - absolutny, Y
192	C0	CPY - natychmiastowy
193	C1	CMP - (pośredni, X)
196	C4	CPY - strony zerowej
197	C5	CMP - strony zerowej
198	C6	DEC - strony zerowej
200	C8	INY - wewnętrzny
201	C9	CMP – natychmiastowy
202	CA	DEX – wewnętrzny
204	CC	CPY – absolutny
205	CD	CMP - absolutny
206	CE	DEC - absolutny
208	D0	BNE - względny
209	D1	CMP - (pośredni), Y
213	D5	CMP - strony zerowej, X
214	D6	DEC - strony zerowej, X
216	D8	CLD - wewnętrzny
217	D9	CMP - absolutny, Y
221	DD	CMP - absolutny, X
222	DE	DEC – absolutny, x
224	E0	CPX – natychmiastowy
225	E1	SBC- (pośredni, X)
228	E4	CPX – strony zerowej
229	E5	SBC - strony zerowej
230	E6	INC - strony zerowej
232	E8	INX - wewnętrzny
233	E9	SBC - natychmiastowy
234	EA	NOP - wewnętrzny
236	EC	CPX - absolutny
237	ED	SBC - absolutny
238	EE	INC - absolutny
240	F0	BEQ - względny
241	F1	SBC-(pośredni), Y
245	F5	SBC - strony zerowej, X
246	F6	I NC – strony zerowej, X
248	F8	SED - wewnętrzny
249	F9	SBC - absolutny, Y
253	FD	SBC - absolutny, X
254	FE	INC - absolutny, X

Brakujące kody nie są rozkazami mikroprocesora 6502.

Rozdział 11

TECHNIKI SORTOWANIA

Umiejętność wykonywania długich żmudnych prac w bardzo krótkim czasie jest jedną z najważniejszych zalet komputera. Do takich prac należy z całą pewnością np. ustawiania w określonym porządku elementów danego zbioru. Elementy te mogą być porządkowane na przykład ze względu na numery, porządek alfabetyczny itp. Dotychczas stworzono wiele algorytmów sortowania
Jak się wydaje nie zawsze ich wydajność zależy od stopnia ich złożoności. W rozdziale tym przedstawiamy cztery spośród nich. Zainteresowanych tym tematem odsyłamy do książki Niklausa Wirtha "Algorytmy + struktury danych = programy". WNT W-wa 1980.

METODA	ZŁOŻONOŚĆ	SZYBKOŚĆ
Sortowanie bąbelkowe	bardzo prosta	bardzo mała
Sortowanie przez proste ustawianie	bardzo prosta	mała
Sortowanie metodą SHELLA	średnia	średnia
Sortowanie szybkie	skomplikowana	bardzo duża

TABELA 12.1. Porównanie metod sortowania

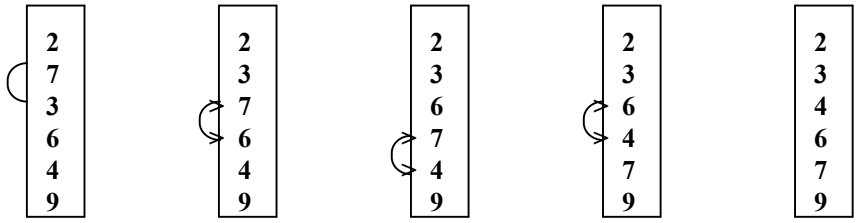
Dla każdej metody zbiory do sortowania zapisuje się najczęściej w tabelach, których wymiary ogranicza jedynie pojemność pamięci RAM.
Każdy ze sposobów sortowania zostanie przedstawiony wraz z przykładem.

11.1 Sortowanie bąbelkowe

Nazwa tej metody pochodzi stąd, że w pionowo ustawionym ciągu, elementy o małej wartości w czasie sortowania "wypływają" do góry, jak bąbelki powietrza, natomiast elementy o dużej wartości "tonuj', upadają na "dno". (Rys. 1). Sortowanie polega na tym, że porządkowana tablica jest stale przeglądana i jeżeli dwa sąsiadujące elementy są przestawiane (tzn. pierwszy ma większą wartość niż drugi) to zostają one zamienione miejscami. Sortowanie trwa dopóki wszystkie elementy nie zostaną ustawione we właściwym porządku.

PROGRAM "SORTOWANIE BĄBELKOWE"

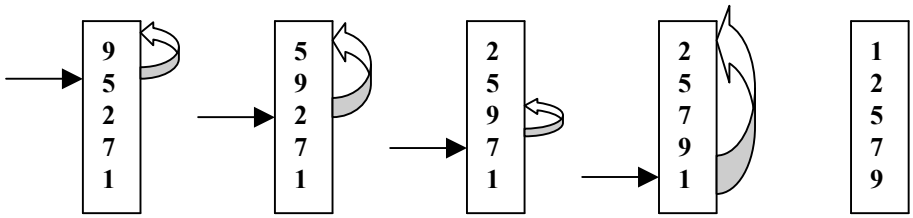
```
1000 REM PROG. "SORTOWANIE BABELKOWE"
1010 REM
1015 FOR J=1 TO N-1
1020 F=0
1025 FOR I=1 TO N-J
1030 IF D(1)<=D (I+1) THEN 1045
1035 F=1
1040 T=D (I+1):D(I+1)= D(I): D(I)=T
1045 NEXT I
1050 IF F=0 THEN 1060
1055 NEXT J
1060 RETURN
```



RYS. 11.1. Sortowanie bąbelkowe

11.2. Sortowanie przez proste wstawianie

Metoda ta umożliwia posortowanie tablicy po jednorazowym jej przeglądnięciu. Jest ona często stosowana np. przez grających w karty. Polega na tym, że przeglądamy po kolei całą tablicę. Jeżeli któryś z elementów nie znajduje się na swoim miejscu, to wstawiamy go tam gdzie powinien się znajdować gdyby zbiór był uporządkowany. (Rys.2).



RYS. 11.2. Sortowanie przez proste wstawianie

PROGRAM "SORTOWANIE PRZEZ WSTAWIANIE"

```
2000 REM PROGRAM
2010 REM "SORTOWANIE PRZEZ WSTAWIANIE"
2015 FOR J=2 TO N
2020 T=D(J)
2025 FOR I=J-1 TO 1 STEP -1
2030 IF D(I)<=T THEN 2045
2035 D(I+1)=D(I)
2040 NEXT I 2045 D(I+1)=T
2050 NEXT J
2055 RETURN
```

11.3. Sortowanie metodą Shella

O ile sortowanie bąbelkowe wymaga porównania obiektów leżących obok siebie, o tyle sortowanie metodą Shella wymaga porównania elementów leżących od siebie daleko. Przy kolejnym przeglądaniu tablicy porównywane (i ewentualnie przestawiane) są elementy oddalone od siebie o tak zwany interwał. Wynosi on z reguły dla pierwszego przeglądania pełną liczbę elementów (to znaczy, że porównujemy pierwszy element z ostatnim), natomiast dla każdego następnego interwału jest o połowę mniejszy. Ostatnia faza sortowania równoważna jest z sortowaniem bąbelkowym. Metoda ta nazywana jest często: sortowanie za pomocą malejących przyrostów.

PROGRAM "SORTOWANIE METODĄ SHELLA"

```
3000 REM PROGRAM
3010 REM "SORTOWANIE METODA SHELLA"
3015 SZ=N
3020 IF SI<1 THEN 3070
3025 SI=INT (SI/2)
3030 F=0
3035 FOR I=1 TO N-SI
3040 IF D(I)<=D (I+SI) THEN 3055
3045 F=1
3050 T=D(I+SI):D(I+SI)=D(I):D(I)=T
3055 NEXT I
3060 IF F=0 THEN 3020
```

```
3065 GOTO 3030
3070 RETURN
```

11.4. Sortowanie szybkie

Na koniec przedstawiamy algorytm sortowania szybkiego. Jest on najbardziej złożony ale w większości przypadków najszybszy spośród przedstawionych. Zasadniczą idei(tego rodzaju sortowania jest podzielenie całej tablicy na małe podgrupy, następnie posortowanie ich metodą bąbelkowa (bardzo dobra dla małej ilości elementów). Jednak, podczas sortowania kolejnych podgrup program musi pamiętać umiejscowienie pozostałych elementów tablicy. Jest to realizowane poprzez zapamiętanie początkowej i końcowej pozycji rzędu elementów, które nie są sortowane, w drugiej tablicy tak, że pierwsze pozycje zapamiętane są pierwszymi cyframi. Taka struktura nazywa się stosem. Sortowanie kończy się w momencie gdy sto sjest pusty. W szybkim sortowaniu używa się dwóch wskaźników. Jeden wskazuje początek, drugi koniec elementów, które są sortowane. Jeden z dwóch wskaźników będzie zawsze wskazywał element, który był początkowo na pierwszej pozycji w rzędzie. Nazywa się on PIVOT. Dwa elementy wyznaczone przez wskaźniki s~ przestawiane, gdy ten znajdujący się bliżej końca jest mniejszy niż ten na początku. Drugi wskaźnik jest wtedy przestawiany o jedna pozycję wzdłuż rzędu w kierunku PIVO'I' i proces się powtarza. Dzieje się tak do chwili, aż dwa wskaźniki spotkają się. W tym przypadku rząd jest podzielony na dwie podgrupy, między którymi jest 1'1 VO'I . 1'rucesjcat następnie powtarzany dla każdej podgrupy. W przypadku gdy ilość informacji jest duża i powoduje "przepełnienie" stosu, to rozmiar sortowanego ciągu musi być zmniejszony.

PROGRAM "SORTOWANIE SZYBKIE"

```
4000 REM PROGRAM"SORTOWANIE SZYBKIE"
4015 PS=7
4020 S(1,1)=1
4025 S(1,2)=N
4030 IF PS=0 THEN 4110
4035 II=S(PS,1): JJ=S(PS,2): PS=PS-1
4040 P=0: I=II: J=JJ
4045 1F D(I)<=D(J) THEN 4060
4050 P=1-P
4055 T=D(L): D(I)=D(J): D(J)=T
4060 IF P=0 THEN I=I+1
4065 IF P=1 THEN J=J-1
4070 IF I<J THEN 4045
4075 IF I>=JJ THEN 4095
4080 PS=PS+1
4085 IF PS>40 THEN PRINT "PRZEPELNIENIE STOSU": END
4090 S(PS,1)=I+1:S(PS,2)=JJ
4095 JJ=I-1
4100 IF JJ>II THEN 4040
4105 GOTO 4030
4110 RETURN
```

11.5.Porównanie metod sortowania

W celu porównania efektywności opisanych metod sortowania spróbuj posortować ten sam zbiór różnymi metodami mierząc czas. Sprawdź, który z algorytmów jest najszybszy dla małej i dużej ilości elementów. Spróbuj wyciągnąć wnioski. Program 'Porównanie metod sortowania wykorzystuje wszystkie podane wcześniej metody sortowania. Dodatkowo mierzy on czas wykonywania programu.

PROGRAM "PORÓWNANIE METOD SORTOWANIA"

```
5 REM PROGRAM
6 REM "POROWNANIE METOD SORTOWANIA"
20 DIM K$(1), N$(4), S(40,2)
25 PRINT CHR$(125)
30 PRINT: PRINT "TECHNIKI SORTOWANIA"
```

```
36 PRINT: PRINT "PODAJ 1LOSC ELEMENTOW SORTOWANIA"
40 INPUT N$:N=INT(VAL(N$))
45 IF N<5 OR N>1000 THEN 25
50 DIM D(N)
55 F'OR J=1 TO N
60 D(J)=INT(RND(0)*2500)
65 NEXT J
70 l'RINT CHR$(125):PRINT: PRINT "PODAJ METODE:"
75 PRINT "1) SORTOWANIE BABELKOWE"
80 PRTNT "2) SORTOWANIE PRZEZ WSTAWIANIE"
85 PRINT "3) SORTOWANIE METODA SHELLA"
90 PRINT "4) SORTOWANIE SZYBKIE"
95 PR1NT
100 1NPUT K$:K=VAL(K$): IF K<7 OR K>4 THEN 95
105 POKE 18,O:POKE 19,O:POKE 20,0
110 ON K GOSUB 1000, 2000, 3000, 4000
115 PRIN'I' : PRINT "DANE SORTOWANIA"
120 T=(65535*PEEK(18)+256*PEEK(19)+PEEK(20))/50
125 PRINT: PRINT "CZAS SORTOWANIA";T;"SEKUND"
130 PRINT "CZY CHCESZ ZOBACZYC DANE""INPUT K$
135 IF K$<>"Y" THEN 165
140 PRINT : PRINT
145 FOR J=1 TO N
150 PRINT D(J)
155 NEXT J
160 INPUT K$
165 IF K$<>"S" THEN 55
170 END
```

11.6. Sortowanie alfabetyczne

Poniższy program jest przykładem wykorzystania metody prostego sortowania do ustawienia pewnego zbioru w porządku alfabetycznym. Do porównania dwóch ciągów używamy generatora "większy niż'(*), przy* czym jeden tekst jest większy od drugiego gdy alfabetycznie położony jest bliżej końca. W tym przypadku pojawił się problem - w ATARI brak jest wielowymiarowych tablic. Przeto żeby zapamiętać szereg ciągów w jednej tablicy np. AS(2000) stosujemy sztuczkę polegającą na tym, że pierwszy ciąg zapisujemy w miejscach 1 - 20 tablicy AS, drugi w miejscach 21 - 40 itd. w końcu ostatni w miejscach 1981 - 2000. Ogólnie można powiedzieć, że n-ty ciąg będzie zapamiętany w miejscach 2(hcn-19 do 20xn.

PROGRAM "SORTOWANIA ALFABETYCZNEGO"

```
5 REM PROGRAM "SORTOWANIE ALFABETYCZNE"
10 DIM A$(2000), B$(20), C$(20)
15 PRINT CHR$(125)
20 PRINT : PRINT "SORTOWANIE ALFABETYCZNE"
25 PRINT: PRINT "PODAJ TEKSTY DO SORTOWANIA"
35 N=O:PRINT
40 N=N+1
45 PRINT N;")";: INPUT B$
50 IF B$="" OR N>100 THEN 70
55 P=N*20-19:A$(P)=B$
60 FOR J=P+LEN(B$) TO N*20:A$(J)=" : NEXT J
65 GOTO 40
70 PRINT : PRINT "POCZATEK SORTOWANIA"
75 FOR J=2 TO N-1
80 B$=A$(J*20-19,J*20)
85 FOR K=J-1 TO 1 STEP -1
90 C$=A$(K*20-19, K*20)
95 IF C$<B$ THEN 110
```

```
100 A$(K*20+1, K*20+20)=C$
105 NEXT K
110 A$(K*20+1, K*20+20)=B$
115 NEXT J
120 PRINT
125 FOR J=1 TO N-1
130 PRINT J; ")"; A$(J*20-19,J*20)
135 FOR K=1 TO 200:NEXT K
140 NEXT J
145 PRINT: PRINT "NACISNIJ RETURN"
150 INPUT B$
155 IF B$="" THEN 15
```

Istnieje jeszcze wiele technik sortowania, ale są one w zasadzie odmianami, modyfikacjami metod przedstawionych powyżej. Programów sortujących można używać do porządkowania różnych katalogów np. kaset, adresów, numerów telefonów itp.

Rozdział 12

PROSTE PROGRAMY OBLICZENIOWE

W rozdziale niniejszym podamy kilka gotowych programów rozwiązujących pewne dość często spotykane problemy matematyczne. Nie bodziemy szczegółowo omawiać teorii dotyczącej danego zagadnienia, również programy nie będą szczegółowo analizowane. Pozostawimy to dociekliwości Czytelników.

Niektóre programy zostały napisane tak, aby można było wykorzystać sam podprogram rozwiązujący dane zagadnienie w jakimś innym większym programie.

