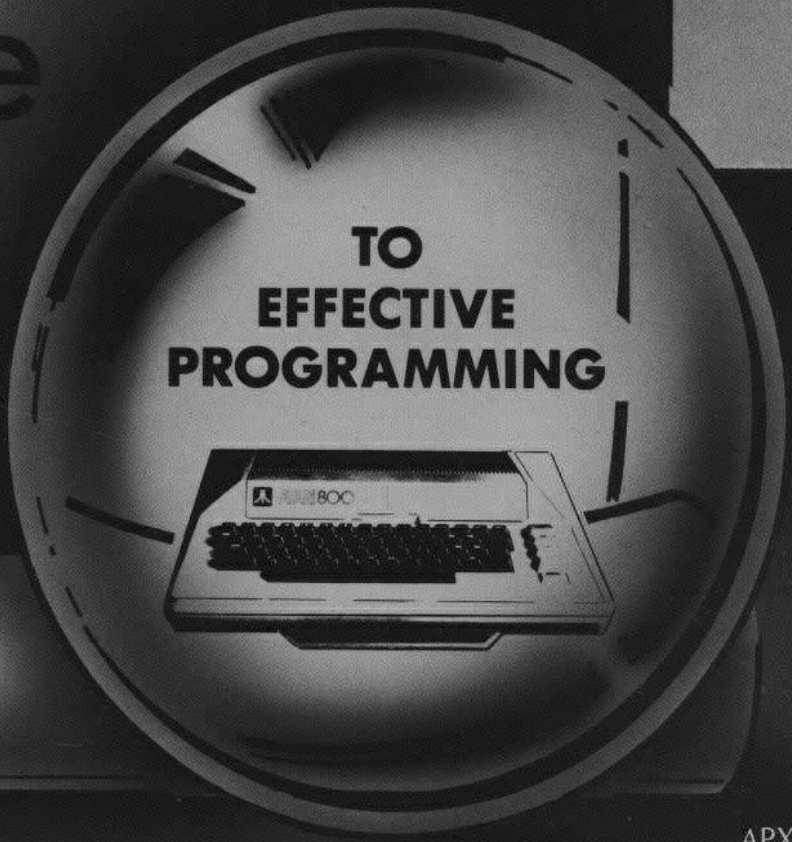
De Re Atari

Anno Domini MCMLXXXI

po polsku

A Guide



400/800™
Home
Computer

L. CROSS '81

APX-90008

WSTĘP

Książka ta dotyczy komputerów domowych Atari, zarówno modelu 400 jak i 800, które pod względem elektronicznym są identyczne, a różnią się wykonaniem mechanicznym, więc klawiaturą czy miejscem dla kartridża. Zadaniem, jakie postawili przed sobą autorzy, było szczegółowe omówienie najlepszych sposobów wykorzystania wszystkich możliwości komputerów Atari. Ponieważ możliwości te są stosunkowo szerokie i wielostronne, staraliśmy się omówić je jak najpełniej. Wynika stąd także, że książka ta nie jest przeznaczona dla początkujących użytkowników Atari, wymaga ona bowiem pewnej podstawowej wiedzy od czytelnika, przede wszystkim należy zapoznać się z językiem Atari Basic, najlepiej z "Atari Basic Reference Manual". Wymagana jest także pewna znajomość języka asemblera. Załączony na końcu słowniczek definiuje i wyjaśnia niektóre rzadziej spotykane pojęcia, nie zawiera jednakże terminów, które dla każdego poważnego użytkownika komputerów osobistych nie powinny być obce.

Napisana ona została jako książka treningowa dla programistów profesjonalnych, pracujących z komputerami Atari, nie wyklucza to jednakże jej przydatności dla szerokiego odbiorcy. Nie może ona ponadto zastąpić żadnej z manualnych instrukcji technicznych Atari, przeznaczonych dla programistów, którzy są już dobrze zaznajomieni z systemem. Należy ją traktować bardziej jako a1manach, wyjaśniający pewne idee oraz ukazujący niezbyt oczywiste możliwości, a nie jako podręcznik omawiający poszczególne rejestry i opisujący kody sterujące.

Jej tytuł, DE RE Atari, jest dość często spotykany w literaturze fachowej. Wiele manuskryptów łacińskich z epoki rzymskiej i średniowiecza nosiło tytuł "De Re Tego" czy "De Re Tamtego". Tak więc "De Re Rustica" była poematem dotyczącym rolnictwa, a "De Re Metallica" opisywała metalurgię. W wolnym tłumaczeniu "Da Re..." znaczy "Wszystko o...".

Autorami książki są pracownicy Program Development Support Group. Rozdziały 1-6 oraz załączniki A i B napisał Chris Crawford. Lane Winner napisał rozdział 10, a wraz z Jimem Coxem załącznik D. Załącznik C jest autorstwa Amy Chen, rozdziały 8 i 9 Jima Duniona. Kathleen Pitta napisała załącznik E, Bob Fraser rozdział 7, Gus Makreas przygotował słowniczek. Całość nie jest pozbawiona niedomówień, ale jesteśmy z niej bardzo dumni.

Autorzy

1. PRZEGLĄD SYSTEMU

Komputery domowe Atari należą do komputerów osobistych drugiej generacji. Przeznaczone są one przede wszystkim dla użytkowników indywidualnych. Wykonanie zewnętrzne komputera jest przejawem realizacji wielu postulatów odbiorców, a o jego orientacji ku szerokiemu użytkownikowi świadczy wiele szczegółów. Po pierwsze, urządzenie jest zabezpieczone przed wieloma z błędów, jakie użytkownik może popełnić. Na przykład wykonanie gniazd i wtyczek chroni całkowicie układ elektroniczny przed błędnym podłączeniem spolaryzowanego napięcia zasilania, czy też niewłaściwym dołączeniem jakichkolwiek urządzeń peryferyjnych. Po drugie, komputer posiada niezwykle rozbudowane możliwości graficzne - obraz znacznie silniej oddziałuje na ludzi niż tekst. Po trzecie, urządzenie ma nieprzeciętne możliwości generowania dźwięku - również dlatego, że odbiór sensoryczny dźwięków jest znacznie lepszy niż w przypadku informacji pisanych, wreszcie, komputer posiada możliwość bezpośredniego podłączenia sterowników, takich jak dżojstik czy paddle, które stanowią bardziej bezpośredni dostęp do komputera niż klawiatura. Wynika stąd nie tylko fakt, iż możliwości tego urządzenia są olbrzymie, ale przede wszystkim to, że owe olbrzymie możliwości są tylko częściami składowymi całej architektonicznej filozofii, mającej służyć użytkownikowi. Ktoś, kto nie zrozumie tej fundamentalnej zasady, będzie się borykał w pracy ze złożonością systemu Atari.

Strukturą wewnętrzną komputery Atari 400/800 różnią się w znacznym stopniu od innych systemów. Oczywiście, znajduje się w niej mikroprocesor - 6502 - RAM, ROM, oraz PIA, lecz są tam także trzy specjalne układy scalone dużej integracji (LSI), zwane ANTIC, CTIA lub GTIA, oraz POKEY. Zostały one zaprojektowane przez inżynierów Atari wyłącznie w celu odciążenia mikroprocesora 6502, co umożliwia skoncentrowanie się na wykonywaniu programów. Z drugiej strony umożliwiły one wyposażenie komputerów Atari w tak olbrzymie możliwości, każdy z tych trzech układów jest niemal tak samo ogromny (w skali płytki krzemowej) jak sam 6502, tak więc stanowią one o wiele większą potęgę niż mikroprocesor. Zarządzanie systemem Atari 400/800 wiąże się głównie z problemami zarządzania tymi właśnie układami.

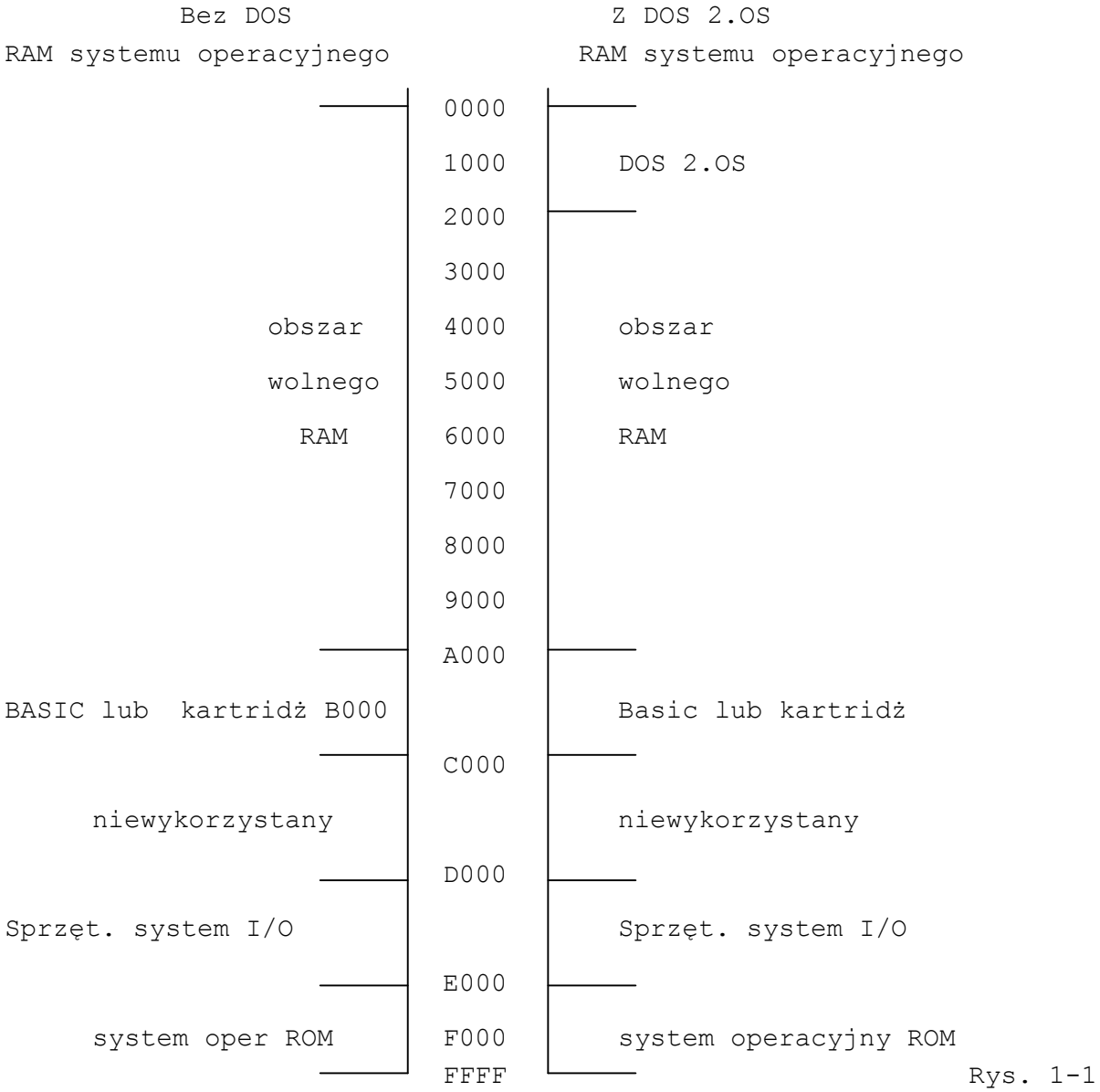
ANTIC jest mikroprocesorem zarządzającym wyświetlanym obrazem. Jest to prawdziwy procesor - posiada swój własny zbiór instrukcji, własny program (zwany listą displejową) oraz zbiór danych. Lista displejowa oraz zbiór danych zapisywane są w RAM przez 6502. ANTIC wykorzystuje te informacje, stosując bezpośredni dostęp pamięci (Direct Memory Access – DMA). Odczytuje instrukcje wyższego rzędu, zawarte w liście displejowej, po czym tłumaczy je na nieprzerwany ciąg prostych instrukcji, realizowanych następnie przez CTIA/GTIA.

CTIA jest układem interfejsu telewizyjnego. Pośrednio większość operacji CTIA jest sterowana przez ANTIC, jednakże 6502 może być zaprogramowany na przejęcie nadzoru niektórych lub wszystkich funkcji realizowanych przez CTIA. CTIA przetwarza ciąg cyfrowych instrukcji z ANTICu (lub 6502) na analogowy sygnał sterujący telewizorem. CTIA ponadto wprowadza do sygnału telewizyjnego pewne własne elementy, jak na przykład kolor obrazu, obecność elementów ruchomych (PMG) oraz detekcję kolizji.

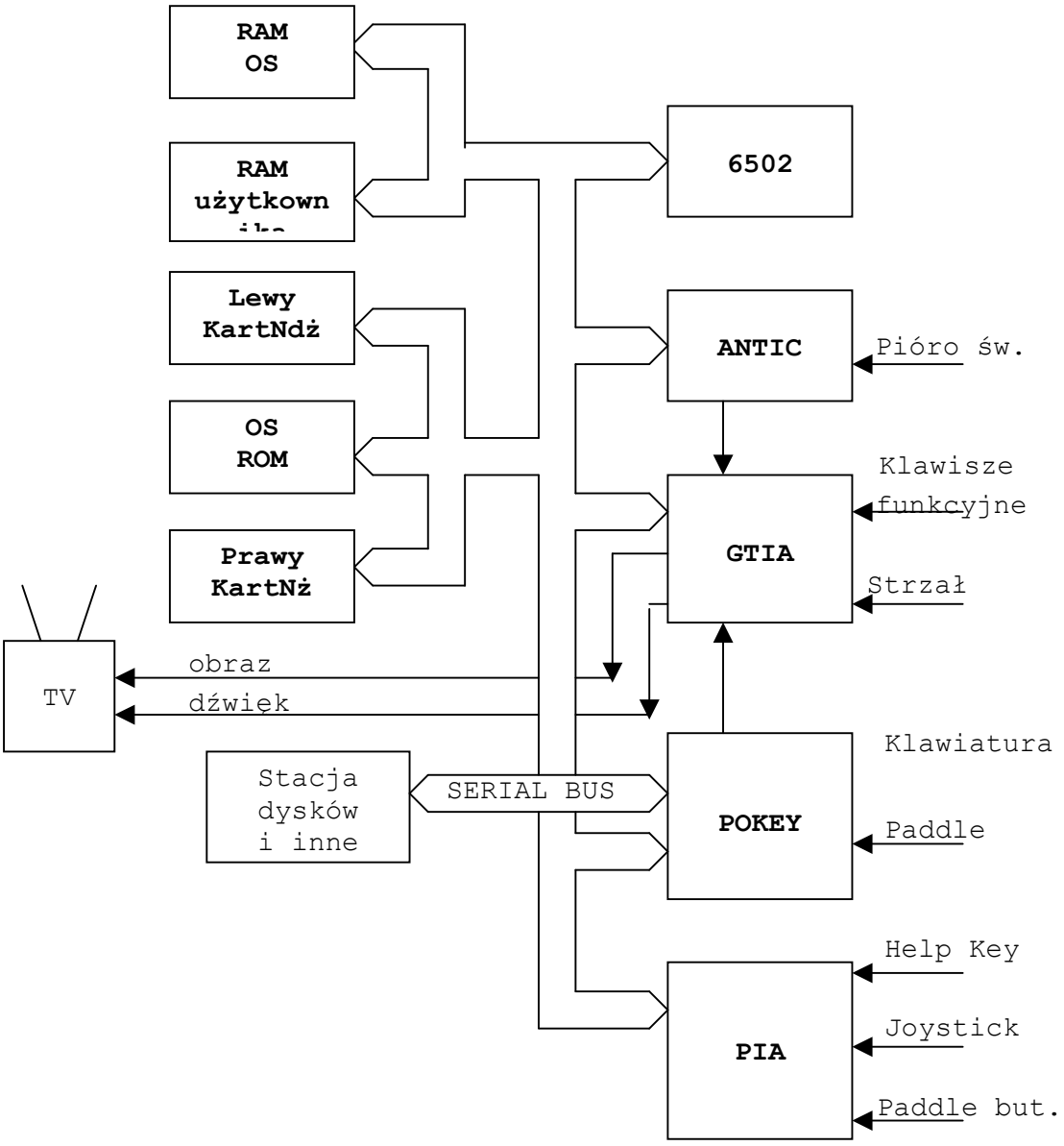
POKEY jest cyfrowym układem wejścia/wyjścia (I/O). Zarządza tak różnymi funkcjami, jak sterowanie równoległą szyną I/O, generacja dźwięku, odczytywanie klawiatury, czy generacja liczb przypadkowych. Ponadto przetwarza na sygnał cyfrowy analogowy sygnał wejściowy z paddle i steruje żądania przerwań maskowalnych (IRQ) przesyłane przez urządzenia peryferyjne.

Wszystkie te cztery układy scalone LSI pracują równocześnie. Podział wykonywanych przez nie funkcji został tak zaprojektowany, by uniknąć jakichkolwiek konfliktów między nimi. Jedynym konfliktem, jaki występuje w tym systemie sprzętowym, jest wykorzystywanie przez ANTIC adresów oraz szyn danych do odczytywania informacji displejowych. W takim momencie wstrzymuje on pracę 6502 i przejmuje sterowanie szynami danych.

Tak jak we wszystkich systemach opartych na procesorze 6502, operacje I/O bazują na mapowaniu pamięci. Rysunek 1-1 przedstawia schematyczną mapę pamięci komputera, natomiast rysunek 1-2 jego architekturę sprzętową.



Rys. 1-1



Rys. 1-2 System Atari 400/800

2. ANTIC I LISTA DISPLEJOWA

Zrozumienie olbrzymich możliwości graficznych komputerów Atari wiąże się nieodłącznie z pewnymi podstawowymi wiadomościami dotyczącymi natury obrazu telewizyjnego. Telewizor pracuje w systemie tworzenia obrazu zwanym rastrowym. Strumień elektronów, generowany przez działo elektronowe umieszczone w tylnej części kineskopu, wyrzucany jest w kierunku przedniej ściany, czyli ekranu. Na drodze strumienia pomiędzy działem a ekranem umieszczony jest zespół cewek, które, po wytworzeniu pomiędzy nimi odpowiedniego pola, powodują odchylenie strumienia elektronów, zarówno w pionie jak i w poziomie. Tą metodą wysoce energetyczny strumień elektronów może pobudzić do świecenia każdy punkt pokrytego luminoforem ekranu. Układ elektroniczny telewizora, sterując zespół odchylający, powoduje, że strumień elektronów przemieszcza się po ekranie w bardzo regularny sposób. Ponadto sterowana elektronicznie jest intensywność strumienia. Jeżeli z działa wyrzucone zostaną elektrony z większą energią, punkt na ekranie, do którego dotrą, zaświeci jaśniej; jeżeli intensywność strumienia zostanie obniżona punkt zaświeci słabo lub w ogóle nie zostanie pobudzony do świecenia.

Strumień elektronów zapala początkowo punkt w lewym górnym rogu ekranu, po czym zaczyna przemieszczać się w poziomie. W trakcie tej drogi zapala poszczególne punkty z różną intensywnością świecenia, tworząc obraz. Po dotarciu do prawej krawędzi ekranu strumień jest wygaszany, po czym przemieszczony z powrotem do lewej krawędzi i przesuwany odrobinę w dół. Tu następuje ponowne włączenie strumienia i rysowanie kolejnej linii obrazu. Proces ten powtarzany jest kolejno, a na narysowanie całego obrazu na ekranie składa się 262 linie (w różnych systemach telewizji ilość linii obrazu jest różna, lecz pomińmy je na razie w tych rozważaniach). Owe 262 linie pokrywają całość ekranu, od góry do dołu.

Po dotarciu do dołu ekranu, czyli po narysowaniu 262-giej linii, strumień elektronów jest wygaszany i przenoszony z powrotem w lewy górny róg ekranu, po czym cały cykl zaczyna się od początku. Kompletny cykl rysowania obrazu na ekranie powtarza się 60 razy na sekundę.

Gwoli ścisłości: linia rysowana przez strumień elektronów na ekranie zwana jest "poziomą linią skaningową". Jest ona podstawową jednostką miary odległości pionowej na ekranie - wysokość obrazu określa się ilością linii potrzebnych do jego narysowania. Okres czasu, kiedy wygaszony strumień elektronów powraca od prawej krawędzi ekranu do lewej, nazywa się "wygaszeniem poziomym". Okres, kiedy wygaszony strumień powraca z prawego dolnego rogu do lewego górnego rogu ekranu, nazywa się "wygaszeniem pionowym". Cały proces rysowania obrazu na ekranie trwa 14634 mikrosekund. Wygaszenie pionowe trwa około 1400 mikrosekund, wygaszenie poziome zaś 14 mikrosekund. Rysowanie poziomej linii na ekranie trwa 64 mikrosekundy.

Odbiorniki telewizyjne regulowane są "overskanningowo", co oznacza, że krawędzie rysowanego obrazu znajdują mię poza krawędziami widocznego ekranu. Gwarantuje to, że na ekranie nie są widoczne ciemne krawędzie dookoła rysowanego obrazu. Jest to bardzo zła sytuacja w przypadku wyświetlania obrazu komputerowego, ponieważ część informacji mogłaby zostać wyświetlona poza granicami ekranu.

Z tego powodu obraz komputerowy musi być nieco mniejszy od tego, jaki telewizor teoretycznie byłby w stanie wyświetlić. Dlatego też Atari wykorzystuje jedynie 192 linie poziome na ekranie telewizora. Stąd też, maksymalna rozdzielczość uzyskiwana w obrazie tworzonym przez komputer wynosi 192 punkty w linii pionowej.

Standardową jednostką odległości poziomej jest "punkt maski". Szerokość obrazu określa się ilością punktów maski potrzebnych do jego narysowania. W jednej poziomej linii skaningowej znajduje się 228 punktów maski, z czego maksymalnie 176 jest widzianych na ekranie. Stąd więc maksymalna rozdzielczość pozioma standardowego ekranu kolorowego wynosi 176 punktów maski. System komputerów Atari stwarza możliwość uzyskania jeszcze lepszej rozdzielczości, poprzez sterowanie poszczególnych połówek punktów maski, co daje rozdzielczość poziomą obrazu 352 punktów. Niezależne sterowanie poszczególnych połówek punktów maski pociąga za sobą powstawanie ciekawych efektów barwnych zwanych "barwnymi artefaktani". Zjawisko to w niektórych przypadkach może być bardzo uciążliwe, a z drugiej strony umożliwia uzyskanie dodatkowych barw obrazu, nie ograniczonych parametrami komputera.

KOMPUTERY A TELEWIZORY

Podstawowym problemem dla mikrokomputera w przesyłaniu obrazu do telewizora pracującego w systemie rastrowym jest fakt, iż tworzenie obrazu jest procesem dynamicznym. Oznacza to w konsekwencji, że telewizor "nie pamięta" obrazu. Wynika stąd, że to komputer powinien posiadać pamięć obrazu i nieustannie wysyłać sygnał sterujący tworzeniem obrazu w telewizorze. Proces przesyłania informacji do telewizora jest więc procesem ciągłym i wymaga nieustannej uwagi komputera. Z tego powodu większość komputerów wyposażona jest w specjalne sprzętowe układy scalone, zarządzające komunikowaniem się z telewizorem. Schemat tworzenia obrazu jest dla wszystkich systemów komputerowych taki sam:

Mikroprocesor → pamięć obrazu → układ scalony wideo → telewizor

Mikroprocesor zapisuje informacje w obszarze pamięci obrazu RAM, gdzie dane dotyczące obrazu są przechowywane. Układ wideo ciągle odczytuje ten obszar pamięci i przekształca dane na analogowy sygnał telewizyjny. Sygnał ten przesłany do telewizora powoduje wyświetlenie na ekranie określonych symboli. Ciąg symboli, które pojawiają się na ekranie, musi być więc umieszczony w pamięci obrazu dokładnie w takiej samej kolejności, w jakiej mają być one wyświetlone. A zatem pierwszy bajt pamięci obrazu opisuje symbol pojawiający się w lewym górnym rogu ekranu, drugi bajt następny symbol w tym samym wierszu, potem trzeci, i tak dalej- ostatni bajt pamięci opisuje symbol wyświetlany w prawym dolnym rogu ekranu.

Jakość obrazu pojawiającego się na ekranie zależy od dwóch czynników: od jakości układu scalonego wideo oraz od ilości RAM przeznaczonej na pamięć obrazu. Najprostsze rozwiązanie można spotkać w komputerach TRS-80 (Radio Shack Co.) oraz Commodore PET. Oba te urządzenia rezerwują specyficzny obszar 1K RAM na pamięć obrazu. Układ wideo po prostu odczytuje zapisane tam dane, zamienia

je na ciąg symboli (wykorzystując zbiór symboli w ROM) i umieszcza na ekranie telewizora. każdy bajt odpowiada jednemu symbolowi spośród standardowego zestawu 256 różnych symboli znajdujących się w ROM. Przy 1K pamięci obrazu może być więc umieszczonych na ekranie około 1000 symboli. Nie jest to zbyt duża ilość. Apple z kolei stosuje znacznie udoskonalony układ wideo w tym samym systemie tworzenia obrazu. Układ ten pozwala na rozróżnienie trzech trybów pracy: tekstowego, graficznego o małej rozdzielczości i graficznego o dużej rozdzielczości. W trybie tekstowym proces odczytywania pamięci obrazu jest identyczny jak opisano poprzednio. W trybie graficznym małej rozdzielczości układ wideo interpretuje pamięć obrazu inaczej. Jeden bajt nie reprezentuje teraz jednego symbolu, lecz jest interpretowany jako dwa nible - wartość każdego nibla opisuje kolor punktu wyświetlanego na ekranie. W trybie graficznym o dużej rozdzielczości każdemu punktowi ekranu przypisany jest tylko jeden bit pamięci obrazu. Jeżeli bit jest równy 1, zapala się punkt w ustalonym uprzednio kolorze; jeżeli bit jest równy 0, punkt nie zostaje wyświetlony na ekranie. Tak wygląda oczywiście tylko podstawowy system pracy, bez wnikania w niuanse architektury Apple. Zasadniczą różnicą jest wyodrębnienie trzech różnych trybów pracy układu wideo, poprzez zastosowanie trzech zupełnie różnych sposobów interpretowania danych zapisanych w pamięci obrazu. Układ wideo Apple, jest na tyle doskonały, że może interpretować zapis w pamięci bądź jako 8-bitowe symbole (tryb tekstowy), jako 4-bitowe kolorowe nible (tryb o małej rozdzielczości), bądź też jako 7 indywidualnych bitów (tryb o dużej rozdzielczości).

ANTIC - MIKROPROCESOR WIDEO

System stosowany w Atari 400/800, oparty na liście displejowej, stanowi generalizację opisanych wyżej systemów. Podczas gdy TRS-80 czy PET mają tylko jeden tryb pracy a Apple trzy tryby, Atari 400/800 dysponuje 14 trybami pracy. Drugą znaczącą różnicą jest możliwość mieszania różnych trybów na ekranie. Oznacza to, że użytkownik nie jest ograniczony do wyboru pomiędzy całym ekranem przeznaczonym dla symboli i całym ekranem otwartym do wyświetlania mniejszych czy większych punktów. Na ekranie może być umieszczony dowolny zestaw spośród 14 dostępnych trybów pracy. Trzecią ważną różnicę stanowi fakt, iż pamięć obrazu w RAM może być umieszczona w dowolnym obszarze pamięci komputera i przesuwana czy rozbudowywana w trakcie wykonywania programu, podczas gdy inne komputery posiadają stały, zarezerwowany obszar pamięci obrazu.

Tak olbrzymie rozbudowanie możliwości komputera było możliwe jedynie dzięki zastosowaniu mikroprocesora wideo, nazwanego ANTIC. Wcześniejsze systemy stosowały mniej czy bardziej rozbudowane układy wideo, ale ANTIC jest już całym, odrębnym mikroprocesorem, posiadającym swój własny zestaw instrukcji, program oraz dane. Program ANTICu zwany jest listą displejową, która konkretyzuje trzy rzeczy: gdzie należy szukać danych dotyczących obrazu, jaki ma być tryb pracy czyli w jaki sposób należy te dane interpretować, oraz jakie specjalne opcje (o ile są) mają zostać wykonane. Praca z systemem opartym na liście displejowej wiąże się nierozerwalnie z jednym - należy przestać traktować ekran jako jednorodne miejsce tworzenia obrazów, a spojrzeć na niego jako na zbiór "linii

trybowych". Linia trybowa to nic innego jak pewna ilość poziomych linii skaningowych telewizora, która określa dany tryb pracy w całym pasie od lewej do prawej krawędzi ekranu. Linia trybowa w trybie graficznym 2 ma wysokość 16 poziomych linii skaningowych, podczas gdy linia w trybie graficznym 7 ma wysokość tylko jednej linii skaningowej. Tryby graficzne, wywoływane z BASICa, są albo całkowicie homogenne, albo dopuszczają jedynie obecność okna tekstowego. Nie należy jednak ograniczać swojej wyobraźni; manipulowanie listą displejową pozwala na tworzenie dowolnej sekwencji linii trybowych na ekranie. Lista displejowa jest bowiem ciągiem kodów, gdzie każdy bajt określa rodzaj linii trybowej.

Zbiór instrukcji mikroprocesora ANTIC jest bardzo prosty, obejmuje bowiem tylko cztery rodzaje instrukcji: dotyczące linii pustych, linii z symbolami, linii wygaszonych oraz skoków programowych. Instrukcje dotyczące linii pustych powodują, że ANTIC wyświetla linię trybową, w której wszystkie punkty na ekranie mają jednakowy kolor - nie są wyświetlane żadne symbole. Instrukcje dotyczące linii z symbolami powodują wyświetlenie linii trybowej i umieszczenie w niej określonych symboli. Instrukcje linii wygaszonych powodują wyświetlenie linii trybowej o kolorze tła. Wreszcie instrukcje skoków są analogiczne do instrukcji skoku mikroprocesora 6502, nazywane JMP; analogicznie powodują one przepisanie licznika programu mikroprocesora ANTIC.

Występują ponadto cztery specjalne opcje, określane przez włączenie odpowiedniego bitu w instrukcjach ANTICu. Są to: przerwanie listy displejowej (DLI), przesuwanie pamięci obrazu (LMS), zwijanie pionowe oraz zwijanie poziome obrazu.

Instrukcje linii pustych powodują wyświetlenie linii trybowej o jednakowej barwie wszystkich punktów. Kolor punktów określony jest przez rejestry kolorów, natomiast wybór rejestru precyzowany jest przez wartość danych w pamięci obrazu. W czterokolorowych trybach graficznych (tryby BASICu 3, 5, 7 i tryby ANTICu 8, A, D, E) para bitów określa wybór rejestru kolorów:

para bitów	wartość	rejestr
00	0	COLBAK
01	1	COLPF0
10	2	COLPF1
11	3	COLPF2

Ponieważ tylko dwa bity precyzują jeden punkt na ekranie, jeden bajt pamięci obrazu charakteryzuje cztery punkty. Na przykład, bajt w pamięci obrazu, zapisany wartością $1B charakteryzuje cztery kolejne punkty: pierwszy o barwie tła, drugi w kolorze określonym przez rejestr koloru 0, trzeci określony przez rejestr koloru 1 i czwarty określony przez rejestr koloru 2:

$1B = 00011011 = 00 01 10 11

W dwukolorowych trybach pracy (tryby BASICu 4, 6, 8 i tryby ANTICu 9, B, C, F) każdy bit precyzuje jeden z dwóch rejestrów

kolorów. Bit wyłączony (równy 0) określa rejestr koloru tła, natomiast bit równy 1 wskazuje na rejestr 0. Stąd jeden bajt pamięci obrazu charakteryzuje osiem punktów wyświetlanego obrazu.

Podsumowując, istnieje osiem różnych trybów wyświetlania linii trybowych na ekranie. Różnią się one między sobą ilością dostępnych kolorów (2 lub 4), wysokością linii trybowej (1, 2, 4 lub 8 poziomych linii skaningowych) oraz ilością punktów, umieszczanych w jednej linii trybowej (40, 80, 160 lub 320 punktów). Oczywiście, można uzyskać tryb o bardzo dużej rozdzielczości, co pociąga jednak za sobą rozszerzenie obszaru RAM przeznaczonego na pamięć obrazu. Poniższa tabela zawiera informacje dotyczące wszystkich trybów pracy.

Numer trybu ANTICu	Numer trybu BASICu	Liczba dostęp-nych kolorów	Ilość linii skaning. w jednej linii trybu	Ilość punktów w jednej linii trybu	Ilość bajtów RAM	
					dla jednej linii trybu	dla całego ekranu
2	0	2	8	40	40	960
3	-	2	10	40	40	760
4	-	4	8	40	40	960
5	-	4	16	40	40	480
6	1	5	8	20	20	480
7	2	5	16	20	20	240
8	3	4	8	40	10	240
9	4	2	4	80	10	4800
A	5	4	4	80	20	9600
B	6	2	2	160	20	1920
C	-	2	1	160	20	3840
D	7	4	2	160	40	3840
E	-	4	1	160	40	7680
F	8	2	1	320	40	7680

Tabela 2-1. Charakterystyka trybów pracy układu ANTIC

Instrukcje dotyczące linii zawierających symbole powodują, że ANTIC wyświetla linię trybową z symbolami. Każdy bajt w pamięci obrazu RAM charakteryzuje jeden symbol. Umieszczanie symboli na ekranie omówione zostanie w rozdziale 3.

Instrukcje dotyczące linii wygaszonych powodują wyświetlenie linii trybowej w kolorze tła. Istnieje osiem instrukcji linii wygaszonych, powodujących wyświetlenie od jednej do ośmiu linii w kolorze tła.

Istnieją tylko dwie instrukcje skoków. Pierwsza, JMP, jest instrukcją bezwarunkową; powoduje ona wpisanie do licznika programu ANTIC adresu, który występuje po instrukcji JMP jako operand. Jej jedyną funkcją jest rozwiązanie następującego problemu: licznik

programu ANTICu jest licznikiem zaledwie 10-bitowym z 6-bitowym rejestrem. Wynika stąd, że lista displejowa nie może być umieszczona po obu stronach granicy 1K przedziałów pamięci. Jeżeli musi ona przekraczać taką granicę, wówczas musi zostać użyta instrukcja JMP do załadowania do licznika adresu z drugiego przedziału RAM. Należy więc zauważyć, że przesuwanie listy displejowej w pamięci w trakcie wykonywania programu wiąże się z pewnymi problemami.

Druga instrukcja skoku, JWB, jest stosowana w każdej liście displejowej. Powoduje ona wpisanie do licznika programu układu ANTIC adresu występującego po instrukcji jako operand, oraz wstrzymanie wykonywania programu do czasu wykonania przez telewizor wygaszenia pionowego. Operandem instrukcji JWB jest zwykle adres początku listy displejowej, a sama instrukcję umieszcza się na końcu listy. Powoduje to zamknięcie programu w nieskończoną pętlę, a oczekiwanie na zakończenie przez telewizor wygaszenia pionowego zapewnia uzyskanie synchronizacji programu wykonywanego w nieskończonej pętli z cyklem tworzenia obrazu w telewizorze. Obie instrukcje skoków, JMP i JWB, są instrukcjami trzybajtowymi, gdzie pierwszy bajt stanowi kod instrukcji, a w dwóch następnych zawarty jest adres skoku w układzie LSB/MSB.

Cztery specjalne opcje, wspomniane uprzednio, będą omówione w rozdziałach 6 i 6. Wydaje się jednak, że instrukcja czytania pamięci obrazu LMS (Load Memory Scan) powinna zostać objaśniona w tym miejscu. Opcja ta wybierana jest poprzez włączenie bitu 6-go w bajcie instrukcji czytania czy to pustej, czy wypełnionej symbolami linii. ANTIC, napotykając instrukcję LMS, zapisuje wewnętrzny licznik pamięci ekranowej adresem występującym po instrukcji LMS jako operand. Ów licznik pamięci ekranowej wskazuje układowi ANTIC gdzie znajdują się dane, które mają zostać umieszczone na ekranie. Czytanie danych do umieszczenia na ekranie rozpocznie się właśnie od tego adresu. Instrukcja LMS jest instrukcją 3-bajtową, gdzie po bajcie kodu instrukcji występuje 2-bajtowy operand adresowy. W liście displejowej najprostszej postaci instrukcja LMS występuje tylko raz, na samym początku listy. W niektórych przypadkach niezbędne jest powtórne użycie instrukcji LMS. Potrzeba taka występuje przykładowo wówczas, kiedy obszar pamięci ekranowej przekracza granicę przedziału 4 K. Licznik pamięci układu ANTIC jest licznikiem 12-bitowym z 4-bitowym rejestrem kluczującym; dlatego też zbiór danych ekranowych nie może znajdować się po dwóch stronach granicy przedziału 4K. Jeżeli zachodzi taka konieczność, należy powtórnie użyć instrukcji LMS w celu wpisania do licznika pamięci ekranowej adresu z nowego przedziału. Należy zauważyć, że w tym świetle umieszczenie obszaru pamięci obrazu w RAM nie może być zupełnie dowolne. Instrukcja LMS posiada szersze zastosowania, które zostaną omówione w dalszej części.

STRUKTURA LISTY DISPLEJOWEJ

Każda lista displejowa powinna zaczynać się od trzech instrukcji: "wyświetl 8 linii wygaszonych". Ma to na celu przesunięcie górnej krawędzi tworzonego obrazu o 24 linie skaningowe w dół i tym samym zapobieżenie wyświetleniu pewnych informacji poza granicą ekranu telewizyjnego. W następnej kolejności powinien zostać sprecyzowany rodzaj pierwszej linii trybowej. Równocześnie należy użyć instrukcji

LMS i zapisać w liczniku pamięci ekranowej adres początkowy obszaru pamięci obrazu. Dalej następuje właściwa lista displejowa, której każdy bajt charakteryzuje kolejną linię trybową. Całkowita ilość poziomych linii skaningowych, które mają być wyświetlone w określonych listą displejową liniach trybowych, może wynosić co najwyżej 192. ANTIC nie jest w żaden sposób związany z czasowym reżimem procesu tworzenia obrazu w telewizorze. A więc polecenie wyświetlenia zbyt wielu linii skaningowych (zawarte w liście displejowej) zostanie wykonane, lecz obraz telewizyjny prawdopodobnie zacznie się przesuwać w pionie. Lista displejowa określająca obraz złożony z mniej niż 192 linii skaningowych nie spowoduje żadnego błędu; spowoduje jedynie wolniejsze wykonywanie programu przez 6502, gdyż więcej czasu będzie potrzebował ANTIC do odczytywania danych z pamięci ekranowej. Ilość linii skaningowych obrazu, określona przez ilość i jakość linii trybowych opisanych instrukcjami listy displejowej, musi być zawsze ściśle skalkulowana i sterowana przez programistę. Lista displejowa kończy się instrukcją JWB. Poniżej przedstawiono typową listę displejową dla standardowego trybu graficznego BASICu 0 (wszystkie wartości w systemie heksadecymalnym):

```
70 wyświetl 8 wygaszonych linii skaningowych
70 +
70 +
42 wyświetl linię trybu ANTICu 2 (tryb BASICu 0), ponadto
20 pamięć ekranowa zaczyna się od adresu 7020
70
02 wyświetl linię trybu ANTICu 2
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
02 +
41 JWB - wykonaj skok do adresu początkowego listy
E0   displejowej, 7BE0, i zaczekaj na wykonanie przez
7B   telewizor wygaszenia pionowego
```

Tab. 2-2. Typowa lista displejowa trybu graficznego 0 (BASICu).

Jak można zauważyć, lista displejowa jest krótka, zawiera jedynie 32 bajty, a większość list displejowych jest krótsza od 100 bajtów. Ponadto, jej struktura jest bardzo przejrzysta, umożliwiająca tworzenie własnych, zróżnicowanych list displejowych.

Tworzenie własnej listy displejowej należy zacząć od określenia formatu obrazu. Najlepiej wykonać to na papierze. Po utworzeniu schematu obrazu, należy przełożyć go na ciąg określonych linii trybowych. Trzeba też skalkulować ilość linii skaningowych zawartych w każdej linii trybowej, posługując się przy tym danymi z tabeli 2-1. Następnie rodzaj kolejnej linii trybowej należy przedstawić w kodzie trybów graficznych ANTICu. Na początku listy muszą się znaleźć trzy instrukcje dotyczące linii wygaszonych ($70). W pierwszej instrukcji określającej rodzaj trybu graficznego należy włączyć bit 6-ty (tzn. nadać starszemu niblowi wartość 4), co spowoduje utworzenie instrukcji LMS. Muszą po niej nastąpić dwa bajty precyzujące adres początkowy pamięci ekranowej (najpierw LSB, potem MSB), po czym kolejno należy umieścić kody charakteryzujące poszczególne linie trybowe. Na końcu musi się znaleźć instrukcja JWB ($41), a po niej adres początkowy listy displejowej. Całość listy może być zapisana w dowolnym miejscu pamięci RAM, należy się jedynie upewnić, czy nie koliduje ona z innym zapisem w pamięci i czy instrukcja JWB zawiera właściwy adres początku listy displejowej. Lista nie powinna zostać zapisana po dwóch stronach granicy przedziału 1K. Jeżeli zachodzi konieczność takiego zapisu, należy tuż przed końcem pierwszego przedziału umieścić instrukcję JMP, której operandem powinien być pierwszy bajt listy leżący w innym przedziale. Następnie należy wyłączyć na chwilę ANTIC podczas przepisywania wskaźnika listy displejowej. Wykonuje się to przez wpisanie wartości 0 do SDMCTL pod adresem $22F. Teraz nowy adres listy displejowej trzeba zapisać w rejestrach $230 i $231, po czym włączyć z powrotem ANTIC wpisując do SDMCTL wartość $22. W trakcie wygaszenia pionowego, kiedy przerwana zostaje praca ANTICu, system operacyjny (OS) dokona przepisania licznika programu ANTICu podanymi wartościami.

UMIESZCZANIE SYMBOLI NA EKRANIE O ZMIENIONEJ LIŚCIE DISPLEJOWEJ

Obszar pamięci ekranowej może być umieszczony w dowolnym miejscu pamięci RAM komputera. Typowa lista displejowa zawiera adres początku pamięci ekranowej jako operand pierwszej instrukcji listy - instrukcji LMS. Jednakże ANTIC może wykonać następną instrukcję LMS, umieszczoną przy jednej z kolejnych instrukcji displejowych, o ile zachodzi taka potrzeba. W ten sposób na jednym obrazie mogą znaleźć się informacje, odczytane z różnych rejestrów rozmieszczonych w dowolnych miejscach całego obszaru RAM. Jest to na przykład metoda umieszczania różnych informacji w niezależnych od siebie a ukazujących się równocześnie oknach tekstowych.

Istnieje jednakże kilka ograniczeń w dowolności rozmieszczania pamięci ekranowej w RAM. Po pierwsze, blok pamięci ekranowej nie może być rozmieszczony po obu stronach granicy 4K pamięci RAM. Jeżeli nie da się tego uniknąć (jak np. w przypadku trybu 8 BASICu,

który wymaga 8K pamięci ekranowej), należy przepisać zawartość licznika pamięci ekranowej przy pomocy kolejnej instrukcji LMS.

Po drugie, jeżeli chce się wykorzystać którykolwiek z podprogramów edytorskich systemu operacyjnego, należy bezwzględnie przestrzegać konwencji używanych przez OS. Może to być szczególnie trudne w przypadku stosowania zmodyfikowanej listy displejowej w programach BASICu, jeżeli dokona się zmian w standardowej liście displejowej programu BASICu, po czym spróbuje wykonać instrukcje PRINT lub PLOT, OS wykona je pod warunkiem, że ich realizacja będzie następowała w identycznych liniach trybowych. W przeciwnym wypadku można uzyskać całkowicie zniekształcony obraz.

Załamanie się wykonywania instrukcji w takim wypadku może być spowodowane trzema przyczynami. Po pierwsze, BASIC może nie wykonać danej instrukcji edytorskiej, ponieważ wykonanie jej jest niemożliwe w tym trybie graficznym, w którym system operacyjny uważa, że się znajduje. OS przechowuje numer trybu graficznego, który według niego powinien znajdować się na ekranie, pod adresem $57. Jest rzeczą stosunkowo prostą oszukać OS w tym miejscu, wpisując do tego rejestru inną wartość. Należy tylko pamiętać, że wpisywane są tu numery trybów BASICu, a nie numery trybów ANTICu.

W innym przypadku załamanie może wystąpić wówczas, kiedy umieszcza się na ekranie linie trybowe o różnej ilości pamięci ekranowej potrzebnej dla jednej linii. Przecież niektóre linie trybowe wymagają aż 40 bajtów pamięci, inne 20, inne tylko 10 bajtów. Załóżmy, że wprowadzimy 20-bajtową linię trybową pomiędzy 40-bajtowe linie na ekranie, po czym instrukcją PRINT umieścimy na nim tekst. Obraz powyżej wprowadzonej linii będzie niezakłócony, lecz poniżej otrzymamy tekst przesunięty o 20 znaków w prawo. Dzieje się tak dlatego, iż OS uważa, że każda linia jest opisana 40 bajtami pamięci, które w niezakłóconej kolejności należy umieszczać jeden za drugim. Tymczasem ANTIC, w którego programie znalazła się linia 20-bajtowa, odczytuje jedynie 20 bajtów z tej części pamięci, którą OS przeznaczył dla linii 40-bajtowej, interpretując zawartość następnych 20 bajtów jako należącą do kolejnej linii i tam też je wyświetlając. Stąd właśnie bierze się owo przesunięcie tekstu o 20 znaków w prawo.

Jedynym rozwiązaniem tego problemu jest niestosowanie instrukcji PRINT i PLOT w przypadku stosowania listy displejowej zawierające j linie trybowe o zróżnicowanych wymaganiach pojemności pamięci ekranowej. Innym wyjściem (przypominającym wyjście drzwiami kuchennymi) jest organizacja ekranu w grupy linii o identycznych całkowitych wymaganiach pojemności pamięci. Tak więc, zamiast umieszczać linię 20-bajtową pomiędzy liniami 40-bajtowymi, umieśćmy dwie linie 20-bajtowe, lub jedną 20-bajtową i dwie 10-bajtowe. Dopóki zachowany zostanie warunek identyczności wymagań pamięciowych dołączanych linii trybowych i linii standardowych znajdujących się na ekranie, nie wystąpi przesunięcie tekstu lub rysunku w poziomie.

Przedstawione wyżej rozwiązanie uwypukla trzeci problem, z którym należy się liczyć przy wprowadzaniu zmian w liście displejowej podczas pracy w BASICu: przesunięcie w pionie. OS umieszcza symbole na ekranie w danym wierszu, odliczając ilość pamięci ekranowej od górnej krawędzi ekranu. Przy standardowej, 40-bajtowej linii trybowej na ekranie, BASIC zacznie umieszczać symbole w 10-tym wierszu, odliczywszy od początku pamięci ekranowej 360 (40*9) bajtów. Jeżeli do listy displejowej dopisane zostały 4 linie 10-

bajtowe, BASIC rozpocznie umieszczanie symboli o trzy wiersze niżej niż należało się tego spodziewać. Ponadto, różnorodne linie trybowe mają różną wysokość, a więc różnią się ilością linii skaningowych potrzebnych do ich utworzenia. Stąd konkretna pozycja na ekranie wczytywana z BASICu musi być ściśle skalkulowana.

Jak więc widać, użycie listy displejowej z różnorodnymi liniami trybowymi w połączeniu z systemem operacyjnym może być stosunkowo trudne. Zwykle zachodzi potrzeba oszukania OS w celu otrzymania żądanego obrazu. Aby otrzymać okno o innym trybie graficznym niż reszta ekranu, gdzie można będzie stosować instrukcje PRINT i PLOT, należy numer trybu okna (numer trybu BASICu) wpisać do adresu $57, a następnie wpisać współrzędne lewego górnego rogu okna do adresów $58 i $59. Przy trybie tekstowym kursor umieszcza się w lewym górnym rogu tegoż okna instrukcją POSITION 0,0. Przy trybach graficznych wszystkie instrukcje PLOT i DRAWT0 będą się odnosiły do lewego górnego rogu okna jako do współrzędnych 0,0.

Metody ingerencji w listę disple3ową mogą być wykorzystane do tworzenia przyciągających uwagę obrazów. Najprostszym zastosowaniem jest stworzenie obrazu z mieszanymi trybami tekstowymi i graficznymi. Na przykład, można przygotować stronę tekstową, z tytułem w trybie graficznym BASIC 2, podtytułem w trybie graficznym 1, oraz zasadniczym tekstem w trybie 0. Pomiędzy tekst w trybie 0 można oczywiście umieścić rysunek, dajmy na to w trybie graficznym 8. Wymownym przykładem stosowania tej techniki jest obraz wielu edukacyjnych programów Atari, jak chociażby program "Nazwy państw i stolic".

Wymienione wyżej problemy znacznie ograniczają stosowanie tej techniki w programach BASICu. Zastosowanie zmodyfikowanej listy displejowej w programach języka asemblera jest znacznie efektywniejsze - pozwala na organizację ekranu w szereg okien, z których każde posiada własną instrukcję LMS oraz niezależny obszar pamięci ekranowej w RAM.

ZASTOSOWANIA LISTY DISPLEJOWEJ

Jednym z prostych przykładów zastosowania modyfikacji listy displejowej jest oddzielenie poszczególnych linii tekstowych poziomymi liniami wygaszonymi. Zabieg ten można stosować w celu rozsunięcia tekstu, zwiększenia czytelności, a także podkreślenia ważniejszych informacji na ekranie.

Innym, bardzo ważnym wykorzystaniem manipulacji 1istą displejową jest możliwość wykorzystania wielu udogodnień, niedostępnych dla użytkownika pracującego w BASICu. Istnieją trzy tryby tekstowe ANTICu, niedostępne dla BASICu - jedynie modyfikacje listy displejowej pozwalają na wykorzystanie tych trybów. Poza tym ANTIC posiada możliwość wykonywania przerwań listy displejowej oraz zwijania ekranu, tak w pionie jak i w poziomie, które to opcje możliwe są tylko poprzez modyfikację listy displejowej. Będą one przedmiotem dyskusji rozdziałów 5 i 6.

Możliwość manipulowania instrukcją LMS oraz jej operandem także stwarza olbrzymie możliwości dla rzutkich programistów. Na przykład, zmiana instrukcji LMS podczas wygaszenia pionowego stwarza możliwość natychmiastowej zmiany całego obrazu. Zmiana ta może być wykonywana jednorazowo lub z niewielką prędkością, co stwarza podstawy do

wcześniejszego przygotowania kilku obrazów a następnie periodycznego ich zmieniania, bez konieczności każdorazowego rysowania całego obrazu. Każdy taki obraz pozostanie zapisany w pamięci (i niestety pochłonie jej część) nawet wówczas, kiedy nie będzie wykorzystany - i natychmiast będzie dostępny w całości. Technika ta może być ponadto wykorzystana do animacji, którą da się zrealizować poprzez szybką zamianę różnych obrazów, proszę zauważyć, że program zmiany obrazu musi manipulować jedynie 2-bajtowym adresem, by móc wykorzystać wiele tysięcy bajtów pamięci RAM.

Istnieje także możliwość nakładania na siebie obrazów poprzez szybką zmianę pamięci ekranowej. Zdolność rozróżniania obrazów przez oko ludzkie ograniczona jest stałą czasową, wynoszącą około 1/16 sekundy. Tak więc program może nałożyć na siebie cztery obrazy, przeskakując pomiędzy nimi z częstotliwością 60 razy na sekundę - każdy obraz będzie się powtarzał 15 razy na sekundę. W ten sposób można jednorazowo zmieścić na ekranie i nałożyć na siebie aż cztery obrazy. Oczywiście, istnieją pewne ograniczenia tej metody. Po pierwsze, stworzenie czterech obrazów pochłonąć może dość pokaźny obszar pamięci RAM. Po drugie, każdy przerzucany w ten sposób obraz będzie nieco rozmyty, jako że czas jego trwania wynosi zaledwie 1/15 sekundy. Muszą to więc być bardzo jasne obrazy na czarnym tle. Ponadto wykorzystanie tej metody powoduje ledwo zauważalne migotanie ekranu, męczące jednak dla oka przy dłuższym przyglądaniu się. Nic nie stoi jednak na przeszkodzie, aby wykorzystać nakładanie się na siebie trzech, czy nawet tylko dwóch obrazów. Metody tej można również zwiększyć zdolności rozdzielcze komputera zarówno jeśli chodzi o barwę obrazu, jak i o jasność obiektów. Uzyskuje się to przez cykliczną zmianę dwóch takich samych obrazów, różniących się między sobą bądź kolorem bądź skalą jasności. Na przykład, załóżmy, że chcemy otrzymać słupek o różnej jasności poszczególnych jego części. Wpiszmy najpierw następujące wartości do rejestrów kolorów:

```
COLBAK: 00
COLPF0: 02
COLPF1: 0A
COLPF2: 0C
```

Wpiszmy teraz następujące wartości do odpowiednich miejsc czterech różnych pamięci ekranowych:

```
     Obraz pierwszy  1  1  1  1  2  3  2  3  2  3  2  3
     Obraz drugi     B  1  1  1  B  B  2  3  2  3  2  3
     Obraz trzeci    B  B  1  1  B  B  B  B  2  3  2  3
     Obraz czwarty   B  B  B  1  B  B  B  B  B  B  2  3
Efektywna jasność X4 2  4  6  8 10 12 20 24 30 36 40 48
```

Otrzymany efekt:

W ten sposób można uzyskać bardziej zróżnicowane jasności poszczególnych punktów ekranu.

I na koniec jeszcze jedna sugestia, związana z tym dość trudnym zagadnieniem, ale myślę, że już teraz choć trochę zrozumiałym: lista displejowa dynamiczna. Chodzi mi tu o listę displejową, którą 6502 zmienia podczas wygaszeń pionowych. Czy można sobie wyobrazić wszystkie ciekawe efekty, jakie dadzą się uzyskać poprzez zastosowanie tak dynamicznie zmieniającej się listy displejowej? Na przykład jedno z prostszych zastosowań: tekstowy program edytorski, natychmiast wpisujący linie wygaszone powyżej i poniżej jakiegoś bloku tekstowego, oddzielające go od innych części ekranu. Linia w liście displejowej może być modyfikowana poprzez przesuwanie kursora w pionie po ekranie. Technika ta z pewnością należy do najtrudniejszych, lecz daje niezwykłe efekty.

3. POŚREDNIE TWORZENIE GRAFIKI. REJESTRY KOLORÓW I ZBIORY SYMBOLI.

Stosowanie metod pośrednich w programowaniu jest techniką dającą bardzo szerokie możliwości. W języku asemblera mikroprocesora 6502 występują trzy poziomy pośredniości w stosunku do operandów. Pierwszym, najbardziej bezpośrednim poziomem jest natychmiastowy tryb adresowania, gdzie jako operand występuje heksadecymalna liczba:

```
LDA #$F4
```

Drugi poziom pośredniości występuje wówczas, kiedy instrukcja odsyła program do adresu pamięci, gdzie przechowywana jest dana liczba:

```
LDA $0602
```

Trzecim, najwyższym poziomem pośredniości w programowaniu 6502 jest odesłanie programu do pary rejestrów adresowych, w których to zapisany jest adres komórki zawierającej daną liczbę. W przypadku 6502 ten tryb adresowania najczęściej połączony jest z dodaniem zawartości jednego z rejestrów indeksowych:

```
LDA ($D0), Y
```

Metody pośrednie zapewniają programiście większy stopień elastyczności, a jednocześnie otwierają przed nim nowe możliwości. Zamiast zapisywać w rejestrach mikroprocesora ciągle te same liczby, można je zapisać w rejestrach pamięci, a pośrednim trybem adresowania odesłać program pod wskazany adres. Zmiana takiego wektora, wskazującego adres w pamięci, może w znaczący sposób zmienić wykonywany program. Tak więc pośrednie tryby adresowania, tak silnie zredukowane w przypadku mikroprocesorów grupy 8080, są niezwykle ważną grupą technik.

REJESTRY KOLORÓW

Pośrednie tworzenie grafiki w komputerach Atari odbywa się dwiema metodami: przy pomocy rejestrów kolorów oraz zbiorów symboli. Programista, stykający się z Atari, a mający doświadczenie w pracy z innymi systemami, pod pojęciem koloru zwykle rozumie jedynie barwę punktu ukazującego się na ekranie. Tymczasem rejestr kolorów w Atari jest pojęciem o wiele bardziej złożonym. Słowo "kolor" możemy odnieść jedynie do barwy, natomiast "kolor" w rejestrze koloru jest traktowany pośrednio, może zawierać wskaźnik dowolnego "koloru". Różnica między obydwoma "kolorami" jest mniej więcej taka jak między kluczem nasadowym a kluczem francuskim. Nasadowy pasuje do nakrętek tylko jednego rodzaju, podczas gdy francuski w zasadzie do wszystkich typów. Jest co prawda trochę więcej zachodu z jego użyciem, ale za to dysponujemy narzędziem w miarę uniwersalnym. Podobnie jest z rejestrem kolorów - również wymaga nieco więcej pracy w użyciu, lecz o ileż bardzie j jest uniwersalny.

Komputery Atari 400/800 posiadają dziewięć rejestrów kolorów. Cztery przeznaczone są dla niezależnych obiektów graficznych (PMG), a z pozostałych pięciu nie wszystkie są zawsze wykorzystywane. Zależy to od trybu graficznego ekranu,niektóre tryby dają możność wykorzystania tylko dwóch rejestrów, inne zaś wszystkich pięciu.

W trybie graficznym 0 BASICu wykorzystywane są 2 i pół rejestru kolorów - ignorowana jest barwa odnosząca się do symboli na ekranie. Barwa symboli, podobnie jak barwa ekranu tekstowego, odczytywana jest z rejestru PF2, lecz jasność symboli na ekranie odczytywana jest z rejestru PF1. Rejestry kolorów znajdują się w obszarze pamięci układu CTIA (lub GTIA) pod adresami od $D016 do SD01A. Są one cieniowane (przepisywane) przez OS z adresów RAM podczas każdego wygaszenia pionowego. Tabela 3-1 przedstawia adresy sprzętowe oraz adresy cieni wszystkich rejestrów kolorów.

Obiekt rejestru	Sprzęte		Cień OS	
	Nazwa	Adres	Nazwa	Adres
Obiekt 0	COLPM0	$D012	PCOLR0	$2C0
Obiekt 1	COLPM1	$D013	PCOLR1	$2C1
Obiekt 2	COLPM2	$D014	PCOLR2	$2C2
Obiekt 3	COLPM3	$D015	PCOLR3	$2C3
Pole 0	COLPF0	$D016	COLOR0	$2C4
Pole 1	COLPF1	$D017	COLOR1	$2C5
Pole 2	COLPF2	$D018	COLOR2	$2C6
Pole 3	COLPF3	$D019	COLOR3	$2C7
Tło	COLBAK	$D01A	COLOR4	$2C8

Tabela 3-1 Nazwy i adresy rejestrów koloru.

W większości przypadków użytkownik steruje rejestry kolorów poprzez zmianę zawartości adresów cieni. Tylko w dwóch przypadkach użytkownik powinien bezpośrednio adresować rejestry CTIA. Pierwszym i najczęściej stosowanym jest wykorzystywanie przerwań listy displejowej, które zostaną omówione w rozdziale 5. Drugi ma miejsce wówczas, kiedy użytkownik wstrzymuje wszelkie procedury, wykonywane normalnie przez OS podczas wygaszenia pionowego - w tym także i te, które przepisują zawartości rejestrów cieni do rejestrów kolorów GTIA. Przerwania wygaszenia pionowego dyskutowane będą w rozdziale 8.

Kolory w rejestrach kolorów kodowane są w bardo prosty sposób. Starszy nibel odpowiada kodowi barwy, który jest identyczny z drugim parametrem występującym po instrukcji SETCOLOR w BASICu. Kody poszczególnych barw przedstawione były niemal w każdej książce, np. w podręczniku "BASIC Reference Manual", tab. 9-3. Młodszy nibel zawiera kod jasności (luminancji) danej barwy. Kod ten jest identyczny z trzecim parametrem występującym po instrukcji SETCOLOR w BASICu, Najmłodszy bit jest nieznaczący, stąd też możliwych jest

tylko osiem różnych jasności danego koloru. W sumie mamy do wyboru paletę 128 barw (8 jasności x 16 barw). W dalszej części tej książki terminem "kolor" będziemy określali kombinowany kod, dotyczący zarówno barwy jak i jasności.

Jednorazowe zapisanie określonego kodu w danym rejestrze koloru wystarczy, by każdy punkt rysowany na ekranie posiadał barwę i jasność, precyzowaną przez odnośny rejestr. W trybach graficznych kolorowych, wykorzystujących cztery rejestry kolorów, dane z pamięci ekranowej konkretyzują, w jakim kolorze ma być wyświetlony kolejny punkt. Ponieważ używane są cztery rejestry kolorów, tylko dwa bity wystarczą do określenia, z którego rejestru ma być odczytana barwa i jasność danego punktu. Tak więc każdy bajt pamięci ekranowej przechowuje dane dotyczące czterech punktów - każdy punkt jest kodowany dwubitowo.

W trybach tekstowych (tryby BASICu 1 i 2) wybór rejestru kolorów zakodowany jest w dwóch najstarszych bitach kodu danego symbolu. Stąd na zdefiniowanie symbolu pozostaje tylko 6 bitów i fakt ten jest przyczyną, dla której w tych trybach dostępny jest zbiór zaledwie 64 symboli.

Pośrednie adresowanie rejestrów kolorów otwiera przed programistą cztery specjalne możliwości. Po pierwsze, indywidualny wybór jednego z dostępnych 128 kolorów umożliwia precyzyjne dobranie odpowiedniej barwy.

Po drugie, manipulowanie rejestrami kolorów w czasie rzeczywistym pozwala na tworzenie ciekawych efektów. Najprostszym tego przykładem jest następująca linia programu BASICu:

```
FOR I=0 TO 254 STEP 2: POKE 712, I : NEXT I
```

Procedura ta zmienia po prostu barwę ramki ekranu przez wszystkie możliwe kolory. jest to dość miły dla oka i z pewnością przyciągający uwagę efekt. Ta właśnie prosta technika zmiany kolorów może być stosowana na wiele różnych sposobów. Przy pewnych modyfikacjach może ona być także zastosowana przy tworzeniu animacji poprzez zmianę zawartości rejestru koloru przypisanego danemu obrazowi zamiast przez zmianę samego obrazu. Poniższy przykład ilustruje tę ideę:

```
10 GRAPHICS 23
20 FOR X=0 TO 39
30 FOR I=0 To 3
40 COLOR I
50 PLOT 4*X+I,0
60 DRAWTO 4*X+I,95
70 NEXT I
80 NEXT X
90 A=PEEK(712)
100 POKE 712,PEEK(710)
110 POKE 710,PEEK(709)
120 POKE 709,PEEK(708)
130 POKE 708,A
140 GOTO 90
```

Trzecim zastosowaniem rejestrów kolorów jest logiczne stosowanie kolorów w zależności od sytuacji. Na przykład, wielostronicowe menu może być o wiele czytelniejsze przez zmianę koloru tła czy koloru ramki w każdej stronie menu; ekran może rozbłyskać czerwono przy naciśnięciu niewłaściwego klawisza. Wykorzystanie różnokolorowych liter z trybów BASICu 1 i 2 może w znacznym stopniu uatrakcyjnić tekst na ekranie. Ważne słowa czy zwroty w tekście mogą zostać wyświetlone w odmiennym kolorze, co uwypukli je na tle reszty tekstu.

Wykorzystanie kolorów w trybach graficznych, nie tekstowych, jest chyba dla wszystkich oczywiste. Każdy obiekt na obrazie może być przedstawiany w kilku różnych kolorach, reprezentujących na przykład różnej ego odmiany. Oczywiście, tworzenie każdego nowego obiektu ekranowego pochłania dość duży obszar pamięci RAM, ale zmiana koloru istniejącego obiektu nie kosztuje prawie wcale. Jeszcze jeden przykład: stworzenie trzech różnych łodzi na obrazie wyścigu wioślarskiego zajmie z pewnością o wiele więcej pamięci, niż stworzenie obrazu jednej tylko łodzi i wyświetlenie jej w różnych miejscach ekranu w odmiennych kolorach.

Czwarte i najważniejsze zastosowanie rejestrów kolorów wiąże się nierozerwalnie z użyciem przerwań listy displejowej. W takim przypadku jeden rejestr koloru może być wykorzystany do wyświetlenia na tym samym obrazie nawet wszystkich 128 kolorów, Ta niezwykle prężna technika omówiona zostanie w rozdziale 5.

ZBIORY SYMBOLI

Pośrednie tworzenie grafiki jest możliwe także poprzez użycie redefiniowalnych zbiorów symboli. Standardowy zbiór symboli znajduje się w ROM, nie istnieją jednak żadne przyczyny, dla których tylko ten zbiór symboli musiałby być wykorzystywany. Użytkownik dla własnych potrzeb może zdefiniować dowolny zbiór symboli. Stworzenie własnego zbioru symboli wymaga trzech następujących etapów. Przede wszystkim zbiór taki musi zostać zdefiniowany przez programistę; jest to najbardziej czasochłonny etap pracy. Każdy symbol wyświetlany jest na ekranie w obszarze rastrowym 8x8 punktów; w pamięci jest on kodowany jako 8-bajtowa tablica. Rysunek 3-2 wyjaśnia sposób kodowania symboli w pamięci.

Symbol	Kod binarny	Kod hex.
	00000000	00
	00011000	18
	00111100	3C
	01100110	66
	01100110	66
	01111110	7E
	01100110	66
	00000000	00

Rys. 3-2. Kodowanie symboli

Pełny zbiór zawiera 128 różnych symboli, drugie 128 to symbole o odwróconych barwach (inverse video). Taki zbiór zajmuje 1024 bajty pamięci (1K) i musi znajdować się w całości w jednym przedziale 1K pamięci. Zbiór symboli dla trybów graficznych BASICu 1 i 2 obejmuje jedynie 64 symbole, wymaga więc tylko 512 bajtów pamięci i może być umieszczony w jednej z dwóch połówek 1K przedziału pamięci. Pierwszych 8 bajtów opisuje symbol zerowy, następnych 8 symbol pierwszy zbioru itd. Jak więc stąd widać definiowanie własnego zbioru symboli jest bardzo żmudną pracą. Na szczęście w wielu przypadkach dostępne są na rynku programy ułatwiające znacznie tę pracę.

Jeżeli zbiór symboli został już zdefiniowany i umieszczony w pamięci komputera, należy jeszcze wskazać mikroprocesorowi ANTIC gdzie należy go szukać. Uzyskuje się to przez wpisanie numeru strony pamięci, gdzie rozpoczyna się zbiór, do adresu SD409 (dziesiętnie 54281). Adres cienia OS, który będzie normalnie używany podczas pracy, zwanego CFMBAS, wynosi $2F4 (dziesiętnie 756). Trzecim etapem jest umieszczenie zdefiniowanego zbioru symboli na ekranie. Wykonuje się to bądź zwyczajną komendą PRINT BASICu, bądź też przez bezpośrednie wpisywanie kodów symboli do obszaru pamięci ekranowej.

Dodatkową możliwością systemu Atari, niedostępną dla BASICu, jest stworzenie symboli 4-kolorowych. W trybach graficznych BASICu 1 i 2 dostępnych jest łącznie 5 kolorów, ale każdy symbol tego zbioru jest symbolem wyłącznie 2-kolorowym: określa się kolor symbolu oraz kolor tła; kolor symbolu może być jednym z 4 dostępnych kolorów, ale równocześnie na ekranie pojedynczy symbol może być tylko jednobarwny. Stanowić to może szczególną niedogodność przy posługiwaniu się zbiorem symboli graficznych.

Istnieją dwa specjalne tryby tekstowe, predysponowane właśnie do wyświetlania symboli graficznych. Są to tryby ANTICu 4 i 5. Każdy symbol w tych trybach ma szerokość tyko czterech punktów rastrowych ekranu, lecz dla każdego z punktów rastrowych można niezależnie wybrać jeden z czterech kolorów (włączając w to kolor tła). Symbole definiowane są w taki sam sposób, jak przedstawiona wyżej metoda dla trybu graficznego 0 BASICu, poza tym, że punkt rastrowy ekranu jest w trybach tych dwa razy szerszy, a dwa bity w kodzie symbolu precyzują rejestr koloru, w jakim dany punkt ma być wyświetlony. W odróżnieniu od trybów ANTICu 6 i 7 (tryby BASICu 1 i 2) wybór rejestru koloru nie jest określony przez sam kod symbolu, ale poprzez definiowanie zbioru symboli. Każdy bajt tablicy symboli czytany jest jako 4 2-bitowe pary, z których każda określa kolor danego punktu rastrowego (z tego powodu każdy symboli może mieć szerokość tylko czterech punktów rastrowych). Najstarszy bit pierwszego bajta w 8-bajtowej tablicy kodu symbolu modyfikuje zakres wykorzystywanych rejestrów kolorów. Wybór rejestru koloru dokonuje się następująco:

Para bitów w kodzie symbolu	D7=0	D7=1
00	COLBAK	COLBAK
01	COLPF0	COLPF0
10	COLPF1	COLPF1
11	COLPF2	COLPF3

Przy użyciu tych trybów tekstowych na ekranie można umieszczać wielobarwnie symbole graficzne.

Innym interesującym trybem tekstowym ANTICu jest tryb małych liter (tryb ANTICu 3). Tryb ten zajmuje dla jednej linii trybowej 10 linii skaningowych, a więc standardowe znaki Atari, mające wysokość 8 linii skaningowych, wyświetlane będą w górnej części linii trybowej, pozostawiając poniżej 2 wolne linie skaningowe. Jeżeli używać będziemy znaków z ostatniej ćwierci standardowego zbioru symboli, wówczas 2 wolne linie skaningowe będą do dyspozycji powyżej wyświetlanych znaków. Pozwala to użytkownikowi na zdefiniowanie własnego zbioru znaków pisarskich, na przykład obejmującego litery alfabetu polskiego: ą, ę; ś, ń, ć itp.

ZASTOSOWANIA ZBIORÓW SYIMBOLI

Wykorzystanie zbiorów symboli w pośrednim tworzeniu grafiki stwarza wiele interesujących i użytecznych możliwości. Jednym z najczęściej spotykanych zastosowań jest modyfikacja standardowego zbioru znaków pisarskich, bądź też wprowadzenie całkowicie nowego zbioru, dającego unikalne możliwości. W ten sposób można zaprojektować zbiory liter alfabetu greckiego, cyrylicy, czy też zbiór specjalnych znaków graficznych. Metoda ta zapewnia także całkowitą dowolność w tworzeniu nowych zbiorów. Na przykład program "Energy Czar" stosuje własny zbiór symboli do tworzenia na ekranie wykresów słupkowych. Jeden symbol w zbiorze standardowym ma szerokość ośmiu punktów rastrowych ekranu, a więc bezpośrednie zmodyfikowanie tego zbioru dałoby możliwość rysowania słupków na rastrze 8 punktowym, co stanowi raczej niewielką dokładność. Program "Energy Czar" wykorzystuje zbiór standardowy, zmodyfikowany w ten sposób, że najmniej używane symbole (symbol funta, paragrafu itp.) zastąpione zostały specjalnymi symbolami do rysowania słupków - niektóre z nich mają szerokość 8 punktów, inne 2, a jeszcze inne 1 punktu, co pozwala na tworzenie słupków o wysokości mierzonej z dokładnością do 1 punktu ekranu. Obraz mieszany, tekstowo-graficzny, jest w tym przypadku nieco mylący. Program wykorzystuje wyłącznie tekstowy tryb graficzny, a wykresy tworzy się poprzez modyfikację standardowego zbioru symboli.

W wielu zastosowaniach symbole, wchodzące w skład zmodyfikowanego zbioru symboli, mogą mieć znaczenie specjalne. Na przykład zdefiniowanie zbioru symboli, odpowiadających znakom kartograficznym, pozwala na uzyskania obrazu mapy dowolnego terenu. Najlepsze rezultaty uzyskuje się przy zdefiniowaniu pięciu do ośmiu znaków dla danego typu terenu. Przy czym należy tak dobrać znaki, by możliwość kombinowania ich na ekranie była dodatkowym źródłem symboli. Mieszanie prostych symboli i łączenie ich w jeden znak pozwala uniknąć monotonii, jaka jest charakterystyczna dla znaków jednakowej wielkości i kroju. W większości dobrze zaprojektowanych obrazów bardzo trudno dostrzec jest, iż skonstruowane one zostały w oparciu o zbiory symboli. Przykładem takiej grafiki może być obraz gry "Eastern Front 1941 ", gdzie ekran, z występującymi na nim jednocześnie 18 kolorami, w żaden sposób nie przypomina standardowego obrazu trybu tekstowego komputera.