12.1 Obliczanie współczynników Newtona

Dwumianem Newtona nazywamy wzór:

$$(a + b)^n = \binom{n}{0}a^n b^0 + \binom{n}{1}a^{n-1} b^1 + \binom{n}{2} a^{n-2} b^2 + \ldots + \binom{n}{n} a^0 b^n$$

współczynnikami są liczby:

$$\binom{n}{0}, \binom{n}{1}, \ldots \binom{n}{n}$$

Obliczenie ich jest zadaniem naszego programu. Dowolny współczynnik dany jest wzorem

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Z rozwinięcia ostatniej formy będziemy korzystać przy obliczeniach_

Program składa się w zasadzie z trzech pętli FOR ... NEXT. Pierwsza zawarta w liniach 40- 130 zmienia k od 11 do n. Wewnątrz niej zawarte są dwie pętle_ Jedna z nich zawarta między liniami 60 a 311 oblicza wartość licznika, natomiast druga zawarta między liniami 90 a 110 dzieli otrzymany wynik przez mianownik.

PROGRAM "WSPÓŁCZYNNIKI NEWTONA"

```
10 REM PROGRAM "WSPOLCZYNNIKI NEWTONA"
                                    20 PRINT "PODAJ N"
30 1NPUT N
40 FOR I=-O TO N
50 S=1
55 IF I-=N THEN 120
60 FOR F=I+1 TO N
                                    70 S-S*F
80 NEXT F
90 FOR F=1 TO N-I
                                    100 S=S/F
110 NEXT F
120 PRINT "C=  "; S
130 NEXT I
140 END
```

12.2. Mnożenie dwóch wielomianów

Dla dwóch danych wielomianów:

$w_1(x) = a_0 + a_1x + a_{2x}{}^2 + \ldots + a_nx^n$

$w_2(x) = b_0 + b_1x + b_2x^2 + \ldots + b_mx^m$

należy wyznaczyć iloczyn postaci:

$$w(x) = w_1(x) * w_2(x) = \sum_{k=2}^{n+m=2} C_{k-2}\, X^{k-2}$$

Współczynniki wielomianu w(x) dane są wzorem:

$$C_{k-2} = \sum_{i\ j} a_i\, b_j$$

gdzie i = 1,2,3,4,....,n, j = 1,2,3,...,m

k = i + j, k = 2, 3, 4,...., n + m + 2

Wiersze 10 do 110 pozwalają wprowadzić do programu współczynniki wielomianów. Następnie zostaje wywołany podprogram obliczający kolejne współczynniki iloczynu i drukujący je.

PROGRAM "ILOCZYN WIELOMIANÓW"

```
10 REM PROGRAM "ILOCZYN WIELOMIANOW"
20 PRINT "PODAJ NAJWYZSZY STOPIEN WIELOMIANU"
30 INPUT N: N=N+1
40 DIM A(N), B(N)
50 M-N
55 PRINT "PODAJ WSPOLCZYNNIKI WIELOMIANOW"
56 PRINT "POCZYNAJAC OD NAJWYZSZEGO STOPNIA"
60 FOR F=N TO 1 STEP -1
70 PRINT "A("; F-1; ")=?, B("; F-1; ")=?"
80 INPUT C, D: A(F)=C: B(F)=D
90 NEXT F
100 GOSUB 1000
110 END
1000 REM PODPROGRAM OBLICZAJACY
1010 REM ILOCZYN DWOCH WIELOMIANOW
1020 FOR K=2 TO N+M+2
1030 S=0
1040 FOR I=1 TO N
1050 J=K-1
1060 IF J>M OR J<1 THEN GOTO 1080
1070 S=S+A(I)*B(J)
1080 NEXT I
1090 PRINT "C("; K-2; ")="; S
1100 NEXT K
1110 RETURN
```

12.3. Dzielenie wielomianu przez dwumian (X+A)

Należy wyznaczyć wielomian $W_2(x)$ oraz resztę R otrzymane z dzielenia wielomianu przez dwumian (X+A). Znaczy to, że dla danego wielomianu:

$$W(x)/(X+A) = W_2(x) + R / (X+A) = c_0 + c_1x + c_2x^2.....+ c_nix^{n}\text{-}' + R/(X+A)$$

Współczynniki wielomianu W_2 (x) dane są wzorami:

$c_{n-1} = a_n$

$c_{n-1} = a_{n-1} * A$ dla i-1, 2, 3,...., n-1

oraz

$R = a_1\text{-}c1\ A$

Struktura programu jest podobna do programu poprzedniego. Współczynniki wielomianu $W_2(x)$ zapisane są w tablicy F3, reszta z dzielenia przechowywana jest w zmiennej R.

PROGRAM "DZIELENIE WIELOMIANU"

```
10 REM PROGRAM "DZIELENIE WIELOMIANU"
15 REM PRZEZ DWUMIAN TYPU X+A"
20 PRINT "PODAJ STOPIEN WIELOMIANU"
30 INPUT N: N=N+1
40 DIM A(N), B(N-1)
50 PRINT "PODAJ WSPOLCZYNNIKI WIELOMIANU"
55 PRINT"POCZYNAJAC OD NAJWYZSZEGO STOPNIA"
60 FOR F=N TO 1 STEP -1
70 PRINT "A("; F-1; ")=?"
80 INPUT C: A(F)=C
90 NEXT F
100 PRINT "PODAJ WYRAZ WOLNY DWUMIANU"
110 INPUT AS
120 GOSUB 1000
130 FOR F=N-1 TO  1 STEP -1
140 PRINT "B("; F-1") ="; B(F)
150 NEXT F
160 PRINT "RESZTA Z DZIELENIA R="; R
170 F:ND
1000 REM PODPROGRAM OBLICZAJACY ILORAZ
1005 REM WIELOMIANU PRZEZ DWUMIAN X+A
1010 B(N-1) = A(N)
1020 FOR I=1 TO N-1
1070 B(N-I-1) = A(N-I) - B(N - I)*AS
1040 NEXT I
I050 R = A(1 -B(1)*As
1060 RE'TURN
```

12.4. Poszukiwanie miejsc zerowych funkcji

Poniżej przedstawiono dwie metody numerycznego rozwiązywania równania postaci: f(x) = 0

gdzie: f - funkcja ciągła

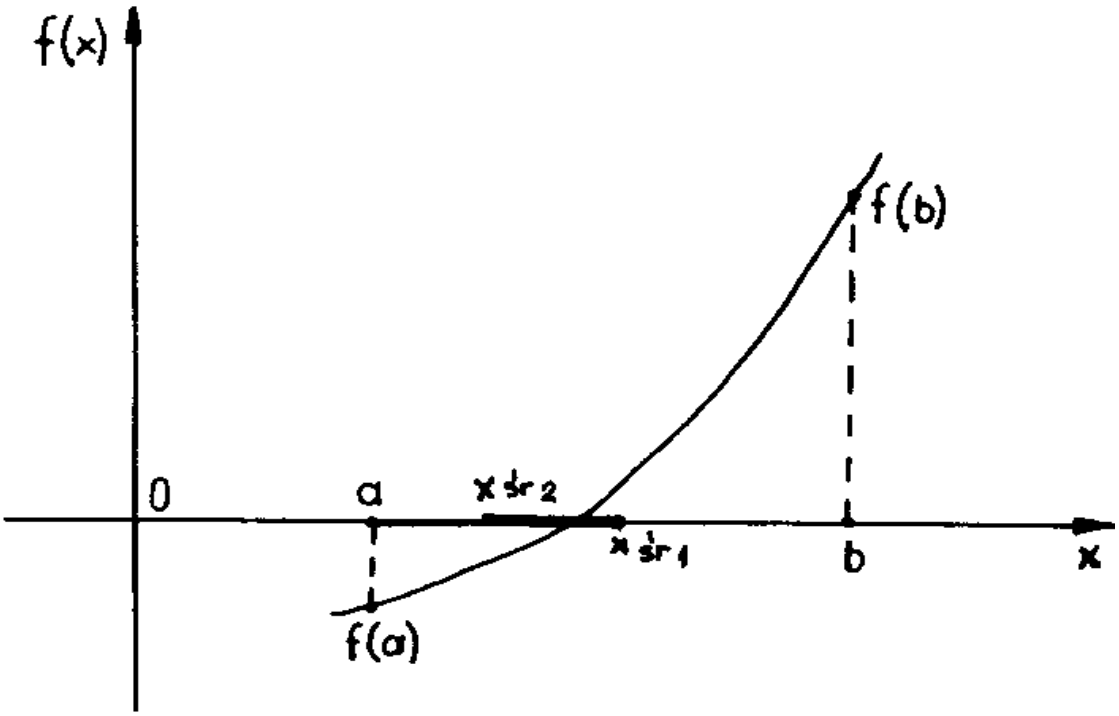
Komputer okazuje się niezwykle pomocnym narzędziem przy rozwiązywaniu tego zagadnienia, gdyż istnieją równania, których analityczne rozwiązanie jest bardzo trudne, niekiedy wręcz niemożliwe, czy po prostu bardzo pracochłonne.

Metoda połowienia przedziału.

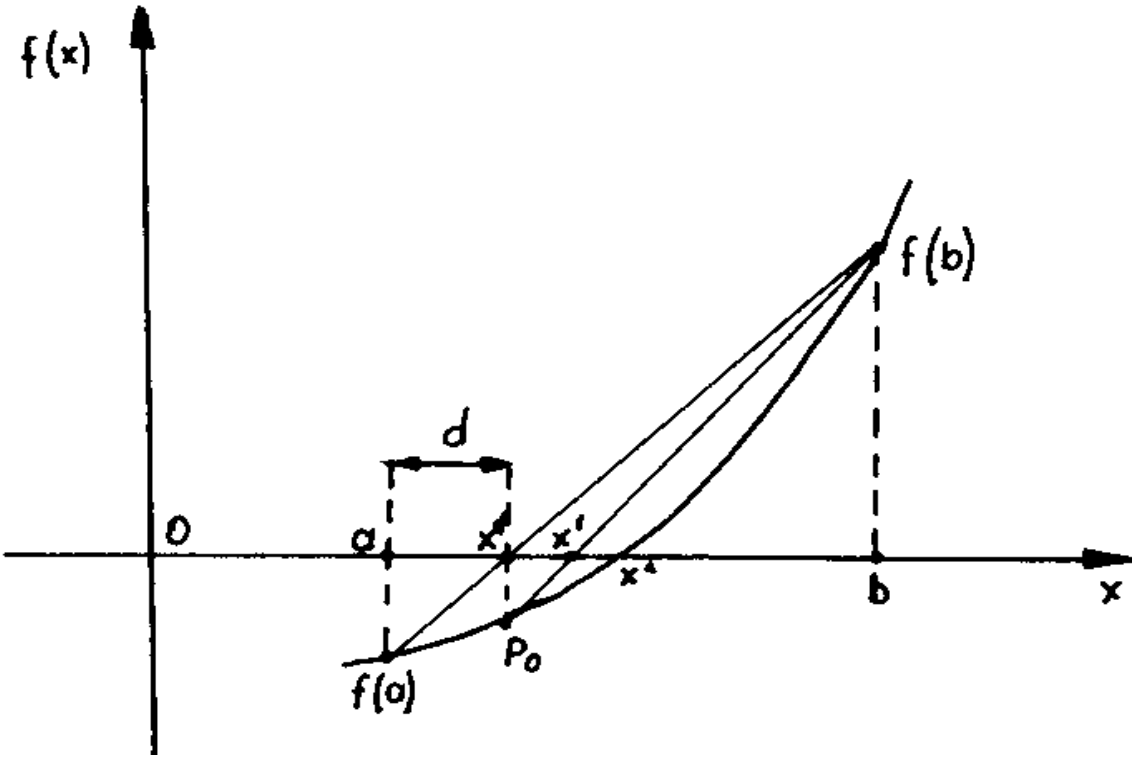
Załóżmy, że potrafimy w przybliżeniu określić przedział, w którym znajduje się pierwiastek, oraz że funkcja jest monotoniczna w tym przedziale. Przy wyznaczaniu pierwiastka metodą połowienia przedziału będziemy stosować następującą metodę: wyznaczamy środek aktualnie badanego przedziału i obliczamy wartość funkcji w tym punkcie. W kolejnym kroku badamy, czy wartość funkcji w badanym przedziale zmieniła znak czy też nic. Jeżeli tak to zawężamy przedział do (x_1, $x_{środek}$), sprawdzamy warunek dokładności i albo drukujemy wynik, alko powtarzamy procedurę. (por. rys. 12.1.)

Metoda siecznych

Załóżmy, że funkcja f(x) jest ciągła i monotoniczna w przedziale (a, b), posiada jedyny pierwiastek w tym przedziale przy czym f(a) * f(b) < 0. Zadaniem naszym jest znaleźć takie x, dla którego f(x) = 0 z pewnym błędem epsilon. Idea tej metody została pokazana na rysunku 12.2



RYS. 12.1.



RYS. 12.2.

Ciąg kolejnych przybliżeń x^0, x^1 ... punktu x jest wyznaczany przez kolejne punkty przecięcia osi x przez sieczne łączące punkty $(x_{i-1}, f(x_{i-1}))$ i $(x_1, f(x_i))$ przy zachowaniu warunku $f(x_i) * f(x_{i-1}) < 0$. Warunek ten

oznacza, że funkcja ma w punktach x_{i-1} oraz x_1 różne znaki, czyli poszukiwane miejsca zerowe. Kolejne wartości punktów ciągu można wyznaczyć przez dodawanie w kolejnych krokach wielkości d, która na przykład dla wyznaczenia x_0 wyraża się wzorem:

$$d = \frac{(b - a) \mid f(a) \mid}{\mid f(a) \mid + \mid f(b) \mid}$$

wtedy $x_0 = a + d$

Opisana metoda zawodzi w przypadku gdy funkcja f (x) jest styczna do osi X.

PROGRAM "MIEJSCA ZEROWE"

```
10 REM PROGRAM
11 REM "SZUKANIE MIEJSC ZEROWYCH
12 REM FUNKCJI METODA POLOWIENIA
13 REM PRZEDZIALU LUB METODA
14 REM RAGULA FALSI"
15 PRINT CHR$ (125)
20 PRINT "PODAJ DANE WEJSCIOWE":PRINT :PRINT
30 PRINT "OKRESL FUNKCJE PRZEZ WPISANIE LINI": PRINT
40 PRINT " 2000 F= funkcja" : PRINT
50 PRINT "NASTEPNIE WYKONAJ ROZKAZ `GOTO 70' ": PRINT: PRINT
60 STOP
70 PRINT "PODAJ PRZEDZIAL W KTORYM ZNAJDUJA SIE PIERWIASTKI"
80 PRINT "X1=?, X2=?"
50 INPUT X1, X2
100 PRINT "PODAJ DOKLADNOSC OBLICZEN"
110 INPUT EPS
120 PRINT "WYBIERZ METODE: "
125 PRINT " 1. POLOWIENIA PRZEDZIALU"
130 PRINT " 2. REGULA FALSI"
140 INPUT M
150 IF M<1 OR M>2 THEN GOTO 120
160 ON M GOSUB 1000, 1500
170 PRINT "SZUKANE MIEJSCE ZEROWE X=  "; X
180 END
1000 REM PODPROGRAM POSZUKUJE MIEJSC
1001 REM ZEROWYCH
1010 X=X1: GOSUB 2000: F1=F
1040 X=(X1 + X2)/2 : GOSUB 2000: FM=F
1070 IF F1*FM=0 THEN GOTO 1140
1080 IF F1*FM>0 THEN GOTO 1110
1090 X2=X: GOTO 1130
1110 F1=FM: XI=X
1130 IF ABS(X2-X1) >EPS THEN GOTO 1040
1140 X=(X1 + X2)/2
1150 RETURN
1500 REM SZUKANIE MIEJSC ZEROWYCH
1501 REM FUNKCJI METODA
1502 REM REGULA FALSI
1510 I=1: N=5000:REM N OKRESLA MAKSYMALNA ILOSC ITERACJI
1520 X=X1:GOSUB 2000:F1=F
1530 X=X2:GOSUB 2000:F2=F
1540 D=(X2-X1)*ABS(F1)/ABS(F1) + ABS (F2))
1550 X=X1 +D: GOSUB 2000: FX=F
1560 IF X=0 OR ABS(D/X) <=EPS OR FX * F1 = 0 THEN RETURN
1570 IF FX*F1>0 THEN GOTO 1590
```

```
1580 X2=X: F'2==FX: GOTO 1600
1590 X1=X: F1=F'X
1600 I=I+1
1610 IF I>=N THEN PRINT "PROCEDURA ZA MALO WYDAJNA DO PODANEJ DOKLADNOSCI":
RETURN
1620 GOTO 1540
1630 RETURN
2000 F=X^3 – 23 * X^2 + 62*X - 40
2010 RETURN
```

12.5. Rozwiązywanie równania różniczkowego metodą Rungego-Kutty rzędu czwartego

Dane jest równanie niestacjonarne (współczynniki są funkcją zmiennej niezależnej t) i nieliniowe równanie różniczkowe:

$dy/dt = f(t, y)$

z warunkiem początkowym

$y(t_0) = y_0$

Należy znaleźć rozwiązanie równania w przedziale $t_0 \le t \le t_k$
Do rozwiązania tak postawionego problemu zastosujemy metodę Rungego-Kutty.