Często spotyka się także zbiór symboli dotyczących elementów elektronicznych - ze znakami odpowiadającymi tranzystorom, diodom,

rezystorom itp. - przy pomocy którego można wyświetlać schematy układów elektronicznych. Na tej samej zasadzie można stworzyć zbiór symboli architektonicznych, wykorzystując go w programach projektowych lub kopiujących. Proszę wierzyć, że wszystkie możliwości, otwierające się poprzez pośrednie tworzenie grafiki w oparciu o modyfikacje i zmiany zbiorów symboli, nie zostały i nie zostaną chyba nigdy do końca wyeksploatowane.

Umieszczenie na ekranie symboli odwróconych do góry nogami jest możliwe przez wpisanie wartości 4 do adresu 755. Jednym z zastosowań tego typu edycji może być stworzenie na ekranie obrazu kart do gry. W górnej części ekranu powinien pojawić się obraz normalny, a przez wykorzystanie przerwań listy displejowej poniżej można stworzyć odwróconą część karty. Możliwość odwracania symboli może być wykorzystana także do tworzenia odbić lustrzanych, na przykład odbić obrazu w tafli jeziora.

Jeszcze szersze możliwości otwierają się przed programistą, który uzmysłowi sobie, iż jest w stanie dowolnie przebudować zbiór symboli w trakcie wykonywania programu. Cały zbiór symboli zajmuje w pamięci 512 bądź 1024 bajty. W najprostszym przypadku można stworzyć kilka zbiorów symboli i w trakcie wykonywania programu przełączać ANTIC na jeden z tych zbiorów. Przy przełączaniu na nowy zbiór symboli należy wziąć pod uwagę trzy różne reżimy czasowe: wolny w stosunku do odbiorcy (ponad 1 sekundę), szybki w stosunku do odbiorcy (od 1/60 do 1 sekundy) oraz maszynowy (szybszy od 1/60 sekundy).

Powolne w stosunku do odbiorcy przełączanie zbiorów jest przydatne do zmiany scenerii obrazu: Na przykład obraz podróży kosmicznej może wykorzystywać jeden zbiór symboli dla opisania jednej planety, drugi dla podróży w przestrzeni, a trzeci dla innej planety. W trakcie trwania programu (podróży) przełączane są zbiory symboli, tworzące nową, egzotyczną scenerię. Program przygodowy może zmieniać scenerię na przykład przy przechodzeniu z jednego do drugiego pomieszczenia.

Szybka w stosunku do odbiorcy zmiana zbiorów symboli ma zastosowanie przede wszystkim podczas tworzenia animacji. Wykorzystuje się tu dwie metody: szybkie zmiany symboli w obrębie jednego zbioru lub też szybką zmianę całego zbioru symboli. Na przykład w programie "Space Invaders" wykorzystano tę pierwszą metodę. Postacie najeźdźców opisane są jednym zbiorem symboli, a szybka zmiana symbolu odpowiadającego jednej z postaci pozwoliła programiście na stworzenie animacji. Nie było to trudne, jako że w programie występuje jedynie sześć różnych postaci, z których każda przyjmować może cztery różne formy.

Szybka, cykliczna animacja całego ekranu możliwa jest przez zdefiniowanie wielu zbiorów symboli, stworzenie postaci ekranowych, a następnie przełączanie komputera na kolejny zbiór symboli. Jeżeli w każdym ze zbiorów symboli każdy z symboli będzie miał zupełnie inny wygląd, wówczas każdy z nich będzie się pojawiał na ekranie sekwencyjnie podczas szybkiego przełączania zbiorów. Jest to metoda na całkowite zapełnienie ekranu wieloma różnymi symbolami, poprzez szybkie przełączanie zbiorów w bardzo prostej pętli programowej. Po zdefiniowaniu wszystkich symboli w dajmy na to 10 różnych zbiorach i umieszczeniu całego zapisu w pamięci, szybką ich zmianę można uzyskać przez tak prostą pętlę jak poniższa:

```
1000 FOR I=1 TO 10
1010 POKE 756,CHARBASE(I)
1020 NEXT I
1030 GOTO 1000
```

Szybkie, na poziomie maszynowym przełączanie zbiorów symboli stosuje się właśnie w celu umieszczenia jednorazowo na ekranie wielu obiektów. Można do tego wykorzystać takie przerwania listy displejowej, które będą przedmiotem rozdziału 5.

Wykorzystanie zbiorów symboli do pośredniego tworzenia grafiki i animacji otwiera niezwykle szerokie możliwości, lecz posiada także swoje ograniczenia. Największą zaletą tej techniki jest fakt, iż pochłania ona bardzo niewiele pamięci RAM. Obraz graficzny oparty na zbiorze znaków z trybu tekstowego 2 BASICu (jak np. w "Eastern Front 1941") stwarza możliwość umieszczenia na ekranie wielu detali oraz daje jeden kolor więcej niż tryb 7 BASICu. Ponadto opis symbolu w tym przypadku zajmuje 200 bajtów, podczas gdy stworzenie analogicznego symbolu w podobnym, nietekstowym trybie graficznym pochłonęłoby aż 4000 bajtów. Standardowy zbiór symboli w tym trybie zajmuje po 8 bajtów dla 64 symboli, a więc tylko 512 bajtów dla jednego zbioru. Wynika stąd, że stworzenie kilku różnych zbiorów symboli nie zajmuje zbyt dużego obszaru pamięci RAM. Manipulacje z ekranem, opartym na grafice zbiorów symboli, są szybsze, ponieważ manipuluje się o wiele mniejszą ilością danych ekranowych. Z drugiej strony grafika taka nie daje dużych możliwości rozdzielczych z powodu ograniczeń dość dużej siatki rastrowej. Nie można każdego symbolu umieścić w dowolnym miejscu ekranu. W wielu zastosowaniach właśnie to ograniczenie uniemożliwia wykorzystanie grafiki opartej na zbiorach symboli. Istnieje jednak bardzo wiele zastosowań graficznych, gdzie program musi umieścić jedynie ograniczoną liczbę zdefiniowanych uprzednio symboli w określonych siatką rastrową miejscach ekranu. W takich przypadkach wykorzystanie trybu tekstowego ze zmodyfikowanym zbiorem symboli otwiera o wiele szersze możliwości niż tryby graficzne.

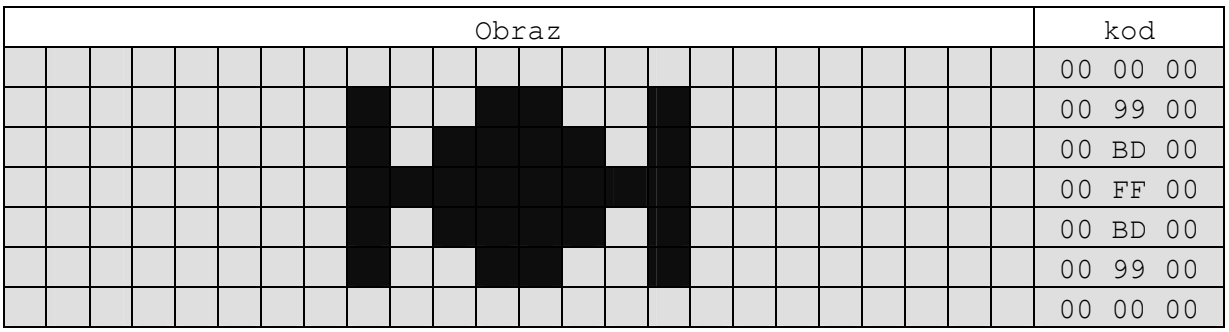
4. GRAFIKA OBIEKTÓW EKRANOWYCH (PMG, PLAYER-MISSILE GRAPHICS)

TRUDNOŚCI W TWORZENIU SZYBKIEJ ANIMACJI

Animacja jest niezwykle ważną funkcją wszystkich systemów komputerów domowych. Aktywność obrazu na ekranie jest nie tylko środkiem przyciągania uwagi, ale nadaje obrazowi cechy realistyczne. Właśnie animacja jest cechą zasadniczą wszystkich gier komputerowych. A co chyba jeszcze ważniejsze, obraz ruchomy przekazuje informacje w bardziej bezpośredni i bardziej wyrazisty sposób niż obraz statyczny. Przede wszystkim zwraca on uwagę na tę część ekranu, która w danej chwili może zawierać najważniejsze informacje. O wiele lepiej ukazuje jakikolwiek proces dynamiczny, bo bezpośrednio, aniżeli najlepszy nawet, lecz pośredni opis tegoż procesu. Zdolności animacyjne muszą być traktowane jako jedna z ważniejszych możliwości graficznych każdego systemu komputerowego.

Konwencjonalną metodą tworzenia animacji w komputerach domowych jest metoda przemieszczania danych, dotyczących obiektu na ekranie, w przestrzeni pamięci ekranowej. Jest to proces dwustopniowy. Najpierw program musi zlikwidować pierwotny obraz, wpisując do pamięci ekranowej, gdzie był on utrwalony, dane dotyczące tła. Następnie program musi umieścić dane dotyczące obrazu w innej części pamięci, odpowiadającej nowej pozycji obiektu na ekranie. Powtarzanie tego procesu stwarza wrażenie ruchu obiektu po ekranie.

Technika ta wiąże się z dwoma problemami. Po pierwsze, jeżeli animacja wykonywana jest w trybie graficznym o stosunkowo dużej siatce rastrowej, nie otrzyma się ruchu płynnego - wyraźnie będzie widoczne przeskakiwanie obrazu. W innych systemach komputerowych jedynym rozwiązaniem tego problemu jest wykorzystanie trybu graficznego o większej rozdzielczości, czyli o mniejszej siatce rastrowej. Drugi problem jest znacznie poważniejszy. Obraz na ekranie jest dwuwymiarowy, lecz organizacja pamięci ekranowej RAM jest (bo musi być) jednowymiarowa. Oznacza to, że obraz spójny na ekranie nie będzie spójny w pamięci - będzie zapisany w rozrzuconych, oddzielonych od siebie adresach pamięci ekranowej. Zróżnicowanie to ilustruje rysunek 4-1.



Rzeczywista kolejność kodów w pamięci ekranowej RAM:
00 00 00 00 99 00 00 BD 00 00 FF 00 00 BD 00 00 99 00 00 00 00

Rys. 4-1. Nieciągłość zapisu w pamięci ekranowej RAM.

Znaczenie tego zróżnicowania staje się jasne dopiero wówczas, kiedy samemu próbuje się napisać program poruszający tego typu obraz. Proszę zwrócić uwagę, jak rozrzucone są kody zapisu w pamięci ekranowej. Aby wykasować ten zapis, program musi szczegółowo obliczyć adresy takiego zapisu, a kalkulacje takie są często bardzo trudne do wykonania. Kod asemblera, przemieszczający w pamięci ekranowej jeden tylko bajt, znajdujący się w pamięci na pozycji x,y (XPOS, YPOS) wygląda jak przedstawiono to poniżej. A więc program ten, zajmujący 40 bajtów pamięci, jest w stanie przemieścić zaledwie jedną linię trybową:

```
LDA SCRNRM      adres początku pamięci ekranowej
STA POINTR      wskaźnik ze strony zerowej
LDA SCRNRM+1    starszy bajt adresu
STA POINTR+1    starszy bajt wskaźnika.
LDA #$00
STA TEMPA+1     rejestr tymczasowy
LDA YPOS        pozycja w pionie
ASL A           razy 2
ROL TEMPA+1     przeniesienie pożyczki do TEMPA+1
ASL A           razy 4
ROL TEMPA+1     ponowne przeniesienie pożyczki
ASL A           razy 8
ROL TEMPA+1     ponowne przeniesienie
LDX TEMPA+1     zapisanie YPOS*8
STX TEMPB+1     w rejestrze TEMPB
STA TEMPB       młodszy bajt
ASL A           razy 16
ROL TEMPA+1
ASL A           razy 32
ROL TEMPA+1
CLC
ADC TEMPB       dodanie YPOS*8 w celu otrzymania YPOS*40
STA TEMPB
IDA TEMPA+1     obliczenie starszego bajta
ADC TEMPB+1
STA TEMPB+1
LDA TEMPB       TEMPB zawiera odległość od szczytu ekranu do
CLC                   danego punktu
ADC POINTR
STA POINTR
LDA TEMPB+1
ADC POINTR+1
STA POINTR+1
LDY XPOS
LDA (POINTR),Y
```

Jak widać, powyższy kod maszynowy, służący do obliczenia przesunięcia punktów jednego wiersza ekranowego, jest dosyć

nieporęczny Z pewnością nie jest to najszybsza i najbardziej przejrzysta metoda rozwiązania problemu. Być może dobry programista mógłby nieco przeorganizować i skrócić tę procedurę. Mimo wszystko program obliczania przesunięcia jest zbyt obszerny, by można go było zastosować do tworzenia szybkiej animacji: Przedstawiona procedura zajmuje około 100 cykli maszynowych, w czasie których przenoszony jest tylko jeden wiersz ekranu. A więc przemieszczenie na ekranie obrazu, który w pamięci ekranowej zajmuje, powiedzmy, 50 bajtów, wymaga wykonania 100 takich obliczeń, czyli około 10000 cykli maszynowych, co stanowi około 10 milisekund: Może nie wydaje się to zbyt długim czasem, ale uzyskanie płynnego ruchu po ekranie wymaga przesuwania obiektu co najwyżej 17 milisekund, a jeżeli chcemy przemieszczać nie jeden obiekt lub wykonywać w tym samym czasie jakieś inne obliczenia, zaczyna nam brakować czasu. Wynika stąd że tego typu animacja (zwana animacją polową) jest w wielu przypadkach za wolna. Można, oczywiście, stosować ten proces, lecz jest się wówczas ograniczonym albo ilością obiektów albo wielkością przesuwanych obiektów, albo szybkością przemieszczania, albo też niemożnością wykonania innych obliczeń w tym samym czasie. A więc warunki stosowania tego typu animacji wiążą się z wieloma ograniczeniami.

ZAŁOŻENIA GRAFIKI OBIEKTÓW EKRANOWYCH

Rozwiązaniem przedstawionego wyżej problemu, zastosowanym w komputerach domowych Atari, jest grafika obiektów ekranowych. Aby zrozumieć zasadę tworzenia obiektów ekranowych, przypomnijmy główny problem, z którym borykała się animacja polowa: przemieścić należało dwuwymiarowy obiekt ekranowy, zapisany w pamięci ekranowej w postaci jednowymiarowej. Rozwiązaniem tego problemu było stworzenie obiektu ekranowego, będącego jednowymiarowym tak na ekranie jak i w zapisie pamięci ekranowej. Obiekt taki (nazywany Player) zapisywany jest w pamięci jako tablica o długości 128 bądź 256 bajtów, która przenoszona jest bezpośrednio na ekran. Pojawia się ona jako pionowy pas, rozciągający się od szczytu do dna ekranu, a każdy bajt pamięci odpowiada jednej lub dwóm poziomym liniom skaningowym, przy czym wybór należy w tym miejscu do użytkownika. Obraz wewnątrz pasa jest bezpośrednim przeniesieniem danych z pamięci na ekran. Jeżeli bit w pamięci równy jest 1, odpowiadający mu punkt pasa pionowego zostaje zapalony; jeżeli bit równy jest 0, punkt pozostaje wygaszony. Jak więc widać, obraz ekranowy w rzeczywistości nie jest jednowymiarowy, a posiada szerokość odpowiadającą 8 bitom pamięci.

W metodzie tej projektowanie obiektów jest niezwykle proste. Najpierw należy żądany obraz rozrysować na papierze milimetrowym, pamiętając, że nie może on być szerszy od 8 punktów rastrowych ekranu. W drugiej kolejności należy przepisać obiekt w binarnym kodzie pamięci ekranowej, zastępując jedynkami punkty, które mają być zapalone. Teraz trzeba obliczyć wartości kolejnych bajtów (w systemie dziesiętnym lub heksadecymalnym, w zależności od wybranego trybu pracy). Wreszcie bajty kodu należy wpisać do pamięci, przeznaczonej na pamięć P1, rozpoczynając od bajta najwyższego wiersza obiektu i następnie wiersza drugiego itd. Im dalej w tym obszarze RAM zostanie bajt zapisany, tym niższy wiersz obiektu będzie opisywał.

PRZEMIESZCZANIE W PIONIE

Przemieszczanie takiego obiektu jest niezwykle proste. Ruch w pionie uzyskuje się poprzez przepisywanie danych wewnątrz obszaru pamięci PMG. Teoretycznie jest to ta sama metoda, którą stosuje się przy animacji polowej, w praktyce jednak występują znaczne różnice: procedura przemieszczania obiektu jest jednowymiarowa, a nie dwuwymiarowa jak przy animacji polowej. Procedura nie musi mnożyć współrzędnych każdego punktu przez 40, a jeszcze częściej nie musi nawet wykorzystywać pośrednich trybów adresowania. Może być nawet tak proste, jak przedstawiona poniżej:

```
          LDX #$01
PĘTLA     LDA PLAYER,X
          STA PLAYER-1,X
          INX
          BNE PĘTLA
```

Procedura ta potrzebuje jedynie 4 milisekund, aby przesunąć w pionie cały obiekt - jest to więc o połowę krótszy czas niż w przypadku animacji polowej, która, dla przypomnienia, przemieszczała 50 bajtów, podczas gdy ta przepisuje 256 bajtów. Jeżeli zachodzi potrzeba bardzo szybkiej animacji, każdą tego typu pętlę można ograniczyć tak, by przemieszczała jedynie zaprojektowany obiekt ekranowy, a nie cały pas playera. W takim przypadku czas przesuwania skrócony zostania do 100-200 mikrosekund. Jak więc widać, stworzenie pionowego ruchu obiektu ekranowego jest zarówno szybsze, jak i prostsze od przesuwania obiektów polowych.

PRZEMIESZCZANE W POZIOMIE

Przemieszczanie obiektu w poziome wydaje się być jeszcze łatwiejsze od animacji ruchu pionowego. W pamięci komputera znajduje się specjalny rejestr, nazywamy rejestrem pozycji poziomej playera. Wartość zapisana w tym rejestrze umiejscawia poziomą pozycję obiektu na ekranie. Przepisanie tej wartości powoduje, że obiekt wyświetlany jest natychmiast w innym miejscu ekranu. Tak więc ruch poziomy obiektu otrzymuje się poprzez cykliczne przepisywanie zawartości wspomnianego rejestru.

Przemieszczanie obiektu w pionie i w poziomie są procesami niezależnymi, które można na siebie nakładać w dowolny sposób.

Skala wielkości, zapisanych w rejestrze pozycji poziomej playera, odpowiada jednemu punktowi w poziomej linii skaningowej ekranu. Stąd zwiększenie zawartości rejestru o jednostkę spowoduje przesunięcie obiektu o jeden punkt linii skaningowej w prawo na ekranie. W jednej linii skaningowej na ekranie znajduje się 228 punktów, z czego część jest niewidoczna, wyświetlana poza ekranem. Wartość wpisana do rejestru może się wahać od 4 do 255, a więc niektóre z nich będą odpowiadały pozycjom znajdującym się poza granicami ekranu. Przy standardowym ekranie lewemu skrajnemu brzegowi pola odpowiada wartość 47, prawemu natomiast pozycja 208. Tak więc widzialna część ekranu odpowiada zawartościom rejestru od 47 do 208. Należy jednak pamiętać, że granice te mogą zależeć od typu i egzemplarza

odbiornika telewizyjnego. Tradycyjnie opisuje się pozycje playera na ekranie wartościami z przedziału od 60 do 200, choć w niektórych przypadkach przedział ten bywa rozszerzany. Technika ta stwarza jeszcze jedną dogodną możliwość: najszybszą metodą usunięcia obiektu z ekranu jest przypisanie mu pozycji poziomej 0 - wymaga to jedynie dwóch instrukcji asemblera (bądź jednego POKE w programie BASICu).

MOŻLIWOŚCI TECHNIKI OBIEKTÓW EKRANOWYCH

Technika, opisana pokrótce powyżej, umożliwia tworzenie szybkiej animacji. Lecz poza opisanymi istnieje jeszcze szereg udogodnień, które znacznie rozszerzają jej możliwości. Pierwszym z nich jest możliwość zaprojektowania czterech całkowicie niezależnych od siebie obiektów ekranowych. Każdy z nich będzie posiadał własne niezależne rejestry sterujące oraz własny obszar pamięci PMG, co pozwala na całkowicie niezależne operowanie poszczególnymi obiektami. Oznacza się je kodami od PO do P3. Można obiekty te wykorzystać łącznie, stwarzając jeden obiekt o szerokości 32 punktów rastrowych, bądź niezależnie jako 4 obiekty o szerokości 8 punktów każdy.

Każdy z obiektów posiada swój własny rejestr kolorów, całkowicie niezależny od rejestrów kolorów używanych do projektowania pola. Rejestry kolorów obiektów nazywane są COLP(X) i cieniowane są rejestrami PCOLR(X). Stwarza to możliwości uzyskania dodatkowych kolorów na ekranie. Jednakże każdy z obiektów może być tylko jednobarwny - stworzenie obiektów wielobarwnych nie jest możliwe bez wykorzystania przerwań listy displejowej (patrz rozdział 5).

Każdy z obiektów posiada projektowalną szerokość - szerokość normalną, podwójną lub poczwórną ustala się, zmieniając zapis w odpowiednim rejestrze SIZEP(X). A więc istnieje możliwość zaprojektowania obiektów o różnych wielkościach. Ponadto programowalna jest rozdzielczość - przy rozdzielczości jednoliniowej jeden bajt pamięci PMG opisuje osiem punktów jednej linii skaningowej; przy rozdzielczości dwuliniowej jeden bajt opisuje punkty w dwóch sąsiednich liniach skaningowych. Stąd właśnie przy rozdzielczości jednoliniowej tablica PMG zajmuje 256 bajtów pamięci, natomiast przy rozdzielczości dwuliniowej tablica PMG zajmuje w RAM tylko 128 bajtów; Jest to jedyna opcja, która nie jest niezależna dla poszczególnych obiektów - wybór rozdzielczości dotyczy wszystkich obiektów jednocześnie. Jest on ustawiany przez bit D4 rejestru DMACTL. Przy rozdzielczości jednoliniowej początkowe 32 bajty pamięci PMG odpowiadają liniom znajdującym się poza standardowym polem ekranowym, natomiast ostanie 32 bajty opisują linie leżące poniżej tego pola. Przy rozdzielczości dwuliniowej liniom leżącym powyżej i poniżej standardowego pola ekranowego odpowiadają początkowe i końcowe 16 bajtów pamięci PMG.

POCISKI (MISSILES)

Kolejnym udogodnieniem techniki PMG są inne obiekty ekranowe, zwane pociskami (missiles). Są to obiekty analogiczne do playerów, opisywane pasem ekranu mapowanym szerokością 2-bitową, Każdy z pocisków związany jest z jednym z playerów; jego kolor także jest opisywany rejestrem kolorów playera. Dane dotyczące kształtu pocisków opisywane są identyczną tablicą w pamięci PMG, znajdującą

się tuż przed tablicami playerów. Mogą one być przemieszczane na ekranie niezależnie od playerów, jako że posiadają niezależne rejestry pozycji poziomej. Posiadają także niezależne rejestry opisujące szerokość pocisku, SIZEM, działające analogicznie jak rejestry szerokości playerów, SIZEP(X). Jedyną różnicą jest fakt, iż wszystkie pociski opisywana są jedyna rejestrem, nie mogą więc różnić się między sobą rozmiarami. W razie konieczności, można je wykorzystać łącznie, tworząc piąty niezależny obiekt typu playera - w takim przypadku będzie on opisywany kolorem z trzeciego rejestru kolorów pola (PF3). Uzyskuje się to przez włączenie bitu D4 w rejestrze kontroli priorytetu (PRIOR). Proszę zwrócić uwagę, że nawet w tym przypadku cztery części tak utworzonego obiektu mogą być przemieszczane niezależnie od siebie, gdyż ich pozycja pozioma opisywana jest niezależnym od innych rejestrem pozycji pocisku. Włączenie bitu D4 w rejestrze PRIOR przełącza jedynie kolor wszystkich pocisków z rejestrów odpowiednich playerów na rejestr PF3.

Przemieszczanie pocisku w pionie wykonuje się tą samą techniką jak w przypadku playerów, poprzez przepisywanie danych w obszarze pamięci PMG przypisanym danemu pociskowi. Utrudnieniem tej procedury jest fakt zastosowania jednej wspólnej tablicy pamięci PMG dla wszystkich czterech pocisków. Przepisanie danych dotyczących jednego pocisku wymaga zastosowania w programie ograniczeń, nie powodujących przepisywania danych innych pocisków.

PRIORYTETY OBIEKTÓW EKRANOWYCH

Olbrzymią zaletą grafiki PMG jest całkowite uniezależnienie za równo playerów jak i pocisków od pola ekranowego. Można je tworzyć w dowolnym trybie graficznym, zarówno tekstowym jak i nietekstowym. Pojawia się tu więc problem: co się stanie, jeżeli niezależny obiekt nałoży się na dowolny element obrazu pola? Który z tych elementów będzie miał wyższy priorytet? Stworzono więc możliwość zaprogramowania priorytetów: w zależności od potrzeby, wszystkie rejestry kolorów obiektów mogą mieć wyższy priorytet od rejestrów kolorów pola, bądź też wszystkie rejestry pola (poza rejestrem tła) mogą mieć wyższy priorytet. Można zaprogramować wyższość obiektów 0 i 1 (w skrócie P0 i P1) nad wszystkimi rejestrami pola, które z kolei będą miały wyższy priorytet od obiektów P2 i P3. Można także zaprogramować wyższość rejestrów pola 0 i 1 (PF0 i PF1) nad wszystkimi rejestrami obiektów, które z kolei będą miały wyższy priorytet niż rejestry PF2 i PF3. Priorytety ustala się poprzez wpisanie odpowiedniej wartości do rejestru kontroli priorytetów (PRIOR), cieniowanego przez GPRIOR. Stwarza to możliwość przemieszczania tego samego obiektu na tle niektórych elementów pola lub też za innymi obiektami, dając wrażenie efektu trójwymiarowego.

SPRZĘTOWA DETEKCJA KOLIZJI

Ostatnim udogodnieniem techniki PMG jest sprzętowa detekcja kolizji. Ma to znaczenie przede wszystkim w programach gier. Istnieje możliwość sprawdzenia, czy którykolwiek z obiektów ekranowych (tak player jak i pocisk) styka się z innym obiektem na ekranie. W szczególności dotyczy to kolizji playerów z pociskami,

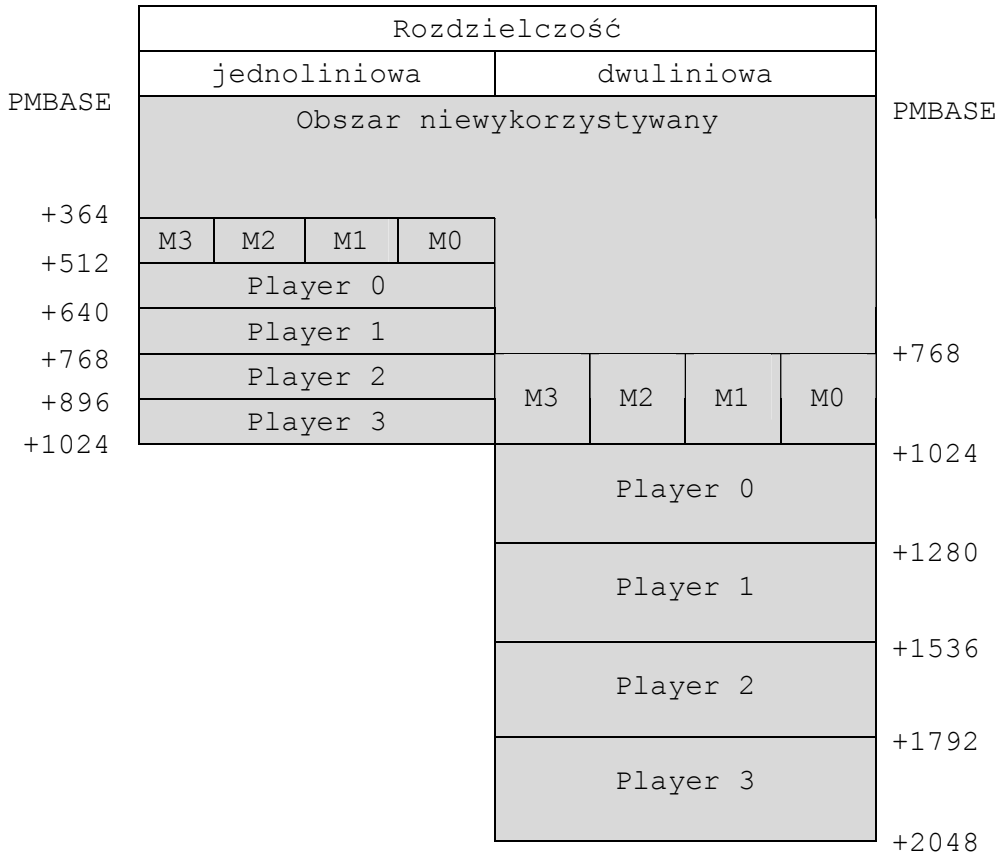
pocisków z elementami pola, playerów z playerami oraz playerów z elementami pola. Istnieją 54 różne typy kolizji, a każdy z nich posiada swój własny bit sterujący - jeżeli bit jest włączony, kolizja miała miejsce. Bity te znajduje się w 15 rejestrach GTIA (w każdym rejestrze wykorzystywane są tylko 4 młodsze bity, a niektóre z nich są nieznaczące). Są to rejestry niezapisywalne i ich zawartość nie może być zmieniona przez program lub użytkownika. Wyzerowanie wszystkich rejestrów uzyskuje się przez zapisanie dowolnej niezerowej wartości do rejestru HITCLR.

Według procedur sprzętowych kolizja ma miejsce wówczas, jeżeli jeden obraz na ekranie zacznie nakładać się na drugi obraz. A więc bit sygnalizujący kolizję nie zostanie włączony, dopóki obiekt ekranowy nie zostanie całkowicie narysowany w danej pozycji. Stąd od momentu wystąpienia kolizji do czasu jej zasygnalizowania może upłynąć nawet 16 milisekund potrzebnych programowi na przeniesienie obrazu. Jeżeli jest to zbyt długi okres czasu, należy procedurę przesuwania obiektu oraz detekcji kolizji wykonać w czasie przerwania wygaszenia pionowego (patrz rozdział 8, gdzie dyskutowane są przerwania wygaszenia pionowego). W takim przypadku najpierw powinna zostać sprawdzona kolizja, następnie wyzerowane rejestry kolizji, po czym wykonane przesunięcie obiektu. W przeciwnym wypadku należy uwzględnić owe 16 milisekund w programie, jakie powinny upłynąć od chwili wystąpienia kolizji do chwili sprawdzenia odpowiedniego rejestru kolizji przez program.

W celu wykorzystania grafiki PMG należy wykonać kilka czynności. Po pierwsze należy wyznaczyć obszar pamięci przeznaczony na pamięć PMG i poinformować komputer, gdzie obszar ten się znajduje. Przy wyborze rozdzielczości jednoliniowej cały zarezerwowany obszar będzie zajmował 1028 bajtów pamięci, przy rozdzielczości dwuliniowej będzie to obszar 640-bajtowy. Dobrze jest wykorzystywać w tym celu RAM znajdujący się tuż poniżej pamięci ekranowej, a więc górną część pamięci. Schematyczną mapę pamięci PMG przedstawia rysunek 4-2.

Wskaźnikiem początku obszaru PMG w pamięci jest rejestr zwany PMBASE. Z powodu ograniczeń, wynikających z konstrukcji układu ANTIC, PMBASE musi znajdować się na początku 2K przedziału pamięci w przypadku rozdzielczości jednoliniowej, lub na początku 1K przedziału przy rozdzielczości dwuliniowej. Jeżeli nie projektuje się wykorzystania wszystkich obiektów ekranowych (dotyczy to zarówno playerów jak i pocisków) wówczas obszar pamięci RAM przeznaczony dla niewykorzystanych obiektów może być wykorzystany do innych celów. Po zaprojektowaniu obszaru PMG w pamięci należy poinformować ANTIC o tym, gdzie ona się znajduje, poprzez wpisanie do rejestru PMBASE w ANTICu numeru strony pamięci, otwierającej ten obszar. Proszę zwrócić uwagę, iż ograniczenia dotyczące PMBASE uniemożliwiają wykorzystanie opisanej poprzednio animacji w pionie, wykorzystującej przepisywanie adresu początku pamięci ekranowej.

Następnym krokiem musi być oczyszczenie tego obszaru pamięci przez wpisanie 0 do wszystkich jej rejestrów. Teraz należy przepisać dane dotyczące poszczególnych playerów i pocisków i umieścić je w odpowiednich przedziałach pamięci PMG.



Rys. 4.2. Schematyczna mapa obszaru PMG w pamięci RAM

W następnej kolejności należy ustalić parametry obiektów przez wpisanie odpowiednich wartości do rejestrów kolorów obiektów, ustalenie wyjściowej pozycji poziomej oraz zapisanie rejestru szerokości obiektu. Jeżeli zachodzi taka konieczność, należy ustalić priorytety w poruszaniu się obiektów po polu. Należy przełączyć ANTIC w zależności od wybranej rozdzielczości - włączając bit D4 w rejestrze DMACTL (cieniowanym przez SDMCTL) dla rozdzielczości jednoliniowej lub wyłączając go dla rozdzielczości dwuliniowej. Na koniec należy uaktywnić grafikę PMG poprzez włączenie PM DMA bitu w rejestrze DMACTL. Należy przy tym zwrócić szczególną uwagę, by jednocześnie nie zmienić wartości innych bitów w tym rejestrze. Poniżej przedstawiamy krótki program BASICu, pozwalający na uaktywnienie jednego obiektu ekranowego i poruszanie go po ekranie przy pomocy dżojstika:

```
1 PMBASE=54279: REM   wskaźnik początku pamięci PMG
2 RAMTOP=106: REM     ustalenie końca wolnego RAM
3 SDMCTL=559: REM     adres cienia rejestru DMACTL
4 GRACTL=53277: REM   rejestr sterujący grafiki CTIA
5 HPOSP0=53248: REM   rejestr pozycji poziomej P0
6 PCOLR0=704: REM     adres cienia rej. kolorów. P0
10 GRAPHICS 0: SETCOLOR 2,0,0: REM czarny kolor tła
20 X=100: REM              pozycja pozioma obiektu
```

```
30 Y=48: REM                pozycja pionowa obiektu
40 A=PEEK(RAMTOP)-8: REM    Ustalenie granicy 2K poniżej TOP
50 POKE PMBASE,A: REM       adres początku pamięci PMG
60 MYPMBASE=256*A: REM      Czytanie adresu z numeru strony
70 POKE SDMCTL,46: REM      Ustalenie rozdzielczości dwuliniowej
80 POKE GRACTL,3: REM       Załączenie grafiki PMG
90 POKE HPOSP0,100: REM     Ustalenie pozycji poziomej
100 FOR I=MYPMBASE+512 TO MYPMBASE+640
110 POKE I,0: REM           Zerowanie pamięci PMG
120 NEXT I
130 FOR I=PMBASE+512+Y TO MYPMBASE+518+Y
140 READ A: REM             Zapis danych obiektu PMG
150 POKE I,A
160 NEXT I
170 DATA 8,17,35,255,32,16,8
180 POKE PCOLR0,88: REM     Różowa barwa obiektu PMG
190 A=STICK(0): REM         Czytanie dżojstika
200 IF A=15 THEN GOTO 190
210 IF A=11 THEN X=X-1: POKE HPOSP0,X
220 IF A=7 THEN X=X+1; POKE HPOSP0,X
230 IF A<>13 THEN GOTO 280
240 FOR I=8 TO 0 STEP -1
250 POKE MYPMBASE+512+Y+I, PEEK(MYPMBASE+511+Y+I)
260 NEXT I
270 Y=Y+1
280 IF A<>14 THEN GOTO 190
290 FOR I=0 TO 8
300 POKE MYPMBASE+511+Y+I, PEEK(MYPMBASE+512+Y+I)
310 NEKT I
320 Y=Y-1
330 GOTO 190
```

Raz wyświetlony obiekt PMG może być dość trudny do usunięcia z ekranu. dzieje się tak dlatego, że procedura uaktywniająca grafikę PMG składa się z kilku etapów: najpierw ANTIC odczytuje dane z pamięci PMG (o ile takie czytanie dozwolone jest przez rejestr DMACTL), następnie przesyła te dane do CTIA (o ile takie przesłanie dozwolone jest przez GRACTL), natomiast CTIA wyświetla wszystko, na co zezwalają mu rejestry od GRAFP0 do GRAFP3 oraz GRAFM. Wielu programistów kasuje obiekty przez wyzerowanie odpowiednich bitów sterujących w DMACTL oraz GRACTL. Zabezpiecza to jednak tylko przed przesyłaniem nowych danych z pamięci PMG przez ANTIC do CTIA; dane znajdujące się w rejestrach GRAF(X) będą nadal wyświetlane na ekranie. Aby całkowicie zlikwidować obiekt należy po wyzerowaniu bitów sterujących w DMACT L i GRACTL wyzerować rejestry GRAF(X). Znacznie prostszym posunięciem jest pozostawienie obiektu, a jedynie przesunięcie go do pozycji poziomej 0. W tym przypadku ANTIC będzie nadal czytał dane z pamięci PMG i przesyłał je do CTIA zużywając na ten proces około 70000 cykli maszynowych na sekundę, blokując jednocześnie na ten czas mikroprocesor 6502.