Krzywa, która jest rozwiązaniem równania zostaje przybliżona łamaną składającą się z odcinków o stałej długości Δt. Kolejne rozwiązania będą obliczane tylko na końcach odcinków i są one dane wzorami:

$y_{n+1} = \Delta t/6*(k_1+2*k_2+2*k_3+k_4)+y_n$
gdzie:
$k_1 = f(t_n y_{n)}$
$k_2 = f(t_n + 0{,}5 * \Delta t, y_n + 0{,}5 * k_1)$
$k_3 = f(t_n + 0{,}5 * \Delta t, y_n + 0{,}5 * k_2)$
$k_4 = F(t_n + \Delta t, y_n + k_3)$

W przedstawionym programie możliwe jest drukowanie wyników co kilka odcinków Δt. Dokonuje się tego przez podanie czasu komunikacji (z operatorem) większego niż krok całkowanie nt.

PROGRAM "RÓWNANIE RÓ~NICZKOWE"

```
1 REM PROGRAM "CALKOWANIE METODA
2 REM RUNGEGO-KUTTY RZEDU 4"
5 PRINT "PODAJ DANE WEJSCIOWE": PRINT :PRINT
7 PRINT "ROZWIAZYWANE ROWNANIE MA POSTAC DY/DT= funkcja  (Y,T)": PRINT
10 PRINT "WPISZ WIERSZ: 2000 C=FUNKCJA(Y,T)": PRINT
12 PRINT "NASTEPNIE WYKONAJ INSTRUKCJE GOTO 20":PRINT:PRINT
15 STOP
20 PRINT "WARTOSC POCZATKOWA Y(0)=?: INPUT Y1
30 PRINT "CZAS POCZATKOWY TO=?": INPUT T1
40 PRINT "CZAS KONCOWY TK=?": INPUT K
50 PRINT "KROK CALKOWANIA H=?":INPUT H
56 PRINT "CZAS KOMUNIKACJI K=?":INPUT F
60 GOSUB 3000
70 END
?000 C=T-SIN(Y)
2010 RETURN
3000 REM PODPROGRAM CALKOWANIA METODA
3001 REM RUNGEGO-KUTTY RZEDU 4
3005 G = F + 0,01
3010 IF G <= F THEN GOTO 3040
```

```
3020 PRINT "DLA T=";T1" Y=";Y1
3030 G = H
3040 G = G + H
3050 IF T1>=K THEN RETURN
3060 Y=Y1:T=T1:GOSUB 2000:L=C
3100 Y=Y1+L/2*H:T=T1+H/2: GOSUB 2000:M=C
3140 Y=Y1+M/2*H:T=T1+H/2:GOSUB 2000:N=C
3180 Y=Y1+N*H:T=T1+H:GOSUB 2000
3210 Y=Y1+(L+2*M+2*N+C)/6*H
3220 T1=T1+H 3230 GOTO 3010 3240 RETURN
```

12.6 Rozwiązywanie układu równań liniowych

Dany jest układ równań liniowych:

$a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + ... + a_n x = b_1$

$a_{21} x_1 + a_{22} x_2 -+ a_{23} x_3 + ... + a_{2n} x = b_2$

:::

a_{n1} xl + $a_{n2} x_2 + a_{n3} x_3 + ... + A_{nn} x = b_n$

co można zapisać w postaci macierzowej:

AX = B

Istnieje wiele metod rozwiązania takiego równania. Najbardziej znana i najmniej efektywna jest metoda Cramera (tzw. metoda wyznaczników), która wymaga w przybliżeniu wykonania N=2(n+ 1)! mnożeń dla rozwiązania układu równań stopnia n.

Poniżej przedstawiamy program działający w oparciu o algorytm eliminacji Gaussa. Metoda eliminacji Gaussa jest bardzo prostym i przy tym niezwykle efektywnym algorytmem służącym do rozwiązania postawionego tu problemu. Oparta jest ona na idei kolejnych eliminacji zmiennych aż do uzyskania jednego równania z jedną niewiadomą, powiedzmy x_n i następnie kolejnym podstawieniu wstecz do poprzednich równań w wyniku czego otrzymujemy pozostałe rozwiązania. Metoda ta wymaga wykonania tylko $N = n^3 /3$ mnożeń. Sprawdźmy, o ile mnożeń to mniej niż w metodzie Cramera np. dla układu pięciu równań (n=5).

PROGRAM " UKŁAD RÓWNAŃ"

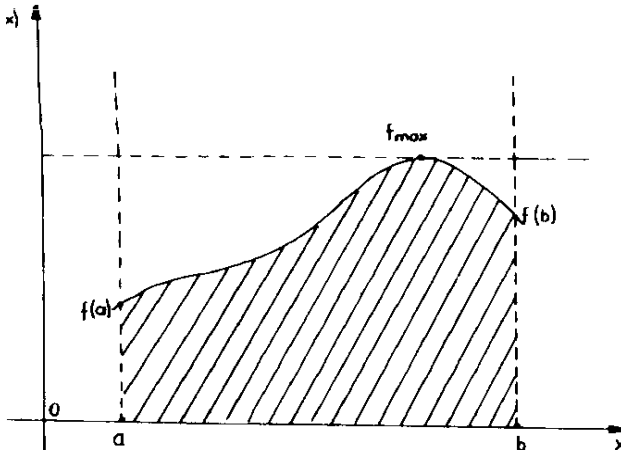
```
10 REM PROGRAM "UKLAD ROWNAN"
90 PRINT CHR$(125)
100 PRINT "PROGRAM ROZWIAZUJE UKLAD n ROWNAN LINIOWYCH METODA ELIMINACJI
GAUSSA"
110 PRINT : PRINT
120 PRINT "PODAJ ILOSC ROWNAN N=";: INPUT N
130 IF N=1 THEN GOTO 90
140 DIM A(N,N), B(N)
150 PRINT :PRINT
160 PRINT "PODAJ WSPOLCZYNNIKI ROWNAN (WIERSZAMI)"
190 FOR I=1 TO N:FOR J=1 TO N
200 PRINT "A(";I; ", "; J;") = ";
210 INPUT Q: A(I,J)=Q
220 NEXT J
230 PRINT "B(";I;")= ";
240 INPUT Q:B(I)=Q
250 NEXT I
270 GOSUB 1000
280 GOSUB 2000
290 PRINT: PRINT "ROZWIAZANIE UKLADU ROWNAN": PRINT
330 FOR I=1 TO N: PRINT "X";I;" = "; B(I): NEXT I
340 END
1000 REM ZEROWANIE SCHODKOWE
```

```
1010 FOR J=1 TO N-1
1020 IF A(J,J)=0 THEN GOSUB 3000
1030 FOR I = J + 1 TO N
1040 IF A(I,J)=0 THEN 1100
1050 M=A(I,J)/A(J,J)
1060 FOR K = J TO N
1070 A (I,K)=A(I,K) - M*A(J,K) 1080 NEXT K
1090 B(I)=B(I) - M*B(J)
1100 NEXT I: NEXT J
1110 RETURN
2000 REM OBLICZANIE WYNIKOW
2010 IF A(N,N)=0 THEN 2110
2020 B(N)=B(N)/A(N,N)
2030 FOR I = N -1 TO 1 STEP -1
2040 D=0
2050 FOR J = 1 + I  TO N
2060 D = D + A(I,J) * B(J)
2070 NEXT J
2080 B(I) = (B(I)-D)/A(I,I)
2090 NEXT I
2100 RETURN
2110 PRINT:PRINT:PRINT "** UKLAD NIEOZNACZONY LUB SPRZECZNY **"
2120 END
3000 REM ZAMIANA WIERSZY
3010 P=J
3020 P=P+1
3030 IF P=N+1 THEN 2110
3040 IF A(P,J)=0 THEN 3020
3050 FOR I=1 TO N
3060 M=A(J,I): A(J,I)=A(P,I):A(P,I)=M
3070 NEXT I
3080 M=B(J):B(J)=B(P): B(P)=M
3090 RETURN
5000 STOP
```

12.7. Obliczanie wartości całki metodą Monte Carlo

Metoda Monte Carlo jest prostą i niekiedy bardzo wygodną metodą obliczania wartości całek na komputerze. Aby wyjaśnić jej działanie należy znać interpretację geometryczną całki.

Geometrycznie całka jest to pole powierzchni zawarte między krzywą całkową a osią x-ów.



RYS. 12.3. Interpretacja geometryczna całki

Zadaniem naszym jest obliczenie wartości całki $\int_{x_1}^{x_2} f(x)dx$

Zbudujmy prostokąt jak na rysunku. Wybierajmy w nim losowo n punktów. Sprawdźmy ile punktów leży pod krzywą całkową. Stosunek ilości wygenerowanych punktów pod krzywą całkową do całkowitej ilości wygenerowanych punktów jest równy stosunkowi pola pod krzywą do pola prostokąta. Pole prostokąta potrafimy łatwo obliczyć, tak więc potrafimy obliczyć pole pod krzywą czyli wartość całki.

Dokładność obliczeń będzie zależała od ilości wygenerowanych punktów oraz równomierności pokrycia całego prostokąta, czyli inaczej mówiąc od jakości generatora liczb losowych.

Program zawiera dwa podprogramy. Pierwszy z nich oblicza maksymalną wartość funkcji w przedziale całkowania (ze względu na prosty i mało efektywny algorytm wykonuje się on dość długo). Drugi podprogram oblicza wartość całki. Ilość wygenerowanych punktów powinna być większa niż 100 ze względu na dokładność obliczeń i mniejsza niż 5000 ze względu na długi czas liczenia.

PROGRAM "MONTE CARLO"

```
10 REM PROGRAM "MONTE CARLO"
11 REM PROGRAM OBLICZA WARTOSC
12 REM CALKI OZNACZONEJ METODA
13 REM MONTE CARLO
14 PRINT CHR$(125): PRINT "PODAJ FUNKCJE PODCALKOWA PRZEZ WPISANIE": PRINT
15 PRINT "WIERSZA 2000 Y= funkcja podcałkowa (x)": PRINT
16 PRINT "NASTEPNIE WYKONAJ 'GOTO 20' ": PRINT
17 STOP
20 PRINT "PODAJ GRANICE CALKOWANIA X1=?, X2=?"
30 INPUT X1, X2
40 PRINT "PODAJ ILOSC GENEROWANYCH PUNKTOW N=?"
50 INPUT N
60 IF N>100 AND N<5000 THEN GOTO 110
70 PRINT "JESTES PEWIEN ? (T/N)",
80 IF PEEK (764)=35 THEN POKE 764,255:PRINT "NIE":GOTO 40
90 IF PEEK (764)=45 THEN POKE 764,255:PRINT "TAK":GOTO 110
100 GOTO 80
110 GOSUB 1000
120 GOSUB 1500
130 PRINT "WARTOSC CALKI C="; CALKA
140 END
1000 REM PODPROGRAM SZACUJE
1010 REM MAKSYMALNA WARTOSC FUNKCJI
1020 EPS -(X2-X1)/N
1030 X=X1: GOSUB 2000: MAX=Y
1040 FOR F=1 TO N
1050 X=X+EPS: GOSUB 2000
1060 IF MAX<Y THEN MAX=Y
1070 NEXT F
1080 PRINT "MAKSYMALNA WARTOSC FUNKCJI MAX="; MAX
1100 RETURN
1500 REM PODPROGRAM OBLICZA WARTOSC
1505 REM LALKI METODA MONTE CARLO 1510 I=O:K=O
1520 FOR I=1 TO N
1530 X=X1+RND(0)*(X2-XI): GOSUB 2000
1540 IF Y>=RND(0)*MAX THEN K=K+1
1550 NEXT I
1560 CALKA=K/N*MAX*(X2-X1)
1570 RETURN
2010 RETURN
```

12.8. Mnożenie macierzy

W tym punkcie przedstawiamy problem mnożenia macierzy. Przy użyciu kartki i ołówka jest tu problem łatwy do rozwiązania, gdy wymiar macierzy jest mały, jednak przy większych rozmiarach macierzy np. 6,7 itd. staje się to uciążliwym zadaniem (praca żmudna i łatwo o pomyłkę).

W podanym programie korzystamy ze znanego wzoru na iloczyn macierzy: n

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\ b_{kj}$$

gdzie: $c_{ij}\ b_{kj}$, a_{ik} , b_{kj} są elementami macierzy

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & .... & a_{1n} \\ a_{21} & a_{22} & a_{23} & .... & a_{2n} \\ . & . & . & . & . \\ a_{m1} & a_{m2} & a_{m3} & .... & a_{mn} \end{bmatrix}$$

$$B=\begin{bmatrix} b_{11} & b_{12} & b_{13} & \ldots & b_{1n} \\ b_{21} & b_{22} & b_{23} & .... & b_{2n} \\ . & . & . & . & . \\ b_{m1} & b_{m2} & b_{m3} & .... & b_{mn} \end{bmatrix}$$

$$C=A*B=\begin{bmatrix} c_{11} & c_{12} & c_{13} & \ldots & c_{1n} \\ c_{21} & c_{22} & c_{23} & .... & c_{2n} \\ . & . & . & . & . \\ c_{m1} & c_{m2} & c_{m3} & .... & c_{mn} \end{bmatrix}$$

Program składa się z trzech części. Pierwsza to wprowadzenie danych do macierzy A (pierwszej) i B (drugiej). Druga część programu to obliczanie iloczynu A*B, natomiast trzecia to wyprowadzenie wyników tzw. macierz C.

PROGRAM "MNOŻENIE MACIERZY"

```
10 REM PROGRAM "MNOZENIE MACIERZY"
20 ? CHR$(125)
100 ? "PODAJ WYMIAR MACIERZY PIERWSZEJ"
105 ? "ILOSC WIERSZY =";:INPUT I: IF I<1 THEN 105
110 ? "ILOSC KOLUMN =";: INPUT J: IF J<1 THEN 110
120 DIM A(I,J)
130 ? "PODAJ WSPOLCZYNNIKI MACIERZY (wierszami)"
140 FOR X=1 TO I:.FOR Y=1 TO J
145 ? "A("; X; ", "; Y;  ")=";
150 INPUT Q:A(X,Y)=Q
160 NEXT Y: ?: NEXT X
200 ? "PODAJ WYMIAR MACIERZY DRUGIEJ"
205 ? "ILOSC WIERSZY=";: INPUT K: IF K<>J THEN 205
210 ? "ILOSC KOLUMN =";: INPUT L: IF L<1 THEN 210
```

```
220 DIM B(K,L), C(I,L)
230 ? "PODAJ WSPOLCZYNNIKI MACIERZY (wierszami)"
240 FOR X-1 TO K: FOR Y=1 TO L
245 ? "B(";X; ", "; Y; ")=";
250 INPUT Q:B(X,Y)=Q
260 NEXT Y:? : NEXT X
300 REM ** MNOZENIE **
310 FOR X=1 TO L: FOR Y=1 TO I
320 D=0: FOR Z=1 TO J
330 D=D+A(Y,Z)*B(Z,X)
340 NEXT Z: C(Y,X)=D
350 NEXT Y: NEXT X
400 REM ** WYPROWADZENIE WYNIKOW **
410 ? CHR$(125)
415 ? "ILOCZYN MACIERZY (wierszami)"
420 FOR X=1 TO I: FOR Y=1 TO L
430 ? "C("; X; ", "; Y; ") = "; C(X,Y)
440 NEXT Y
445 ? : ? "NACISNIJ SPACE"
450 IF PEEK(764)=255 THEN 450
460 ?: POKE 764,255
470 NEXT X
```

12.9. Regresja liniowa

Niekiedy zachodzi potrzeba przeprowadzenia linii prostej przez szereg punktów na wykresie XY. Ułatwi nam tu program badający równanie tej prostej. Jest kilka sposobów rozwiązania tego problemu. Posłużmy się najpopularniejszą metodą tzw. najmniejszych kwadratów. Polega ona na takim doborze parametrów prostej, by suma kwadratów różnic wartości y_1 oraz obliczonych $y_1 = ax_x + b$ osiągnęła minimum.

$$s^2 = \sum_{i=1}^{n} [y_i - (ax_i + b)^2 = \min$$

Zakładamy ponadto, że wszystkie punkty są obarczone takim samym błędem przypadkowym o rozkładzie Gaussa.