ZASTOSOWANIA GRAFIKI PMG

Grafika obiektów ekranowych stwarza wiele nowych, interesujących możliwości. Jej znaczenie przy tworzeniu animacji jest powszechnie znane. Największym jej ograniczeniem jest możliwość stworzenia tylko czterech (pięciu) obiektów, każdy o szerokości zaledwie 8 bitowej. Jeżeli zachodzi potrzeba stworzenia animacji z użyciem większych obiektów, trzeba powrócić do skomplikowanej i czasochłonnej animacji polowej. Jednakże w większości przypadków ograniczenia grafiki obiektów ekranowych nie są decydujące przy tworzeniu animacji.

Istnieje także możliwość pominięcia pośredniej drogi przez ANTIC i wpisania danych obiektów ekranowych bezpośrednio do odpowiednich rejestrów GRAF(X) w CTIA. Uzyskuje się przez to bardziej bezpośrednią kontrolę nad grafiką PMG, lecz jednocześnie znacznie bardziej zwiększa czasochłonność projektowania obiektów. Szczególnie ważne jest w tym przypadku dokładne wyliczenie czasu i wpisanie danych do rejestrów CTIA w odpowiednim momencie. Ponadto 6502 musi zostać zablokowany na okres rysowania obiektu na ekranie (patrz dyskusja o kernelach w rozdziale 5), Generalnie jest to więc technika przynosząca niezbyt wielkie efekty kosztem olbrzymiego nakładu pracy programisty. Jeżeli chce się ominąć pośrednią rolę ANTICu przy tworzeniu obiektów ekranowych, należy wówczas przejąć na siebie wszystkie jego zadania.

Obiekty ekranowe można ponadto wykorzystać do stworzenia trójwymiarowego ruchu na ekranie. Sytuację tę komplikują nieco ograniczenia szerokości obiektów. Każdy z obiektów może być opisany w pamięci kilkoma mapami o różnej ilości bitów. Załóżmy, że w jednej mapie jest on opisany jako obiekt o szerokości 6-bitowej, a w drugiej o szerokości 8-bitowej. A więc będzie on mógł mieć szerokość 6 bądź 8 punktów rastrowych, w zależności od wykorzystywanej części pamięci PMG. Następnym etapem może być przełączenie grafiki PMG na podwójną szerokość, co da możliwość uzyskania obiektów o szerokości 12 i 18 punktów, w zależności od wykorzystywanej tablicy. Przełączenie na szerokość poczwórną pozwoli na uzyskanie obiektów o szerokościach odpowiednio 24 i 36 punktów. W sumie więc można uzyskać zmianę szerokości obiektu od 6 do 36 punktów rastrowych ekranu. Efekt ten został znakomicie wykorzystany w grze "Star Raiders".

Grafika obiektów ekranowych stwarza wiele możliwości nie tylko w zakresie tworzenia animacji. Jej wykorzystanie jest znakomitą metodą na uzyskanie dodatkowych kolorów na ekranie, poprzez wprowadzenie czterech niezależnych rejestrów kolorów. Jedynym ograniczeniem tej metody jest limit szerokości obiektów ekranowych. Istnieje jednak sposób ominięcia tej trudności, który w niektórych przypadkach może znaleźć szerokie zastosowanie. Należy przełączyć grafikę PMG na poczwórną szerokość i umieścić obiekt na ekranie, programując priorytety tak, by kolor obiektu miał niższy priorytet niż kolor tła. Teraz należy do rejestru koloru pola wpisać ter sam kolor co kolor tła, a więc obiekt powinien zniknąć za polem. Teraz na polu należy umieścić obraz w kolorze tła (a więc niewidoczny w normalnych warunkach) a w miejscu tym pokaże się kolor obiektu posiadającego niższy priorytet od koloru pola, lecz wyższy od koloru tła.

Oto prosty program BASICu, pozwalający na wykorzystanie tej metody:

```
1 RAMTOP=105: REM          adres wskaźnika końca pamięci
2 PMBASE=54279: REM        wskaźnik pamięci PMG
3 SDMCTL=559: REM          cień DMACTL
4 GRACTL=53277: REM        rejestr sterujący grafiki CTIA
5 HPOSP0=53248: REM        rejestr pozycji poziomej P0
6 PCOLR0=704: REM          cień rejestru kolorów PO
7 SIZEP0=53256: REM        rejestr sterujący szerokości P0
8 GPRIOR=623: REM          rejestr kontroli priorytetów
10 GRAPHICS 7
20 SETCOLOR 4,8,4
30 SETCOLOR 2,0,0
40 COLOR 3
50 FOR Y=0 TO 79: REM      pętla zapełniania ekranu
60 PLOT 0,Y
70 DRAWTO 159,Y
80 NEXT Y
90 A=PEEK(RAMTOP)-20: REM  ustalenie początku pamięci PMG
100 POKE PMBASE,A
110 MYPMBASE=256*A
120 POKE SDMCTL, 46
130 POKE GRACTL, 3
140 POKE HPOSP0, 100
150 FOR I=MYPMBASE+512 TO MYPMBASE+640
160 POKE I, 255: REM       opis koloru playera
170 NEXT I
180 POKE PCOLR0, 88
190 POKE SIZEP0, 3: REM    poczwórna szerokość playera
200 POKE GPRIOR, 4: REM    ustalenie priorytetów
210 COLOR 4
220 FOR Y=30 TO 40
230 PLOT Y+22, Y
240 DRAWTO Y+43, Y
250 NEXT Y
```

program ten tworzy następujący obiekt:

Rys. 4-3. Wykorzystanie PMG jako płaszczyzny o dodatkowym kolorze ekranu

ZNAKI SPECJALNE

Kolejnym zastosowaniem grafiki PMG jest tworzenie znaków specjalnych. W szczególności odnosi się to do znaków, które muszą posiadać większą wysokość niż standardowy znak 8-liniowego trybu tekstowego. Można to osiągnąć przez zaprojektowanie zbioru znaków specjalnych, większych od standardowych. Lecz znacznie prostszą metodą jest wykorzystanie do tego celu grafiki PMG. Przykładem takich znaków może być umieszczanie na ekranie podpisów, czy symboli matematycznych, sumy, całkowania itp. Oto przykład prostego programu BASICu:

```
1 RAMTOP=106: REM          adres wskaźnika końca pamięci
2 PMBASE=54279: REM        adres pamięci PMG
3 SDMCTL=559: REM          cień DMACTL .
4 GRACTL=53277: REM        rejestr sterujący grafiki CTIA
5 POOSP0=53248: REM        rejestr pozycji poziomej P0
6 PCOLR0=704: REM          cień rejestru kolorów P0
10 GRAPHICS 0: A=PEEK(RAMTOP)-16: REM przesunięcie końca pamię
20 POKE PMBASE,A: REM           ci dla rozdzielcz. jednolin.
30 MYPMBASE=256*A
40 POKE SDMCTL, 62
50 POKE GRACTL, 3
60 POKE HPOSP0, 102
70 FOR I=MYPMBASE+1024 TO MYPMBASE+1280
80 POKE I,0
90 NEXT I
100 POKE PCOLR0, 140
110 FOR I=0 TO 15
120 READ X
130 POKE MYPMBASE+1100+I,X
140 NEXT I
150 DATA 14, 29, 24, 24, 24, 24,  24, 24
160 DATA 24, 24, 24, 24, 24, 24, 184, 112
170 ? "↖": REM             kasowanie ekranu
180 POSITION 15, 6
190 ? "xdx"
```

Program ten tworzy następujący obraz:

Rys. 4-4. Wykorzystanie PMG do tworzenia znaków specjalnych

Szczególnie wartościowym wykorzystaniem Grafiki PMG jest projektowanie kursorów. Obiekty PMG nadają się znakomicie do tego celu, jako że z łatwością mogą być przemieszczane po całym ekranie, nie zmieniając jednocześnie jego zawartości. Łatwe jest także

stworzenie kursora zmieniającego kolor, w zależności od położenia na ekranie i od zawartości danej części ekranu.

Grafika PMG otwiera szerokie pole do popisu dla programistów. Jej zastosowanie przy animacji i grach komputerowych jest chyba oczywiste. Wykorzystanie jej do uzyskania dodatkowego koloru, kursorów i znaków specjalnych jest tylko przykładem. Należy wykorzystywać te możliwości przy każdej okazji.

5. PRZERWANIA LISTY DISPLEJOWEJ

Przerwania listy displejowej (DLI - Display List Interrupts) są jedną z "najpotężniejszych broni", w jakie wyposażono komputery domowe systemu Atari 400/800. Jest to jednocześnie jedna z najrzadziej wykorzystywanych możliwości systemu[1] - podobnie jak kilka innych właściwości tych urządzeń - jako że wymaga dokładnego poznania systemu oraz języka asemblera mikroprocesora 6502. Przerwania listy displejowej jako takie nie stwarzają nowych możliwości komputera; dopiero zastosowanie ich w połączeniu z innymi technikami programowania, takimi jak grafika obiektów ekranowych, pośrednie wykorzystywanie zbiorów symboli czy rejestrów kolorów, otwiera drogę do pełnego i efektywnego wykorzystania powyższych technik programowych.

TEORIA OPERACJI

Przerwania listy displejowej opierają się na wykorzystaniu sekwencyjnego charakteru tworzenia rastrowego, skaningowego obrazu telewizyjnego. Przypomnijmy, że telewizor tworzy obraz na ekranie sekwencyjnie. Rysowanie całego obrazu, począwszy od górnej, a skończywszy na dolnej krawędzi ekranu, trwa około 17000 mikrosekund. Dla oka ludzkiego jest to niedostrzegalna chwila, lecz w skali pracy komputera jest to dość długi okres czasu. Zdecydowanie na tyle długi, by mikroprocesor mógł wprowadzić zmiany w danych dotyczących rysowanego obrazu. Oczywiście, zmiana taka musi być przeprowadzana każdorazowo w momencie rysowania ekranu, czyli 60 razy na sekundę. Ponadto (i jest to najbardziej krytyczny moment) zmiana taka musi być podczas każdorazowego rysowania ekranu wprowadzana dokładnie w tym samym momencie, Znaczy to, że cykl zmian parametrów obrazu musi być zsynchronizowany z cyklem rysowania obrazu na ekranie. Jedną z metod uzyskania takiej synchronizacji może być zamknięcie programu pracy 6502 w ciasną pętlę, której częstotliwość wykonywania jest dokładnie taka sama jak częstotliwość rysowania ekranu. Oczywiście, w takim przypadku praktycznie nie ma możliwości wykonania jakichkolwiek innych obliczeń oprócz zmiany parametrów obrazu. Poza tym wykonywany program byłby całkowicie jednostajny. Znacznie efektywniejszą metodą jest przerwanie pracy 6502 tuż przed obliczonym momentem zmiany parametrów obrazu. 6502 przerywa pracę, zmienia parametry obrazu, po czym wraca do wykonywania głównego programu. Zaprogramowanie takiego przerwania wiąże się ze ścisłym wyliczeniem czasu - tak, by zmiana parametrów obrazu następowała w każdym cyklu dokładnie w tym samym momencie procesu rysowania obrazu. Właśnie takie przerwania, o dokładnie wyliczonym czasie, wykonywane są przez układ ANTIC, a noszą one nazwę przerwań listy displejowej (DLI).

Dokładne wyliczenie czasu oraz realizacja procedury przerwania mogą się wydać skomplikowane. Spróbujmy więc na początku prześledzić sekwencję wydarzeń przy poprawnie zastosowanym DLI. Proces rozpoczyna się w momencie, kiedy ANTIC, wykonując kolejne instrukcje listy displejowej, natrafia na instrukcję, w której włączony został

[1] Należy pamiętać, że książka ta powstała ponad 20 lat temu i jej autorzy nie oglądali efektów pracy polskiej sceny ☺

bit (D7) przerwania. Teraz ANTIC czeka, aż zostanie narysowana ostatnia linia skaningowa danej linii trybowej, po czym sprawdza rejestr sterujący NMIEN, czy przerwania listy displejowej są dopuszczalne. Jeżeli odpowiedni bit rejestru jest równy 0, wówczas ANTIC ignoruje żądanie przerwania i kontynuuje wykonywanie instrukcji listy displejowej. Jeżeli bit został włączony, ANTIC przesyła wezwanie NMI do 6502, po czym powraca do wykonywania dalszych instrukcji listy displejowej. 6502, kierując się wektorem NMI, przechodzi do procedury obsługi przerwania, mieszczącej się w systemie operacyjnym. Procedura ta w pierwszej kolejności sprawdza żądanie przerwania. Jeżeli rzeczywiście żądanie dotyczy DLI, procedura przeskakuje do adresu $0200, $0201 (LSB i MSB), gdzie znajduje się adres procedury obsługi DLI. Procedura obsługi DLI dokonuje zmiany jednego lub więcej rejestrów kolorów, dotyczących rysowanego obrazu. Na koniec 6502 wykonuje instrukcję RTI i powraca do wykonywania głównego programu.

Wykorzystanie procedury DLI wymaga więc kilku kolejnych etapów. Przede wszystkim należy samemu stworzyć procedurę DLI. Trzeba pamiętać, że wszystkie rejestry 6502, które będą przez nią wykorzystywane, muszą zostać przeniesione na stos, jako że procedura obsługi przerwań OS nie czyni tego automatycznie - samoczynnie 6502 przenosi na stos jedynie rejestr wskaźników procesora. Procedura DLI musi być krótka, działająca szybko - powinna zmieniać tylko te rejestry kolorów, które w danym momencie dotyczą rysowanego obrazu. Powinna się ona kończyć powrotnym przepisaniem zawartości rejestrów ze stosu do 6502. W następnej kolejności należy zapisać przygotowaną procedurą gdziekolwiek w pamięci - najlepszym do tego celu miejscem jest strona 6 pamięci. W adresach $0200, $0201 należy umieścić wektor, wskazujący początek procedury DLI. Teraz należy obliczyć pionową pozycję na ekranie miejsca, w którym procedura ma zostać wykonana, po czym odnaleźć odpowiednią instrukcję listy displejowej i włączyć bit D7 w instrukcji poprzedniej. Na koniec trzeba zezwolić na wykonanie przerwania poprzez włączenie bitu D7 w rejestrze NMIEN pod adresem $D40E. Poprawnie przygotowane DLI powinno w tym momencie zacząć działać.

WYLICZENIE CZASU DLI

Podobnie jak w przypadku procedur obsługi przerwań innego typu, krytycznym momentem może być dokładne wyliczenie czasu. ANTIC nie przesyła sygnału NMI do 6502 natychmiast po jego odczytaniu w liście displejowej; sygnał ten przesyłany jest dopiero wówczas, gdy narysowana zostanie do końca ostatnia linia skaningowa dane j linii trybowej, w instrukcji której znalazło się żądanie przerwania.

Poza tym procedura obsługi przerwań OS wykonuje kilka czynności, które powodują dodatkowe opóźnienie wykonania procedury DLI. Stąd procedura obsługi przerwania DLI zostanie wykonana w chwili, kiedy strumień elektronów będzie się znajdował gdzieś w połowie ostatniej linii skaningowej rysowanej linii trybowej, w której miało nastąpić przerwanie. Załóżmy, że procedura DLI dotyczyła zmiany rejestrów kolorów; w takim przypadku lewa część tejże linii skaningowej będzie miała kolor stary, a prawa jej część kolor zmieniony. Dodatkowo, z powodu niestabilności czasowej odpowiedzi 6502 na żądanie przerwania, granica pomiędzy dwoma kolorami tej linii nie będzie

stała, lecz będzie przesuwana w pewnych granicach bądź w lewą bądź w prawą stronę, powodując irytujące "skakania" linii na ekranie.

Problem ten został rozwiązany systemowo. Wprowadzono specjalny rejestr synchronizacji poziomej, nazywany WSYNC. Jeżeli rejestr ten jest zapisany jakąkolwiek wartością, ANTIC wysyła do 6502 sygnał RDY. Powoduje on zatrzymanie pracy 6502 do momentu, aż WSYNC przepisany zostanie przez impuls synchronizacji poziomej. Oznacza to, że praca 6502 wstrzymana zostanie do momentu osiągnięcia przez strumień elektronów w kineskopie prawej krawędzi ekranu. Jeżeli w procedurze obsługi DLI znajdzie się instrukcja STA WSYNC, poprzedzająca bezpośrednio instrukcję zapisującą rejestr kolorów nową zawartością, wówczas nowy zapis w rejestrze kolorów pojawi się w momencie, kiedy strumień elektronów będzie się znajdował poza widzialną częścią ekranu. A więc nowy kolor pojawi się na ekranie o jedną linię skaningową niżej, lecz granica między kolorami będzie ostra i stabilna.

Z tego właśnie powodu właściwe użycie DLI wymaga wpisania żądania przerwania nie do instrukcji listy displejowej dotyczącej linii trybowej, w której zmiana jest wymagana, lecz do instrukcji poprzedniej. Procedura obsługi DLI musi w pierwszej kolejności przenieść na stos zawartość odpowiednich rejestrów 6502, następnie zapisać w tych rejestrach wielkości, które mają być zmienione, po czym wykonać STA WSYNC. Teraz należy przepisać nowe wartości do odpowiednich rejestrów ANTICu lub CTIA, ponownie zapisać rejestry 6502 utrwalonymi na stosie wartościami, po czym powrócić z przerwania. Zachowanie wszystkich tych kroków zapewnia, że zawartość rejestrów kolorów zostanie przepisana w momencie, kiedy strumień elektronów rysujących obraz znajdzie się poza ekranem - a więc nowy kolor pojawi się na ekranie od początku kolejnej linii skaningowej.

PRZYKŁAD DLI

Poniżej przedstawiamy prosty program, demonstrujący wykorzystanie przerwania listy displejowej:

```
10 DLIST=PEEK(560)+256*PEEK(561): REM: początek listy displej.
20 POKE DLIST+15,130: REM              żądanie przerwania DLI
30 FOR I=0 TO 19: REM               pętla wpisująca procedurę ob.-
40 READ A: POKE 1536+I,A: NEXT I: REM sługi DLI w kodzie masz.
50 DATA 72, 138, 72,1 69, 80, 162, 88
60 DATA 141, 10, 212, 141, 23, 208
70 DATA 141, 24, 208, 104, 170,104, 64
80 POKE 512,0: POKE 513, 6: REM  wpisanie wektora procedury
90 POKE 54286,192: REM                zezwolenie na DLI
```

Program powyższy zapisuje na stronie 6 pamięci następującą procedurę obsługi DLI w kodzie maszynowym:

```
     PHA            zapisz zawartość akumulatora
     TXA            przepisz rejestr X do akumulatora
     PHP            zapisz zawartość (rejestru X)
     LDA #$50       kod koloru czarnego
```

```
LDX #$58        kod koloru różowego
STA WSYNC       czekaj na sygnał synchronizacji
STA COLPF1           przepisz kod do rejestru COLPF1
STX COLPF2           przepisz kod do rejestru COLPF2
PHA
TAX
PLA                  przepisz zawartości rejestrów
RTI                  wróć z przerwania
```

Jest to przykład bardzo prostej procedury DLI. Zmienia ona kolor pola z niebieskiego na różowy. Zamienia także kolor symboli, będą one teraz wyświetlane jako czarne na różowym tle. Można się zastanawiać, dlaczego w górnej części ekranu pole nadal pozostaje niebieskie, skoro zawartość odpowiedniego rejestru kolorów została przez DLI przepisana. Odpowiedź jest następująca: w trakcie każdego przerwania wygaszenia pionowego procedura tegoż przerwania, wykonywana przez system operacyjny, wpisuje do rejestru kolorów dane pochodzące z cienia OS, odpowiadającego danemu rejestrowi kolorów, a więc kolor niebieski. Każdy sprzętowy rejestr kolorów jest cieniowany przez odpowiednie komórki pamięci w RAM. Cienie te, mieszczące się pod adresami od 708 do 712, były już poprzednio omawiane. W większości przypadków zmianę kolorów przeprowadza się poprzez zmianę zawartości odpowiedniego cienia. Jeżeli przepisana zostanie zawartość sprzętowego rejestru kolorów OS dokona zmiany zawartości tego rejestru co najwyżej po upływie 1/60 sekundy. Jednakże w przypadku posługiwania się procedurą DLI zmiany muszą być dokonywane właśnie w sprzętowych rejestrach kolorów. Stąd wypływa poważne ograniczenie: nie można zmienić koloru pierwszej linii trybowej rysowanego obrazu; kolor tej linii będzie zawsze pochodził z przepisanego podczas przerwania wygaszenia pionowego rejestru kolorów. DLI może zmienić kolor dopiero drugiej i następnych linii trybowych.

TRYB PRZYCIĄGANIA UWAGI

Bezpośrednie adresowanie sprzętowych rejestrów kolorów stwarza kolejny problem: zabronione zostaje automatyczne włączenie trybu przyciągania uwag.: Tryb ten, włączany przez OS, polega na tym, iż po dziewięciu minutach od chwili ostatniego naciśnięcia klawisza komputera kolory na ekranie poczynają zmieniać się cyklicznie, przy czym barwy wybierane są losowo, natomiast wartości luminancji obniżono do wartości 50%. Tryb ten ma na celu zabezpieczenie kineskopu telewizora przed wypaleniem luminoforu w przypadku pozostawienia nieużywanego komputera włączonego na dłuższy okres czasu. Bardzo łatwo jest samodzielnie zaprojektować tryb przyciągania uwagi i włączyć go do procedury obsługi przerwania listy displejowej. Posługując się przedstawionym wyżej przykładem procedury DLI tryb przyciągania uwagi włączamy poprzez dołączenie dwóch linii kodu maszynowego do przedstawionej procedury:

```
stara procedura       nowa
LDA #NEWCOL           LDA   #NEWCOL
STA WSYNC             EOR   COLRSH
STA COLPF2            AND   DRKMSK
```

```
STA  WSYNC
STA  COLPF2
```

gdzie DRKMSK oraz COLRSH są adresami ze strony zerowej ($4E i $4F), zapisywanymi przez OS podczas przerwania wygaszenia pionowego. Jeżeli nie jest włączony tryb przyciągania uwagi, COLRSH zapisany jest wartością 00, natomiast DRKMSK SFF. Włączenie trybu przyciągania uwagi powoduje zapisanie COLRSH wartością przypadkową, zmienianą przez OS co około 4 sekundy, natomiast do DRKMSK zostaje wpisana wartość $F6. Tak więc barwa ekranu podczas trybu przyciągania uwagi odczytywana jest z COLRSH, natomiast luminancja z rejestru DRKMSK.

CZASOCHŁONNOŚĆ PROCEDURY DLI

Realizacja trybu przyciągania uwagi poprzez zastosowanie DLI jeszcze raz zwraca uwagę na bardzo ważny problem: niezwykle krótki czas, w jakim procedura DLI musi zostać wykonana. Szczegółowy opis czasochłonności procedury może rzucić nieco światła na ten problem. Wykonanie procedury możemy rozbić na trzy fazy:

- Faza pierwsza obejmuje okres od startu procedury do instrukcji STA WSYNC. W trakcie tej fazy strumień elektronów w kineskopie rysuje jeszcze ostatnią linię skaningową tej linii trybowej, w instrukcji której znalazło się żądanie przerwania listy displejowej.
- Faza druga obejmuje okres do momentu pojawienia się strumienia elektronów na ekranie kineskopu. Odpowiada ona więc wygaszeniu poziomemu; wszystkie zmiany, jakie procedura DLI ma wprowadzić wśród parametrów rysowanego obrazu, muszą zostać dokonane w tej fazie.
- Faza trzecia obejmuje czas od chwili zapalenia strumienia elektronów w kineskopie do momentu zakończenia procedury DLI. Czas wykonywania tej fazy nie jest krytyczny.

Rysowanie jednej poziomej linii skaningowej na ekranie telewizora trwa dokładnie 114 cykli zegarowych procesora, żądanie DLI dociera do 6502 w cyklu ósmym. Od 8 do 14 cykli zajmuje 6502 odpowiedź na żądanie. Procedura obsługi przerwań OS, potwierdzająca żądanie i przeskakująca przez wektor DLI do umieszczonej w RAM procedury DLI, zajmuje 11 cykli maszynowych. W tym czasie od 1 do 3 cykli poświęconych zostanie na czytanie danych z pamięci ekranowej. Tak więc procedura DLI będzie mogła się rozpocząć po upływie od 28 do 36 cykli zegarowych. Przy obliczeniach wykonywanych dla projektowanej procedury DLI należy przyjąć najgorszy przypadek i założyć, że wykonywanie procedury rozpocznie się w 36 cyklu zegarowym. Z drugiej strony, instrukcja STA WSYNC musi zostać osiągnięta nie dalej, jak w cyklu o numerze 100, co z kolei redukuje czas dostępny w fazie pierwszej o dalsze 14 cykli. Wreszcie kilka cykli zostanie poświęconych na wykonywanie instrukcji DMA przez ANTIC, a kolejnych 9 potrwa odczytywanie pamięci ekranowej przez DMA. Stąd za absolutne maksimum czasu dostępnego w fazie pierwszej należy przyjąć 55 cykli zegarowych. Maksimum to dostępne jest dla procedury DLI tylko w przypadku wyświetlania na ekranie linii wygaszonych. W przypadku linii trybowych, czy to graficznych czy tekstowych, trzeba od czasu tego odjąć po jednym cyklu zegarowym na każdy bajt pamięci ekranowej umieszczany na ekranie. Stąd w najgorszym przypadku, przy trybach

BASICu 0, 7 i 8, opisywanych 40 bajtami dla jednej linii trybowej, na wykonanie pierwszej fazy procedury DLI pozostanie zaledwie 15 cykli zegarowych. Reasumując - na wykonanie fazy pierwszej procedury DLI należy liczyć od 15 do 55 cykli zegarowych, w zależności od wyświetlanej aktualnie linii trybowej.

Faza druga, najbardziej krytyczna, trwa 27 cykli maszynowych czasu rzeczywistego. Podobnie jak w fazie pierwszej część tego czasu trzeba odliczyć na instrukcję DMA ANTICu. Jeżeli wykorzystywana jest grafika PMG, zabierze ona dalszych 5 cykli; 1 cykl potrwa wykonywanie instrukcji listy displejowej; jeżeli instrukcja ta zawiera opcję LMS, trzeba odliczyć kolejne dwa cykle. Wreszcie 1 lub 2 cykle potrwa czytanie pamięci ekranowej lub przesyłanie danych ekranowych. Stąd, w fazie drugiej dostępnych jest od 17 do 26 cykli zegarowych.

Teraz chyba problem czasochłonności procedury DLI stał się bardziej jasny. Wczytanie, zamiana i przepisanie zawartości jednego rejestru kolorów zajmie 14 cykli zegarowych. Przepisanie zawartości rejestrów A, X i Y na stosy po czym wczytanie, zmiana i przepisanie trzech rejestrów kolorów przy wykorzystaniu wszystkich rejestrów 6502 trwać musi 47 cykli maszynowych - a więc olbrzymią większość jeżeli nie całość czasu dostępnego w fazie pierwszej. Z doświadczenia wiemy że próby zastosowania DLI do poważnych zmian w parametrach obrazu wymagają niezwykle dokładnych i pracochłonnych wyliczeń czasowych procedury DLI. Nie dotyczy to, rzecz jasna, początkujących programistów, dla których obliczenia takie nie będą niezbędne. Jeżeli DLI aa dokonać zmiany tylko jednego rejestru kolorów czy też innej prostej operacji na danych, liczenie cykli maszynowych i optymalizacja szybkości są całkowicie zbędne.

Jeżeli natomiast wymagana jest zmiana większej ilości niż trzy rejestry kolorów przy pomocy jednej procedury DLI, trudno jest znaleźć jakieś proste wyjście. Zmiana zawartości czwartego rejestru kolorów może być dokonana na początku fazy trzeciej - o ile kolor ten nie pojawi się w lewej części ekranu. Analogicznie, jeżeli jakiś kolor nie pojawia się przy prawej krawędzi ekranu, zmiana zawartości odpowiedniego rejestru może być dokonana pod koniec fazy pierwszej. Innym rozwiązaniem jest rozbicie jednej, zbyt długiej procedury DLI na dwie krótsze, z których każda wykonywałaby tylko część zadań. Druga DLI mogłaby w takim przypadku zostać poprzedzona wyświetleniem jednej linii skaningowej wygaszonej (zawierającej żądanie DLI), znajdującej się tuż poniżej linii trybowej, w której wykonywana byłaby pierwsza procedura DLI. Rzeczywiście, pochłonęłoby to pewną część obrazu ekranowego.

Innym rozwiązaniem pośrednim tego problemu może być dokonywanie zamiany odpowiednich parametrów w trakcie przerwania wygaszenia pionowego. W tym celu należałoby zapisać w RAM dwie tablice wartości rejestrów kolorów w tablicy 1 wartości, które mają zostać umieszczone na ekranie przez DLI, w drugiej zaś wartości, które mają zostać zmienione, Zaprojektowana przez użytkownika procedura musiałaby podczas wygaszenia pionowego odczytać wartości z pierwszej tablicy, zmienić je i wpisać do tablicy drugiej. Wówczas procedura DLI odczytywałaby dane bezpośrednio z drugiej tablicy, nie tracąc czasu na zamianę wartości.

WIELOKROTNE PROCEDURY DLI

Często wymagane jest wprowadzenie kilku procedur DLI, wywoływanych w różnych liniach trybowych rysowanego obrazu. Jest to bardzo ważna metoda uzyskiwania wielokolorowych obrazów. Niestety, w RAM znajduje się tylko jeden wektor procedury DLI; jeżeli chcemy zastosować wielokrotne procedury DLI, wektor ten musi być przepisywany każdorazowo przez samą procedurę. Istnieje kilka metod wykorzystania procedur wielokrotnych. Jeżeli DLI ma wykonywać te same zmiany parametrów, różniących się tylko wartościami, wówczas można wykorzystać tablicowanie wartości parametrów. Podczas każdorazowego wykonywania procedury DLI zwiększana będzie zawartość licznika, wykorzystywanego do indeksowego adresowania tablic wartości. Oto przykład prostej procedury DLI, działającej w ten właśnie sposób:

```
          PHA
          TXA
          PHA
          INC   COUNTR
          LDX   COUNTR
          LDA   COLTAB,X        wykorzystaj str. 2 na tabl. kol.
          STA   WSYNC           czekaj
          STA   COLBAK
          CPX   #$4F            czy to ostatnia linia?
          BNE   ENDDLI          nie, zakończ procedurę
          LDA   #$00            tak, wyzeruj licznik
          STA   COUNTR
ENDDLI    PLA
          TAX
          PLA                   przepisz akumulator
          RTI
```

A teraz program BASICu do uruchomienia procedury

```
10 GRAPHICS 7
20 DLIST=PEEK(560)+256*PEEK(561): REM początek listy displej:
30 FOR J=6 TO 84: REM      zapis w każdej instr.listy żądania DLI
40 POKE DLIST+J,141: REM   tryb 7 BASICu z żądaniem DLI
50 NEXT J
60 FOR J=0 TO 30
70 READ A: POKE 1536+J,A: NEXT J: REM zapis procedury DLI
80DATA 72, 138, 72, 238, 32, 6, 175, 32, 6
90 DATA 189, 0, 240, 141, 10, 212, 141, 26, 208
100 DATA 224, 79, 208, 5, 169, 0
110 DATA 141, 32, 6, 104, 170, 104, 64
120 POKE 512,0: POKE 513,6: REM zapis wektora procedury DLI
130 POKE 54286,192: REM zapis zezwolenia na DLI
```

Program ten umieści 80 różnych kolorów na ekranie.

Są jeszcze inne metody zastosowanie wielokrotnych procedur DLI. Jedną z nich jest wykorzystanie licznika DLI jako warunku do wykonania skoku do właściwej procedury DLI. Metoda ta wydłuża czas do rozpoczęcia wykonywania procedury DLI, szczególnie w przypadku procedur zapisanych na końcu, a więc takich, w stosunku do których musi zostać wykonanych kilka pętli sprawdzających. Znacznie lepszym sposobem jest procedura DLI, która przepisuje wektor DLI pod adresami $200, $201 tak, by wskazywał początek następnej procedury DLI. Proces ten musi być wykonywany w fazie trzeciej. Jest to chyba najlepszy sposób na wykorzystanie wielokrotnych procedur DLI. Dodatkową zaletą tej metody jest fakt, że przepisywanie wektora DLI odbywa się w fazie trzeciej przerwania, a więc nie zabiera czasu z fazy pierwszej i drugiej procedury.

Istnieje procedura systemu operacyjnego, potwierdzająca sygnałem dźwiękowym naciśnięcie klawisza w klawiaturze, która przeszkadza w wykonywaniu procedur DLI, jako że wykorzystuje się w niej kilkakrotnie instrukcję STA WSYNC do kalkulacji czasu sygnału dźwiękowego. Może ona spowodować błędne wyliczenie czasu projektowanej procedury DLI, co objawi się przesunięciem o jedną linię skaningową pojawiającego się na ekranie koloru trwającym ułamek sekundy. Nie istnieje proste rozwiązanie tego problemu. Jedno z rozwiązań opiera się na wykorzystaniu rejestru WCOUNT - rejestru typu ROM znajdującego się w ANTICu, wskazującego numer linii skaningowej aktualnie rysowanej przez ANTIC. Procedura DLI musiałaby w tym przypadku sprawdzać zawartość tego rejestru przed przejściem do wykonywania zmiany w zawartości rejestru kolorów. Innym wyjściem jest zablokowanie procedury klawiaturowej OS i wprowadzenie na jej miejsce własnej procedury klawiaturowej. Można się domyślać, że nie będzie to łatwe zadanie. Ostatnim rozwiązaniem jest zamknięcie kanału przyjmującego polecenia z klawiatury. Jeżeli naciśnięcie klawisza nie będzie czytane przez system, nie wystąpi tym samym konflikt obu procedur.

Możliwość wykorzystania przerwań listy displejowej zastąpiła w systemie Atari 400/800 bardziej prymitywną, programowo-sprzętową technikę, zwaną kernelem. Kernel jest to pętla programowa 6502, dokładnie zsynchronizowana z cyklem rysowania obrazu na ekranie telewizora. 6502 może arbitralnie sterować wszystkie parametry graficzne całego rysowanego obrazu, poprzez sprawdzanie zawartości rejestru WCOUNT i konsultowanie go z tablicą zmian parametrów, zapisanych w funkcji wartości rejestru WCOUNT. Za tę niezwykłą możliwość sterowania całego ekranu płacona jest bardzo wysoka cena: 6502 nie może wykonywać żadnych innych obliczeń w trakcie każdorazowego rysowania obrazu na ekranie, czyli przez 75% czasu. Ponadto wykonywane obliczenia nie mogą zabierać 6502 więcej czasu niż około 4000 cykli maszynowych, czyli tyle, ile trwa wygaszenie pionowe oraz rysowanie poza krawędziami widzialnej części ekranu. Ograniczenia te wskazują, że technika kerneli może być wykorzystywana tylko w programach nie wymagających wielu obliczeń, jak np. programy gier zręcznościowych. Jednym z programów wykorzystujących tę technikę jest gra "Basketball"; program ten nie wymaga skomplikowanych obliczeń, za to tworzy wielokolorowy obraz. Wielobarwne sylwetki graczy nie mogłyby być utworzone przy pomocy przerwań listy displejowej, jako że DLI adresowana jest liniami trybowym pola, a nie rejestrami pozycji playerów w grafice PMG.