W wyniku obliczeń otrzymamy wzory na wartość i odchylenia standardowe parametrów a i b z równania prostej y = ax + b.

gdzie:

$$a = {}^{1}\backslash W \; (n \sum x_i y_i - \sum x_i \sum y_i)$$
$$a = {}^{1}\backslash W \; (\sum y_i \sum x_i - \sum x_i \sum x_i y_i)$$

$$sa = \sqrt{n / (n-2)} \qquad \sqrt{(y_i - ax_i - b^2) / W}$$

$$sb = sa \sqrt{x_i \backslash n}$$

$$W = n \sum x_{i2} - (\sum x_i)^2$$

Program składa się z trzech części. Pierwsza to wprowadzanie współrzędnych punktów. Gdy chcemy zakończyć wpisywanie danych należy wprowadzić słowo END. Program jest przeznaczony do wprowadzenia 100 punktów. Gdy mamy większą ilość punktów należy zwiększyć wymiar macierzy X i Y w linii

10 DIM X(100), Y(100).....

Druga część programu oblicza wszystkie potrzebne wartości występujące we wzorze na a, b, sa, sb. Trzecia część podaje nam wzór na prostą łącznie z odchyleniami standardowymi.

PROGRAM "REGRESJA LINIOWA"

```
1 REM PROGRAM "REGRESJA LINIOWA"
10 DIM X(100), Y(100), P$(10), R$(10)
20 N=1
25 PRINT CHR$(125): PRINT: PRINT "PODAJ WSPOLRZEDNE PUNKTU X i Y"
30 PRINT "PUNKT"; N: INPUT P$: IF P$(1,1)="E" THEN N=N-I: GOTO 200
50 INPUT R$:Y(N)=VAL(R$): X(N)=VAL(P$)
60 N=N+1:GOT0 30
200 A=0: B=0: C=0: D=0
210 FOR F=1 TO N
220 A=A+X(F): B=B+Y(F): C=C+X(F)*X(F): D=D+Y(F)*X(F)
230 NEXT F
240 W=N*C - A*A: S=(N*D-A*B)/W: T=(B*C-A*D)/W
250 FOR F=1 TO N
260 E=E+(Y(F)-S*X(F)-T)^2
270 NEXT F
280 SA=SQR(N*E/N/(N-2)): TB=SA*SQR(C/N)
290 PRINT: PRINT"ROWNANIE PROSTEJ MA POSTAC Y=AX+B": PRINT: PRINT"gdzie": PRINT
300 PRINT "A=";S;"+-";SA: PRINT" B=";T;" +-";TB
```

12.10. Statystyka

Niżej przedstawiony program oblicza nam średnią arytmetyczną ~, wariancję rozkładu w próbie s^2, wariancję rozkładu w populacji s^2 , oraz moment drugiego rzędu x^2 z wprowadzonych próbek.

gdzie:

$$x = {}^{1}\!\setminus_{n} \sum_{i=1}^{n} x_i$$

$$s^2 = {}^{1}\!\setminus_{n} \sum_{i=1}^{n} (x_i - x)^2$$

$$s^2 = {}^{1}\!\setminus_{n-1} \sum_{i=1}^{n} (x_1 - x)^2$$

$$x^2 = {}^{1}\!\setminus_{n} \sum_{i=1}^{n} x_{i2}$$

n - ilość próbek

W pierwszej części programu wprowadzamy wartości próbek. Zakończenie programu uzyskujemy przez wpisanie słowa END. W wyniku działania programu otrzymujemy na ekranie wartości wg wyżej przedstawionych wzorów.

PROGRAM "STATYSTYKA"

```
10 REM PROGRAM "STATYSTYKA"
80 ? CHR$(125)
90 DIM A(200), Q$(10): F=1
100 ? "PODAJ LICZBY"
110 INPUT Q$
120 IF Q$="END" THEN F=F-I: GOTO 150
130 A(F)=VAL(Q$): F=F+1: GOT0 110
150 ?:? "ILOSC PROB":?"N=";F
200 D=0:FOR I=1 TO F
210 D=D+A(I):NEXT I
220 D=D/F
230 ?:?"SREDNIA ARYTMETYCZNA"
```

```
240 ?"-": ?"X="; D
300 E=0
310 FOR I=1 TO F
320 E=E+(A (I)-D)*(A(I)-D)
330 NEXT I
340 E=E/F
350 ?: ?"WARIANCJA ROZKLADU W PROBIE"
360 ?"2": ?"S="; E
400 G=E/(F-1)
410 ?: ?"WARIANCJA ROZKLADU W POPULACJI"
420 ?"2": ?"S="; G
450 H=O:FOR I=1 TO F
460 H=H+A (I)*A(I)
470 NEXT I
490 ?:? "MOMENT DRUGIEGO RZEDU"
495 ? "2"; ? "2": ? "Y=" : H/F
```

Rozdział 13

OPTYMALIZACJA PROGRAMÓW

W rozdziale tym zastanowimy się nad sposobem pisania programów eleganckich i efektywnych. Omawiane tutaj zagadnienia nabierają szczególnego znaczenia przy pisaniu dużych programów, ale poniższe uwagi warto jest stosować na co dzień aby stały się dobrym nawykiem. Rozważania nasze dotyczyć będą w zasadzie języka ATARI BASIC, ale ich charakter jest ogólny i można je wykorzystać programując w innym języku.

13.1. Styl programowania i czynność programu

Jeżeli pisany przez nas program jest duży i skomplikowany to w czasie jego uruchamiania będzie on wielokrotnie czytany i może wymagać wprowadzenia wielu zmian. Często do pracy nad programem powracamy po dużej przerwie lub też program pisany przez nas jest uruchamiany bądź modyfikowany przez innego użytkownika. Dlatego należy założyć, że odbiorcą programu jest przede wszystkim człowiek, a w następnej kolejności komputer. Przejrzystości programu należy pod-porz4dkować inne mniej ważne kryteria jak na przykład jego sprawność (przez sprawność rozumiemy szybkość wykonywania programu). Stosowanie podanych poniżej wskazówek znacznie ułatwi nam pisanie przejrzystych programów, które będzie później łatwo uruchamiać i poprawiać.

Starajmy się:

Stosować znaczące nazwy zmiennych. Nie używajmy nazw np. WWWX13, AAAA itp. Używanie nazw opisowych zwiększa czytelność programu.

Cyfry umieszczać na końcu nazwy. Pamiętajmy przy tym, że cyfry 1, 0, 2, 5 mogą mylić się z literami l, O, Z, S.

Nie używać juko nazw zmiennych słów kluczowych języka. Mimo, że poniższe linie są w ATARI BASIC syntaktycznie poprawne - mylą zupełnie:

```
155 LET IF=1: LET THEN =75: LET GOTO 175
175 IF IF=THEN THEN GOTO GOTO
```

W miarę możliwości unikać stosowania zmiennych roboczych. Mimo, że zmienne robocze są często potrzebne nigdy nic twórzmy ich zbyt wiele. Duża liczba zmiennych roboczych zwiększa ilość szczegółów, które trzeba analizować przy czytaniu programu. Pamiętajmy ponadto, że w ATAKI 13ASIC można używać jedynie l28 zmiennych.

Nic zmieniać wartości zmiennej sterujących wewnątrz pętli. Może to prowadzić du trudnych du wykrycia błędów. Na przedstawionym poniżej przykładzie widać do jakich nieporozumień może doprowadzić niestosowanie tej zasady:

```
115 FOR I=1 TO N
125 IF W THEN I=N+1 źle
195 NEXT I

115 FOR I=1 TO N
125 IF W THEN GOTO 205 dobrze
195 NEXT I
205 następna linia
```

Starajmy się dzielić program logicznie na podprogramy. Ułatwia to uruchomienie i późniejsze zmiany.

Pisząc programy uważajmy aby w podprogramach nie wykorzystywać tych samych zmiennych co w innych segmentach. Nie zamierzona zmiana przez podprogram wartości zmiennych w innym segmencie to częste źródło trudnych do wykrycia błędów.

Zwróćmy uwagę szczególnie przy stosowaniu instrukcji porównania IF, że liczby zmiennopozycyjne reprezentowane są w komputerze ze skończoną dokładnością. Znaczy to po prostu, że np. 11,5 * 2,5 nie musi dać wyniku dokładnie 26.25.

14.2. Szybkość działania programu

Efektywność działania programu starajmy się osiągnąć przez inteligentny dobór algorytmów i struktur danych a nie przez trywialne chwyty w rodzaju np. eliminacji wyliczania indeksów przy korzystaniu z tablic. Pamiętajmy zawsze o tym, że dopóki program nie działa poprawnie jego efektywność nie ma żadnego znaczenia. Szybki lecz działający błędnie program jest bezużyteczny. Starajmy się:

Nigdy nie ulepsz«ć niczego bez koniecznej potrzeby. Ulepszania programu nie należy traktować jako sztuki dla sztuki. Program można ulepszać tylko wtedy gdy występuje taka konieczność i wiemy, która część programu wymaga udoskonalenia.

Nie poświęcać czytelności dla efektywności. Większość spotykanych sugestii odnośnie poprawy efektywności opiera się na zawitych trikach obniżających czytelność.

Stosować makro a nie mikro oszczędności. Pamiętajmy, że np. niektóre algorytmy (co widać szczególnie na przykładach algorytmów wyszukiwania i sortowania) są setki razy szybsze od innych. Niżej podamy kilka sposobów poprawy efektywności programów. Stosujmy je z umiarem i tylko do tych fragmentów programu, które rzeczywiście tego wymagają. W typowym programie większość czasu przypada na jego niewielką część (często 5%), z reguły są to wielokrotnie wykonywane pętle lub wielokrotnie wywoływane podprogramy. Tam właśnie dokonać można ulepszeń. Wprowadzanie ich w innych miejscach może dać niezauważalne skrócenie czasu wykonania przy kompletnym zaciemnieniu całego programu.

Pamiętajmy, że operacje matematyczne wykonywane są z różnymi szybkościami. Oto lista operacji arytmetycznych uszeregowanych od najszybszych do najwolniejszych.

(1) dodawanie lub odejmowanie
(2) mnożenie
(3) dzielenie
(4) potęgowanie

Widzimy, że dodawanie jest szybsze niż mnożenie, zatem mnożenie przez niewielkie liczby rzeczywiste można zastąpić wielokrotnym dodawaniem np. zamiast 3*I można zapisać I + I + I

W analogiczny sposób można podnoszenie do niewielkiej potęgi zastąpić wielokrotnym mnożeniem.

Można pozbywać się operacji drogą przestawienia wyrażeń. Na przykład X=2*Y+(A-1)/P+2*T można zamienić na X=2*(X+T)+(A-1)/P eliminując jedno mnożenie. Dzielenie jest wolniejsze niż mnożenie, więc zamiast dzielić należy, jeżeli jest to możliwe, mnożyć przez odwrotność. Zamiast dzielić przez 4 można mnożyć przez 0,25 itp. Jeżeli wielokrotnie dzielimy przez tę samą zmienną można zastosować zmienną roboczą w której zapamiętujemy odwrotność i następnie mnożyć przez tę zmienną.

Użycie zmiennej roboczej może być także korzystne w wypadku obliczeń w pętlach. Np. zamiast

```
115 FOR I=1 TO 1000
125 ALFA1=SIN(BETA)      11
135 ALFA2=SIN(BETA)      22
175 ALFA9=SIN(BETA)      99
185 NEXT I
```

Można zapisać:

```
115 POMOC=SIN(BETA)
125 FOR I=1 TO 1000
135 ALFA1=POMOC          11
175 ALFA9=POMOC          99
185 NEXT I
```

W pierwszym przypadku SIN(BETA) jest obliczany 9000 razy, w drugim tylko jeden raz przed wejściem w pętlę. Jeżeli pętli nie da się uniknąć pamiętajmy aby wszystkie obliczenia, których wykonanie w pętli nie jest konieczne umieścić przed pętlą. Niewielkie rachunki można często wykonywać w ogóle bez posługiwania się pętlami.

Przy optymalizacji pętli najwięcej uwagi starajmy się zawsze poświęcić pętli wewnętrznej. Oto przykład:

```
115 FOR A=1 TO 50
125 FOR B=1 TO 50
135 FOR C=1 TO 50
215 NEXT C
225 NEXT B
235 NEXT A
```

W pętli C każda operacja wykonywana jest 75000 razy. Bardzo niewielkie ulepszenie działań wewnątrz pętli C zapewni nam dużo większy zysk niż znaczne nawet zmiany w pętlach A i B.

Raz jeszcze przypomnijmy: ulepszenia stosujmy tylko tam, gdzie jest to konieczne czyli w wielokrotnie wykonywanych pętlach, lub wielokrotnie (z pętli) wywoływanych podprogramach Koza tymi miejscami zupełnie się to nie opłaca.

Teraz kilku dodatkowych uwag specjalnie dla pracujących w ATARI BASIC.

Jeżeli konieczne jest zwiększenie szybkości działania naszego programu możemy to uzyskać przez:

- umieszczenie wielokrotnie wykonywanych pętli i często używanych podprogramów w liniach o jak najniższych numerach,
- umieszczenie pętli FOR-NEXT w jednej linii. Pętla taka wykonywana jest nieco szybciej niż umieszczona w kilku liniach.

Co natomiast można zrobić, jeżeli nasz program nie mieści się w pamięci? Jeżeli nie możemy zmniejszyć zajętości pamięci przez zmianę algorytmu, ograniczenie wymiarów tablic itp. spróbujmy:

- przypisać często występujące stałe (np. 157, 35, "ALFA" itp.) do zmiennych i w programie odwołujemy się wówczas do odpowiednich zmiennych,
- używać podprogramów zamiast wielokrotnie występujących linii programu,
- wykorzystać elementy tablic o zerowych indeksach np. X(0),
- zastosować READ i DATA do inicjacji zmiennych zamiast instrukcji podstawiania lub jeszcze lepiej INPUT lub GET oraz odpowiednich zbiorów,
- zastosować w jednej linii po kilka instrukcji,
- usunąć linie komentarza REM.

Pamiętajmy jednak, że powyższe zmiany mogą znacznie obniżyć czytelność programu. Ich stosowanie starajmy się ograniczyć do naprawdę koniecznych przypadków.

Na koniec jeszcze jedna uwaga. Znaczne zwiększenie szybkości działania programu i zmniejszenie zajmowanego obszaru pamięci możemy uzyskać stosując segmenty w kodzie maszynowym, co jednak jest sprzeczne z założeniem przejrzystości programu. W ATARI BASIC do wywołania procedur w kodzie maszynowym służy funkcja USR.

Nic wyczerpaliśmy tu oczywiście ani wszystkich problemów związanych z tzw. dobrym stylem programowania, ani z poprawą szybkości działania programów. Zainteresowanych bliżej tym problemem zachęcamy do lektury literatury fachowej, jak np. książka G..J. Meyersa "Projektowanie niezawodnego oprogramowania", czy D. Van Tassela "Praktyka programowania". Ponadto zawsze należy zapoznać się szczegółowo z fabrycznymi instrukcjami do użytkowanych komputerów lub interpreterów aby w pełni wykorzystać dawane przez nie możliwości.

Dodatek A

KOMUNIKATY BŁĘDÓW

Błędy oznaczone numerami mniejszymi niż 128 mają podwójne znaczenie w zależności czy używany jest DOS czy tylko BASIC.