Istnieje możliwość zastosowania techniki kerneli do rysowania w pojedynczej linii skaningowej ekranu i zmiany zawartości rejestrów kolorów "w locie". W ten sposób jeden rejestr kolorów może wprowadzić kilka kolorów do pojedynczej linii skaningowej obrazu. Punkt linii skaningowej, w trakcie rysowania którego następuje zmiana zawartości rejestru kolorów, określony jest przez ilość czasu, jaka musi upłynąć do momentu dokonania przepisania rejestru. Stąd, przy dokładnym wyliczeniu ilości cykli zegarowych, można umieścić w jednej linii skaningowej kilka różnych kolorów. Niestety, zastosowanie tej techniki w praktyce jest niezwykle utrudnione. Fakt iż część czasu 6502 jest "kradziona" przez ANTIC na wykonanie DMA, w zasadzie uniemożliwia wykonanie precyzyjnych obliczeń czasowych. Najdokładniejsze nawet obliczenia nie przynoszą spodziewanych rezultatów. Owszem, istnieje możliwość wyłączenia instrukcji DMA ANTICu. w takim przypadku 6502 mógłby całkowicie przejąć kontrolę nad wyświetlanym obrazem, ale jednocześnie musiałby przejąć na siebie wszystkie funkcje układu ANTIC. Wynika stąd, że stosowania techniki kerneli najczęściej jest o wiele bardziej kosztowne, niż uzyskiwane rezultaty. Jednakże - jeżeli dwa obrazy na ekranie, które mają być rysowane w różnych kolorach z jednego rejestru kolorów, są od siebie oddzielone, powiedzmy, dzieli je 20 lub więcej punktów rastrowych ekranu - wówczas odległość ta powinna zrekompensować niedokładności w obliczeniach czasowych i umożliwić zastosowanie techniki kerneli.

ZASTOSOWANIA PRZERWAŃ LISTY DISPLEJOWEJ

Myślę, że teraz prawdziwe znaczenie pośredniego tworzenia grafiki i modyfikowalnych w każdej chwili sprzętowych rejestrów kolorów stało się dla każdego jasne. Przerwania listy displejowej umożliwiają dokonywanie zmian zawartości tych rejestrów niejako w biegu. Z ich pomocą można uzyskać na ekranie wielokolorowe obrazy, specjalne znaki czy też unikalne efekty graficzne. Najpowszechniej wykorzystywanym zastosowaniem DLI jest umieszczenie na ekranie dodatkowych kolorów. Dotyczy to zarówno rejestrów kolorów pola, jak i rejestrów kolorów playerów. A więc dysponujemy 9 rejestrami kolorów, z których każdy może przenieść na ekran 128 różnych kolorów. Czy to wystarczająca ilość kolorów dla twoich potrzeb? Rzecz jasna, normalnie działający program nie może poświęcić aż tyle czasu, by umieścić na ekranie tak wiele kolorów. Stworzenie zbyt wielu procedur DLI spowoduje znaczne spowolnienie wykonywania programu głównego. Czasami też jednym z ograniczeń może być struktura listy displejowej, która nie zaakceptuje zbyt wielu procedur DLI: W praktyce wyświetlenie 12 kolorów na ekranie jest niezwykle proste, uzyskanie 24 kolorów wymaga dość poważnej pracy programisty - otrzymanie większej ilości kolorów jest możliwe tylko w wyjątkowych sytuacjach. Przerwania listy displejowej umożliwiają nie tylko uzyskanie dodatkowych kolorów - także w znaczny sposób rozszerzają możliwości grafiki PMG. procedura DLI może przepisywać rejestr pozycji poziomej obiektu na ekranie - umożliwia to na przykład rozbicie obiektu na kilka części, bądź umieszczenie jego części poza ekranem. Każdy z obiektów ekranowych może przy pomocy DLI przyjmować kilka różnych postaci. Jeżeli zaprogramujemy PMG jako pionową kolumnę, zawierającą kilka postaci, właśnie DLI należy wykorzystać jako nożyczki, służące do pocięcia kolumny na oddzielne postacie i umieszczenia ich w różnych częściach ekranu. Oczywiście,

żadna z części tej kolumny nie będzie mogła zostać umieszczona w jednej linii poziomej z inną jej częścią, jakkolwiek w jednej linii będą mogły być umieszczona dwie postacie tego samego playera. Tak więc, jeżeli tworzony przez ciebie obraz zawiera obiekty, które nigdy nie powinny się znaleźć w jednej linii poziomej, możesz wszystkie te obiekty zaprojektować przy pomocy jednego tylko playera PMG oraz procedury DLI.

Innym zastosowaniem DLI w stosunku do PMG jest zmiana wielkości obiektów PMG oraz priorytetów ekranowych. Może to być często wykorzystywane w połączeniu z trikiem maskowania, wykorzystującym priorytety, opisanym w rozdziale 4.

Ostatnim zastosowaniem DLI jest zmiana zbioru symboli w trakcie rysowania obrazu. Właśnie w ten sposób uzyskuje się możliwość rysowania symboli graficznych w głównej części ekranu, podczas gdy w oknie tekstowym występują symbole literowe. Możliwe jest wykorzystanie wielu różnych zbiorów symboli: program może umieszczać znaki graficzne jednego typu w górnej części ekranu, znaki drugiego typu w środkowej części ekranu oraz standardowy tekst w części dolnej. W programie "Rosetta Stone" wykorzystano na przykład DLI do uzyskania znaków tekstowych różnej wielkości w różnych częściach ekranu. Przy pomocy procedury DLI można także uzyskać lustrzane odbicie tekstu, gdzie w górnej części ekranu tekst będzie normalny, natomiast w dolnej połowie odwrócony do góry nogami.

Właściwe wykorzystanie procedur DLI wymaga dokładnego zaprojektowania architektury ekranu. Programista musi szczegółowo zaplanować kolejne poziome struktury obrazu. Rastrowy skaningowy system telewizyjny nie jest symetryczny w obu wymiarach; znacznie silniej zaznaczona jest struktura pionowa obrazu niż pozioma. Szybkość rysowania obrazu w poziomie jest 262 razy większa niż szybkość rysowania w pionie. System tworzenia obrazu w komputerach domowych Atari jest ściśle związany z rastrowym skaningowym systemem telewizyjnym i dokładnie odzwierciedla anizotropię tegoż systemu. Obraz tworzony przez komputery Atari nie jest czymś podobnym do białej kartki papieru, na której programista noże umieszczać dowolnie znaki i symbole; należy go raczej traktować jak zbiór wąskich poziomych linii, z których każda opisywana jest oddzielnymi parametrami. Programista, który spodziewa się możliwości wykorzystania całkowicie izotropowego ekranu, będzie musiał stoczyć ciężką walkę z systemem. Najlepsze rezultaty uzyskuje się wówczas, gdy od samego początku organizuje się informację, która ma być umieszczona na ekranie, przy rygorystycznym uwzględnieniu skaningowej struktury obrazu. Dopiero takie podejście zapewnia możliwości całkowitego wykorzystania olbrzymich możliwości, jakie stwarzają procedury przerwań listy displejowej.

6. PRZESUWANIE EKRANU

Dosyć często zdarza się, iż ilość informacji, którą programista chce umieścić na ekranie, przekracza ilość, która może zmieścić się tam zmieścić. Jednym z rozwiązań w takim przypadku jest przesuwanie informacji po ekranie. Na przykład listowanie programów BASICu wykorzystuje przesuwanie pionowe w kierunku od dna do szczytu ekranu. Wszystkie komputery osobiste posiadają takie możliwości. Jednakże komputery domowe Atari wyposażone zostały w dwie dodatkowe opcje, poszerzające zakres manewrowania ekranem. Pierwsza z nich to wykorzystanie instrukcji LMS, druga natomiast to płynne przesuwanie ekranu.

Inne komputery zwykle wykorzystują przesuwanie wierszowe, które polega na zmianie pozycji ekranowej punktów rastrowych trybu, wyświetlających określone symbole. W technice oznacza to przepisywanie danych wewnątrz obszaru pamięci ekranowej. Skokiem takiego przesuwania jest wielkość punktu rastrowego, co praktycznie najczęściej oznacza ośmioliniowy symbol - stąd można nazwać tę technikę przesuwaniem wierszowym. Dla użytkownika oznacza to °przeskakiwanie" symboli na sąsiednią pozycję - czy to w pionie czy w poziomie - co nie jest zbyt miłym dla oka efektem, Ponadto technika ta musi przepisywać tysiące danych ekranowych wewnątrz obszaru pamięci ekranowej, co jest metodą dość żmudną i spowalniającą pracę mikroprocesora.

W niektórych komputerach można uzyskać bardziej płynne przesuwanie ekranu poprzez wyświetlanie informacji w trybie graficznym o większej rozdzielczości. Równa się to jednak większej ilości danych, które muszą być przemieszczane w pamięci ekranowej, a tym samym dalszemu spowolnieniu pracy mikroprocesora. Rozwiązanie to jednak opiera się nadal na tej samej zasadzie - przepisywania bajtów informacji wewnątrz pamięci ekranowej,

Komputery Atari 400/800 umożliwiają zastosowanie zupełnie innej techniki do uzyskania analogicznego efektu przesuwania wierszowego: metody przesuwania pamięci ekranowej wśród bajtów informacji. W liście displejowej każdego rodzaju musi znaleźć się instrukcja adresowania pamięci ekranowej - LMS - która informuje ANTIC gdzie zaczyna się obszar pamięci ekranowej. Instrukcja ta była szczegółowo opisana w rozdziale 2. Standardowa lista displejowa zawiera tylko jedną instrukcję LMS, znajdującą się na początku listy; operandem tej instrukcji jest adres początkowy liniowej sekwencji bajtów, które mają zostać umieszczone na ekranie, czyli tak zwanego obszaru pamięci ekranowej. Manipulowanie dwoma bajtami operandu instrukcji LMS umożliwia uzyskanie standardowego wierszowego przesuwania ekranu. Efekt, uzyskany tą metodą, można porównać do przesuwania całoekranowego okna ponad znacznie większym obszarem, zawierającym cały obraz. A więc poprzez manipulowanie dwoma bajtami adresowymi możemy uzyskać taki sam efekt, jak przez przepisywanie tysięcy bajtów informacji wewnątrz pamięci ekranowej. Poniższy program ilustruje tę technikę.

```
10 DLIST=PEEK(560)+256*PEEK(561): REM początek listy displejowej
20 LMSLOW=DLIST+4: REM          młodszy bajt operandu LMS
30 LMSHIGH=DLIST+5: REM         starszy bajt operandu LMS
40 FOR I=0 TO 255: REM          pętla zewnętrzna
50 POKE LMSHIGH, I
```

```
60 FOR J=0 TO 255: REM          pętla wewnętrzna
70 POKE LMSLOW, J
80 FOR Y=1 TO 50: NEXT Y: REM   opóźnienie
90 NEXT J
100 NEXT I
```

Program ten przesuwa pamięć ekranową - a więc tym samym umieszcza na ekranie - ponad całą dostępną pamięcią RAM i ROM komputera. Ukazane w tym przykładzie przesuwanie jest kombinacją przesuwania pionowego i przesuwania poziomego. Przesuwania pionowe może być uzyskane poprzez dodanie lub odjęcie zmiennej liczby (długości wiersza ekranowego w bajtach) do/od operandu instrukcji LMS. Oto przykład takiego programu:

```
10 GRAPHICS 0
20 DLIST=PEEK(560)+256*PEEK(561)
30 LMSLOW=DLIST+4
40 LMSHIGH=DLIST+5
50 SCREENLOW=0
60 SCREENHIGH=0
70 SCREENLOW=SCREENOW+40: REM         następny wiersz
30 IF SCREENLOW<256 THEN GOTO 120: REM poza ekranem
90 SCRENLOW=SCREENLOW-256: REM        tak, wyzeruj wskaźnik
100 SCREENHIGH=SCREENHIGH+1
110 IF SCREENHIGH=256 THEN END
120 POKE LMSLOW, SCREENLOW
130 POKE LMSHIGH,SCREENHIGH
140 GOTO 70
```

PRZESUWANIE W POZIOMIE

Przesuwanie obrazu w poziomie nie da się tak prosto zrealizować jak przesuwanie w pionie. Problem polega znowu na organizacji pamięci ekranowej. Dane ekranowe dla każdej linii obrazu umieszczone są w pamięci sekwencyjnie, gdzie bajty dotyczącej danej linii umieszczone są bezpośrednio za danymi dotyczącymi linii poprzedniej. Przesuwanie linii ekranowych w poziome uzyskuje się poprzez przemieszczanie bajtów w pamięci w lewo - czyli poprzez zmniejszanie wartości operandu instrukcji LMS. Jednakże w tym przypadku lewe skrajne bajty każdej linii będą wyświetlane z prawej strony linii znajdującej się bezpośrednio powyżej. Problem ten pojawił się już w pierwszym przykładowym programie, przedstawionym powyżej.

Rozwiązaniem tegoż problemu jest powiększenie obszaru pamięci ekranowej i rozbicie jej na serię niezależnych regionów, zawierających dane dotyczące jednej tylko linii. Idea ta przedstawiona została schematycznie na rys. 6-1.

Rozmieszczenie danych w pamięci

normalne do przesuwania w poziomie

Rys. 6-1. Organizacja pamięci ekranowej

Lewy rysunek przedstawia normalną organizację pamięci ekranowej. Jednowymiarowy ciąg danych w pamięci jest umownie podzielony na ciąg obszarów, odpowiadających poszczególnym liniom obrazu. Z prawej ukazano organizację pamięci, jaka potrzebna jest do uzyskania poziomego przesuwania obrazu. Oczywiście, organizacja danych w pamięci pozostać musi jednowymiarowa, lecz należy ją zupełnie inaczej zorganizować. Obszar pamięci dla każdej linii obrazu jest znacznie większy, niż w danej linii na obrazie można przedstawić informacji. Rzecz jasna, nie jest to przypadkowe - generalnym założeniem przesuwania ekranu jest konieczność przedstawienia większej ilości danych niż pojedynczy ekran może pomieścić. A nie da się zrealizować tego celu w inny sposób, niż poprzez rozbudowanie objętości pamięci ekranowej. Tylko przy takiej organizacji pamięci ekranowej możliwe będzie uzyskanie przesuwanego ponad danymi całoekranowego okna, bez niepotrzebnego przemieszczania danych pomiędzy wierszami.

Pierwszym krokiem do przeorganizowania pamięci ekranowej musi być określenie maksymalnej ilości znaków w wierszu oraz zaplanowanie odpowiedniego obszaru pamięci RAM. Następnie należy stworzyć całkowicie nową listę displejową, w której instrukcja każdej linii trybowej będzie jednocześnie zawierała instrukcję LMS. Oczywiście, lista taka będzie znacznie dłuższa od standardowej, lecz nie istnieje przecież ograniczenie długości programu ANTICu. Jakich wartości użyć jako operandów instrukcji LMS? Wygodnie jest zastosować adresy pierwszego bajta każdej linii w pamięci ekranowej. Tak więc każda instrukcja wyświetlenia danej linii trybowej będzie jednocześnie zmieniała adres pamięci ekranowej w liście displejowej. Teraz trzeba listę displejową umieścić w pamięci, wskazać układowi ANTIC, gdzie ona się znajduje, a następnie wpisać do pamięci ekranowej dane, które będą umieszczane na ekranie. Uzyskanie przesuwania danych na ekranie uzyskamy poprzez cykliczne zwiększanie operandu adresowego przy przesuwaniu w prawo lub zmniejszanie go przy przesuwaniu w lewo. Program realizujący przesuwanie musi ponadto zawierać opcję, zapewniającą nieprzekroczenie granicy obszaru pamięci zawierającego dane dla jednej linii trybowej - w przeciwnym wypadku uzyskamy znów przeskakiwanie danych z jednej linii do drugiej. Przy ustalaniu takiej opcji należy pamiętać, że operand instrukcji LMS wskazuje aktualnie na pierwszy bajt wyświetlany w danej linii. A więc maksymalna wartość każdego operandu musi być równa adresowi ostatniego bajta danej linii trybowej minus ilość bajtów linii, które aktualnie wyświetlane są na

ekranie. Należy także pamiętać, że obszar pamięci ekranowej dedykowany jednej linii trybowej na ekranie nie może być rozmieszczony po obu stronach 4K przedziału pamięci, gdyż w tym przypadku przekroczenie limitu licznika LMS spowoduje błędne wyświetlenie danych na ekranie.

Ponieważ proces ten wydaje się być nieco skomplikowany, spróbujmy prześledzić go na przykładzie. Po pierwsze, musimy ustalić maksymalną ilość znaków w wierszu. Wybierzmy wiersz 256-znakowy, gdyż uprości to obliczenia. Tak więc dane dotyczące jednej linii będą umieszczone w pamięci na oddzielnej stronie RAM. Załóżmy, iż wykorzystamy tryb BASICu 2, który wyświetla 12 linii trybowych na ekranie. Stąd potrzebować będziemy 12 stron pamięci, czyli 3K RAM; Dla uproszczenia przykładu (a jednocześnie zapewnienia, iż ten obszar nie będzie pustym obszarem pamięci) wykorzystajmy najniższe 3K pamięci RAM. Obszar ten zapisywany jest przez OS oraz DOS, powinien więc zawierać wiele interesujących danych. Aby doświadczenie uczynić jeszcze bardziej interesującym, umieśćmy listę displejową na 6 stronie pamięci - a więc ona również będzie mogła być wyświetlona na ekranie podczas przesuwania. W takim przypadku wartości początkowe operandów instrukcji LM będą niezwykle proste do obliczenia: młodszy bajt będzie w każdym przypadku równy 0 (początek strony pamięci), starszy zaś dla każdej linii trybowej będzie kolejnym numerem strony pamięci. A oto gotowy program, który wykonuje wszystkie opisane wyżej funkcje i przesuwa obraz w poziomie:

```
10 REM          zapisywanie w pamięci nowej listy displejowej
20 POKE 1536,112: REM      wyświetl 8 linii wygaszonych
30 POKE 1537,112: REM      wyświetl 8 linii wygaszonych
40 POKE 1538,112: REM      wyświetl 8 linii wygaszonych
50 FOR I=1 TO 12: REM      pętla zapisująca instr. 12 linii tryb.
60 POKE 1536+3*I,71: REM   tryb 2 BASICu z instrukcją LMS
70 POKE 1536+3*I+1,0: REM  młodszy bajt operandu LMS
80 POKE 1536+3*I+2,I: REM  starszy bajt operandu LMS
90 NEXT I
100 POKE 1575,65: REM      instrukcja JWB ANTICu
110 POKE 1576, 0: REM      młodszy bajt adresu listy displejowej
120 POKE 1577, 6: REM      starszy bajt adresu równego $0600
130 POKE 560,0: REM        zapis adresu listy dla ANTICu
140 POKE 561,6
150 REM teraz wykonujemy przesuwanie ekranu
160 FOR I=0 TO 235: REM    pętla przepisująca LSB operandu LMS
170 REM używamy 235, a nie 255, ponieważ linia zawiera 20 znaków
180 FOR J=1 TO 12: REM     przepisywanie w każdej instrukcji listy
190 POKE 1536+3*J+1, I: REM zapis młodszego bajta operandu LMS
200 NEXT J
210 NEXT I
220 GOTO 160: REM zamknięcie niekończącej się pętli
```

Program ten przesuwa dane po ekranie od strony prawej do lewej. Jeżeli dochodzi on do końca strony pamięci, zaczyna ponownie

wyświetlać dane od początku każdej strony. Ponieważ lista displejowa umieszczona została na stronie 6 pamięci, można ją znaleźć w 6-tym wierszu przesuwających się na ekranie znaków. Pojawia się ona jako ciąg podwójnych znaków zapytania.

Następnym krokiem może być wymieszanie przesuwania poziomego i pionowego w celu otrzymania przesuwania ukośnego. Przesuwanie poziome realizuje się poprzez cykliczne dodawanie lub odejmowanie jedności do operandu instrukcji LMS. Przesuwanie pionowe uzyskiwane jest przez dodawanie lub odejmowanie od operandu LMS wielkości równej długości linii ekranowej w bajtach. Wykonanie jednoczesne obu tych operacji da w rezultacie ukośne przesuwanie informacji na ekranie. Przy przesuwaniu ukośnym istnieją cztery możliwe kierunki przesuwania. Jeżeli, na przykład, długość linii ekranowej wynosi 256 bajtów i chcemy przesunąć ją skośnie w dół i w prawo, musimy do każdego operandu instrukcji LMS w liście displejowej dodać wartość 256+(-1)=255. Będzie to dodawanie wartości 2-bajtowych; przedstawiony wyżej program przykładowy BASICu może nie zrealizować w pełni takiego dodawania, ale generalnie przeadresowywanie takie nie powinno nastręczać trudności. Jednakże we wszystkich przypadkach prawdziwie szybki, dwuwymiarowy efekt uzyskuje się jedynie poprzez napisanie programu w języku asemblera.

Przesuwanie oparte na przepisywaniu operandu instrukcji LMS zezwala na stosowanie wszelkiego rodzaju trików. I tak wiersze ekranu mogą przesuwać się kolejno, jeden za drugim, bądź też przeskakiwać jeden nad drugim. Możliwa jest dowolna konfiguracja operandów instrukcji LMS i dowolna organizacja pamięci ekranowej. Dodatkowe efekty można uzyskać stosując tradycyjne przepisywanie danych wewnątrz obszaru pacnięci ekranowej. Największą zaletą tej techniki przesuwania jest jej prostota i szybkość. Zamiast przepisywania tysięcy bajtów wewnątrz pamięci ekranowej stosuje się przeadresowywanie kilku lub co najwyżej kilkudziesięciu instrukcji LMS w liście displejowej.

PRZESUWANIE PŁYNNE

Drugim, dodatkowym udogodnieniem, stosowanym w komputerach Atari, jest możliwość płynnego przesuwania ekranu. Polega ono na tym, iż można każdy punkt rastrowy ekranu przesuwać z krokiem mniejszym od wielkości danego punktu. W przesuwaniu wierszowym każdy punkt rastrowy był przesuwany o jedną pozycję, czyli krok przesuwania był równy wielkości danego punktu rastrowego - przesuwanie płynne natomiast przesuwa obraz ze skokiem jednej linii skaningowej ekranu w pionie i ze skokiem jednego punktu ekranu w poziomie (jednego piksela). Ograniczeniem płynnego przesuwania jest niezbyt duża odległość, na jaką można przesunąć obraz. W celu uzyskania przesuwania obrazu dłuższymi odcinkami konieczne jest zastosowanie jednoczesnego płynnego i wierszowego przesuwania.

Uzyskanie płynnego przesuwu wymaga jedynie dwóch kroków. Po pierwsze, należy włączyć bit zezwolenia płynnego przesuwania w bajtach instrukcji listy displejowej dotyczących tych linii trybowych, które mają być przesuwane. Oczywiście, w większości przypadkach trzeba będzie włączyć bity wszystkich instrukcji, jako że zwykle przesuwa się cały obraz. Bit D5 instrukcji listy displejowej jest bitem włączającym przesuwanie pionowe, natomiast

bit D4 włącza przesuwanie poziome obrazu. Następnie należy wpisać żądane wartości do odpowiednich rejestrów przesuwania obrazu. Są dwa takie rejestry, jeden dla przesuwania pionowego, drugi dla przesuwania poziomego. Rejestr przesuwania poziomego (HSCROL) znajduje się pod adresem $D404, natomiast rejestr przesuwania pionowego (WSCROL) pod adresem $D405. Przy przesuwaniu poziomym do rejestru należy wpisać ilość punktów ekranu, o jaką obraz ma być przesunięty. Przy przesuwaniu pionowym do rejestru należy wpisać ilość poziomych linii skaningowych, o jaką trzeba przesunąć cały, bądź wyszczególnione linie trybowe obrazu. Przesunięcie o określony wektor zostanie wykonane w stosunku do wszystkich linii trybowych, których instrukcje zezwalają na wykonanie przesuwania.

Stosowanie płynnego przesuwania związane jest z dwiema komplikacjami; obie wynikają z faktu, iż przy przesuwaniu należy w pamięci ekranowej zawrzeć więcej informacji, niż może się jej pomieścić na pojedynczym ekranie. Spróbujmy sobie na przykład wyobrazić, co się stanie, jeżeli przesuniemy w poziomie jeden wiersz tekstu o połowę znaku w lewo. W jednym wierszu znajduje się 40 znaków. Połowa pierwszego znaku zniknie poza krawędzią pola obrazu, znak 40-ty przesunie się o pół znaku w lewo, a co pojawi się na wolnym miejscu? Powinna to być połowa następnego znaku, czyli w kolejności 41-go. Ale przecież pamięć ekranowa zorganizowana jest w wiersze 40-znakowe?

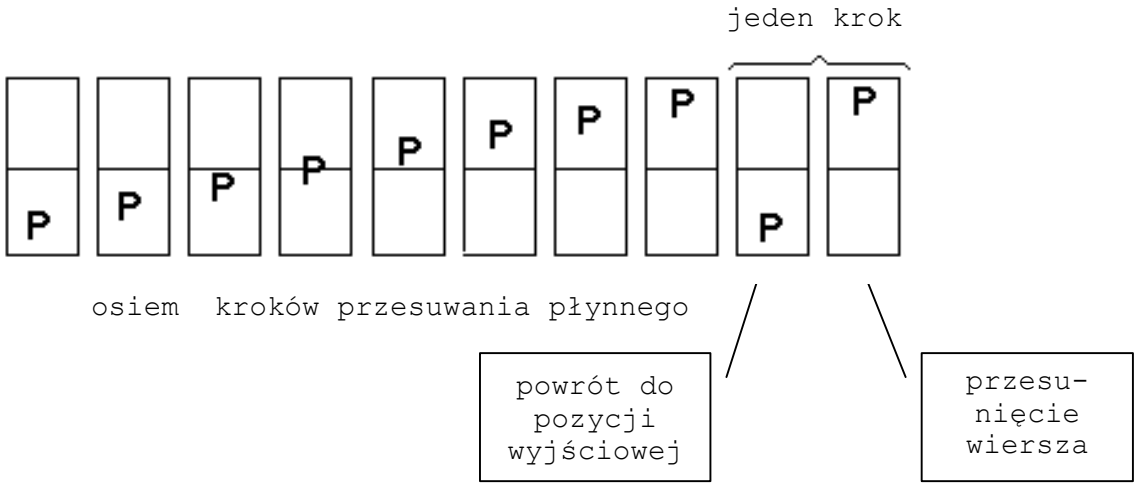
Jeżeli dokładnie zapoznaliśmy się z przesuwaniem wierszowym, wiemy, iż na miejsce przesuniętych w lewo znaków pojawi się znak 41-szy. Jeżeli nie została przeorganizowana pamięć ekranowa, będzie to ponownie znak 1-szy, który zniknął poza lewą krawędzią pola. Problem ten w przypadku przesuwania płynnego rozwiązany został sprzętowo. Mamy do wyboru trzy opcje dotyczące szerokości pola ekranowego: pole wąskie o szerokości 128 punktów ekranu, pole standardowe o szerokości 160 punktów i pole szerokie o szerokości 192 punktów ekranu. Opcje te wybiera się poprzez włączenie odpowiedniego bitu w rejestrze DMACTL. Jeżeli wykorzystujemy płynne przesuwanie, ANTIC automatycznie pobiera z pamięci ekranowej więcej danych, niż jest wyświetlanych w jednej linii trybowej. Jeżeli na przykład bit sterujący w DMACTL jest włączony na standardową szerokość pola, co dla trybu 0 BASICu równe jest 40 znakom w linii, ANTIC będzie odczytywał dane dla pola szerokiego, co równoznaczne jest 48 bajtom dla jednej linii trybowej. Powoduje to, że część danych wyświetlana jest poza ekranem i w trakcie przesuwania może pojawić się na ekranie. Oczywiście, problem ten przestaje istnieć, jeżeli zorganizujemy pamięć ekranową w długich, niezależnych wierszach, jak pokazano to na rys. 6-1.

Analogiczna niedogodność pojawiająca się przy przesuwaniu pionowym może być usunięta dwiema metodami. Najprościej jest zignorować zwiększony obszar pamięci ekranowej i zabezpieczyć się przed wyświetleniem części zamiast całych znaków na obu krawędziach pola ekranowego. W przeciwnym wypadku linie trybowe na dole ekranu nie będą płynnie przesuwane, a będą pojawiały się dopiero po powstaniu wystarczającej ilości miejsca dla jednego wiersza. Uzyskanie prawidłowego przesuwania nie wymaga zbyt wiele pracy. Należy w tym celu poświęcić jedną linię trybową na bufor symboli. Uzyskuje się to poprzez wyłączenie bitu, zezwalającego na przesuwanie pionowe, w instrukcji listy displejowej dotyczącej ostatniego wiersza przesuwanej strefy ekranu. Zapewnia to uzyskanie

płynnego przesuwu, choć z drugiej strony skraca obraz na ekranie o jedna linię trybową.

Bardzo łatwo jest zorganizować pamięć ekranową w taki sposób, by zapisać w niej obraz większy od 192 poziomych linii skaningowych ekranu. Nie ma to żadnego znaczenia przy obrazie statycznym, natomiast przy wykorzystywaniu przesuwania pionowego pozwala na umieszczenie części obrazu powyżej i poniżej widzialnej części ekranu, które to części w trakcie przesuwania będą mogły ukazać się w polu ekranowym.

Przesuwanie płynne pozwala na uzyskanie bardzo szybkiego przesuwania. Jest ono jednak ograniczone dość wąskim limitem - przy przesuwaniu pionowym maksymalne przesunięcie może wynosić 16 linii skaningowych, przy przesuwaniu poziomym natomiast 16 punktów ekranowych. Jeżeli zapiszemy rejestry przesuwania wyższymi wartościami, ANTIC po prostu zignoruje te zapisy. Otrzymanie płynnego przesuwania całego ekranu, z szybkością regulowaną przez użytkownika, możliwe jest tylko przez połączenie przesuwania płynnego i przesuwania wierszowego. Najpierw trzeba przesunąć płynnie obraz o maksymalną dla danego trybu graficznego ilość linii skaningowych (lub punktów ekranowych)- czyli o ilość równą ilości linii w danym trybie po czym należy wyzerować rejestr przesuwania, wykonać jedno przesunięcie wierszowe i ponownie włączyć przesuwanie płynna (rys. 6-2).



Rys: 6-2. Połączenie przesuwania płynnego i wierszowego

A oto prosty przykład programu wykonującego przesuwanie płynne:

```
1 HSCROL=54275
2 WSCROL=54277
10 GRAPHICS 0: LIST
20 DLIST=PEEK(560)+256*PEEK(561)
30 POKE DLIST+10,50: REM   przesunięcie w obu kierunkach
40 POKE DLIST+11,50: REM   dwóch linii trybowych
50 FOR Y=0 TO 7
50 POKE WSCROL,Y:    REM   przesuwanie pionowe
```

```
70 GOSUB 200:        REM   pętla opóźnienia
80 NEXT Y
90 FOR X=0 T o 3
100 POKE HSCROL,X:   REM   przesuwanie poziome
110 GOSUB 200:       REM   pętla opóźnienia
120 NEXT X
130 GOTO 40
200 FOR J=1 TO 200
210 NEXT J: RETURN
```

Program ten realizuje przesuwanie dwóch wierszy ekranowych z bardzo małą prędkością. Ukazuje on jednocześnie kilka trudności związanych z wykorzystaniem przesuwania płynnego. Po pierwsze, tekst znajdujący się poniżej przesuwanych linii jest przesunięty w prawo. Wynika to z faktu, że ANTIC automatycznie odczytuje 8 bajtów pamięci ekranowej dla danej linii, podczas gdy wyświetlanych jest 40. Oczywiście, problem ten jest znaczący w zasadzie tylko w tak nierealistycznych programach, jak przedstawiony wyżej. Przy zastosowaniu organizacji pamięci ekranowej w niezależne obszary odpowiadające poszczególnym liniom trybowym, jak przedstawiono na rys. 6-1, problem ten nie występuje w ogóle. Drugi, poważniejszy problem powstaje wówczas, kiedy zawartości rejestrów przesuwania są przepisywane w tym samym czasie, kiedy ANTIC wykonuje rysowanie obrazu. Powoduje to błędy w wykonywaniu płynnego przesuwania i "przeskakiwanie" obrazu na ekranie. Rozwiązaniem tego problemu jest przepisywanie zawartości rejestrów przesuwania tylko w trakcie okresów wygaszenia pionowego. To z kolei zmusza do tworzenia programów przesuwających jedynie w języku asemblera. Należy więc przyjąć, że pełne wykorzystanie możliwości płynnego przesuwania informacji na ekranie jest możliwe tylko przy stosowaniu programów w języku assemblera.

ZASTOSOWANIA PRZESUWANIA EKRANU

Zastosowania całoekranowego płynnego przesuwania obrazów w programach graficznych należą do rzadkości. Najczęściej stosuje się je przy wyświetlaniu obrazów typu dużych map, tworzonych poprzez grafikę zbiorów symboli. Przy wykorzystaniu trybu 2 BASICu sam stworzyłem mapę ZSRR, o powierzchni przekraczającej pojemność 10 pełnych ekranów, gdzie ekran stanowił okno do przeglądania fragmentów mapy a użytkownik mógł je przesuwać przy pomocy dżojstika. Tworzenie tego typu obrazów jest niezwykle pracochłonne; wspomniany program, łącznie z pamięcią ekranową, listą displejową i zbiorem symboli ekranowych zajął ponad 4K pamięci RAM.

Istnieje wiele innych zastosowań techniki przesuwania płynnego. Każdy duży obraz, stworzony przy pomocy grafiki symboli, niezbyt nadaje się do jej wykorzystania - lecz z drugiej strony stworzenie dużego obrazu przy pomocy trybów nietekstowych jest o wiele bardziej pracochłonne i zajmuje dużo więcej pamięci. Metodą tą można na przykład opracowywać duże schematy elektroniczne. Joystick może być wykorzystywany zarówno do przesuwania okna ponad obrazem, jak i do wskazywania poszczególnych elementów obrazu, które użytkownik pragnie adresować. Wszelkiego typu duże schematy czy rysunki mogą

być wyświetlane metodą przesuwania - o ile nie zachodzi potrzeba ukazania ich w całości. Okno całoekranowe jest w stanie ukazać większą ilość szczegółów,

Przesuwanie ekranu stosowane jest przede wszystkim podczas wyświetlania na ekranie informacji pisanych - choć czasami spotyka się niepraktyczne programy, gdzie przesuwane są pod ekranem bardzo duże bloki tekstów, co niezwykle utrudnia ich czytanie. System ten świetnie nadaje się do niezależnych on siebie bloków tekstowych. Jednym z najpowszechniej stosowanych sposobów jest technika menu. Oto na przykład program wyświetla powitanie, wskazując jednocześnie na kilka oddzielnych menu, znajdujących się na jednym większym obrazie - zarazem wyświetlana jest instrukcja, w jaki sposób należy przesunąć ekran ponad obrazem w tę czy w tamtą stronę. Użytkownik przesuwa obraz do kolejnych menu, dajmy na to przy pomocy dżojstika, po czym dokonuje wyboru danych opcji. Wybór może być dokonywany poprzez przemieszczenie kursora do danej pozycji i naciśnięcie przycisku dżojstika. Nie jest to oczywiście generalna reguła do tworzenia wszystkich programów, lecz przy określonych typach programów może to być najlepsza metoda wyboru opcji programowych.

Istnieją ponadto dwa "niezbadane regiony" w technice płynnego przesuwania ekranu, Pierwszy z nich to płynne przesuwanie selektywne, to znaczy takie, w którym różne linie trybowe obrazu posiadają różne możliwości przesuwania w tę czy w tamtą stronę. Zwykle wykorzystuje się przesuwanie całego ekranu, nie musi być to jednak regułą - można wybrać jedną linię trybową do przesuwania tylko w poziomie, inną do przesuwania tylko w pionie itp. - wszystko zależy od tego, jak widzisz realizację założeń swojego programu. Drugim niezbadanym obszarem jest projekt wykorzystania przerwań listy displejowej do przepisywania zawartości rejestrów przesuwania HSCROL i WSCROL niejako "w locie". Nigdy nie próbowałem tego czynić, wydaje się jednak, że przepisanie WSCROL przez DLI może całkowicie zdezorientować ANTIC i spowodować zniekształcenia obrazu. Przepisanie HSCROL przez DLI także należy do trudnych zadań, wydaje się jednak, że nie powinno ono powodować błędów ekranowych.

7. DŹWIĘK

Komputery domowe Atari 400/300 wyposażone zostały w szerokie sprzętowe możliwości generacji dźwięku. Posiadają one cztery niezależnie od siebie sterowane kanały dźwiękowe, które mogą być wykorzystywane jednocześnie. Każdy z kanałów posiada własny rejestr częstotliwości, określający wysokość dźwięku, oraz rejestr sterujący regulujący głośność oraz zawartość szumów. Kilka opcji pozwala na dołączenie filtrów górnoprzepustowych, wybór szybkości taktowania, wybór kilku trybów realizacji dźwięku oraz regulację liczników poly.

DEFINICJE TERMINÓW I TECHNIK

Dla celów niniejszego opisu musimy najpierw zdefiniować kilka terminów i pojęć:

1 Hz /herc/ oznacza 1 puls na sekundę

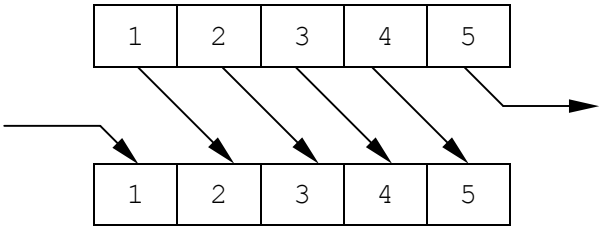
1 kHz /kiloherc/ oznacza 1000 pulsów na sekundę

1 MHz /megaherc/ oznacza 1000000 pulsów na sekundę

Puls, albo impuls, oznacza napięcie zmienne, cyklicznie wzrastające i opadające. Pojedynczy impuls słyszany będzie przez głośnik jako krótki stuk.

Fala oznacza ciągłą sekwencję impulsów. Rozróżniamy różne rodzaje fal, różniących się między sobą sposobem narastania i opadania napięcia pojedynczego impulsu. Generatory komputerów Atari wytwarzają fale prostokątne (jak na rys. 7-2). Typowe instrumenty dęte są źródłem fal trójkątnych, natomiast głos ludzki jest falą o przebiegu sinusoidalnym (jak na rys. 7-15).

Rejestr przesuwający jest rejestrem analogicznym do komórki pamięci (przechowującej dane w postaci binarnej), który - po włączeniu - przesuwa wszystkie bity w nim zapisane o jedną pozycję w prawo. Oznacza to, że bit, który znajdował się na pozycji 5, będzie na pozycji 4, bit, który znajdował się na pozycji 4, będzie na pozycji 3 itd. Tak więc bit znajdujący się na skrajnej prawej pozycji zostanie zapomniany, natomiast na skrajną lewą pozycję zostanie zapisany bit znajdujący się w obwodzie wejściowym rejestru. Schemat takiego przesunięcia przedstawiono na rys: 7-1.



Rys. 7-1. Przepisywanie bitów w rejestrze przesuwającym

AUDF1-4 są rejestrami częstotliwości dźwięku odpowiednich kanałów (1-4). Znajdują się one odpowiednio pod adresami: $D200, $D202, $D204, $D206 (53760, 53762, 53764, 53766).

AUDC1-4 są rejestrami sterującymi dźwięku odpowiednich kanałów 1-4. Znajdują się one pod adresami: $D201, $D203, $D205, $D207 (53761, 53763, 53765, 53767).

Dla potrzeb tej dyskusji określmy częstotliwość jako miarę ilości impulsów w jednostce czasu. Oznacza to, że przy częstotliwości na przykład 100 herców, w ciągu 1 sekundy przesłanych zostanie dokładnie 100 impulsów. Im szybciej przesyłane będą impulsy, tym wyższy otrzymamy dźwięk. Przyjmijmy, że określenia "ton", "częstotliwość" i "wysokość dźwięku" stosować będziemy zamiennie.

Podobnie zamiennie używać będziemy określeń "szum" i "dystorsja", choć w rzeczywistości nie są one równoznaczne. W przypadku komputerów Atari bardziej adekwatnym określeniem wydaje się być "szum".

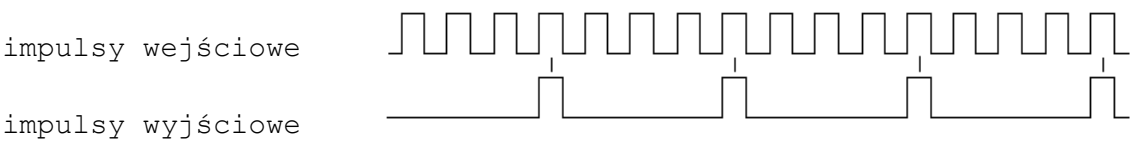
Wszystkie zawarte tu przykładowe programy napisane zostały w BASICu, poza jednym, napisanym w języku asemblera. Wykorzystanie tych przykładów powinno pomóc w zrozumieniu natury syntezy dźwięku. Należy jedynie pamiętać, by stosować je dokładnie tak, jak pojawiają się w tekście - nie wszystkie bowiem programy zawierają kolejne numery linii, a często w jednej linii trybu bezpośredniego BASICu wpisywanych jest kilka instrukcji.

SPRZĘTOWY SYSTEM SYNTEZY DŹWIĘKU

W komputerach Atari dźwięk generowany jest przez układ zwany POKEY, który ponadto zarządza szeregową szyną I/O oraz klawiaturą. Prawidłowa praca tego układu zależna jest od jego zainicjalizowania, które jest niezbędne po wykonaniu każdej operacji I/O, czy to w przypadku kasety, stacji dysków, drukarki, czy też łącza RS-232. Inicjalizacja w BASICu polega na wykonaniu instrukcji SOUND 0,0,0,0; SOUND 1,0,0,0 itd. W języku asemblera należy wpisać 0 do rejestru AUDCTL (SD208; 53768) oraz 3 do rejestru SKCTL (SD20F; 53775) - cieniowanego przez $232 (562).

AUDF1 - 4

Każdy kanał syntezy dźwięku posiada własny rejestr częstotliwości wytwarzanego dźwięku. Rejestr ten zapisany jest wartością N, wykorzystywaną przez układ dzielenia przez N. Nie jest to oczywiście dzielenie w sensie matematycznym, ale coś znacznie prostszego: z każdych N pulsów wchodzących do układu na wyjście przepuszczany jest 1 impuls. Dla przykładu na rys. 7-2 pokazano schemat dzielenia impulsów wejściowych przez 4.



Rys. 7-2. Schemat operacji dzielenia częstotliwości przez 4

Stąd właśnie, jeżeli wzrasta wartość N, maleje częstotliwość impulsów wyjściowych, co powoduje powstawanie niższych dźwięków.

AUDC1-4

Każdy kanał posiada także odpowiadający mu rejestr sterujący. Rejestry te pozwalają na sterowanie natężeniem oraz zawartością szumów dźwięku wytwarzanego przez każdy kanał. Architektura bitów w rejestrach AUDC1-4 jest następująca:

Nr bitu AUDC							
7	6	5	4	3	2	1	0
dystorsja dźwięku			tylko regul. głośn.	głośność dźwięku			

Rys. 7-3. Architektura bitów w rejestrach AUDC1-4,

- Sterowanie głośnością

Sterowanie natężeniem dźwięku każdego kanału dokonywane jest metodą bezpośrednią. Cztery młodsze bity rejestru sterującego zawierają 4-bitową liczbę określającą głośność danego kanału. Wartość 0 oznacza całkowite wyciszenie, wartość 15 oznacza maksymalną głośność syntezy dźwięku. Suma wartości, określających głośność wszystkich czterech kanałów, nie powinna przekraczać wartości 32, ponieważ przy większym natężeniu dźwięku występuje wzajemna modulacja kanałów dźwiękowych, objawiająca się obniżeniem głośności poszczególnych kanałów oraz pojawieniem się nieprzyjemnych, "warczących" przydźwięków.

- Sterowanie dystorsji

Rysunek 7-3 ukazuje; że w każdym rejestrze sterującym znajdują się ponadto 3 bity, sterujące dystorsję każdego kanału. Efekty dystorsyjne wykorzystywane są w przypadkach syntezy specjalnych efektów dźwiękowych.

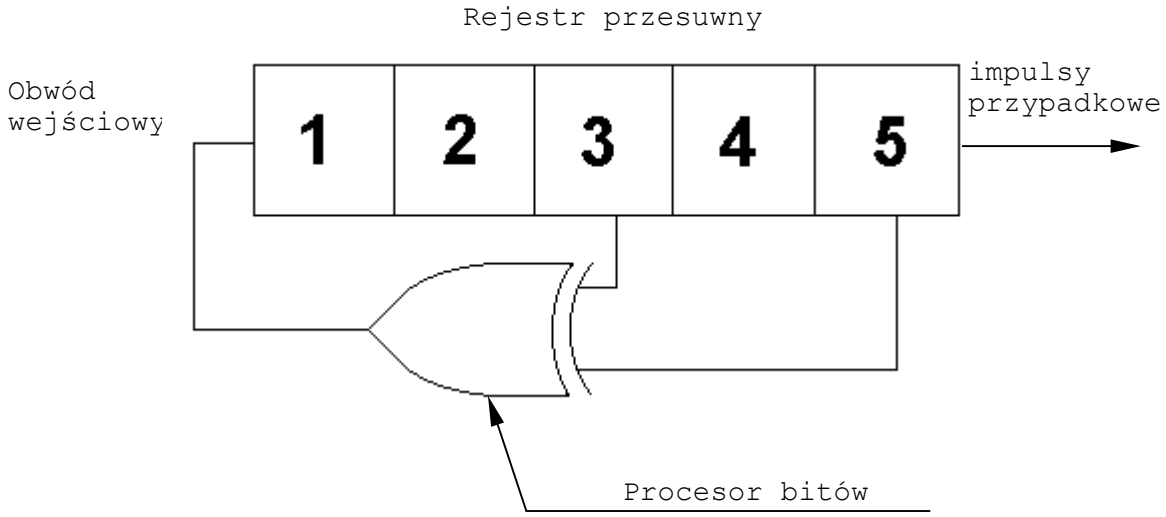
Możliwość wykorzystania efektów dystorsyjnych w dźwiękach syntetyzowanych przez komputer stwarza szeroką gamę jakości uzyskiwanego brzmienia. Z ich pomocą można programować niemal nieskończoną ilość odmian brzmieniowych, od dudnienia i stukotu przez pisk do dzwonienia i szumu, włączając w to syntezę perkusyjnego podkładu muzycznego.

Stosowane w tym przypadku określenie "dystorsja" nie jest równoznaczne z jego standardowym znaczeniem. Dystorsja intermodulacyjna i dystorsja harmoniczna są kryteriami, stosowanymi przy określaniu jakości dźwięków przenoszonych przez systemy stereo hi-fi.

W takim znaczeniu dystorsja odnosi się do zniekształceń fali dźwiękowej, gdzie widmo fali ulega odkształceniom wskutek wadliwego funkcjonowania układów elektronicznych. W komputerowej syntezie dźwięku fale nic ulegają zniekształceniom, są to w każdym przypadku

fale o przebiegu prostokątnym - tutaj dystorsja odnosi się do „wycięcia" niektórych impulsów z sekwencji występującej w fali. Tak więc technika ta niewiele ma wspólnego z określeniem "dystorsja". O wiele bardziej adekwatnym jest pojęcie zawartości szumu w syntezowanym dźwięku, jednak w literaturze bardzo często mówi się o dystorsji syntetyzowanych dźwięków.

Pełne zrozumienie pojęcia dystorsji dźwięku jest nierozerwalnie związane z zasadą działania liczników poly. W komputerach Atari 400/800 liczniki poly wykorzystywane są jako źródło impulsów przypadkowych, wykorzystywanych przy generacji szumu. Wszystkie liczniki poly wykorzystują rejestr przesuwający pracujący z częstotliwością 1:79 MHz. Zawartość wszystkich bitów rejestru jest przesuwana o jedną pozycję, a na wolne miejsce wpisywany jest bit będący kombinacją dwóch innych bitów tego rejestru. Metoda ta powoduje, iż na wyjściu rejestru otrzymuje się pseudolosowy ciąg bitów. Przykład działania licznika poly przedstawiono schematycznie na rysunku 7-4. Bit 5 rejestru przesuwnego przesyłany jest na wyjście jako kolejny impuls przypadkowy, a na pozycję 1 rejestru wpisywany jest bit będący funkcją wartości bitów 3 i 5.



Rys. 7-4. 5-bitowy licznik poly

Na wyjściu liczników poly nie uzyskuje się impulsów całkowicie przypadkowych - po pewnym czasie sekwencje impulsów na wyjściu licznika zaczynają się powtarzać. Rzecz jasna, czas powtarzania sekwencji impulsów jest uzależniony od wielkości licznika - im większa jest pojemność licznika (rejestru przesuwnego), tym dłuższy czas musi upłynąć, nim impulsy zaczną być powtarzane.

Dystorsja dźwięku uzyskiwana jest w komputerach Atari poprzez wykorzystanie impulsów przypadkowych z liczników poly w obwodach selekcji. Obwody te to nic innego jak komparatory cyfrowe, lecz określenie "obwody selekcji" znacznie lepiej oddaje zasadę ich działania. Na wyjście takiego obwodu przesyłane są tylko te impulsy wejściowe, które koincydują z impulsami licznika poly. Tak więc pewna ilość impulsów wejściowych jest w tym obwodzie eliminowana w sposób przypadkowy. Efekt ostateczny jest następujący: niektóre impulsy, pochodzące z dzielnika częstotliwości, są w tym obwodzie

"wykasowywane”. Ponieważ zmieniana jest ilość (i częstotliwość) impulsów w fali, metodą tą uzyskuje się inny brzmieniowo dźwięk. W ten sposób otrzymuje się dystorsję dźwięku, albo inaczej szum.

Ponieważ, jak już wspomniano, liczniki poly powtarzają co pewien czas wyjściową sekwencję impulsów, wyjściowy ciąg impulsów: przypadkowych jest cykliczny. Tym samym fala, otrzymywana na wyjściu obwodu selekcji, będzie również składała się z powtarzających się cyklicznie ciągów impulsów. W ten właśnie sposób powstają dźwięki naśladujące odgłos pracy silnika, wystrzałów i temu podobne.

Komputery Atari posiadają trzy liczniki poly o różnej długości, które mogą być ze sobą w określony sposób kombinowane. Liczniki o małej pojemności, 4- i 5-bitowe, powtarzają wytwarzane sekwencje impulsów w dość krótkich odstępach czasu, wykorzystywane więc być mogą do wytwarzania dźwięków o powtarzającym się cyklu; licznik o dużej pojemności, 17-bitowy, zaczyna powtarzać wytwarzaną sekwencję impulsów po tak długim okresie czasu, że dystorsja dźwięku uzyskana przy jego pomocy nie odznacza się praktycznie żadną cyklicznością. Licznik ten może być wykorzystywany do tworzenia odgłosów eksplozji, deszczu itp., to znaczy takich dźwięków, gdzie potrzebne są impulsy przypadkowe i szum, a nie cyklicznie powtarzający się ciąg impulsów, Nieregularność pracy tego licznika jest na tyle duża, że przy jego pomocy wytwarzać można szum biały o widmie niemalże regularnie ciągłym.

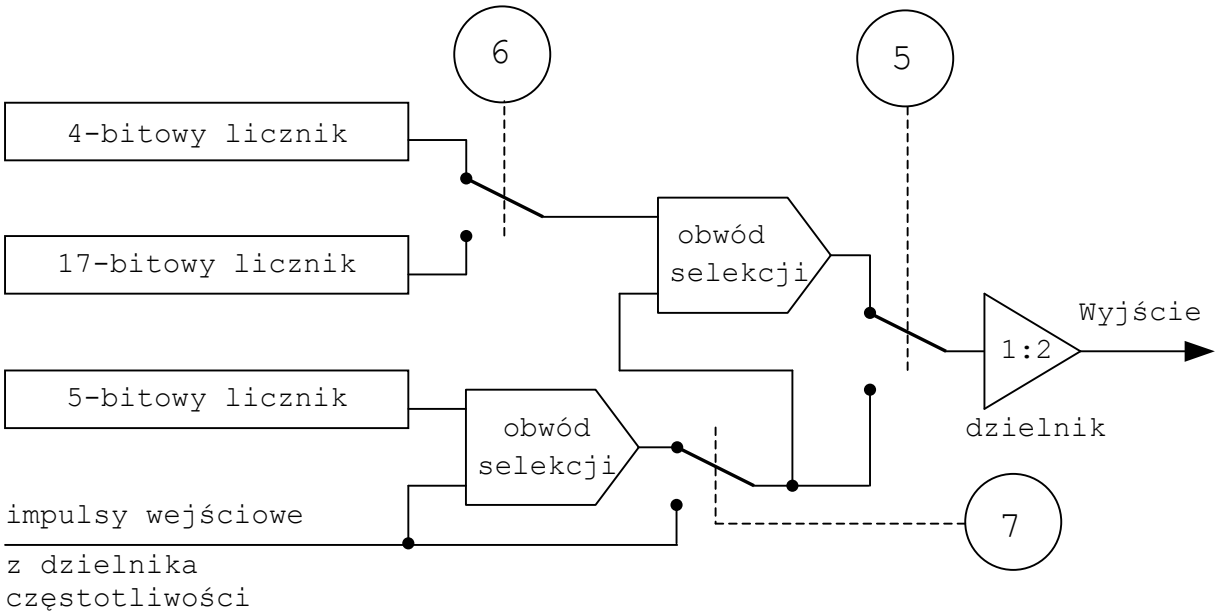
Wszystkie trzy liczniki poly mogą być stosowane w sześciu różnych kombinacjach niezależnie dla każdego z czterech kanałów syntezy dźwięku, jak pokazano to na schemacie 7-1.

A U D C 1 - 4							
7	6	5	4	3	2	1	0

0	0	0	podziel częstotliwość wyjściową, w obwodzie-selekcji zastosuj najpierw licznik 5-bitowy potem 17-bitowy i podziel częstotliwość przez 2
0	X	1	podziel częstotliwość wejściową, w obwodzie selekcji zastosuj licznik 5-bitowy i podziel częstotliwość przez 2
0	1	0	podziel częstotliwość wejściową, w obwodzie selekcji zastosuj najpierw licznik 5-bitowy potem 4-bitowy i podziel częstotliwość przez 2
1	0	0	podziel częstotliwość wejściową, w obwodzie selekcji zastosuj licznik 17-bitowy i podziel częstotliwość przez 2
1	X	1	podziel częstotliwość wejściową, podziel częstotliwość przez 2 (nie stosuj obwodów selekcji)
1	1	0	podziel częstotliwość wejściową, w obwodzie selekcji zastosuj licznik 4-bitowy i podziel częstotliwość przez 2

Uwaga - X oznacza nieznaczącą wartość danego bitu

Rys. 7-6. Możliwe kombinacje liczników poly w obwodzie selekcji.



Rys. 7-7. Schemat blokowy działania rejestrów AUDC1-4

Cztery najstarsze bity każdego z rejestrów AUDC1-4 sterują więc trzy elektroniczne przełączniki, co pokazano na rysunku 7-7. Rysunek ten powinien pomóc w zrozumieniu, dlaczego taka a nie inna jest architektura najstarszych bitów w rejestrach AUDC1-4 i jakie efekty uzyskuje się przez odpowiednie ich stosowanie.

Każda z kombinacji liczników poly pozwala na uzyskanie różnych brzmieniowo dźwięków, Ponadto, przy każdej ze stosowanych kombinacji liczników uzyska się inny dźwięk przy innych częstotliwościach impulsów wejściowych. Stąd też, aby uzyskać najbardziej adekwatny do danych potrzeb dźwięk, najlepiej posłużyć się metodą prób i błędów. Poniżej przedstawiam tabelę, która pozwoli zorientować się jakie rodzaje dźwięków można uzyskać przy różnych kombinacjach liczników poly oraz różnych zakresach częstotliwości.

AUDC bit#			Częstotliwości		
7	6	5	niskie	średnie	wysokie
0	0	0	licznik Geigera	płomień	pęd powietrza	strumień
0	X	1	karabin masz.	samochód	silnik elektr.	transformator
0	1	0	ognisko	samochód	wizg silnika
1	0	0	walący się budynek	zakłócenia radiowe	wodospad
1	X	1	c z y s t e d ź w i ę k i
1	1	0	samolot	kosiarka	golarka elektryczna

Tab. 7-8. Rodzaje odgłosów wytwarzane przy różnych dystorsjach dźwięku i różnych zakresach częstotliwości.

- Sterowanie tylko głośności

Bit 4 rejestrów AUDC1-4 włącza tryb pracy, w którym sterowana jest jedynie głośność. Włączenie tego bitu powoduje, że na wyjście układu generacji dźwięku przesyłana jest wartość zapisana w bitach 0-3 rejestrów AUDC1-4. Wartość ta nie jest modulowana przez impulsy, których częstotliwość określają rejestry AUDC1-4.

Aby w pełni zrozumieć wykorzystanie tego trybu pracy, musimy wpierw uzmysłowić sobie mechanizm działania głośnika - czyli mechanizm tworzenia dźwięków przez impulsy przesyłane do głośnika. Głośnik wyposażony jest w elastyczną membranę, która może przesuwać się w przód i w tył. Pozycja membrany w każdym momencie jest wprost proporcjonalna do napięcia sygnału wysyłanego przez komputer. Jeżeli napięcie sygnału spada do zera, membrana powraca do pozycji spoczynkowej. Ruch membrany wywołuje drganie powietrza, które odbierane jest przez ucho ludzkie jako dźwięk.

Z naszej definicji impulsu wynika, że jest to sygnał napięciowy, gdzie po wzroście napięcia do pewnej wartości następuje jego spadek do zera. Impuls taki przesłany do głośnika spowoduje, że w miarę wzrostu napięcia sygnału membrana będzie się przesuwała ku przodowi, natomiast przy spadku napięcia będzie cofała się do pozycji spoczynkowej, co w rezultacie wywoła falę powietrza, odebraną przez nasze ucho jako pojedynczy stuk. Przesłanie pojedynczego impulsu do głośnika, co słyszane będzie jako pojedyncze uderzenie, można uzyskać sekwencją instrukcji:

```
POKE 53761, 31: POKE 53761,16
```

Ciąg impulsów, czyli fala, będzie powodowała ciągły ruch membrany głośnika, a wiec słyszalny będzie ciągły dźwięk. Im częściej będą przesyłane impulsy, czyli im większa będzie ich częstotliwość, tym wyższy dźwięk będzie produkowany przez membranę głośnika. W ten sposób generowane są dźwięki w głośniku telewizora.

Należy podkreślić, że przy trybie pracy sterującym jedynie głośność wytwarzanych dźwięków natężenie dźwięku nie będzie spadało automatycznie do zera, lecz pozostanie niezmienne, dopóki zmiana głośności nie zostanie przeprowadzona przez program. Program, aby wytwarzać szum, musi dostatecznie często zmieniać głośność dźwięku: Spróbujmy teraz następującego ciągu instrukcji, wsłuchując się uważnie po wczytaniu każdej z nich: .

```
POKE 53761,31
POKE 53761,31
```

Poprzednim razem słychać było pojedyncze uderzenie - do głośnika przesyłany był pojedynczy impuls napięciowy. Teraz nie było słychać nic. Ponieważ membrana głośnika znajdowała się już w pozycji spoczynkowej, kolejna tego typu instrukcja nie zmieniła jej położenia, a więc nie wywołała jej ruchu, tym samym nie powstał żaden dźwięk. Spróbujmy teraz innych instrukcji:

```
POKE 53761,16
POKE 53761,16
```

Tak jak za pierwszym razem słychać było stukot, kiedy membrana zmieniała swe położenie. Lecz podczas wykonywania drugiej instrukcji nie było słychać nic, ponieważ membrana głośnika po wykonaniu pierwszej instrukcji znajdowała się już w danym położeniu, a więc druga instrukcja nie spowodowała zmiany jej położenia.

W ten właśnie sposób, poprzez ustalenie trybu pracy sterującego tylko natężeniem dźwięku, program może bezpośrednio sterować położeniem membrany głośnika przez cały czas. Powyższe przykłady ukazały zastosowanie jedynie dwóch krańcowych położeń membrany, nie jest to jednak żadne ograniczenie. Program może wykorzystywać każde z 16 różnych położeń membrany głośnika, na jakie pozwala zakres regulacji zmiany natężenia dźwięku.

Dla przykładu, generacja prostej fali o przebiegu trójkątnym (podobnej do tej, jaką wytwarzają blaszane instrumenty dęte) można uzyskać poprzez narzucenie kolejnych wartości głośności dźwięku:

8, 9, 10, 11, 10, 9, 8, 7, 6, 5, 6, 7 - i powrocie 8 - oraz powtarzanie tej sekwencji z duża szybkością. Oczywiście, metodą tą nożna uzyskać generację fal dźwiękowych o dowolnym widmie, nie wyłączając syntezy głosu ludzkiego. Wymaga to jednak stosowania wyłącznie programów maszynowych. W dalszej części rozdziału technika ta zostanie omówiona szerzej.

- AUDCTL

W uzupełnieniu rejestrów sterowania niezależnych kanałów syntezy dźwięku (AUDC1-4) istnieje dodatkowo rejestr sterujący trybem pracy wszystkich kanałów jednocześnie. Każdy bit rejestru AUDCTL posiada odrębną funkcję:

AUDCTL bit#	Znaczenie bitu
0	Ustawienie częstotliwości zegara: 0=64 kHz 1=15 kHz
1	Filtr górnoprzepustowy taktowany z: 0-kanał 4, 1-kanał 2
2	Filtr górnoprzepustowy taktowany z: 0-kanał 3, 1-kanał 1
3	Kanał 3 i 4: 0-niezależne, 1-połączone (sterowanie 16bitowe)
4	Kanał 1 i 3: 0-niezależne, 1-połączone (sterowanie 16bitowe)
5	Kanał 3 taktowany: 0 - 15 lub 64kHz, 1 – 1,79 MHz
6	Kanał 1 taktowany: 0 - 15 lub 64kHz, 1 – 1,79 MHz
7	0 – 17-bitowy licznik poly, 1 – 9-bitowy licznik poly

Rys: 7-9. Znaczenie poszczególnych bitów rejestru AUDCTL

- Taktowanie

Zanim zajmiemy się szczegółowym omawianiem znaczenia poszczególnych bitów rejestru AUDCTL wyjaśnijmy jeszcze jedno pojęcie: taktowanie. Generalnie taktujące impulsy zegarowe służą do synchronizacji milionów wewnętrznych operacji, które mają miejsce w komputerze w ciągu każdej sekundy. Zegar bazowy komputera nieprzerwanie wytwarza impulsy taktujące, które nakazują obwodom

wykonać następny krok w wykonywaniu operacji. Zasadą tworzenia dźwięku w komputerach Atari jest praca dzielników częstotliwości, dzielących sygnał wejściowy przez N, czyli przesyłających na wyjście jeden spośród N sygnałów wejściowych. Nie wyjaśniliśmy wówczas skąd pochodzą impulsy wejściowe. Komputer wyposażony jest w jeden główny zegar bazowy, pracujący z częstotliwością 1,79 MHz - impulsy wejściowe mogą pochodzić od niego. Istnieje ponadto cały szereg, liczników drugorzędnych, które również mogą być źródłem impulsów wejściowych. Rejestr AUDCTL umożliwia między innymi dokonanie wyboru, który spośród zegarów układu będzie licznikiem źródłowym dla dzielników częstotliwości.

Załóżmy, na przykład, że wykorzystujemy jako źródło impulsów zegar 15 kHz, a rejestr częstotliwości zapisany jest liczba 8. Oznacza to, że dzielniki częstotliwości dzielą sygnał wejściowy, 15 kHz przez 8, a więc na wyjściu uzyskujemy falę o częstotliwości poniżej 2 kHz. Przełączmy teraz główny zegar syntezy dźwięku na licznik, pracujący z częstotliwością 64 kHz, nie zmieniając jednocześnie rejestru częstotliwości. Dzielniki częstotliwości będą nadal przesyłały na wyjście co ósmy impuls wejściowy - lecz przecież impulsy wejściowe posiadają teraz częstotliwość 64 kHz, a więc częstotliwość impulsów wyjściowych będzie wynosiła 8 kHz.

Obliczenie częstotliwości impulsów wyjściowych jest bardzo proste:

$$\text{częstotliwość wyjściowa} = \frac{\text{częstotliwość zegara}}{N}$$

Włączenie bitu 1 rejestru AUDCTL przełącza układ syntezy dźwięku z zegara bazowego 64 kHz na zegar 15 kHz. Umożliwia to uzyskanie dźwięków o bardzo niskich częstotliwościach. Należy też zapamiętać, że przełączenie takie dotyczy wszystkich kanałów syntezy dźwięku jednocześnie. Analogicznie włączenie bitów 5 lub 6 rejestru AUDCTL pozwala na przełączenie odpowiednio kanałów 3 lub 1 na centralny zegar komputera 1.79 MHz, co pozwala na uzyskanie bardzo wysokich częstotliwości. Poniższy przykład ilustruje efekt przełączenia kanału dźwiękowego na inny zegar bazowy:

```
SOUND 0,255,10,8      Niski dźwięk wytwarzany w kanale 1
POKE 53768, 64        włączenie bitu 6 rejestru ALDCTL
```

- 16-bitowe sterowanie częstotliwości

Rozdzielczość częstotliwości dźwięku, jaką stwarzają 8-bitowe rejestry sterowania częstotliwością, w większości wypadków jest wystarczająca do uzyskania żądanej częstotliwości impulsów wyjściowych. Jednakże czasami rozdzielczość ta jest zbyt mała. Spróbujmy na przykład wykonać następujący ciąg instrukcji:

```
FOR I=255 TO 0 STEP -1: SOUND 0,I,10,8: NEXT I
```

W początkowej fazie narastania dźwięku zmiany są płynne, lecz im wyższe są wytwarzane częstotliwości dźwięku, tym coraz wyraźniej odbiera się skokową zmianę wysokości dźwięku. Dzieje się tak dlatego, że impulsy wejściowe z licznika bazowego dzielone są kolejno przez coraz mniejsze liczby. 15 kHz podzielone przez 255 daje prawie taką samą częstotliwość jak 15 kHz podzielone przez 254; lecz różnica częstotliwości między 15 kHz podzielone przez 3 a 15 kHz podzielone przez 1 jest olbrzymia. Jedynym rozwiązaniem tego problemu jest użycie większych liczb określających krok dzielników częstotliwości, co pozwoli na określenie częstotliwości impulsów wyjściowych z dużo większą precyzją. Możliwość taka została zaprojektowana w układzie POKEY.

Bity 3 i 4 rejestru AUDCTL pozwalają na łączenie dwóch kanałów dźwiękowych w jeden o znacznie rozszerzonej dynamicznej skali częstotliwości. W warunkach normalnych krok dzielników częstotliwości każdego kanału może być określony liczbą z przedziału od 0 do 255 (8 bitów rejestru sterującego częstotliwością). Połączenie dwóch kanałów pozwala na określenie kroku dzielników częstotliwości liczbą z zakresu od 0 do 65535 (16 bitów połączonych rejestrów kontroli częstotliwości). Jednocześnie ten tryb pacy pozwala na obniżenie dolnej granicy wytwarzanych częstotliwości poniżej 1 Hz. Oto przykładowy program, który łączy dwa kanały dźwiękowe w jeden o 16-bitowym sterowaniu oraz wykorzystuje dwa paddle w porcie 1 jako sterowniki częstotliwości wyjściowej:

```
10 SOUND 0,0,0,0: REM  Inicjalizacja kanału dźwiękowego 0
20 POKE 53768, 80: REM  Zegar bazowy kanału 1 = 1,79 MHz
                              połączenie kanałów 1 i 2
30 POKE 53761,160: POKE 53763,163: REM Wyłączenie kanału 1, czyste
                                       dźwięki w kanale 2
40 POKE 53760,PADDLE(0): POKE 53762,PADDLE(1)
50 GOTO 40: REM Pętla czytająca paddle
```

Prawy paddle zmienia dźwięki skokowo, podczas gdy lewy umożliwia dokładne dostrojenie częstotliwości.

Program ten włącza początkowo bity 4 i 6 rejestru AUDCTL, które przełączają kanał 1 na centralny zegar bazowy komputera 1,79 MHz oraz łączą karały 1 i 2. W ten sposób 8-bitowe rejestry sterujące częstotliwością obu kanałów zostały połączone i będą odczytywane jako rejestr 16-bitowy, zawierający liczba kroków dzielnika częstotliwości. Następnie głośność kanału 1 ustalamy na 0. W układzie tym kanał 1 nie jest bezpośrednio dołączony do wyjścia układu syntezy dźwięku, a więc jego głośność jest nieznacząca. Rejestr sterowania częstotliwością kanału 1 jest używany jako precyzujący dokładnie częstotliwość, czyli zawiera młodszy bajt liczby sterującej pracą dzielnika częstotliwości. Na przykład zapisanie rejestru sterowania częstotliwością kanału 1 liczbą 1 spowoduje, że impulsy wejściowe połączonych kanałów 1 i 2 dzielone będą przez 1; zapisanie tej samej liczby w rejestrze kontroli częstotliwości kanału 2 spowoduje dzielenie przez 256; zapisanie liczby 1 w obu rejestrach równocześnie da w efekcie dzielenie impulsów przez 257.

Bit 3 rejestru AUDCTL pozwala na połączenie kanałów 4 i 3 we wspólny kanał o sterowaniu 16-bitowym w dokładnie taki sam jak opisany wyżej sposób.

Poniższy ciąg instrukcji demonstruje kilka ciekawych aspektów 16-bitowego sterowania częstotliwością dźwięku:

```
SOUND 0, 0, 0, 0
POKE 53768,24
POKE 53761,168
POKE 53763,168
POKE 53765,168
POKE 53767,168
POKE 53760,240: REM należy spróbować...
POKE 53764,252: REM wpisać inne wartości
POKE 53762,28: REM do tych czterech
POKE 53766,49: REM rejestrów AUDF1-4
```

- Filtry górnoprzepustowe

Bity 1 i 2 rejestru AUDCTL dołączają na wyjście kanałów odpowiednio 2 i 1 filtry górnoprzepustowe. Filtry te przepuszczają jedynie impulsy o dużych częstotliwościach, wyższych od częstotliwości granicznej filtra. W tym przypadku częstotliwość graniczną określają impulsy znajdujące się na wyjściu innego kanału dźwiękowego, określonego przez kombinację bitów rejestru AUDCTL. Dla przykładu, jeżeli kanał 3 zaprogramujemy na tworzenie ryczenia krowy, po czym włączymy bit 2 rejestru AUDCTL, wówczas z dźwięków wywarzanych w kanale 1 słyszalne będą jedynie te o częstotliwościach wyższych, niż produkowane w kanale 3 ryczenie krowy - dźwięki o częstotliwościach niższych zostaną odfiltrowane. Sytuacja ta przedstawiona została na rys. 7-10.

Częstotliwość graniczna filtrów górnoprzepustowych jest programowalna w czasie rzeczywistym, jako że dźwięk wytwarzany w kanale sterującym pracą filtru może być zmieniony w każdej chwili. Otwiera to bardzo szerokie możliwości programowe.