Najpierw podamy interpretację tylko dla BASICA:

2. Niewystarczająca pamięć. Jeżeli niemożliwe jest powiększenie RAM-u można zrezygnować z nieużywanych zmiennych lub zmniejszyć wymiary zmiennych. Błąd ten może być spowodowany także instrukcją GOSUB przy zbyt dużej ilości zagnieżdżonych podprogramów.
3. Wartość numeryczna zmiennej wykracza poza dopuszczalny zakres 1E-99 do 1E+98 albo zmiennej przyporządkowana jest wartość ujemna, podczas gdy winna być dodatnia, albo osiągnięto koniec zbioru przy użyciu instrukcji STATUS.
4. Zbyt dużo zmiennych tzn. więcej niż dopuszczalne 128.
5. Przekroczony wymiar zmiennej tekstowej. Należy zmniejszyć tekst lub zwiększyć wymiar zmiennej
6. Błąd w czytaniu danych. Niezgodna ilość danych po DATA dla zadeklarowanych zmiennych instrukcją READ albo próba ponownego odczytania stałych po DATA bez wcześniejszego użycia RESTORE.
7. Numer linii w programie jest liczbą ujemną lub większą niż maksymalnie możliwa 32 767. 8. Nic można zmiennej numerycznej przyporządkować wartości numerycznej.
9. W programie użyto niezadeklarowanej zmiennej tekstowej lub tablicy, albo zadeklarowano tę samą zmienną powtórnie; albo wymiar DIM przekracza 32 767 dla zmiennych tekstowych lub 5 4611 dla tablic.
10. Zbyt wiele nawiasów w wyrażeniu albo wyrażenie jest zbyt długie, albo użyto zbyt wielu instrukcji GOSUB.
11. Dzielenie przez zero albo wynik działania przyjął wartość spoza dopuszczalnego przedziału (1 E-99 do 1 E+98).
12. Nie ma linii wskazanej jako adres po IF - THEN, ON- GOSIJB, ON - GOTO , GOSUB, (:O"1'O.
13. Użyto NEXT bez właściwego FOR.
14. Linia zbyt długa -dopuszczalna 140 znaków.
15. Użyto NEXT albo RETURN gdy właściwa instrukcja FOR albo GOSUB została wcześniej skasowana.
16. Użyto RETURN bez właściwego GOSUB.
17. Użyto niewłaściwie instrukcji POKE, zepsuty RAM albo instrukcje zostały napisane niezgodnie ze składnią języka BASIC.
18. Niewłaściwy znak początku zmiennej albo pierwszy znak argumentu instrukcji VAL nie jest cyfrą.
19. ładowany program jest zbyt długi i nie mieści się w pamięci.
20. Użyto niewłaściwego numeru urządzenia zewnętrznego - spoza przedziału 1 - 7.
21. Użyto niewłaściwej instrukcji do ładowania programu. Właściwe pary instrukcji to: LIST ENTER oraz SAVE-LOAD(RUN).

Interpretacje błędów przy użyciu DOS-a

2. Przy użyciu funkcji X - USER - DEFINED nie ma żadnego zbioru *.CMD na dysku 1.
3. Przy użyciu funkcji RENAME nie podano nowej nazwy albo jest zbyt wiele podobnych nazw aby stosować nazwy niepełne (wild cards) - albo naciśnięto tylko RETURN bez podania żądanego określenia zbioru. Także przy instrukcji STATUS gdy osiągnięto koniec zbioru.
4. Użyto "I'O - CARTRIDGE gdy nie ma żadnego dodatkowego ROM-u albo BASIC - wewnętrzny został wyłączony podczas włączania komputera do sieci (przez naciśnięcie OI'TION). S. Błąd wejścia / wyjścia np. nie podłączona drukarka.
6. Przy instrukcji SAVE użyto adresu końcowego nie większego od adresu początkowego.
7. Przy użyciu funkcji MEM-SAV komputer nie zapisał żadnego programu na dyskietce tak że po pojawieniu się Menu DOS-a program został skasowany. Powodem mogła być zniszczona dyskietka albo brudna głowica stacji dysków.
8. Błąd podobny do poprzedniego z tym, że sygnalizowany już w czasie próby zapisu programu przy użyciu funkcji ME:M.SAV.
9. Błędnie podano określenie urządzenia zewnętrznego.
l0. Błędnie podano określenie zbioru lub programu.

Interpretacja błędów oznaczonych numerami od 128 do 177 nie jest uzależniona od użycia bądź nie użycia DOS-a

128. Klawisz BREAK był użyty podczas trwania operacji wyjścia / wejścia co zatrzymało operację.

129. Użyto instrukcji OPEN dla zbioru już otwartego. Po wykryciu tego błędu system operacyjny automatycznie zamyka zbiór.

130. Użyto określenia nieistniejącego urządzenia dla operacji wejścia/wyjścia albo nie nazwano urządzenia podając tylko nazwę zbioru.

131. Nie można czytać ze zbioru otwartego do zapisywania. Zbiór można otworzyć jednocześnie dla czytania i zapisywania.

132. Niewłaściwy kod instrukcji XIO albo IOCB.

133. Nie zastała użyta instrukcja OPEN dla danego zbioru lub urządzenia.

134. Użyto niewłaściwego indeksu dla instrukcji IOCB.
Dla BASIC'a właściwy jest zakres 1 - 7.
ASSEMBLER wymaga użycia wielokrotności 16 mniejszej od 128.

135. Błąd analogiczny do 131. Nie można zapisywać w zbiorze otwartym tylko do czytania.

136. Osiągnięto koniec zbioru. Nie ma więcej danych.

137. Próbowano czytać rekord większy od dozwolonego, albo użyto instrukcji INPUT do czytania zbioru utworzonego instrukcją PUT.

13ft. Skończył się limit czasu przeznaczonego dla urządzenia zewnętrznego. Powody takiej sytuacji:
- złe określenie numeru urządzenia,
- nic ma takiego urządzenia,
- urządzenie nie jest właściwie połączone, - urządzenie nie jest włączone do sieci.
W przypadku magnetofonu: źle jest ustawiona taśma, taśma ma wady, prędkość obrotów jest niewłaściwa.
W przypadku stosowania interface'u:
- sygnał nie jest dostatecznie dopasowany.

139. Niesprawne urządzenie zewnętrzne albo niewłaściwie podłączone. Błąd w parametrach sterujących urządzeniem albo niewłaściwa instrukcja sterująca.

140. Bardzo rzadki błąd. Uszkodzona komunikacja komputera z peryferiami. Jeżeli po sprawdzeniu połączeń i instrukcji sterujących błąd pojawia się ponownie może to świadczyć o niesprawności komputera albo urządzenia zewnętrznego. Błąd ten może niekiedy wystąpić w wypadku czytania uszkodzonej taśmy lub dyskietki. "testowanie systemu wg zaleceń jak w ICH.

141. Parametry położenia kursora na ekranie ustalono niewłaściwie w danej grafice. Należy zmienić parametry.

14?. Uszkodzenie kanału transmisji szeregowej.
Jeżeli po sprawdzeniu wg 138 błąd ten pojawi się powtórnie świadczy to o uszkodzeniu któregoś z urządzeń, uszkodzeniu taśmy lub dyskietki.

143. Nie zgadza się suma kontrolna podczas transmisji danych. Nie można jednoznacznie stwierdzić gdzie tkwi błąd - w programie czy w sprzęcie, czy też w uszkodzonym nośniku.

144. Urządzenie zewnętrzne nie jest w stanie wypełnić czynności podanych w instrukcji np. zapisać programu na zabezpieczonej przed zapisywaniem dyskietce.

145. Stacja dysków wskazuje na różnice pomiędzy informacją otrzymaną a informacjo zapisaną właśnie na dyskietce, albo niewłaściwie użyto instrukcji dotyczących grafiki w danym trybie graficznym.

146. Użyto niewłaściwej instrukcji dla danego urządzenia np. PUT dla klawiatury.

147. Zajęta została cała pamięć RAM (podobnie jak w przypadku błędu 2 BASIC). Błąd ten może wystąpić np. przy zmianie trybu grafiki.

1511. Próba otwarcia portu szeregowego uprzednio otwartego.

151. Brak zezwolenia na współbieżny tryb pracy portu we/wy. Np. port przed wykonaniem XIO 40 należy wcześniej otworzyć dla trybu współbieżnego.

152. Zła długość bufora i jego adres został użyty przy operacji we/wy w trybie współbieżnym.

153. Program chce podłączyć wejście lub wyjście do portu szeregowego podczas gdy inny port szeregowy został otwarty i uaktywniony w trybie współbieżnym.

154. Próba wykonania operacji we/wy przez port szeregowy, który wymaga uaktywnienia go w trybie współbieżnym.

160. Zły numer stacji dysków. Prawidłowy 1-4.

161. Zbyt dużo otwartych zbiorów. Tylko 4 mogą być otwarte równocześnie.

162. Nie ma więcej miejsca na dyskietce.

163. Przekłamanie w programie DOS-a.

164. Parametry instrukcji POINT są źle dobrane dla otwartego w danej chwili zbioru. Nie ma takiego sektora w otwartym zbiorze.

165. Użyto niewłaściwego znaku w nazwie zbioru czy programu.
166. Nie ma takiego bajta w określonym sektorze jaki występuje w instrukcji POINT**.**
167. Próba zapisu na zabezpieczonym zbiorze.
168. Niewłaściwa instrukcja sterująca urządzeniem zewnętrznym.
169. Maksymalnie można zapisać na dyskietce 64 zbiory i liczba ta została już przekroczona.
170. Nic ma takiego zbioru na dyskietce. Należy sprawdzić właściwą nazwę.
171. Zbiór został otwarty albo złe parametry instrukcji POINT..
172. Nie ma dostępu do zbiorów DOS1 za pomocą DOS2 i odwrotnie. Należy najpierw przeprowadzić konwersję.
173. Zły sektor na dyskietce. Należy sformatować dyskietkę ponownie, gdy nie daje to rezultatu należy zmienić dyskietkę na inną. Błąd ten może także wystąpić przy próbie czytania programów fabrycznie zabezpieczonych przed skopiowaniem.
174. Ponownie użyto tej samej nazwy zbioru np. przy instrukcji RENAME. Należy zmienić nazwę
175.Tego zbioru nie można ładować instrukcją LOAD z DOS-a.
176. Analogicznie jak 172 - dotyczy DOS2 i DOS3.
177. Uszkodzona dyskietka. Przy użyciu DOS3 może wystąpić jeżeli dyskietka nie jest zapisana w DOS-ie 3.

BOOT ERROR - błąd powstający przy uruchamianiu systemu gdy:
- źle ustawiono taśmę (nie na początku programu),
- nie włożono dyskietki, -źle włożono dyskietkę,
- nie domknięto stacji dysków,
- uszkodzony jest magnetofon albo stacja dysków.

Dodatek B

Kody ATASCII i PEEK / POKE

ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE
♥	0	64	→	31	-	<	62	30
▙	1	65	SPACJA	32	0	?	63	31
▌	2	66	!	33	1	@	64	32
▟	3	67	″	34	2	A	65	33
▖	4	68	#	35	3	B	66	34
▜	5	69	$	36	4	C	67	35
▗	6	70	%	37	5	D	68	36
▝	7	71	&	38	6	E	69	37
◢	8	72	`	39	7	F	70	38
▪	9	73	(	40	8	G	71	39
◣	10	74	)	41	9	H	72	40
▪	11	75	*	42	10	I	73	41
▪	12	76	+	43	11	J	74	42
▬	13	77	,	44	12	K	75	43
▂	14	78	-	45	13	L	76	44
▪	15	79	.	46	14	M	77	45
♣	16	80	/	47	15	N	78	46
▛	17	81	0	48	16	O	79	47
▬	18	82	1	49	17	P	80	48
✚	19	83	2	50	18	Q	81	49
●	20	84	3	51	19	R	82	50
▂	21	85	4	52	20	S	83	51
▌	22	86	5	53	21	T	84	52
▀	23	87	6	54	22	U	85	53
▟	24	88	7	55	23	V	86	54
▊	25	89	8	56	24	W	87	55
▙	26	90	9	57	25	X	88	56
▒	27	-	:	58	26	Y	89	57
▲	28	-	;	59	27	Z	90	58
▼	29	-	<	60	28	[	91	59
◀	30	-	=	61	29	\	92	60

ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE
]	93	61		128	-	#	163	-
^	94	62		129	-	$	164	-
_	95	63		130	-	%	165	-
	96	96		131	-	&	166	-
a	97	97		132	-	‘	167	-
b	98	98		133	-	(	168	-
c	99	99		134	-	)	169	-
d	100	100		135	-	*	170	-
e	101	101		136	-	+	171	-
f	102	102		137	-	,	172	-
g	103	103		138	-	-	173	-
h	104	104		139	-	.	174	-
i	105	105		140	-	/	175	-
j	106	106		141	-	0	176	-
k	107	107		142	-	1	177	-
l	108	108		143	-	2	178	-
m	109	109		144	-	3	179	-
n	110	110		145	-	4	180	-
o	111	111		146	-	5	181	-
p	112	112		147	-	6	182	-
q	113	113		148	-	7	183	-
r	114	114		149	-	8	184	-
s	115	115		150	-	9	185	-
t	116	116		151	-	:	186	-
u	117	117		152	-	;	187	-
v	118	118		153	-	<	188	-
w	119	119		154	-	=	189	-
x	120	120	RETURN	155	-	>	190	-
y	121	121		156	-	?	191	-
z	122	122		157	-	@	192	-
	123	123		158	-	A	193	-
	124	124		159	-	B	194	-
	125	-	SPACJA	160	-	C	195	-
	126	-	!	161	-	D	196	-
	127	-	„	162	-	E	197	-

ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE	ZNAK	KOD ATASCII	PEEK POKE
F	198	-	Z	218	-	n	238	-
G	199	-	[	219	-	o	239	-
H	200	-	\	220	-	p	240	-
I	201	-	]	221	-	q	241	-
J	202	-	^	222	-	r	242	-
K	203	-	_	223	-	s	243	-
L	204	-	◆	224	-	t	244	-
M	205	-	a	225	-	u	245	-
N	206	-	b	226	-	v	246	-
O	207	-	c	227	-	w	257	-
P	208	-	d	228	-	x	248	-
Q	209	-	e	229	-	y	249	-
R	210	-	f	230	-	z	250	-
S	211	-	g	231	-	♠	251	-
T	212	-	h	232	-	▌	252	-
U	213	-	i	233	-	↖	253	-
V	214	-	j	234	-	◀	254	-
W	215	-	k	235	-	▶	255	-
X	216	-	l	236	-			
Y	217	-	m	237	-			

Dodatek C

KODY KLAWISZY (PEEK (764))

KLAWISZ	KOD	KLAWISZ	KOD	KLAWISZ	KOD
L	0	4	24	W	46
J	1	3	26	Q	47
;	2	6	27	9	48
K	5	ESC	28	0	50
+	6	5	29	7	51
*	7	2	30	DELETE	52
O	8	1	31	8	53
P	10	,	32	<	54
U	11	.	34	>	55
RETURN	12	N	35	F	56
I	13	M	37	H	57
–	14	/	38	D	58
=	15	◩	39	CAPS	60
V	16	R	40	G	61
C	18	E	42	S	62
B	21	Y	43	A	63
X	22	TAB	44		
Z	23	T	45		

Podane w tabeli kody pojawiają się w komórce pamięci o adresie 764 (dec). W przypadku naciśnięcia kombinacji klawiszy do podanych wyżej wartości należy dodać:

Kombinacja SHIFT + Klawisz +64
Kombinacja CONTROL + Klawisz +128

Dodatek D

KOMPENDIUM JĘZYKA BASIC

Formaty instrukcji zapisano w jednolity sposób, stosując przy tym ogólnie przyjęte symbole:
{ } oznacza elementy, z których jeden musi być użyty,
[ ] oznacza elementy, które mogą być użyte dodatkowo (opcje),
... oznacza, że elementy występujące przed tym znakiem mogą być powtórzone, " (): muszą być użyte tak jak w formacie,
DUŻE LITERY muszą być napisane tam gdzie zastosowane są w formacie,
kursywa służy jedynie do zapisania nazw elementów formatu, nie występuje w rzeczywistych instrukcjach i funkcjach.