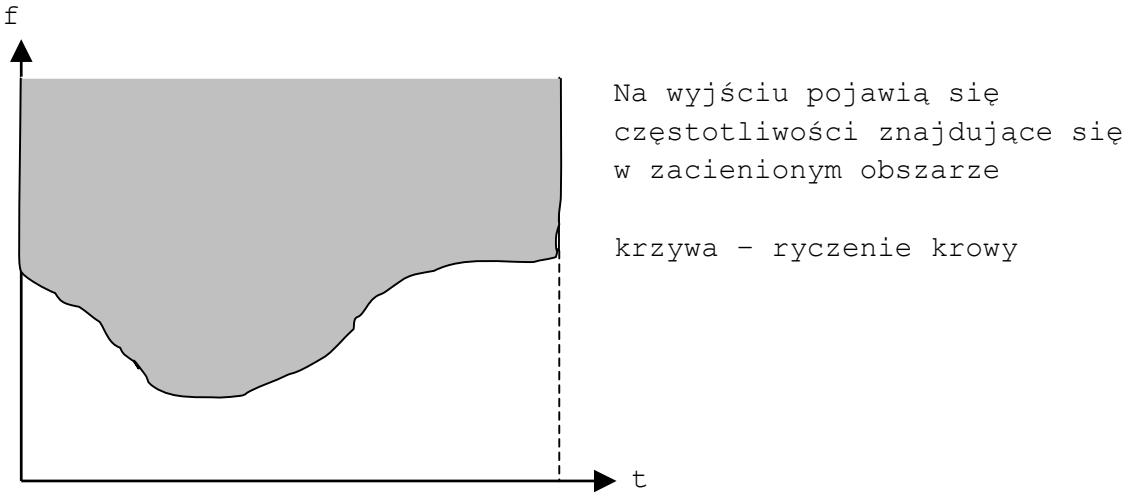
Filtry te wykorzystywane są głównie do wytwarzania efektów specjalnych. Proszę spróbować wykonać kolejno następujące instrukcje:

```
SOUND 0, 0, 0, 0
POKE 53768, 4
POKE 53761,168
POKE 53765,168
POKE 53760, 254
POKE 53764,127
```

- 9-bitowy licznik poly

Bit 7 rejestru AUDCTL zamienia 17-bitowy licznik poly w licznik 9-bitowy. Przypomnijmy, że im krótszy jest rejestr przesuwny licznika poly, tym częściej powtarza się wytwarzany przez niego ciąg impulsów pseudoprzypadkowych, czyli tym mniej impulsy stają się przypadkowe. Stąd zmiana licznika 17-bitowego w licznik 9-bitowy spowoduje częstsze powtarzanie sekwencji impulsów, a zatem mniejszą ich przypadkowość. Spróbujmy wykonać następującą demonstrację działania liczników poly 17- i 9-bitowego, wsłuchując się uważnie w zmiany, jakie spowoduje wykonanie instrukcji POKE:

```
SOUND 0,80,8,8 Wykorzystanie 17-bitowego licznika
POKE 53768,128 Przekształcenie go w licznik 9-bitowy
```



Rys. 7-10. Efekt użycia filtru górnoprzepustowego, sterowanego przez kanał 3, w kanale dźwiękowym 1

PROGRAMOWE TECHNIKI GENERACJI DŹWIĘKU

Istnieją dwie podstawowe metody generowania dźwięku w komputerach domowych systemu Atari: metoda statyczna oraz dynamiczna. Statyczna generacja dźwięku jest metodą prostszą: program ustala parametry generowanych dźwięków, powraca na jakiś czas do wykonywania innych operacji, po czym przerywa bądź zmienia parametry generacji dźwięku. Dynamiczna generacja dźwięku jest znacznie trudniejsza: komputer przez cały czas steruje generacją dźwięku podczas wykonywania programu. Oto przykład:

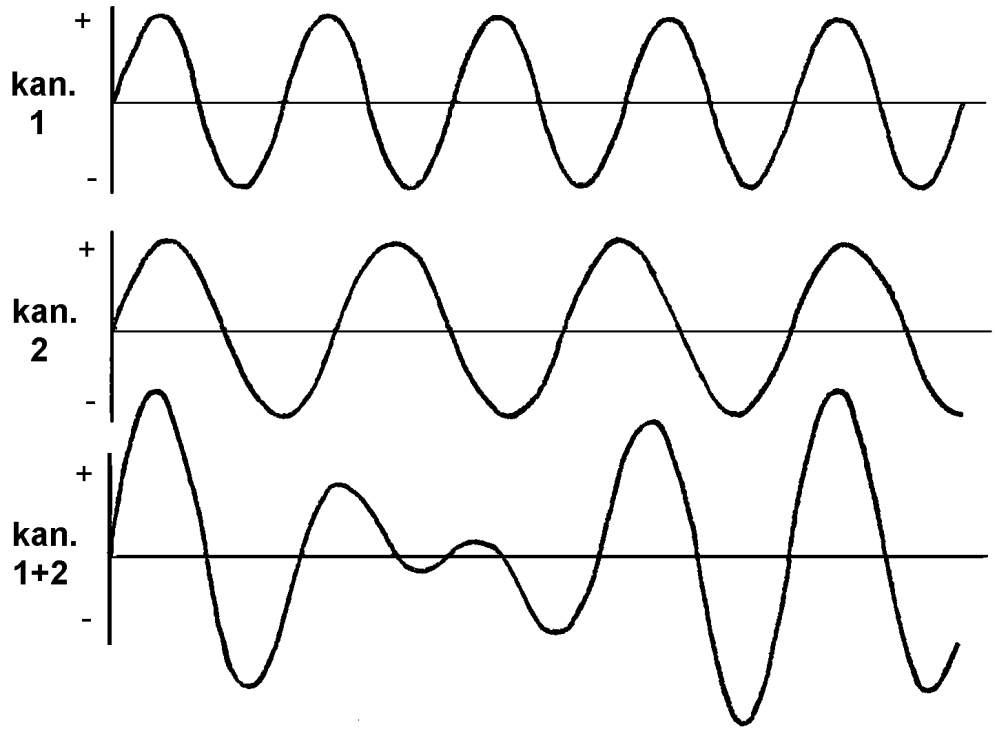
```
Dźwięk statyczny
SOUND 0, 120, 8, 8
```

```
Dźwięk dynamiczny
FOR X=0 TO 255
SOUND 0, X, 8, 8
NEXT X
```

- Dźwięk statyczny

Dźwięk statyczny normalnie ograniczony jest do wytwarzania takich sygnałów dźwiękowych jak: piski, warkot, czy standardowe komputerowo „beepy”. Wyjątkami od tej reguły są dwa przykładowe programy, przedstawione przy omawianiu filtrów górnoprzepustowych oraz 16-bitowego sterowania częstotliwości. Inną metodą uzyskania ciekawych efektów techniką statycznej generacji dźwięków jest wykorzystanie interferencji, jak w poniższym przykładzie:

```
SOUND 0,255,10,8
SOUND 1,254,10,8
```



Rys. 7-11.Dwie fale o różnych częstotliwościach oraz ich suma

Ten dziwny efekt jest rezultatem interferencji dwóch fal o niewiele różniących się częstotliwościach. Dodawanie się fal ukazano na rys. 7-11. Fale sinusoidalne, wytwarzane w kanałach 1 i 2, niewiele różnią się częstotliwościami, a w głośniku pojawia się ich suma. Proszę zwrócić uwagę, w jak znaczny sposób różni się fala sumaryczna od czysto sinusoidalnych fal, przez których dodanie powstała.

Rys. 7-11 ukazuje, że w pewnych momentach czasowych obie fale wzmacniają się nawzajem, podczas gdy w innych interferują, niemal całkowicie się wygaszając. Dodanie do siebie dwóch fal w momencie, kiedy ich maksima się pokrywają, da w rezultacie falę o podwójnej

amplitudzie, podczas gdy dodanie takich samych fal w chwili, kiedy jedna znajduje się w maksimum a druga w minimum, spowoduje ich całkowite wygaszenie. Efekt ten widoczny jest na przebiegu sumy dwóch fal z rys. 7-11. Na obu krańcach wykresu, gdzie maksima obu fal składowych koincydują ze sobą, fala sumaryczna osiąga maksymalną amplitudę. W środku wykresu, obie fale wygaszają się wzajemnie, a fala sumaryczna ma niewielką, amplitudę. Interesujące w niektórych przypadkach może być zsumowanie 3 lub 4 fal o niewiele różniących się częstotliwościach, co doprowadzi w efekcie do powstania unikalnych efektów dźwiękowych.

Im mniejsze są różnice pomiędzy częstotliwościami dodawanych fal, tym dłuższy będzie czas powtarzania tej samej sekwencji amplitudy fali sumarycznej. Łatwo można się o tym przekonać, rysując wykresy podobne do rysunku 7-11, gdzie częstotliwości dodawanych fal ulegałyby zmianom. Jako przykład proszę spróbować następującej sekwencji instrukcji:

```
SOUND 0, 255, 10, 8
SOUND 1, 254, 10, 8
SOUND 1, 253, 10, 8
SOUND 1, 252, 10, 8
```

W miarę wzrostu różnicy częstotliwości fal obu kanałów dźwiękowych skraca się czas powtarzania sekwencji amplitudy fali sumarycznej.

- Dźwięk dynamiczny

Uzyskanie bardziej kompleksowych efektów dźwiękowych wymaga wykorzystania techniki generacji dźwięku dynamicznego. W technice tej można wyróżnić trzy metody generacji dźwięku: programowanie w BASICu, wykorzystanie przerwań wygaszenia pionowego oraz programowanie w kodzie maszynowym.

- Generacja dźwięków w BASICu

BASIC posiada pewne ograniczenia w przypadku wykorzystania do generacji dźwięku dynamicznego. Jak można się było przekonać, wprowadzenie nowej instrukcji SOUND kasuje wszelkie wprowadzone uprzednio zapisy w rejestrze AUDCTL. Problem ten może być ominięty poprzez bezpośrednie zapisywanie rejestrów dźwiękowych instrukcją POKE i nie korzystanie z instrukcji SOUND.

Dodatkowo BASIC ograniczony jest niewielką szybkością działania. Jeżeli program nie jest w całości poświęcony generacji dźwięków, czasu procesora wystarczy zaledwie na wytworzenie dźwięków statycznych lub bardzo ograniczoną generację dźwięku dynamicznego. Jedynym wyjściem z tej sytuacji jest okresowe wstrzymywanie innych operacji mikroprocesora i w tym okresie generowanie dźwięku.

Kolejny problem wyłania się wówczas, gdy chcemy generować dźwięk jednocześnie przez więcej niż jeden kanał. Przy wykorzystaniu czterech kanałów dźwiękowych czas pomiędzy wykonaniem instrukcji dla pierwszego kanału a wykonaniem instrukcji dla kanału czwartego jest

wyraźnie wyczuwalny. Oto przykład rozwiązania przedstawionego problemu:

```
10 SOUND 0, 0, 0, 0: DIM SIMUL$(16)
20 RESTORE 9999: X=1
25 READ Q: IF Q<>-1 THEN SIMUL$(X)=CHR$(Q): X=X+1: GOTO 25
30 READ F1, C1, F2, C2, F3, C3, F4, C4
40 IF F1=-1 THEN END
50 X=USR~(ADR(SIMUL$), F1, C1, F2, C2, F3, C3, F4, C4
60 GOTO 30
100 DATA 182, 168, 0, 0, 0, 0, 0, 0
110 DATA 162, 168, 182, 166, 0, 0, 0, 0
120 DATA 144, 168, 162, 166, 35, 166, 0, 0
130 DATA 128, 168, 144, 166, 40, 166, 35, 166
140 DATA 121 ,168, 128, 166, 45, 166, 40, 166
150 DATA 108, 168, 121, 166, 47, 166, 45, 166
160 DATA 96, 168, 96, 166, 60, 166, 47, 166
170 DATA 91, 168, 108, 166, 53, 166, 53, 166
999 DATA -1, 0, 0, 0, 0, 0, 0, 0
9000 REM
9010 REM
9020 REM poniższe dane stanowią program maszynowy
9030 REM czytany pod adresem SIMUL$
9999 DATA 104, 133, 203, 162, 0, 104, 104, 157, 0, 210, 232, 228,
          203, 208, 246, 96, -1
```

W programie tym SIMUL$ jest krótką procedurą maszynową, zapisującą rejestry dźwiękowe wszystkich czterech kanałów bardzo szybko. A więc program BASICu, wykorzystujący procedurę SIMUL$ może generować dźwięk, operując wszystkimi czterema kanałami równocześnie. Procedurę tę może wykonywać jakikolwiek program, który dane dla rejestrów dźwiękowych wpisze instrukcją USR, jak w linii 50 programu demonstracyjnego. Parametry powinny być wpisane w takiej kolejności, jak w podanym programie, to znaczy wartość rejestru sterującego za wartością rejestru częstotliwości. Parametry powinny zawierać od jednej do czterech par, w zależności od tego, ile kanałów dźwiękowych będzie wykorzystywał dany program. W równym stopniu procedura SIMUL$ może zapisywać rejestry dźwiękowe dla 3, 2 lub 1 kanału dźwiękowego, wystarczy jedynie pominąć niepotrzebne parametry w części adresowej instrukcji USR.

Procedura ta omija jednocześnie jeszcze jedną wadę BASICu. Jak wspomniano poprzednio, każda instrukcja SOUND przepisuje od nowa zawartość rejestru AUDCTL. Przy wykorzystaniu procedury SIMUL$ nie wykonywana jest żadna instrukcja SOUND oprócz inicjalizacyjnej, tak więc zaprogramowany rejestr AUDCTL pozostanie niezmieniony, bez względu na to, jakie parametry SIMUL$ będzie wpisywała do rejestrów częstotliwości i rejestrów sterujących.

BASIC może wykorzystywać jeszcze jedną technikę generacji dźwięku w Atari 400/800, choć jest to niepraktyczne. Chodzi tu o technikę sterowania jedynie głośności jednego z czterech kanałów dźwiękowych poprzez włączenie bitu 4 odpowiedniego rejestru AUDC1-4,

Spróbujmy wykonać następujące instrukcje:

```
SOUND 0, 0, 0, 0
10 POKE 53761,16: POKE 53761, 31: GOTO 10
```

Program ten włącza bit kontroli głośności kanału 1, po czym moduluje głośność tego kanału tak szybko, jak tylko pozwala na to BASIC. Pomimo wykorzystania całego czasu, jaki dla operacji mikroprocesora udostępnia BASIC, efektem tegoż programu jest jedynie basowy pomruk.

- Wykorzystanie przerwań wygaszenia pionowego

Technika ta jest prawdopodobnie najbardziej elastyczna i praktyczna spośród wszystkich metod generowania dźwięku jakie dostępne są w systemie Atari 400/800.

Precyzyjnie co 1/60 sekundy[2], układy sprzętowe komputera wykonują procedury przerwania wygaszenia pionowego. W tym czasie mikroprocesor wstrzymuje na krótko wykonywanie głównego programu, przechodzi do krótkiej procedury obsługi przerwania, a po jej wykonaniu 6502 przechodzi do kilku instrukcji maszynowych, zwanych wyjściem z procedury przerwania, które z kolei powodują powrót do wykonywania głównego programu. Oczywiście, obie procedury obsługi przerwania wygaszenia pionowego w żaden sposób nie zakłócają wykonywania przez 6502 programu głównego.

Procedura obsługi przerwania VBLANK, znajdująca się w ROM systemu Atari 400/800, ma za zadanie zmianę wartości liczników systemowych, przekazanie informacji kontrolnych, oraz wykonanie innych drobnych operacji, które wymagają "ciągłego" nadzoru mikroprocesora.

Zanim procesor, po wykonaniu procedury obsługi przerwania, powróci do wykonywania głównego programu, może wykonać inną procedurę przerwania, zaprojektowaną przez użytkownika - na przykład procedurę zapisania rejestrów dźwiękowych. Technika ta wydaje się być idealna do celów syntezy dźwięku, jako że wykonywanie procedury VBLANK odbywa się co pewien precyzyjnie odmierzony interwał czasowy. Ponadto, mikroprocesor może wykonywać inny program i nie jest zmuszony do poświęcenia całego czasu syntezie dźwięku. Dodatkową zaletą tej techniki jest jej uniwersalność. Procedura maszynowa generacji dźwięku wykonywana w okresach VBLANK może współpracować z każdym innym programem - niezależnie od języka i trybu pracy tegoż programu. Co najwyżej procedura taka będzie wymagała kilku nieznacznych przeróbek, by mogła współpracować z programem napisanym w innym języku - FORTHie, BASICu, asemblerze itp.

Najbardziej nadaje się do tego celu procedura, wykonywana w oparciu o stablicowane dane - to znaczy taka, której wykonanie polega na odczytaniu kolejnej danej z tablicy danych i zapisaniu jej w odpowiednim rejestrze, W przypadku procedury generującej dźwięk, danymi stablicowanymi byłyby parametry częstotliwości generowanych dźwięków i ewentualnie parametry rejestrów sterujących, w tym także natężenia dźwięku. Przy wykorzystaniu tej techniki mnożna osiągnąć

[2] Tak jest w komputerach działających w systemie kodowania koloru NTSC. W przypadku systemu PAL jest to 1/50 sek.

częstość zmiany wysokości dźwięku równą 60 Hz, co jest raczej wystarczającą szybkością dla większości zastosowań.

Po zaprojektowaniu tego typu procedury i umieszczeniu jej w pamięci RAM (powiedzmy, na stronie 6), należy dołączyć ją jako integralną część procedury obsługi przerwania wygaszenia pionowego. Czyni się to opisaną już uprzednio metodą przepisania wektorów.

Adresy pamięci $224, $225 są wektorem (czyli zawierają adres) wspomnianej procedury wyjścia z przerwania - XITVBL (eXiT Vertical Blank interrupt service routine); procedura ta wykonywana jest po zakończeniu wszelkich procedur przerwania i powoduje powrót do wykonywania programu głównego.

Uruchomienie własnej procedury generacji dźwięku podczas przerwań wygaszenia pionowego wymaga następujących kroków:

1. Należy umieścić własną procedurę w pamięci.
2. Jako ostatnią instrukcję tej procedury należy wpisać JMP $E462 (jest to adres XITVBL, który spowoduje powrót do głównego programu po wykonaniu obsługi przerwania).
3. Wpisać do rejestru X starszy bajt adresu procedury (w naszym przykładzie 6).
4. Wpisać do rejestru Y młodszy bajt adresu procedury (w naszym przykładzie 0).
5. Wpisać do akumulatora liczbę 7 (oznaczającą procedurę VBLANK).
6. Wykonać JSR $E45C (adres procedury SETVBV).

Kroki od 3 do 6 mają na celu bezbłędne przepisanie wartości wektora pod adresami $224, $225. Wywołuje się w tym celu systemową procedurę przepisującą wartość wektora obsługi przerwania - SETVBV (Set Vertical B1ank Vectors), która zapisuje dane umieszczone w rejestrach X, Y i A 6502 pod adresem $224, $225. Po wykonaniu powyższych operacji obsługa przerwania wygaszenia pionowego będzie wyglądała następująco:

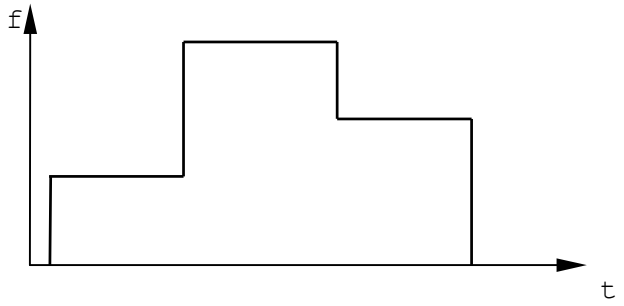
1. Komputer wykona systemową procedurę obsługi przerwania VBLANK.
2. Poprzez wektor $224, $225 przejdzie do wykonywania opóźnionej procedury VBLANK, która teraz będzie naszą procedurą generacji dźwięku. `
3. Wykona zaprojektowaną procedurę.
4. Po jej zakończeniu wykona skok do XITVBL.
5. XITVBL spowoduje powrót komputera do wykonywania głównego programu.

Na rynku dostępny jest program firmowy Atari, zwany INSOMNIA (Interrupt Sound Initializer/Alterer), który jest gotową procedurą syntezy dźwięku, wykorzystującą przerwania wygaszenia pionowego. Użytkownik musi tylko wpisać własne dane dotyczące wysokości poszczególnych dźwięków, ich głośności oraz parametry sterujące. Program ten jest procedurą maszynową i może współpracować z dowolnym programem głównym napisanym w dowolnym języku.

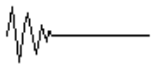
- Generacja dźwięku w programach maszynowych

Bezpośrednie sterowanie rejestrami dźwiękowymi, sprawowane przez główny program w kodzie maszynowym otwiera kolejne drzwi do uzyskania dodatkowych efektów w syntezie dźwięku. Technika ta jest bardzo zbliżona do opisanej poprzednio, a wykorzystującej przerwania VBLANK; tutaj również program maszynowy wykorzystuje stablicowane parametry, lecz tym razem jest on wykonywany jako program główny. Umożliwia to nam poświęcenie całego dostępnego czasu procesora na generację dźwięku, a co za tym idzie uzyskanie dźwięku wyższej jakości.

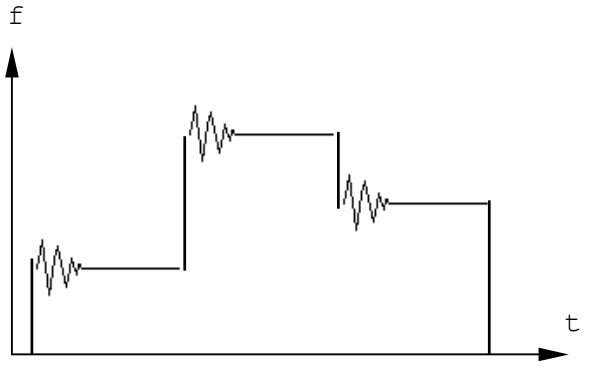
Rozważmy przypadek trzech różnych dźwięków, wytwarzanych przez typową procedurę dźwiękową wykonywaną w czasie przerwań VBLANK. Przedstawiony on został na rys. 7-12.



Rys. 7-12. Przykład trzech nut generowanych przy wykorzystaniu procedury maszynowej wykonywanej w trakcie przerwania VBLANK



Rys. 7-13. Sekwencja zmian częstotliwości w brzmieniu fortepianu



Rys. 7-14. Przykład trzech nut z rys. 7-12 z nałożoną sekwencją zmian częstotliwości symulującą brzmienie fortepianu.

Ponieważ teraz w maszynowym programie głównym możemy na generację dźwięku poświęcić dużo więcej czasu procesora, możliwe jest dokonywanie zmiany częstotliwości impulsów wyjściowych z dużą

szybkością już w trakcie trwania danego dźwięku co w efekcie pozwala na symulowane brzmienia instrumentów klasycznych. Spróbujmy dla przykładu wytworzyć dźwięk naśladujący brzmienie fortepianu, który przy każdym uderzeniu w klawisz wytwarza charakterystyczną sekwencję zmian częstotliwości, jak pokazano na rys. 7-13.

Brzmienie fortepianu można uzyskać poprzez nałożenie szybkich zmian częstotliwości na początkowy, "płaski" przebieg dźwięku z rys. 7-12. A więc w efekcie musimy otrzymać taki przebieg zmian częstotliwości dźwięku, jak pokazany został na rys. 7-14.

Nadal wytwarzany będzie ten sam dźwięk, jak na rys. 7-12, zmieni się jedynie początkowa część naszej wygrywanej nuty, nadając jej charakterystyczne brzmienie fortepianu - czyniąc ją jednocześnie bardziej przyjemną dla ucha niż czysto maszynowy dźwięk o płaskiej charakterystyce częstotliwości. Niestety, technika ta wykorzystuje w całości czas mikroprocesora, nie pozostawiając miejsca na wykonanie innych operacji. Dane dla każdego kanału dźwiękowego nie mogą teraz być zmieniane co 1/60 sekundy - jeżeli operację tę procesor ma wykonywać ciągle, w tym samym czasie musi ponad 100 razy zapisać rejestr częstotliwości tą samą wartością.

- Generacja dźwięków poprzez sterowanie tylko głośności

Eksperymentowaliśmy już z bitem rejestrów AUDC1-4, lecz okazało się, że szybkość, jaką oferuje BASIC, jest za mała, by uzyskać tą techniką jakiekolwiek dźwięki poza basowym pomrukiem. Wykorzystanie w tej technice programów maszynowych omija barierę szybkości.

Wykorzystanie tego bitu stwarza możliwość generacji fal dźwiękowych o dowolnym kształcie widma. Jedynymi ograniczeniami tej metody jest czas mikroprocesora oraz zakres regulacji głośności poszczególnych kanałów dźwiękowych. Zamiast symulować brzmienie fortepianu przez nakładanie mian częstotliwości na płaski przebieg generowanych dźwięków, w technice tej możemy bezpośrednio zaprogramować częstotliwościowe widmo brzmienia fortepianu. Niestety, nie będzie to nigdy precyzyjna replika brzmienia fortepianu - czy jakiegokolwiek innego instrumentu. Tylko 4 bity, a więc 16 wartości, sterujących głośność, to zdecydowanie zbyt mała skala, by uzyskać bardzo dobre rezultaty. Mimo wszystko technika ta pozwala na generowanie dźwięków naprawdę wysokiej jakości. Poniższy program, napisany w języku assemblera, demonstruje wykorzystanie techniki syntezy dźwięku metodą sterowania tylko głośności jednego kanału.

```
           0100
           0110 ; VONLY              Bob Fraser    23-7-81
           0120 ;
           0130 ;
           0140 ; program testowy wykorzystujący bit sterowania
           0150 ; tylko głośności w rejestrach AUDC1-4
           0160 ;
           0170 ;
           0180 ;
D208       0190 AUDCTL     =     $D208
D200       0200 AUDF1      =     $D200
D201       0210 AUDC1      =     $D201
D20F       0220 SKCTL      =     $D20F
```

```
          0230 ;
          0240 ;
0000      0250            *=    $B0
00B0 01         0260 TEMPO      .BYTE 1
00B1 00         0270 MSC        .BYTE 0
                0280 ;
                0290 ;
                0300 ;
00B2            0310        *=    $4000
4000 A900       0320        LDA   #0
4002 8D08D2     0330        STA   AUDCTL
4005 A903       0340        LDA   #3
4007 8D0FD2     0350        STA   SKCTL
400A A200       0360        LDX   #0
                0370 ;
400C A900       0380        LDA   #0
400E 8D0ED4     0390        STA   $D40E      ; zabronienie VBI
4011 8D0ED2     0400        STA   $D20E      ; zabronienie IRQ
4014 8D00D4     0410        STA   $D400      ; zabronienie DMA
                0430 ;
                0430 ;
                0430 ;
4017 BD5240     0450 L00    LDA   DTAB,X
401A 85B1       0460        STA   MSC
                0470
401C BD3640     0480        LDA   VTAB,X
401E A4B0       0490 L0     LDY   TEMPO
4021 8D01D2     0500        STA   AUDC1
4024 88         0510 L1     DEY
4025 D0FD       0520        BNE   L1
                0530 ;
                0540 ;
4027 C6B1       0550        DEC   MSC
4029 D0F4       0560        BNE   L0
                0570 ;
                0580 ;
                0590 ;      kolejna nuta
                0600 ;
402B E8         0610        INX
402C EC3540     0620        CPX   NC
402E D0E6       0630        BNE L00
                0640 ;
                D650 ;
4031 A200       0660        LDX   #0
4033 F0E2       0570        BEQ   L00
                0680 ;
                0690 ;
4035 1C         0700 NC     .BYTE 28   ; licznik nut
                0710 ;
                0720 ; tablica kolejnych wartości natężenia dźwięku
                0730 VTAB
4036 18         0740        .BYTE 24,25,26,27,28,29,30,31
4037 19
4038 1A
4039 1B
403A 1C
```

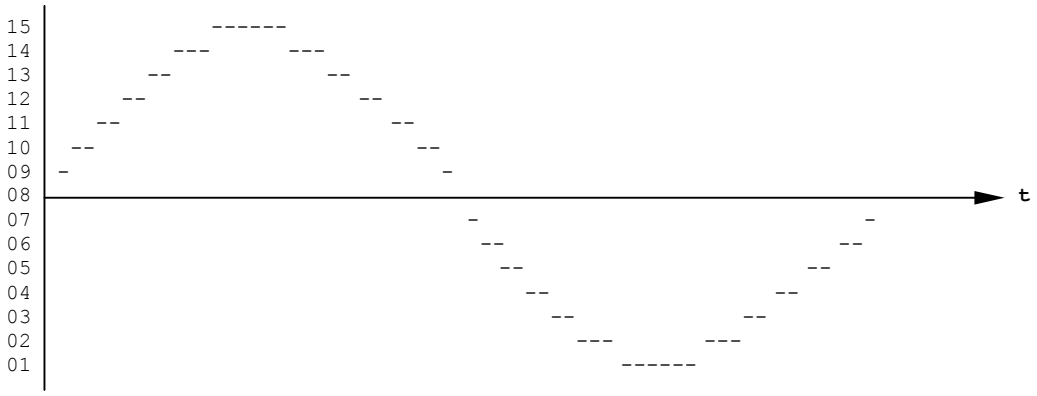
```
403B 1D
403C 1E
403D 1F
403E 1E          0750       .BYTE 30,29,28,27,26,25,24.
403F 1D
4040 1C
4041 1B
4042 1A
4043 19
4044 18
4045 17          0760       .BYTE 23,22,21,20,19,18,17
4046 16
4047 15
4048 14
4049 13
404A 12
404B 11
4040 12          0770       .BYTE 18,19,20,21,22,23
404D 13
404E 14
404F 15
4050 16
4051 17
                 0780 ;
                 0790 ; tabela ta zawiera czas trwania każdej z nut
                 0800 DTAB
4052 01          0810       .BYTE 1,1,1,2,2,2,3,6
4053 01
4054 01
4055 02
4056 02
4057 02
4058 03
4059 06
405A 03          0820       .BYTE 3,2,2,2;1,1,1
405B 02
405C 02
405D 02
405E 01
405F 01
4060 01
4051 01          0830       .BYTE 1,1,2,2,2,3,6
4062 01
4063 02
4064 02
4065 02
4066 03
4067 06
4068 03          0840       .BYTE 3,2,2,2,1,1
4069 02
406A 02
406B 02
406C 01
406D 01
```

Można się przekonać, że parametrem decydującym o jakości dźwięku w tym przypadku wcale nie jest szybkość zmian natężenia dźwięków. Wytworzenie jednego pełnego okresu fali sinusoidalnej wymaga ponad 60 kroków, a mimo to program może wytwarzać dźwięki do częstotliwości około 10 KHz.

Spróbujmy teraz usunąć z programu linie 400-410 i spróbować jeszcze raz. Można zauważyć wyraźne załamywanie się wykonywania programu. Przyczyną jest wykonywanie systemowych przerwań VBLANK - wyraźnie słychać zniekształcenia dźwięku, spowodowane przerywaniem wykonywania programu podczas każdego okresu VBLANK.

Linia 420 programu wstrzymuje wykonywanie DMA przez układ ANTIC. To właśnie jest przyczyną całkowitego wygaszenia ekranu podczas wykonywania powyższego programu. Wstrzymanie DMA ma na celu dwie rzeczy: przyspieszenia pracy mikroprocesora oraz precyzyjne określenie czasu. Wykonywanie DMA wiązałoby się z przerywaniem co jakiś czas pracy procesora, a tym samym z przerywaniem wykonywania programu, analogicznym do obserwowanego podczas przerwań VBLANK.

Powyższy program demonstracyjny wytwarza regularną falę o przebiegu sinusoidalnym. Dokładność sinusoidalnych zmian natężenia dźwięku jest na tyle duża, że wyraźnie słyszy się różnicę w brzmieniu tego programu z tradycyjnie wytwarzanymi dźwiękami o przebiega prostokątnym. Naniesienie zmieniających się kolejno głośności dźwięku w relacji czasu - co w tym przypadku równałoby się przedstawieniu widma generowanej fali - dałoby obraz ukazany na rys. 7-15.



Rys. 7-15. Schemat danych fali sinusoidalnej dla programu syntezy dźwięku metodą kontroli wyłącznie głośności.

W rozdziale tym przedstawiono dyskusję na temat technicznych aspektów syntezy dźwięku w komputerach domowych systemu Atari 400/800. Do pełnego wykorzystania możliwości dźwiękowych tych komputerów jest jeszcze - a może przede wszystkim - potrzebne zrozumienie przez użytkownika znaczenia dźwięku w programach.

Impresjonistyczne efekty dźwiękowe silnie oddziaływają na umysł człowieka, a także na jego podświadomość. Wiąże się to głównie z faktem, że obecność informacji dźwiękowych oddziaływuje na użytkownika stale, bez względu na to, czy i ile uwagi poświęca on efektom dźwiękowym. Zupełnie inaczej oddziaływują bodźce wizualne,

które wymagają od człowieka skupienia uwagi. Jeżeli więc zachodzi potrzeba oderwania się od wizualnych informacji dostarczanych przez komputer za pośrednictwem odbiornika telewizyjnego, obserwowany obraz bardzo szybko ulatuje z pamięci. Generacja dźwięku oferuje programistom bezpośrednią drogę do umysłu i wyobraźni odbiorcy - drogę omijającą procesy myślowe człowieka, lecz za to silnie oddziaływującą na jego stan emocjonalny.

8. SYSTEM OPERACYJNY

WPROWADZENIE

Każdy komputer domowy systemu Atari wyposażony jest w 10K ładunek systemu operacyjnego. Bardzo często zapomina się, jak ważna jest to część komputera. Bez tego systemu dysponowalibyśmy jedynie mniejszym czy większym potencjałem obliczeniowym - lecz niczym poza tym. Sytuacja ta nie dotyczy wyłącznie komputerów Atari, ale wszystkich komputerów. Urządzenie jako takie jest jedynie zestawem sprzętowych układów scalonych. Wykonanie jakichkolwiek operacji przy ich pomocy wymaga wstępnej organizacji całego systemu. Gdyby każdy programista musiał zaczynać od organizowania systemu, a dopiero potem mógł wprowadzić swój program, z pewnością istniałoby o wiele mniej programów niż istnieje ich dzisiaj. Rozwiązaniem, które wprowadzono już przed laty, było stworzenie programu zarządzającego wszystkimi dostępnymi układami sprzętowymi, który pozwalał na proste przejęcie kontroli nad nimi przez oprogramowanie. Program ten znany jest pod różnymi nazwami: System Operacyjny, Sterujący Program Zarządzający, Zarządca Systemu, Monitor Systemu itp. W systemie komputerów domowych Atari występuje on pod nazwą Systemu Operacyjnego, oznaczanego w skrócie OS.

Pierwszym krokiem w zapoznawaniu się z systemem operacyjnym musi być rozpoznanie sprzętowego potencjału, jakim dysponuje komputer Atari, a którym zarządzać musi OS. Potencjał ten składa się z:

- mikroprocesora 6502
- pamięci RAM (o zmiennej pojemności)
- układu scalonego LSI ANTIC
- układu scalonego LSI CTIA (lub GTIA)
- układu scalonego LSI POKEY
- układu scalonego interfejsu peryferyjnego PIA

Przy wykorzystaniu tego potencjału OS może współpracować i jednocześnie sterować wiele zewnętrznych urządzeń, jak odbiornik telewizyjny lub monitor, klawiatura, przełączniki konsoli, dżojstiki, paddle, magnetofon kasetowy, stacja dysków, drukarki, plotery, interfejs RS-232 i od nich pochodne.

System operacyjny Atari można podzielić na siedem podstawowych części, które dalej omówione zostaną szczegółowo:

MONITOR. Monitor OS jest procedurą systemową, wykonywaną w momencie włączenia zasilania komputera lub po naciśnięciu klawisza RESET. Procedura ta pozwala OS na przejęcie kontroli nad systemem. Po jej wykonaniu OS może przekazać kontrolę nad systemem programowi lub bezpośrednio użytkownikowi tylko na wyraźne żądanie użytkownika. Monitor inicjalizuje system zarządzania pamięcią, inicjalizuje podsystem obsługi wejścia/wyjścia (I/O), ustala wektory systemowe i wybiera odpowiedni sposób przekazania kontroli po zakończeniu procedur inicjalizacyjnych.

STRUKTURA OBSŁUGI PRZERWAŃ. Komputer wykorzystuje standardową strukturę obsługi przerwań mikroprocesora 6502, rozszerzoną nieco o

pewne elementy charakterystyczne dla systemu Atari. Przerwania wykonywane są poprzez kilka rodzajów żądań przerwań: wprowadzanie danych z klawiatury, wciśnięcie klawisza BREAK, przekroczenie zakresu liczników systemowych, oraz podczas wygaszenia pionowego w odbiorniku telewizyjnym i w niektórych przypadkach podczas przesyłania danych szyną szeregową.

WEKTORY SYSTEMOWE OS. Wektory systemowe są narzędziem pozwalającym użytkownikowi na odwoływanie się do procedur systemowych bądź na wykorzystywanie OS w przypadkach specjalnych. Najczęściej spotykanym zastosowaniem wektorów jest odwoływanie się do procedur systemowych I/O, wykorzystywanie liczników systemowych oraz przekazywanie kontroli procedurom sprzętowym. Procedury systemowe mogą być wywoływane dwoma sposobami. Wektory ROM są rejestrami Zawierającymi instrukcje JMP do odpowiednich procedur systemowych - zawartość tych rejestrów nie może być zmieniona. Wektory RAM, które mogą zostać przepisane, są rejestrami zawierającymi adresy procedur systemowych. Adresy procedur, zawarte w odpowiadających sobie wektorach RAM i ROM są identyczne.

PODSYSTEM OBSŁUGI WEJSCIA/WYJSCIA. OS stwarza szerokie możliwości programowego sterowania wszystkich urządzeń peryferyjnych komputera. Podsystem obsługi wejścia/wyjścia jest zbiorem procedur, zwanych sterownikami urządzeń peryferyjnych, które mogą nadzorować komunikację komputera z wszelkimi dołączonymi peryferiami.