Zbiór nazw elementów używanych w formatach, został ograniczony a same nazwy często są skrótami. W takich przypadkach nie użyto kropki na końcu skrótu gdyż w języku BASIC jest ona zarezerwowana do innych celów.

Niektóre nazwy mają szerokie znaczenie, na przykład *wyrażenie* oznacza *wyrażenie numeryczne* lub logiczne a także zmienną lub stałą numeryczną lub funkcję. Nazwy, w których występuje słowo *numer* oznaczają ogólnie wyrażenia *numeryczne*, ich wartości spełniają określone role.

Nazwa *opis zbioru* oznacza stałą lub zmienną tekstową: "C:", "E:", "K:", "R[n]:", "S:", *"D[n]:nazwa[identyfikator]"*

ABS

Funkcja, która przyporządkowuje liczbie jej w2rtoSC.bezwzględną.

Format: *ABS(wyrażenie numeryczne)*
Przykład: A=ABS(B) ADR

Funkcja przyporządkowująca zmiennej lub stałej tekstowej adres jej pierwszego znaku, wyrażony dziesiętne.

F: ADR(*zmienna lub stała tekstowa)*
P: A=ADR(B$) A=ADR("STRING")

AND

Operator logiczny i

ASC

Funkcja przyporządkowująca pierwszemu znakowi tekstu numer jego kodu ATASCII.

F: ASC*(zmienna lub szata tekstowa)*
P: A=ASC ("COMPUTER")
B =ASC(A$)

ATN

Przyporządkowuje liczbie (w radianach) wartość jej arcusa tangensa.

F: ATN *(wyrażenie numeryczne)*
P: A=ATN(B)

BYE lub **B.**

Powoduje przejście komputera z BASIC'a do procedury testującej (SELF TEST). Całkowicie wymazuje pamięć komputera.

F: BYE
P: BYE

CLOAD

Instrukcja, która ładuje program z kasety do komputera.

F: CLOAD
P: CLOAD

CHR$

Przyporządkowuje liczbie odpowiadający jej znak wg kodu ATASCII.

F: CHR$*(wyrażenie numeryczne)*
P: A$=CHR$(65)

CLOG

Przyporządkowuje liczbie wartość jej logarytmu dziesiętnego.

F: CLOG*(wyrażenie numeryczne)*
P: A=CLOG(B)

CLOSE lub CL.

Zamyka kanał wejścia / wyjścia.

F: CLOSE#numer *kanału*
P: CLOSE#1
CLOSE#UNITA

CLR

Zeruje wszystkie zmienne *numeryczne* i tekstowe.

F: CLR
P: CLR

COLOR lub **C.**

Określa rejestr koloru, który ma być użyty w następnej instrukcji PLOT lub DRAWTO. W trybie graficznym 1 2 i 3 określa znak, który zostanie wygenerowany w instrukcji PLOT lub DRAWTO.

F: COLOR *wyrażenie numeryczne*
P: COLOR 2

COM

Identyczna z **DIM**

CONT

Wznawia wykonywanie programu (od następnej linii) po zatrzymaniu klawiszem BREAK, instrukcją END lub STOP, o ile wznowienie jest możliwe.

F: CONT
P: CONT

COS

Przyporządkowuje liczbie (w radianach) wartość jej cosinusa.

F: COS(*wyrażenie numeryczne*)
P: COS(3.1415)

CSAVE lub **CS.**

Instrukcja, która wyprowadza program z pamięci komputera na kasetę.

F: CSAVE
P: CSAVE

DATA lub **D.**

Instrukcja tworząca listę liczb lub tekstów, które będą użyte w instrukcji READ.

F: DATA stała [,stała...]
P: DATA 5, 8, 21, 45, 11

DEG

Zamienia jednostkę argumentów funkcji trygonometrycznych na stopnie (zamiast radianów).

F: DEG
P: DEG

DIM

Rezerwuje miejsce w pamięci dla tekstów i ciągów liczb. Każdy znak tekstu ma zarezerwowany jeden bajt, element ciągu *numeryczne*go - sześć bajtów.

F: DIM { *zmienna tekstowa (wyr.num.)* / *zmienna numeryczna(wyr.num.[wyr.num.])* } [, { *zm. tekst.(wyr.num)* / *zm.num.(wyr.num[wyr.num.])* } ....]

P: DIM A$(15)
DIM B(8)
DIM C(12,12)
DIM D$(7), E(12)

DOS

Powoduje wyprowadzenie na monitor "menu" DOS(używane tylko przy współpracy ze stacją dysków).

F: DOS
P: DOS

DRAWTO lub **DR.**

Rysuje linię prostą pomiędzy punktem ostatniego położenia kursora a punktem o podanych współrzędnych.

F: DRAWTO nr kolumny, nr wiersza
P: DRAWTO X,Y

END

Zatrzymuje wykonanie programu, zamyka wszystkie kanały, wyłącza dźwięk.

F: END
P: END

ENTER lub **E.**

Instrukcja, która wprowadza program do pamięci komputera z urządzenia zewnętrznego.

F: ENTER *opis zbioru*
P: ENTER"C'
ENTER "D:MYFILE.LST "

EXP

Przyporządkowuje liczbie wartość stałej e podniesionej do potęgi o wykładniku równym tej liczbie

F: EXP*(wyrażenie numeryczne)*
P: A = EXP(B)

FOR lub **F.**

Instrukcja powtarzająca ciąg działań i kroków zawartych między FOR a NEXT

F: FOR *zm num = wyr num*1 TO *wyr num2* [STEP *wyr num3*]
P: FOR A =1 TO 256 STEP 2
FOR X = S TO 125

FRE

Podaje liczbę dostępnych wolnych bajtów pamięci komputera.

F: FRE(*dowolna liczba)*
P: A=FRE(0)

GET

Pobiera pojedynczy bajt z wyszczególnionego urządzenia i podstawia do wskazanej zmiennej.

F: GET# *nr kanału*, *zmienna*
P: GET#1,A

GOSUB lub **GOS.**

Powoduje przejście wykonywania programu do linii o wskazanym numerze, początku podprogramu.

F: GOSUB *numer linii*
P: GOSUB 45

GOTO lub **G.**

Instrukcja skoku do linii o podanym numerze.

F: GOTO *numer linii*
P: GOTO1255

GRAPHICS lub **GR.**

Przygotowuje komputer do prac w trybie graficznym, może dodatkowo kasować zawartość ekranu.

F: GRAPHICS *wyrażenie numeryczne*
P: GRAPHICS7
GRAPHICS 8 + 16

Tryby graficzne o numerach od 0 do 15 pozostawiają na dole ekranu okienko tekstowe, o numerach powiększonych o 16 rezerwują cały ekran, a o numerach powiększonych o 32 nie powodują wymazania poprzedniej zawartości ekranu.

IF ...THEN

Powoduje wykonanie instrukcji występującej po THEN jeżeli *wyrażenie* umieszczone po IF jest prawdą, w przeciwnym wypadku wykonywana jest następna linia programu.

F: IF *wyrażenie* THEN instrukcja *[:instrukcja ...]* IF *wyrażenie* THEN *numer* linii
P: IF A > B THEN PRINT "C"
IF C=DELTA THEN 545

INPUT lub **I.**

Instrukcja umożliwiająca wprowadzenie danych i przyporządkowanie ich podanym zmiennym.

F: INPUT *[#numer kanału {,' ,}]zmienna[,zmienna...]*
P: INPUT A
INPUT A,B,C
INPUT A$
INPUT # 1,A$,B

Jeżeli nie podamy numeru kanału komputer automatycznie wybiera klawiaturę jako urządzenie wejścia.

INT

Przyporządkowuje liczbie największą liczbę całkowitą nie większą od niej samej.

F: INT*(wyrażenie numeryczne)*
P: A=INT(B-C)

LEN

Przyporządkowuje zmiennej lub stałej tekstowej liczbę równą ilości jej znaków (długość tekstu).

F: LEN*(zmienna lub stała tekstowa)*
P: A=LEN(A$)

LET

Nadaje wartość zmiennej. Słowo kluczowe LET może być opuszczone.

F: LET *zmienna = wyrażenie*
P: LET A=B
C = 1985
A$ = "ATARI"

LIST lub **L.**

Instrukcja służąca do wyprowadzania części lub całości programu z pamięci komputera na ekran monitora lub wskazane urządzenie zewnętrzne.

F: LIST *opis zbioru [,numer linii t, numer linii]]*
LIST *[nr linii 1(, nrlinii2JJ*

LIST	cały program na ekran
LIST 15	linia 15 na ekran
LIST 115,245	od linii 115 do 245 na ekran
LIST "P:"	cały program na drukarkę
LIST "P:",12,158	od linii 12 do 158 na drukarkę
LIST "D2:MYFILE.LST'	cały program na dysk 2
LIST'D:MYFILE.LST',15,58	od linii 15 do 58 na dysk 1
LIST'C:"	na kasetę

LOAD lub **LO.**

Instrukcja służąca do ładowania programu z urządzenia zewnętrznego do pamięci.

F: LOAD *opis zbioru*
P: LOAD "C:"
LOAD "D:PROGRAMl"

LOCATE lub **LOC.**

Odczytuje kod znaku lub punktu graficznego o podanych współrzędnych i nadaje jego wartość wskazanej zmiennej *numerycznej*, (po uprzednim użyciu instrukcji OPEN dla S: i otwarciu IOCB #6 dla czytania lub dla czytania i pisania).

F: LOCATE *nr kolumny, numer wiersza zmienna numeryczna*
P: LOCATE 5, 12, PIXEL

LOG

Oblicza wartość logarytmu naturalnego z podanej liczby.

F: LOG (*wyrażenie numeryczne)*
P: M=LOG(A-B)

LPRINT lub **LP.**

Instrukcja drukowania na drukarce.

F: LPRINT [*wyrażenie*][{ *j} ... [wyrażenie]] ...*
P: LPRINT A,B
LPRINT AS;B,"ATAKI"

NEW

Kasuje program i wszystkie zmienne w pamięci komputera

F: NEW
P: NEW

NEXT lub **N.**

Instrukcja, która kończy pętlę programu rozpoczętą przez FOR

F: NEXT *zmienna numeryczna*
P: NEXT D

NOT

Operator logiczny nie

NOTE lub **NO.**

Instrukcja określająca wartość chwilowego znacznika zbioru na dysku.

F: NOTE# *numer kanału, zmienna num, zmienna num*
P: NOTE #1, SEC, BY'TE

ON

Instrukcja skoku uzależnionego od wartości wyrażenia.

F: ON *wyrażenie num* GOSUB *numer linii[,numer linii...]*
ON *wyr.num* GOTO *numer linii[,numer linii...]*
P: ON A GOSUB 12,315,55
ON D-6 GOTO 24,55,125

OPEN lub **O.**

Instrukcja otwarcia kanału komunikacji o podanym numerze do operacji wejścia / wyjścia dla urządzenia o określonym symbolu.

F: OPEN *numer kanału, kod operacji, wyrażenie pomocnicze, opis zbioru*
P: OPEN #1,4,0,"K:"
OPEN #2,8,0,"P:"

Dalszy opis parametrów zamieszczono w rozdziale 3.

OR

Operator logiczny lub

PADDLE

Funkcja, przyporządkowuje liczbę położeniu wskazanego manipulatora analogowego (potencjometru).

F: PADDLE *(wyrażenie numeryczne)*
P: A=PADDLE(3)

PEEK

Funkcja umożliwiająca odczytanie zawartości danej komórki pamięci. Przyporządkowująca numerowi komórki pamięci wartość dziesiętną jej zawartości.

F: PEEK(adres)
P: P=PEEK(564)

PLOT lub **PL.**

Umieszcza znak lub punkt graficzny na ekranie w miejscu o podanych współrzędnych.

F: PLOT nr *kolumny, nr wiersza*
P: PLOT X,Y

POINT lub **P.**

Określa wartość chwilowego znacznika zbioru na dysku przy odczytywaniu rekordów w trybie bezpośredniego dostępu.

F: POINT nr *kanału, zm num. zm num.*
P: POINT#5, SCTR, B

POKE

Instrukcja umieszczająca podaną liczbę we wskazanej komórce pamięci.

F: POKE *adres, wyrażenie numeryczne*
P: POKE 715,1

POP

Instrukcja, która przy wykonywaniu FOR, GOSUB lub ON ...GOTO powoduje zignorowanie adresu powrotu.

F: POP
P: POP

POSITION lub **POS.**

Instrukcja określająca pozycję na ekranie wyprowadzanej informacji.

F: POSTITON *nr kolumny, nr wiersza*
P: POSTI'ION 15,8

PRINT PR. lub **?**

Wyprowadza informację na ekran lub inne, wskazane urządzenie.

F: PRINT [{ *wyrażenie* / *# numer kanału* }] [{ ; / , } .... [*wyrażenie*]]

P: PRINT "ATARI",C
?#1;A$

PTRIG

Funkcja przybierająca wartość 0 gdy jest naciśnięty przycisk wskazanego manipulatora analogowego, a wartość 1 gdy nie jest on naciśnięty.

F: PTRIG(*wyrażenie numeryczne)*

P: A=PTRIG(1)

PUT

Instrukcja służąca do przesyłania pojedynczego bajta do otwartego uprzednio kanału wyjścia.

F: PUT # *nr* kanału, *wyrażenie numeryczne*

P: PUT# 1,A

RAD

Instrukcja powodująca, że wartość argumentów funkcji trygonometrycznych będzie odczytywana w radianach.

F: RAD

P: RAD

READ

Instrukcja odczytania kolejnej luby lub tekstu umieszczonych po instrukcji DATA.

F: READ *zmienna num lub tekstowa[,zm num lub tekstowa..]*

P: READ A,B$,C

REM RE. R. lub **.**

Instrukcja, która powoduje, że wszystkie znaki umieszczone po niej w linii są komentarzem (nie są interpretowane przez komputer).

F: REM *komentarz*

P: REM Podprogram - obliczenie A

RESTORE lub **RES.**

Instrukcja powodująca powrót do pierwszej lub wskazanej instrukcji DATA.

F: RESTORE [*nr linii*]

P: RESTORE

RETURN lub **RET.**

Instrukcja, która powoduje powrót wykonywania programu do linii następnej po ostatnio wykonywanej instrukcji GOSUB lub ON ... GOSUB.

F: RETURN

P: RETURN

RND

Funkcja generująca liczbę pseudolosową z przedziału < 0,1).

F: RND*(dowolna liczba)*

P: A=RND(1)

RUN lub **RU.**

Instrukcja rozpoczynająca wykonanie programu BASIC, może ładować do pamięci program z urządzenia zewnętrznego.

F: RUN *[opis zbioru]*

P: RUN
RUN "C:"
RUN "D2:BUDGET.BAS"

SAVE lub **S.**

Wyprowadza program BASIC z pamięci komputera do urządzenia zewnętrznego.

F: SAVE *opis zbioru*

P: SAVE "C:"
SAVE "D:PROGRAMI"
S. PRGMS

SETCOLOR lub **SE.**

Instrukcja, która przypisuje kolor i luminancję (jasność) wskazanemu rejestrowi koloru.

F: SETCOLOR *nr rejestru, kod koloru, luminancja*
P: SETCOLOR 4, 1, 0

SGN

Funkcja przyporządkowująca liczbie 1, gdy jest ona dodatnia, -1 gdy jest ujemna i 0 jeśli jest równa 0.

F: SGN(*wyrażenie numeryczne*)
P: A=SGN(B)

SIN

Oblicza sinus podanej liczby.

F: SIN(*wyrażenie numeryczne)*
P: A = SIN(B)

SOUND lub **SO.**

Włącza (lub wyłącza) wskazany generator dźwięku a także określa częstotliwość, brzmienie i głośność.

F: SOUND *nr generatora, okres, brzmienie, głośność*
P: SOUND 2,112,12,15

SQR

Oblicza wartość pierwiastka kwadratowego z podanej liczby.

F: SQR(*wyrażenie numeryczne)*
P: A=SQR(C)

STATUS lub **ST.**

Nadaje podanej zmiennej wartość kodu reprezentującego stan wskazanego kanału po ostanie] operacji wejścia lub wyjścia.

F: STATUS# *nr kanału, zmienna numeryczna*
P: STATUS# 1, A

STICK

Funkcja, której wartość podaje aktualne położenie wskazanego joysticka.

F: STICK(*wyrażenie numeryczne)*
P: A=STICK(B)

STRIG

Funkcja przybiera wartość 0 gdy jest naciśnięty przycisk wskazanego joysticka, a wartość 1 gdy nie jest naci3nięty.

F: STRIG(*wyrażenie numeryczne*)
P: A=STRIG(1)

STOP

Instrukcja zatrzymująca wykonywanie programu, nie zamyka kanałów, nie wyłącza dźwięku.

F: STOP
P: STOP

STR$

Funkcja, która zamienia liczbę w ciąg znaków (tekst).

F: STRS(*wyrażenie numeryczne)*
P: AS=STRS(243.5)

TRAP TR. lub **T.**

Instrukcja, która w przypadku wystąpienia błędu w trakcie wykonywania programu powoduje skok do linii o wskazanym numeru.

F: TRAP *nr linii*
P: TRAP 2155

Przy pomocy PEEK(195) można uzyskać nr błędu, a przy pomocy 256*PEEK(187) +PEEK(186) numer linii, w której ten błąd wystąpił.

USR

Przekazanie sterowania przebiegiem programu z BASIC'a do procedury napisanej w języku maszynowym.

F: USR (*adres*[,*wyrażenie numeryczne ...*])
P: A=USR(BCD)

VAL

Funkcja, która zamienia ciąg numeryczny(tekst) na liczbę.

F: VAL(*tekst*)
P: A= VAL ("123")

XIO lub **X.**

Ogólna instrukcja wejścia/wyjścia.

F: XIO *kod operacji, nr kanału, wyr.num , wyr. num ,opis zbioru*
P: XIO 18, #6, 0, 0, "S:"

Dodatek E

Instrukcje XIO

Rozszerzeniem instrukcji we/wy są instrukcje XIO. Wiele z nich związanych jest z operacjami ze zbiorami na dyskietkach i odpowiada bezpośrednio dyrektywom dyskowego systemu operacyjnego.

XIO *nr dyrektywy, nr kanału, auxl, aux2, opis zbioru*
nr dyrektywy -liczba całkowita określająca operacje we/wy - patrz tabela E1
nr kanału –numer kanału we/wy
auxl, aux2 - zmienne, których użycie jest dokładnie opisane w tabelach.
ops zbioru - określa urządzenie i nazwę zbioru – patrz tabela

Przykład: XIO 18, #6, 0, 0, ”S: ”
XIO 32, #1, 0, 0, ”D1: nowy, stary”

*TABELA **E 1.** Komendy XIO*

Działanie	Numer dyrektywy	Odpowiednik BASIC'a	numexpr1	numexpr2
Ogólne:				
Otwórz kanał	3	OPEN	Tabela E12	Tabela E13
Czytaj linię	5	INPUT	0	0
Pobierz znak	7	GET	0	0
Zapisz linię	9	PRINT	0	0
Przekaż znak	11	PUT	0	0
Zamknij kanał	12	CLOSE	0	0
Status kanału	13	STATUS	0	0
Operacje graficzne:				
Rysuj linię [1]	17	DRAWTO	0	0
Zamaluj obszar [2]	18	-	0	0
Operacje dyskowe [3]				
Zmień nazwę pliku [4]	32	DOS Menu	0	0
Skasuj plik [5]	33	DOS Menu	0	0
Zabezpiecz plik [5]	35	DOS Menu	0	0
Odbezpiecz plik [5]	36	DOS Menu	0	0
Przenieś znacznik [5,6]	37	POINT	0	0
Szukaj znacznika pliku [5,6]	38	NOTE	0	0
Formatuj aktualny dysk [5]	254	DOS Menu	0	0
RS-232 port szeregowy [7]				
Wyślij blok danych	32	-	0	0
Sterowanie DTR, RTS, XMT	34	-	Tabela E2	0
Parametry transmisji	36	-	Tabela E3	Tabela E3
Tryb translacyjny	38	-	Tabela E4	kod ATASCII
Tryb współbieżny	40	-	0	0

[1] Przed użyciem XIO17 ustaw kursor na początku linii instrukcją POSITION.

[2] Przed użyciem XIO18 ustaw kolor (POKE 765,...) i narysuj pionowe granice zamalowywanego obszaru.

[3] DOS musi być załadowany do pamięci.

[4] W XIO 32 parametr *opis zbioru* określa nową i starą nazwę pliku.

[5] Parametr *opis zbioru* określa nazwę kasowanego pliku, a nie plik otwarty w danym kanale.

[6] Niedostępne w DOS 1.0.

[7] Program obsługi urządzenia RS-232 musi być załadowany do pamięci.

TABELA **E 2** Parametry ***aux1*** dla XIO 34

Zsumuj wartości z poszczególnych kolumn aby otrzymać wartość ***aux1***				Możliwe wartości ***aux1***			
	DTR	RTS	XMT	DTR	RTS	XMT	Wartość
Bez zmian	0	0	0	Wył.	Wył.	0	162
Wyłączenie	128	32	2	Wył.	Wył.	1	163
(XMT→ 0)				Wył.	Zał.	0	178
Załączenie	192	48	3	Wył.	Zał	1	179
(XMT→ 1)				Zał.	Wył.	0	226
				Zał.	Wył.	1	227
				Zał.	Zał.	0	242
				Zał.	Zał	1	243

TABELA **E 3** Parametry ***aux1*** **i** ***aux2*** dla XIO 36

Zsumuj wartości z poszczególnych kolumn aby otrzymać wartość ***aux1***						Wartość ***aux2***			
Bit stopu	Wartość	Długość słowa	Wartość	Szybkość transmisji (baud rate)	Wartość	DSR	CTS	CRX	Wartość
1	0	8 bit	0	300	0	Nie	Nie	Nie	0
2	128	7 bit	16	45,5	1	Nie	Nie	Tak	1
		6 bit	32	50	2	Nie	Tak	Nie	2
		5 bit	48	56,875	3	Nie	Tak	Tak	3
				75	4	Tak	Nie	Nie	4
				110	5	Tak	Nie	Tak	5
				134,5	6	Tak	Tak	Nie	6
				150	7	Tak	Tak	Tak	7
				300	8				
				600	9				
				1200	10				
				1800	11				
				2400	12				
				4800	13				
				9600	14				
				19200	15				

TABELA **E 4** Parametry ***aux1*** dla XIO 38

Zsumuj po jednej wartości z każdej kolumny aby otrzymać wartość ***aux1***							
Wysuw linii (LF)		Translacja ATASCII-ASCII		Wejście parzystości		Wyjście parzystości	
Dodany?	Wartość	Tryb	Wartość	Tryb	Wartość	Tryb	Wartość
Nie Tak *	0 64	Słaba Mocna Brak	0 16 32	Pomiń+ Nieparzysty+ Parzysty +	0 4 8	Bez zmian Nieparzysty Parzysty Bit włączony	0 1 2 12

* Wysuw o 1 linię po powrocie karetki (EOL w ATASCII)
+ Gdy kontrola parzystości zostanie wykryta – bit parzystości zostaje wyzerowany.

Parametry instrukcji **OPEN**
Format: OPEN *#nr kanału, kod operacji, wyr. pomocn., opis zbioru*
Instrukcja została opisana w Rozdz. 8, tu opisane zostaną znaczenia poszczególnych zmiennych.

TABELA **E 5**. *Kody operacji* instrukcji OPEN

Urządzenie	Kod operacji	Opis działania			
Magnetofon (C:)	4	Odczyt			
	8	Zapis			
Stacja Dysków	4	Odczyt			
(D[n]:plik.ext)	6	Odczyt katalogu dyskietki			
	8	Zapis (nowego pliku)			
	9	Zapis (dopisywanie)			
	12	Odczyt i zapis (modyfikacja pliku)			
Edytor ekranowy	8	Wyjście na ekran			
(E:)	12	Wejście z klawiatury i wyjście na ekran			
	13	Wejście i wyjście na ekran			
Klawiatura (K:)	4	Odczyt klawiatury			
Drukarka (P:)	8	Zapis na drukarkę			
Port szeregowy	5	Odczyt współbieżny			
(R[n]:)	8	Zapis bloku			
	9	Zapis współbieżny			
	13	Zapis i odczyt współbiezny			
		Kasowanie zawartosci	Okno tekstowe +	Odczyt	Zapis
Ekran (S:)	8	Tak	Nie	Nie	Tak
	12	Tak	Nie	Tak	Tak
	24	Tak	Tak	Nie	Tak
	28	Tak	Tak	Tak	Tak
	40	Nie*	Nie	Nie	Tak
	44	Nie*	Nie	Tak	Tak
	56	Nie*	Tak	Nie	Tak
	60	Nie*	Tak	Tak	Tak

* Ekran jest zawsze kasowany w trybie graficznym 0.
+ Brak jest oddzielnego okna tekstowego w trybie graficznym 0.

TABELA **E 6**. Wartości *wyrażenia pomocniczego* instrukcji OPEN

Urządzenie	Opis funcji	Wartość
Magnetofon (C:)	Normalne przerwy międzyrekordowe	0
	Krótkie przerwy międzyrekordowe	128
Stacja dysków (D[n]:plik.ext)	Ignorowany	0
Edytor ekranowy (E:)	Ignorowany	0
Klawiatura (K:)	Ignorowany	0
Drukarka (P:)	Normalne znaki	0
	Boczna kolumna znaków (ATARI 820)	83
Port szeregowy (R[n]:)	Ignorowany	0
Ekran (S:)	BASIC tryb graficzny 0	0
	BASIC tryb graficzny 1	1
	BASIC tryb graficzny 2	2
	BASIC tryb graficzny 3	3
	BASIC tryb graficzny 4	4
	BASIC tryb graficzny 5	5
	BASIC tryb graficzny 6	6
	BASIC tryb graficzny 7	7
	BASIC tryb graficzny 8	8

TABELA **E 7**. Nazwy *opisu zbioru* instrukcji OPEN

Urządzenie	Opis zbioru
Magnetofon	C:
Stacja dysków	D[n]:plik[.ext]
Edytor ekranowy	E:
Klawiatura	K:
Drukarka	P:
RS-232 port szeregowy	R[n]:
Ekran	S:

ANTIC i Lista Displejowa

ANTIC jest procesorem graficznym, i jak każdy procesor ma też swoje rozkazy. Można więc pisać programy dla ANTICa tak, jak dla każdego innego procesora. Program dla ANTICa ma swoją nazwę: Lista Displejowa (ang. Display List). To, co widać na ekranie, to wynik realizacji Listy Displejowej przez ANTICa

Zestaw instrukcji ANTICa pokazany jest w tablicy E 7.

TABELA **E 8** Instrukcje Listy Displejowej.

	Kod instrukcji		Tryby graficzne BASIC'a	Ilość pikseli w poziomie	Bajtów na linię	Użytych linii skaningo-wych	Bitów na piksel	Dane (D) maski wyboru koloru (C)
	Dec	Hex						
Wyjście pustych linii skaningo-wych	0	00	-	-	-	1	-	
	16	10	-	-	-	2	-	
	32	20	-	-	-	3	-	
	48	30	-	-	-	4	-	
	64	40	-	-	-	5	-	
	80	50	-	-	-	6	-	
	96	60	-	-	-	7	-	
	112	70	-	-	-	8	-	
Tryby tekstowe	2	02	0	40	40	8	8	DDDDDDDD *
	3	03	-	40	40	10	8	
	4	04	-	40	40	8	8	CCCCCCCC
	5	05	-	40	40	16	8	
	6	06	1	20	20	8	8	CCDDDDDD *
	7	07	2	20	20	16	8	
Tryby graficzne	8	08	3	40	10	8	2	CC +
	9	09	4	80	10	4	1	C=0 -tło, 1- rej. 1
	10	0A	5	80	20	4	2	CC + -patrz kod 6,7,8
	11	0B	66	160	20	2	1	patrz kod 9
	12	0C	-	160	20	1	1	
	13	0D	7	160	40	2	2	CC + -patrz kod 6,7,8
	14	0E	-	160	40	8	2	
	15	0F	8	320	40	1	1	D *

* D=0 – rejestr 2
D=1 – rejestr 1

\+ CC=00 – rejestr 0
CC=01 – rejestr 1
CC=10 – rejestr 2
CC=11 – rejestr 3

TABELA **E 9**. Dostępne numery rejestrów koloru

SETCOLOR numer rejestru	COLOR *r* (wartość w trybach graficznych). Patrz też Rozdz. 5. *		
	3, 5, 7	4, 6	8 **
0	1	1	-
1	2	-	1
2	3	-	0
3	-	-	-
4	0	0	-

* W trybie 0, 1 i 2 kolor znaku jest określony przez jego rodzaj (duże, małe litery i ich inwersja).

** W trybie 8 *r* określa luminancję.Rejestr 2 zawsze określa odcień.

TABELA **E 10**. Domyślne wartości rejestru koloru (SETCOLOR X, .., .).

Numer rejestru koloru	Odcień	Jaskrawość	Kolor
0	2	8	Pomarańczowy
1	12	10	Zielony
2	9	4	Niebieski ciemny
3	4	6	Czerwony
4	0	0	Czarny

Grafika Gracz – Pocisk (Player Missile)

TABELA **E 11**. Kolory i odpowiadające im wartości tła, gracza i pocisku.

Wartość koloru *		
Kolor	Dec	Hex
Szary	0	0
Złoty	16	10
Pomarańczowy	32	20
Czerwony	48	30
Różowy	64	40
Fioletowy	80	50
Purpurowy	96	60
Niebieski	112	70
Niebieski	128	80
Niebieski jasny	144	90
Turkusowy	160	A0
Niebiesko-zielony	176	B0
Zielony	192	C0
Żółto-zielony	208	D0
Pomarańczowo-zielony	224	E0
Pomarańczowy jasny	240	F0

* Aby ustawić jaskrawość dodaj parzystą liczbę z zakresu 2...14.
0 – brak jaskrawości, 14 – maksimum.

TABELA **E 12**. Możliwe wartości rejestru sterowania DMA dla gracza i pocisku.

Wartość dla instrukcji POKE	Rezultat
4	Włączone DMA tylko dla pocisku
8	Włączone DMA tylko dla gracza
12	Włączone DMA dla gracza i pocisku
dodanie 16	Rozdzielczość jednej linii (domyślnie ustawiana jest rozdzielczość 2 linie)

Dźwięk

SOUND ***gen, częst, zniekszt, siła***

TABELA **E 13.** Charakterystyka zniekształceń dźwięku (*zniekszt*).

Wartość zniekształceń *	Wyciszenia **	Tony wtórne ***	Opis
14	Brak	Brak	Czysty ton
12	Wiele	Wiele	Wysokie tony nie zniekształcone
10	Brak	Brak	Czysty ton
8	Brak	Brak	Statyczne (niskie tony) do szumu białego (wysokie tony)
6	Kilka	Kilka	Brak zmian przy wartości siły głosu poniżej 200
4	Kilka	Kilka	Statyczne (niskie tony) do pulsującego (wysokie tony)
2	Kilka	Kilka	Brak zmian przy wartości siły głosu poniżej 200
0	Kilka	Brak	Zmieszane 4 i 8.

* Nieparzyste wartości zniekształceń generują stuk przy włączeniu i wyłączeniu dźwięku.
** Niektóre kombinacje zniekształceń i siły głosu dają w wyniku ciszę.
*** Niektóre kombinacje zniekształceń i siły głosu dają w wyniku tony harmoniczne.

Różne

TABELA **E 14**. Znaczenie bitów rejestru statusu / zezwolenia przerwań.

Bit	Przerwanie
0	Timer 1
1	Timer 2
2	Timer 4
3	Wyjście szeregowe (bajt) – koniec transmisji
4	Potrzebne dane do wyjścia szeregowego
5	Gotowość wejścia szeregowego do odbioru danych
6	Naciśnięty został jakiś klawisz (z wyjątkiem BREAK)
7	Niaciśnięty został klawisz BREAK

TABELA **E 15**. ROM Systemu Operacyjnego

Adres	Przeznaczenie
55296-57393	Procedury zmiennoprzecinkowe
57344-58367	Zestaw znaków
58368-58533	Wektory OS (patrz tabela E16)
58534-59092	CIO
59093-59715	Obsługa przerwań
59716-60905	SIO
60906-61047	Obsługa stacji dysków (boot)
61249-61666	Obsługa magnetofonu
61667-62435	Monitor
62436-65535	Obsługa ekranu i klawiatury

TABELA **E 16**. Wektory Systemu Operacyjnego

Adres	Przeznaczenie
58368-58383	Edytor
58384-58399	Ekran
58400-58415	Klawiatura
58416-58431	Drukarka
58432-58447	Magnetofon
58448-58495	Wektory skoków
58496-58533	Wektory inicjalizacji RAM

TABELA **E 17**. Adresy układów wejścia / wyjścia.

Adres	Typ	Przeznaczenie	Obszar
53248-53503	I/O	CTIA lub GTIA	256 bajtów
53504-53759	I/O	Wolny	256 bajtów
53760-54015	I/O	POKEY	256 bajtów
54016-54271	I/O	PIA	256 bajtów
54272-54783	I/O	ANTIC	512 bajtów
54784-55295	I/O	Wolny	512 bajtów

TABELA **E 18**. RAM używany przez Sustem Operacyjny,
kartridż lub Free RAM

Adres	Przeznaczenie
0-127	Zerostronnicowa pamięć Systemu Operacyjnego
128-255	Zerostronnicowa pamięć użytkownika
256-511	Stos procesora
512-1151	RAM Systemu Operacyjnego (Tablica E19)
1152-1791	RAM użytkownika
1792-2047	Obszar startowy użytkownika (Boot Area)

TABELA **E 19**. RAM Systemu Operacyjnego

Adres	Przeznaczenie
512-553	Wektory przerwań
554-623	Różne
624-647	Dżojstiki
648-655	Różne
656-703	RAM ekranu (w trybach graficznych)
704-711	Kolory
712-735	Wolny
736-767	Różne
768-779	DCB
780-793	Różne
794-831	Obsługa tablic adresowych
832-847	I/O Channel 0 (IOCB0)
848-863	I/O Channel 1 (IOCB1)
864-879	I/O Channet 2 (IOCB2)
880-895	I/O Channel 3 (IOCB3)
896-911	I/O Channel 4 (IOCB4)
912-927	I/O Channel 5 (IOCB5)
928-943	I/O Channel 6 (IOCB6)
944-959	I/O Channel 7 (IOCB7)
960-999	Bufor drukarki
1000-1020	Wolny
1021-1151	Bufor magnetofonu

Program zamieniający liczby dziesiętne na heksadecymalne i na odwrót.

```
1 REM PROGRAM "DEC-HEX"
10 DIM HE$(16), B$(4): HE$="0123456789ABCDEF"
15 PRINT CHR$(125)
20 PRINT: PRINT "WYBIERZ KONWERSJE": PRINT
30 PRINT " 1 - ZAMIANA DEC na HEX";: PRINT: PRINT " 2 - ZAMIANA HEX na DEC";: INPUT A
40 IF A=2 THEN 100 50 IF A<>1 THEN 20
60 PRINT:, PRINT "PODAJ LICZBE DZIESIETNA";: INPUT A: PRINT PRINT " "; A; " = " ;
70 FOR F=3 TO 0 STEP -1
80 D=INT (A/16^F): PRINT HE$ (D+1,D+1);
90 A=A-D*16^F: NEXT F
95 PRINT: GOTO 20
100 PRINT: PRINT "PODAJ LICZBE HEXSADECYMALNA";:INPUT B$:PRINT:PRINT " "; B$;" =";
110 A=0: FOR Q=1 TO LEN (B$): FOR F=1 TO 16
120 IF B$ (Q,Q)=HE$(F,F) THEN GOTO 140
130 NEXT F
140 A=A+(F-1)* 16^ (LEN (B$) - Q): NEXT Q
150 PRINT A: PRINT
160 GOTO 20
```

Program rysujący wykresy fukcji.

```
5 DIM FUNKT$(120), DISK$(120), OUT$(20): HO=319: VE=191: HOP=799: VEP=479
7 OPEN#1, 4, 0, "K:"
10 GRAPHICS 0:PR=0:? "WYKRES DOWOLNEJ FUNKCJI"
20 ? " ↓ y=f (x) "
30 ? " ↓ od Xstart do Xkon"
50 ? " ↓ y PODAJ"
60 ? " ↓ Xstart,Xkon ";: ZNPUT XSTART, XEND
80 ? " ↓ Y= ";: INPUT FUNKT$
90 RAD: ? DEG CZY RAD ";:INPUT DISK$: IF LEN (DISK$) THEN IF DISK$ (1,1) = "D" THEN
DEG
91 ? " ↓ Krok obliczen ";: INPUT KR
100 GO=103: GOSUB 1000
103 ? " "
104 ? " ↓ Poczekaj chwile - obliczam YMIN,YMAX"
105 GOSUB 5200
108 ? " " :? "Ymin , "YmdX "; YMIN;" ";YMAX
146 TRAP 150: ? "KOLOR, RAMKA, TLO, FUNKCJA";: INPUT CO, L, RAND, HELHIN, HELFUN:
GOTO 170
150 COL=14: RAND=14: HELHIN=0: HELFUN=15
170 GRAPHICS 24: POKE 752,255
180 SETCOLOR 4, RAND, B:SETCOLOR 1, 12, HELFUN:SETCOLOR 2, COL,HELHIN: COLOR 1
190 GOSUB 3000: X=XSTART: TRAP 220:ERR=0
200 Y=SIN(X"2)*COS(X"3)
205 TRAP 220
210 Y=VE-(VE-1)*(Y-YMIN)/(YMAX-YMIN): IF (Y>=0) AND (Y<=E) THEN PLOT 0, Y: ERR=1
220 H=(XEND-XSTART)/HO:FOR I=1 TO HD
230 X=X+H: TRAP 260:ERR2=ERR:ERR=0
240 Y=SIN(X^2)*COS(X^3)
245 TRAP 260
250 Y=VE-(VE-1)*(Y-YMIN)/(YMAX-YMIN): IF (Y<0) OR (Y>VE) THEN 260
```

```
251 IF ERR2=0 THEN PLOT I,Y:ERR=1: GOTO 260
252 DRAWTO I,Y:ERR=1
260 NEXT I
270 OPEN #7, 4, 0, "K:":GET #7,A:CLOSE #7
280 IF A=ASC("F") THEN GOSUB 4000:GOT0 300
281 IF A=ASC("X") THEN GOSUB 4000:GOT0 400
282 IF A=ASC("Y") THEN GOSUB 4000:GOT0 500
284 IF A=ASC("S") THEN GOTO 10
285 IF A=ASC("K") THEN END
286 IF A=ASC("M") THEN 2000
290 GOTO 270
300 ? " y=";:INPUT FUNKT$:? " "
310 GO=190:POKE 752,1: GOSUB 1000
400 ? "POZ.OX na DY,SKALA,OPIS OSI": INPUT Y,SC,NUM: TRAP 65000
405 IF NUM<>0 AND NOT TDEVICE THEN ? "CHWILECZKE...":GOSUB 32000
410 ? " ": TRAP 270:GOSUB 3000
420 PL=VE-(VE-1)*(Y-YMIN)/(YMAX-YMIN):PLOT O,PL
430 DRAWTO HO,PL
440 IF SC=O THEN 270
442 MIN=PL-3: IF MIN<0 THEN MIN=0
443 MAX=PL+3: IF MAX>VE THEN MAX=VE
444 DUI=O:DUA=0
445 IF XSTART<0 AND INT (XSTART/SC)<> XSTART/SC THEN DUI=SC
446 IF XEND<0 AND INT(XEND/SC)<> XEND/SC THEN DUA=SC
450 FOR I=INT(XSTART/SC)*SC+DUI TO INT (XEND/SC)*SC+DUA STEP SC
460 PL=(HO-1)*(I-XSTART)/(XEND-XSTART)
470 PLOT PL,MIN:DRAWTO PL,MAX
471 IF NUM=0 THEN 480
472 OUT$=STR$(I)
473 IF NUM=1 THEN YPOS=MIN-IO:XPOS=PL-4*LEN(OUT$)
 474 IF NUM=-1 THEN YPOS=MAX+2: XPOS=PL-4*LEN(OUT$)
475 IF XPOS<0 THEN XPOS=0
476 IF XPOS+LEN(OUT$)*8>HO THEN XPOS=HO-LEN(OUT$)*B
477 POSITION XPOS,YPOS: ?#3;OUT$;
480 NEXT I
490 GOTO 270
500 ? "POZ.OY NA OX, SKALA, OPIS USI":INPUT X,SC,NUM:TRAP 65000
505 IF NUM<>0 AND NOT TDEVICE THEN ? "CHWILECZKE....": GOSUB 32000
510 ? "}": TRAP 270: GOSUB 3000
520 PL=(HO-1)*(X-XSTART)/(XEND-XSTART):PLOT PL,O
530 DRAWTO PL,VE
540 IF SC=0 THEN 270
542 MIN=PL-3: IF MIN<0 THEN MIN=0
543 MAX=PL+3: IF MAX>HO THEN MAX=HO
544 DUI=O:DUA=0
545 IF YMIN<0 AND INT(YMIN/SC)<>YMIN/SC THEN DUI=SC
546 IF YMAX AND INT (YMAX/SC) THEN DUA=SC
550 FOR I=INT(YMIN/SC)*SC+DUI TO INT (YMAX/SC)+DUA STEP SC
560 PL=VE-(VE-1)*(1-YMIN)/(YMAX-YMIN)
570 PLOT MIN,PL:DRAWTO MAX, PL
571 IF NUM=0 THEN 580
572 OUT $=STR$(I)
573 IF NUM=1 THEN YPOS=PL-4:XPOS=MAX+2
574 IF NUM=-1 THEN YPOS=PL-4:XPOS-2-8*LEN(OUT$)
575 IF YPOS<0 THEN YPOS=O
576 ZF YPOS+B>VE THEN YPOS=VE-8
577 POSITION XPOS, YPOS:? #3;OUT$; 580 NEXT I
590 GOTO 270
```

```
1000 REM ZAPIS FUNKCJI NA KASETE
1005 CEL=0
1010 IF FUNKT$="" THEN 1200
1020 ? " Przewin kasete do ";CEL;" i nacisnij dow.klawisz";:GET #1,A
1030 OPEN #3,8,0,"C:"
1100 DISK$="200 Y=": DISK$(7)=FUNKT$
1120 ?#3;DISK$
1122 DISK$(1,3)="240" .
1123 ? #3;DISK$
1124 DISK$(1,4)="5120"
1125 ? #3;DISK$
1126 DISK$(1,4)="5550"
1127 ? #3;DISK$
1130 CLOSE #3
1140 POP:? " Przewin kasete do ";CEL
1145 ? "j GOTO ";GO; "t": POKE 764,12
1150 ENTER "C:"
1200 ? " Przewin kasete do ";CEL
1210 OPEN #3,4,0,"C:"
1220 INPUT #3,DISK$:FUNKT$=DISK$(7)
1230 CLOSE #3: GOTO 1140
2000 GOSUB 4500
2010 ? " MENU"
2020 ?  " ↓ F Podanie wzoru funkcji"
2030 ?  " ↓ X Opis osi X"
2040 ?  " ↓ Y Opis osi Y" 2060 ? "j S Start"
2080 ?  " ↓ K Koniec"
2081 ?  " ↓ Pozostale klawisze-wykres"
2090 GET #1,A
2110 GOSUB 3500:GOT0 280
3000 GRAPHICS 8+16+32: POKE 752,255
3100 SETCOLOR 4, RAND, 8:SETCOLOR 1,12, HELFUN:SETCOLOR 2, COL,HELHIN: COLOR 1
3200 RETURN
3500 POKE 106,PEEK(106) + 32:GOT0 3000
4000 GRAPHICS 8+32:POKE ?52,0
4100 SETCOLOR 4, RAND,8:SETCOLOR 1,12, HELFUN SETCOLOR 2, COL, HELHIN:COLOR 1
4200 RETURN
4500 POKE 106, PEEK(106)-32:GRAPHICS O:RETURN 5120 Y=SIN(X"2)*COS(X"3)
5150 Y=LOG(X"3-0.5) 5200 NW=5.0E+91:NM=-NW
5220 FOR X=XSTART TO XEND STEP KR
5550 Y=SIN(X"2)*COS(X"3)
5600 IF Y>NM THEN NM=Y
5610 IF Y<NW THEN NW=Y
5620 NEXT X
5630 YMIN=NW:YMAX=NM
5640 RETURN
32000 REM TEKST
32010 RESTORE 32100:TDVICE=1:PRUEF=0
32020 FOR I=1536 TO 1536+255
32030 READ BYTE:PRUEF=BYTE+PRUEF
32040 POKE I, BYTE
32045 NEXT I
32050 BYTE=USR(1536):POKE 765,1
32090 OPEN #3,4,0,"T:"
32095 IF PRUEF=28449 THEN RETURN
32100 DATA 104,162,0,189,26,3,240,12,201,84,240,23,232, 232,232,224
32110 DATA 36,48,240,96,169,84,157,26,3,169,36,157,27,3, 169,6
```

32120 DATA 157,28,3,96,192,6,192,6,47,6,50,6,47,6,47,6 32130 DATA 160,146,96,133,208,41,127,201,32,16,6,2,4,105, 64,76,72

```
32140 DATA 6,201,96,16,3,56,233,32,162,0,134,204,1,33, 203,134,206
32150 DATA 134,207,160,16,208,13,24,165,206,105,8,133 206,165,207,105
32160 DATA 0,133,207,70,207,102,206,102,204,102,20,3, 136,48,4,144,243
32170 DATA 176,228,24,165,204,109,244,2,133,204,16,0, 0,17?,203, 162,7
32180 DATA 10,72,165,208,144,10,201,128,176,13,32,220, 6,76,151,6
32190 DATA 201,128,144,3,32,220,6,230,85,208,2,230, 86,104,202,16
32200 DATA 223,200,192,8,16,20,72,56,165,85,233,8,133, 85,165,86
32210 DATA 233,0,133,86,104,230,84,76,124,6,56,165,84, 233,7,133
32220 DATA 84,160,1,96,173,253,2,141,251,2,169,17,141, 162,3,169
32230 DATA 0,141,168,3,141,169,3,162,96,76,86,228,138, 72,152,72
32240 DATA 165,85,133,91,165,86,133,92,165,84,133,90, 208,5,230,90
32250 DATA 76,245,6,198,90,32,196,6,32,196,6,104,1, 68,104,170,96
32260 GRAPHICS 0:?"POMYLKA W DANYCH"
```

LITERATURA

1. B.Albrecht, L.Finkel, J.Brown, "ATARI BASIC
2. H.Moore,J.Lower,B.Albrecht, "ATARI SOUND AND GRAPHICS"
3. L.Poole,M.McNiff, S.Cook "YOUR ATARI COMPUTER"
4. P.Luptorn,.Robinson'THE ATARI XL HANDBOOK"
5. "ATARI XL USER'S HANDBOOK"
6. "COMPUTE'S FIRST (SECOND,THIRD) BOOK OF ATARI"
7. "COMPUTE'S FIRST (SECOND)BOOK OF ATARI GRAPHICS"
8. "COMPUTE'S MAPPING THE ATARI"
9. "COMPUTE'S FIRST (SECOND) BOOK OF MACHINE LANGUAGE"
10."HOW TO PROGRAM YOUR ATARI IN 6502 MACHINE LANGUAGE"
11. M.Harrison, S.Trotter, "REFERENCE XL BOOK"
12. "OPERATING SYSTEM USER'S MANUAL"
13. "ATARI HARDWARE MANUAL"
14. "ATARI BASIC REFERENCE MANUAL"
15. 'TECHNICAL REFERENCE NOTES - ATARI SYSTEM"
16. 'TRICKS AND TIPS FOR YOUR ATARI - HACKER BOOK"
17. M.E11is,R.Ellis "ATARI USER'S GUIDE"
18. H.Moore "SHAPES AND SOUNDS FOR THE ATARI"
19. R.Zaks "PROGRAMMING THE 6502"

Periodyki dotyczące ATARI:

ANTIC
ANALOG
ATARI CONNECTION
HI-RES
ATARI USER
MONITOR
APX