PROGRAMOWANIE W CZASZE RZECZYWISTYM. Komputery domowe Atari stosunkowo dobrze realizują zadania, związane z "programowaniem w czasie rzeczywistym". OS wykorzystuje dwa typy liczników: liczniki sprzętowe i systemowe liczniki programowe. Liczniki sprzętowe są licznikami odliczającymi do zera, które mogą być wykorzystane w różnych zliczeniach o skoku od 0,5 mikrosekundy do kilku sekund. Systemowe liczniki programowe pracują z częstotliwością 60 Hz i mogą być wykorzystywane jako zegary seryjnego przesyłania danych lub do generowania efektów dźwiękowych.

STANDARDOWY ZBIOR SYMBOLI. ROM komputera wyposażony jest w tzw. "programowy" standardowy zbiór symboli, tzn. taki, który może być zmieniony przez użytkownika. Standardowy zbiór symboli, znajdujący się w pamięci ROM, jest wykorzystywany do wyświetlania standardowych znaków tekstowych i graficznych; jest on inicjalizowany podczas załączania komputera.

ŁADUNEK OPERACJI ZMIENNOPRZECINKOWYCH. Ładunek operacji zmiennoprzecinkowych jest zbiorem procedur matematycznych, rozszerzających możliwości wykonywania obliczeń arytmetycznych przez komputer. Procedury te wykorzystują binarny kod zapisu liczb dziesiętnych (BCD) przy obliczaniu działań podstawowych +, -, *, /, wykonują ponadto obliczenia eksponencjalne i logarytmiczne oraz przeprowadzają konwersję kodów z ATASCII na BCD i odwrotnie.

MONITOR

Monitor OS jest tą częścią systemu operacyjnego, która zarządza systemem zarówno podczas załączania komputera jak też podczas naciśnięcia klawisza RESET. Obie te procedury przejmują kontrolę nad wszystkimi układami komputera i zapewniają odpowiednią inicjalizację poszczególnych jego elementów przed przekazaniem częściowej kontroli

programowi aplikacyjnemu użytkownika. Obie procedury realizują podobne zadania i w dużej części ich przebieg jest identyczny.

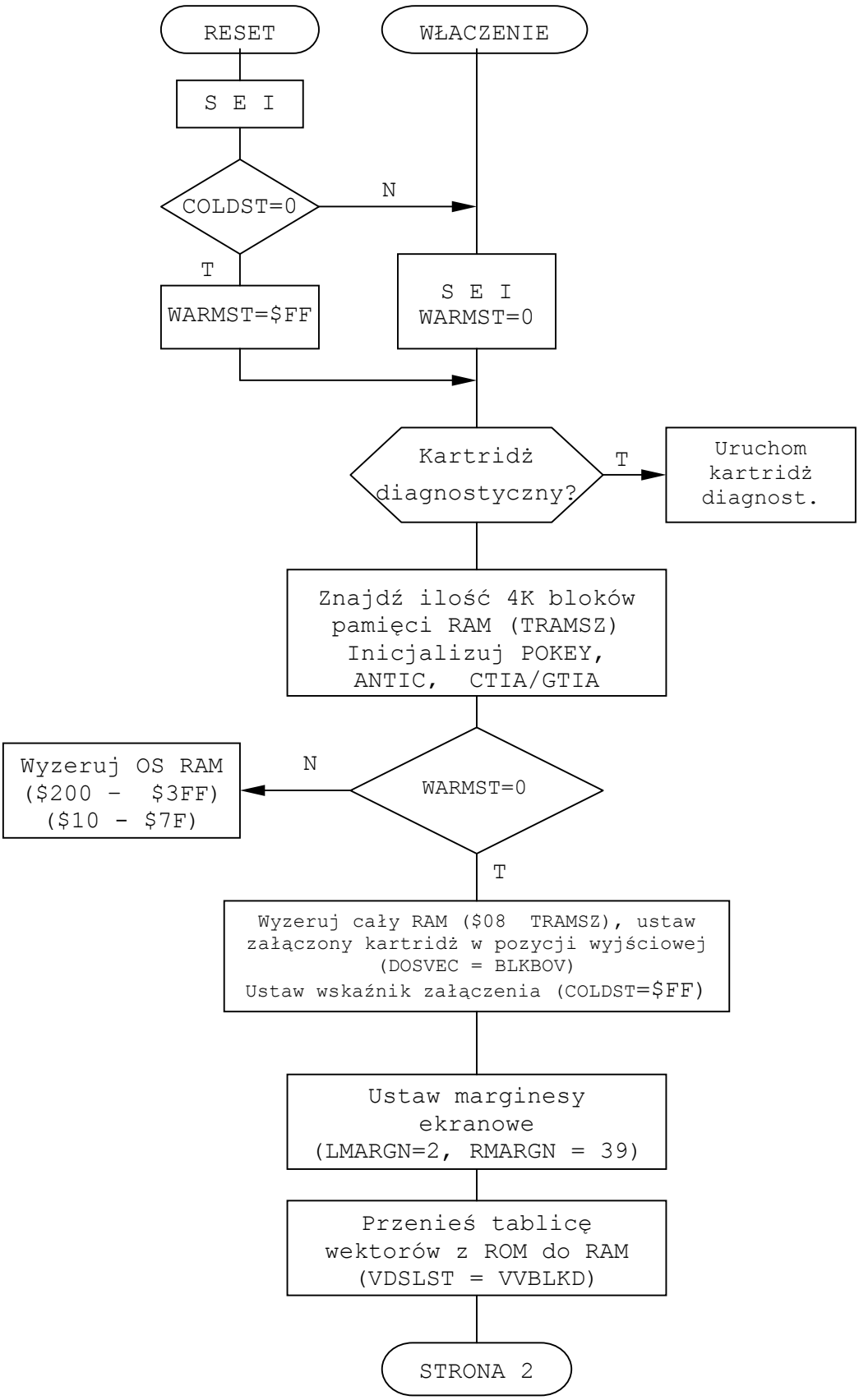
Procedura załączania komputera (start zimny) wywoływana jest przez włączenie zasilania komputera bądź przez przekazanie kontroli instrukcją JMP przez wektor COIDSV ($E477). Najważniejszymi operacjami wykonywanymi przez tę procedurę są:

1. Wyzerowanie wszystkich rejestrów pamięci RAM oprócz adresów $0000 - $000F.
2. Wykonanie bootingu bądź kasety bądź dysku. BOOT ($0009) jest rejestrem flagowym, wskazującym powodzenie bądź niepowodzenie wykonanej operacji bootingu. Włączenie bitu 0 oznacza powodzenie bootingu kasety, włączenie bitu l jest równoznaczne powodzeniu bootingu dysku.
3. COLDST ($0244) jest rejestrem flagowym wskazującym jakiego typu inicjalizacja ma być wykonana. Zapisany wartością 0 oznacza, że naciśnięty został klawisz RESET. Zapisany inną wartością wskazuje, że inicjalizacja ma charakter włączenia komputera. Rejestr ten może być wykorzystany przy zabezpieczaniu programów. Jeżeli COIDST zostanie zapisany wartością niezerową w trakcie wykonywania programu, wówczas wciśnięcie klawisza RESET będzie równoznaczne z inicjalizacją typu załączenia zasilania. Chwyt ten może zabezpieczyć program przed przejęciem przez użytkownika kontroli nad programem w trakcie jego wykonywania.

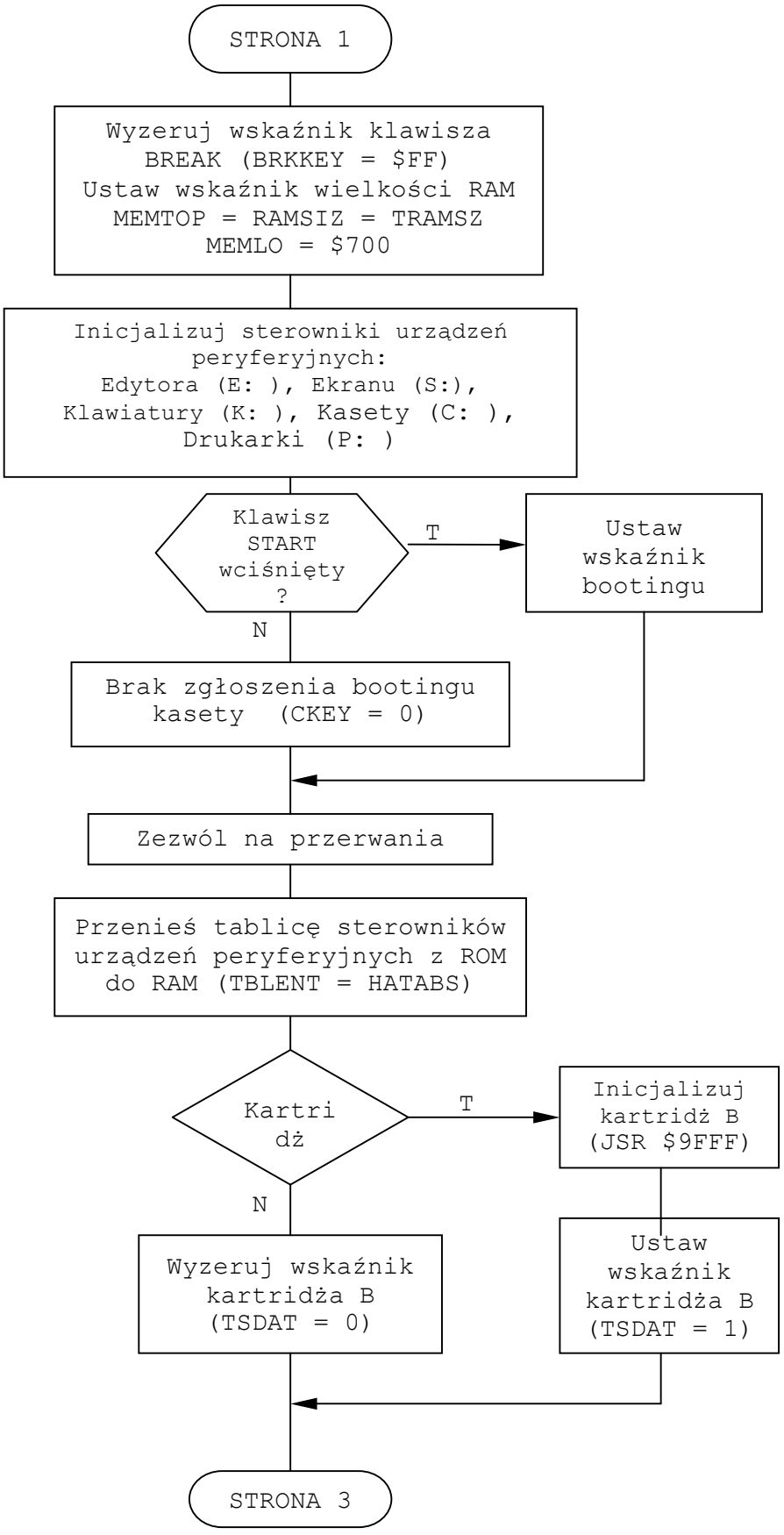
Naciśnięcie klawisza RESET uruchamia procedurę resetowania komputera, zwaną często startem gorącym. warto pamiętać, że w trakcie jej wykonywania przeprowadzane są następujące operacje:

1. Wszystkie wektory systemowe RAM przepisywane są z ROM zarówno podczas startu zimnego jak i gorącego. Jeżeli zachodzi potrzeba ingerencji w proces przenoszenia wektorów, należy zabezpieczyć się przed wykonaniem niektórych podprocedur startu gorącego. Pewne sugestie na ten temat przedstawione zostały dalej, w podrozdziale o gospodarowaniu pamięcią.
2. Wartości MEMLO, MEMTOP, APPMHII; RAMSIZ oraz RAMTOP są uaktualniane podczas startu gorącego. Jeżeli zachodzi potrzeba zmiany tych wskaźników RAM, np. w celu zarezerwowania miejsca w pamięci dla procedur maszynowych wywoływanych z BASICU, należy również zabezpieczyć się przed wykonaniem pełnej procedury resetowania systemu. Przykład takiego programu przedstawiony został na rys. 8-3.

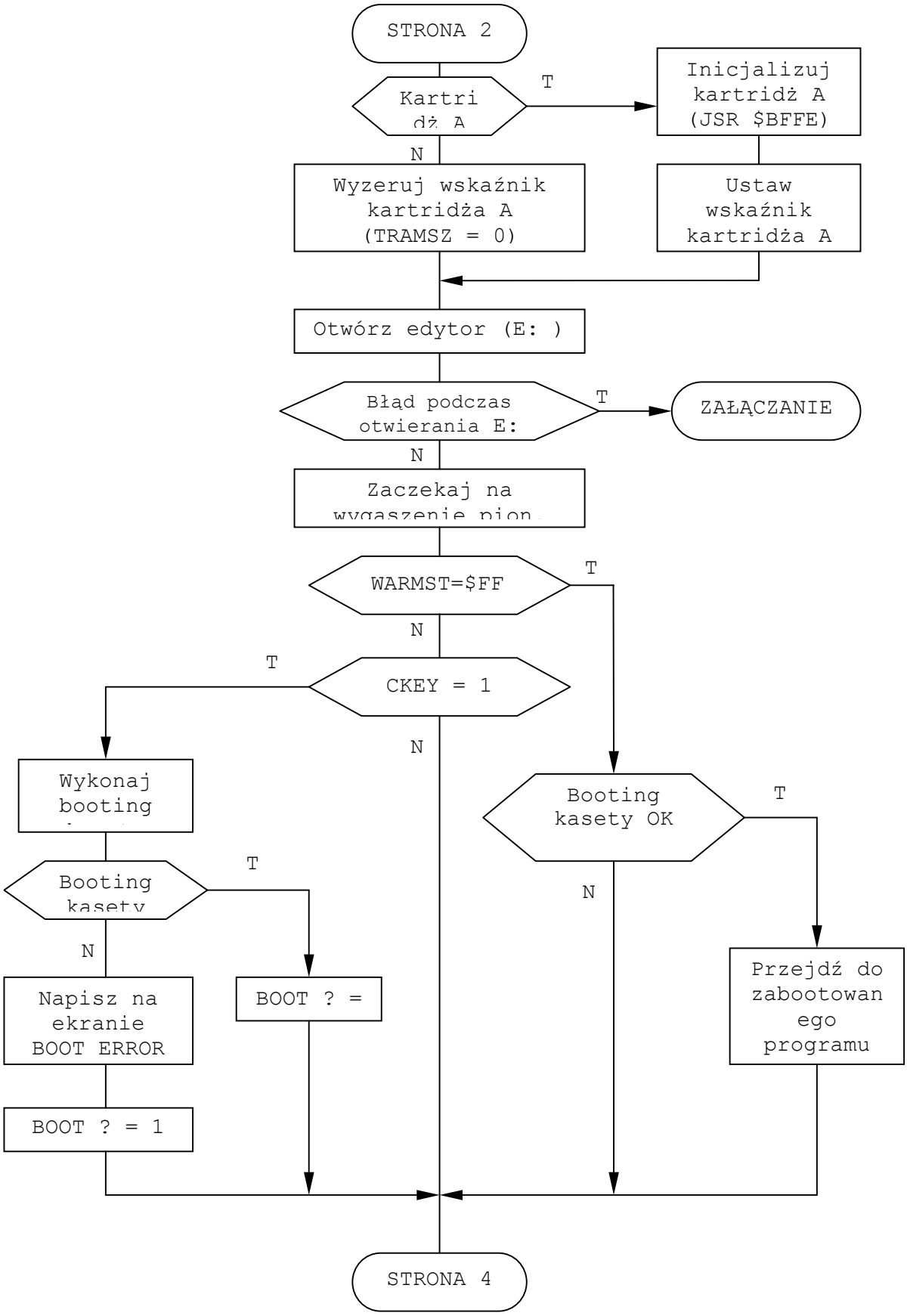
Rysunki, przedstawione na kilku następnych stronach, stanowią szczegółowe schematy przebiegu obu procedur startowych.



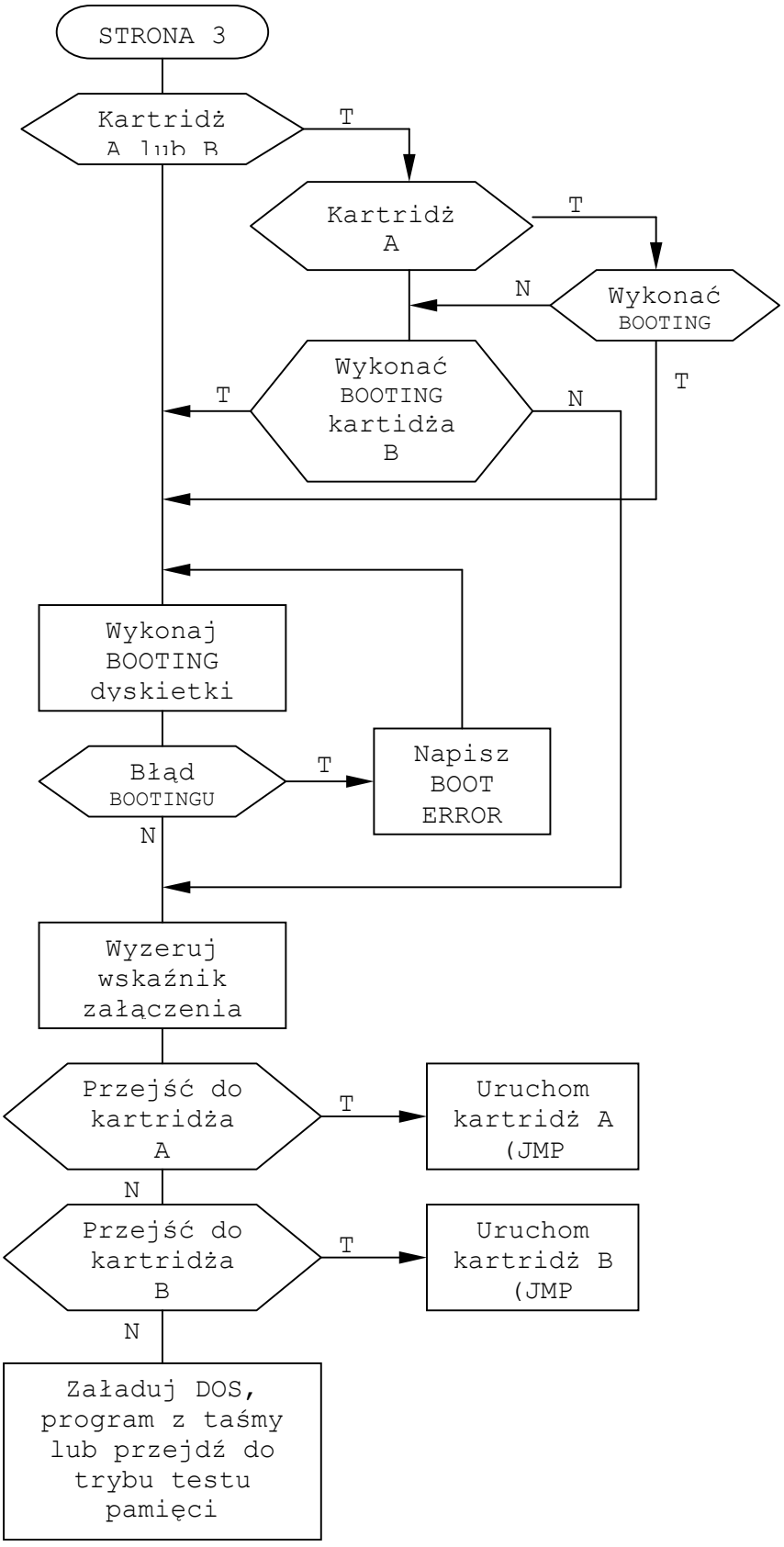
Rys. 8-1.1. Inicjalizacja systemu



Rys. 8-1.2. Inicjalizacja systemu



Rys. 8-1.3. Inicjalizacja systemu



Rys. 8-1.4. Inicjalizacja systemu.

GOSPODAROWANIE PAMIĘCIĄ

Fakt, iż OS przeznaczony jest dla mikroprocesora 6502, zadecydował o kilku ważkich decyzjach dotyczących gospodarowania pamięcią komputera. W przestrzeni adresowej pamięci obsługiwanej przez 6502 wyróżnić można trzy specjalne obszary. Znaczenie strony zerowej jest powszechnie znane. Operowanie na danych zawartych na tej stronie przyspiesza i ułatwia wykonywanie programu przez 6502. Niektóre instrukcje mikroprocesora przeznaczone są wyłącznie do pracy ze stroną zerową pamięci. Strona pierwsza również posiada szczególne znaczenie - przeznaczona ona została na stos mikroprocesora. Trzecim szczególnym obszarem są rejestry $FFFA - $FFFF, zarezerwowane dla sprzętowych wektorów procedury resetowania oraz obsługi przerwań.

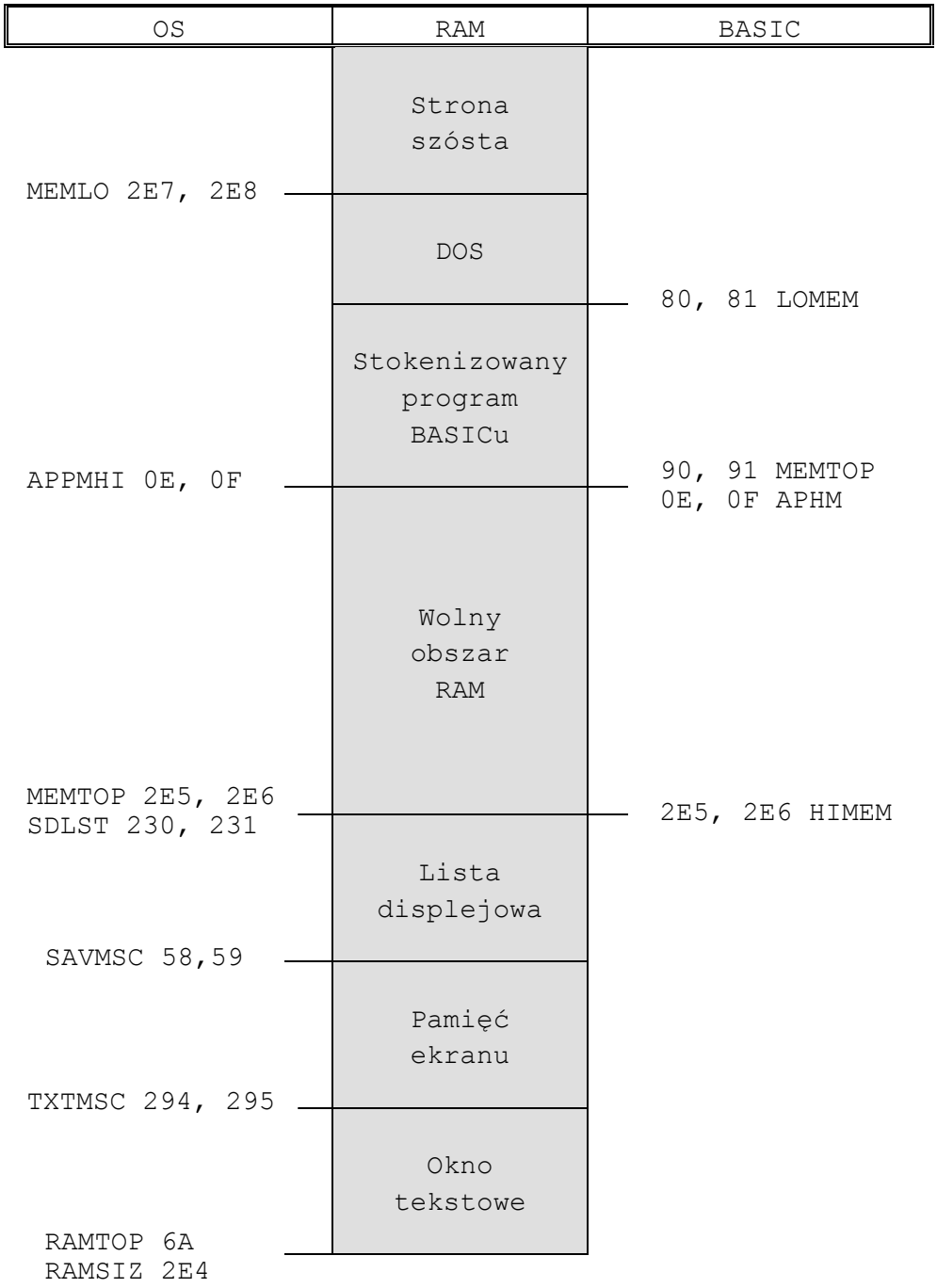
System operacyjny rezyduje w najwyższej części przestrzeni adresowej komputera, w rejestrach od $D800 do $FFFF. Tuż poniżej tego obszaru znajduje się rejon zarezerwowany dla sprzętowych rejestrów układów ANTIC, GTIA oraz POKEY. Obejmuje on adresy od $D000 do $DFFF.

Na drugim końcu przestrzeni adresowej pamięci OS rezerwuje na swój użytek połowę strony zerowej. Ponadto, do użytku systemu operacyjnego przeznaczone są strony druga, trzecia, czwarta i piąta pamięci. Z punktu widzenia programisty, użytkowa przestrzeń adresowa komputera rozciąga się od adresu $0600 do $BFFF:

Jedną z pierwszych operacji, wykonywanych przez OS podczas procedury załączenia zasilania komputera, jest określenie wielkości dostępnego obszaru pamięci RAM. Dokonywane jest to poprzez sprawdzenie zawartości pierwszej komórki każdego z 4K bloków pamięci, rozpoczynając od adresu $1000. Zawartość każdej komórki jest odczytywana, sprawdzana, po czym zapisywana ponownie. Jeżeli wynikiem tej operacji jest zero, wartość tymczasowego wskaźnika wielkości pamięci zostaje zwiększona o 1. Proces ten trwa dopóty, dopóki OS nie natrafi na komórkę, której zawartość nie może być przepisana. Dwa rejestry wskaźnikowe, RAMTOP oraz RAMSIZ, zostają zapisane ilością dostępnych stron pamięci. W uzupełnieniu, procedury gospodarowania pamięcią OS zapisują pozostałe rejestry wskaźnikowe - MEMLO, MEMTOP oraz APPMHI. Zależności pomiędzy wskazaniami poszczególnych rejestrów, ukazane zostały na rys. 8-2, będącym schematyczną mapką pamięci komputera.

MEMLO jest rejestrem dwubajtowym, w którym zapisany zostaje adres, od którego może rozpocząć się zapisywanie programu aplikacyjnego. Wskaźnik ten może być zmodyfikowany, np. w celu zarezerwowania przestrzeni dla procedur maszynowych wywoływanych z BASICu. BASIC wykorzystuje wskaźnik MEMLO do określenia adresu startowego programów (patrz dyskusja o strukturze programów BASICu w rozdziale 10). Jeżeli w MEMLO zapisany zostanie adres wyższy, BASIC od niego rozpocznie zapisywanie programów. Czynność ta musi być jednakże przeprowadzona zanim stera nad systemem przekazana zostanie do kartridża BASICu. Przepisanie zawartości MEMLO jest także dość ryzykowne, jako że w rejestrze tym adres jest aktualizowany każdorazowo po wywołaniu procedur startowych (RESET lub załączenie).

Jeżeli program aplikacyjny wczytywany jest z dysku, rezerwacja przestrzeni adresowej poprzez przepisanie MEMLO może być dokonana z programu typu AUTORUN.SYS. Jednakże DOS jest również inicjalizowany



Rys. 8-2. Rejestry wskaźnikowe OS i BASICu.

podczas procedury resetowania - poprzez Wektor DOSINI ($000C). Wektor ten zawiera adres procedury inicjalizacyjnej DOSu, wywoływanej przez monitor inicjalizujący system. DOSINI jest ponadto jedynym rejestrem, umożliwiającym ingerencję w procedury resetowania. Ponieważ inicjalizacja DOSu musi mieć miejsce niezależnie od zawartości MEMLO, normalne procedury inicjalizacyjne muszą zostać wykonane przed zmianą wektora DOSINI. Można to uczynić

poprzez przeniesienie zawartości DOSINI do innego dwubajtowego adresu, do którego system będzie kierowany instrukcją JSR z programu AUTORUN.SYS. Zaraz po wykonaniu instrukcji JSR musi znaleźć się program zapisujący MEMLO żądaną wartością. Dopiero tu powinna znaleźć się instrukcja RTS. Tak więc DOSINI musi być zapisany adresem podprogramu, wywoływanego przez JSR. Podczas resetowania systemu wywoływany jest ten podprogram, którego instrukcja - JSR zawartość DOSINI - inicjalizuje DOS, po czym wykonywana jest dalsza jej część, zmieniająca zawartość MEMLO, i wreszcie wykonywana jest reszta procedur inicjalizacyjnych. Rys. 8-3 przedstawia przykład programu, wykonującego opisane operacje.

Przedstawiona technika może być także stosowana w odniesieniu do MEMTOP - wskaźnika końca dostępnej pamięci RAM. Rejestr ten zawiera najwyższy adres, dostępny dla programów aplikacyjnych. Adres ten nie jest równoznaczny najwyższemu adresowi pamięci RAM, jako że OS w najwyższej części pamięci RAM umieszcza listę displejową oraz dane pamięci ekranowej. Tutaj podobnie można zarezerwować przestrzeń adresową dla procedur maszynowych lub danych, poprzez zmniejszenie zawartości MEMTOP zapisanej procedurami startowymi. Wykorzystanie do tego celu MEMTOP jest jednakże nieco bardziej skomplikowane.

```
                0010 ; ZMIANA WARTOSCI WSKAŹNIKA MEMLO
                0020 ;
0600            0030 START      = 5600
000C            0040 DOSINI     = $0C
02E7            0050 MEMLO      = $2E7
3000            0060 NEWMEM     = $3000 ;NOWA WARTOSC MEMLO
                0065 ;
                0070 ;PROGRAM TEN REZERWUJE MIEJSCE DLA PROCE-
                0090 ;DUR MASZYNOWYCH PRZEZ PRZEPISANIE MEMLO.
                0100 ;TO PROGRAM TYPU AUTORUN.SYS. PO RESECIE
                0120 ;MEMLO ZAPISYWANE JEST WARTOSCIA NEWMEM.
                0130 ;TA CZĘŚĆ MUSI ZAWSZE REZYDOWAĆ W ZBIORZE.
                0140 ;WEKTOR DOSINI ZOSTAL PRZENIESIONY I ZNAJ-
                0150 ;DUJE SIĘ W CZESCI ADRESOWEJ INSTRUKCJI
                0160 ;JSR TROJAN.
                0170 ;PODCZAS RESETOWANIA DOSINI WSKAZUJE NA
                0180 ;INITDOS. JSR TROJAN INICJALIZUJE PROCE-
                0185 ;DURY DOS, MEMLO MA NOWA WARTOSC A KON-
                0190 ;TROLA PRZEKAZANA JEST DO MONITORA.
0000            0200            *=   START
                0210 INITTDOS
0600 200D06     0220       JSR TROJAN ; INICJALIZUJ DOS
0603 A900       0230       LDA #NEWMEM&255
0605 8DE702     0240       STA MEMLO
0608 A930       0250       LDA #NEWMEM/256
060A 8DE802     0260       STA MEMLO+1
                0270 TROJAN
060D 60         0280       RTS
                0290 ;TA CZESC WYKONUJE PODCZAS ZALACZANIA I
```

```
                0300 ;MOZE BYC SKASOWANA PO ZALACZENIU. PRZEPISUJE
                0330 ; ONA DOSINI DO ADRESU INSTRUKCJI JSR TROJAN.
                0350 ;REJESTRTR DOSINI ZAPISYWANY JEST NOWA
                0370 ; WARTOSCIA, WSKAZUJACA ADRES INITDOS
                0390 GRABDOSI
060E A50C       0400       LDA DOSINI ; PRZEPISANIE DOSINI
0610 8D0106     0410       STA INITDOS+1
0613 A50D       0420       LDA DOSINII+1
0615 8D0206     0430       STA INITDOS+2
0618 A900       0440       LDA #INITDOS&255 ; ZAPISANIE DOSINI
061A 8500       0450       STA DOSINI
061C A906       0460       LDA #INITDOS/256;
061E 850D       0470       STA DOSINI+1
6620 A500       0480       LDA NEWMEM&255 ; ZAPISANIE MEMLO
0622 8DE702     0490       STA MEMLO
0625 A930       0500       LDA NEWMEM/256
0627 8DE802     0510       STA MEMLO+1
062A 60         0520       RTS
0628            0530       *= $2E2
02E2 0E06       0540       .WORD GRABDOSI ; ZAPIS ADRESU STARTU
02E4            0550       .END
```

Rys. 8-3. Program przepisujący wskaźnik MEMLO.

Wartość MELLO zależy zarówno od objętości dostępnej pamięci RAM, jak i od objętości pamięci ekranowej, uzależnionej od wybranego trybu graficznego. Fakt ten stwarza trudności w przewidzeniu wartości MEMTOP - jest to jednak możliwe pod warunkiem ciągłego jej sterowania oraz dokładnego ustalenia trybów graficznych, wywoływanych przez program. Niepewność co do końcowych adresów pamięci RAM skłania do umieszczania w tej części pamięci jedynie procedur maszynowych całkowicie relokowalnych.

APPMHI jest rejestrem wskaźnikowym, zawierającym najniższy adres pamięci RAM, do którego może być rozszerzona pamięć ekranowa - jest to pierwszy wolny adres powyżej stokenizowanego programu BASICu. Zapisanie tego rejestru właściwą wartością może zabezpieczyć każdy program przed skasowaniem go w chwili rozszerzania części ekranowej RAM, spowodowanej zmianą trybu graficznego.

RAMSIZ, podobnie jak MEMTOP, może być także wykorzystamy do rezerwowania przestrzeni adresowej pamięci na procedury maszynowe lub dane. Ponieważ RAMSIZ jest rejestrem jednobajtowym, zawierającym ilość dostępnych stron pamięci RAM, obniżenie jego wartości o 1 spowoduje zarezerwowanie 256 bajtów pamięci. Podstawowa różnica pomiędzy posługiwaniem się RAMSIZ oraz MEMTOP polega na tym, że przepisanie zawartości RAMSIZ spowoduje zarezerwowanie miejsca powyżej pamięci ekranowej RAM i tym samym przesunięcie danych ekranowych i listy displejowej do niższych adresów - podczas gdy zmiana zawartości MEMTOP rezerwuje przestrzeń adresową poniżej danych ekranowych i listy displejowej.

STRUKTURA OBSŁUGI PRZERWAŃ

Możliwość selektywnego dostosowywania się do jakichkolwiek specjalnych wezwań (np. wezwań przerwań), wywoływanych czy to sprzętowo czy to programowo, świadczy o wielkiej elastyczności systemu komputerowego. Jak w każdym systemie, opartym na mikroprocesorze 6502, w Atari wyróżnić możemy dwa rodzaje żądań przerwań na poziomie mikroprocesora - przerwania maskowalne (IRQ) oraz niemaskowalne (NMI). Wyższy poziom kontroli przerwań realizowany jest przez pozostałe uk3ady: ANTIC, POKEY oraz PIA. Każdy z tych układów jest odpowiedzialny za wykonanie przerwań wywoływanych przez kilka różnych sytuacji. Jeżeli poszczególne przerwania są dozwolone na poziomie każdego z tych układów, wówczas wysyłają one żądanie wykonania przerwania do 6502. Przerwaniami NMI zarządza ANTIC, natomiast przerwaniami IRQ po części POKEY oraz PIA.

Dostępne są następujące rodzaje przerwań:

Nazwa przerwania	Wektor	Typ	Funkcja	wykorzystywane przez
Listy displejowej	VDSLST	NMI	Organizacja ekranu	Użytk.
Resetowania	-	NMI	Inicjalizacja systemu	OS
Wygaszenia pionowego	VVBLKI VVBLKD	NMI	Tworzenie obrazu	OS Użytk.
Gotowości odczytu szer.	VSERIN	IRQ	Operacje wejścia	OS
Gotowości przesł. szer.	VSEROR	IRQ	Operacje wyjścia	OS
Zakończ. przesł. szer.	VSEROC	IRQ	Operacje wyjścia	OS
Licznika nr 1 POKEYa	VTIMR1	IRQ	Licznik sprzętowy	Użytk.
Licznika nr 2 POKEYa	VTIMR2	IRQ	Licznik sprzętowy	Użytk.
Licznika nr 4 POKEYa *	VTIMR4	IRQ	Licznik sprzętowy	Użytk.
Klawiatury	VKEYBD	IRQ	Odczyt klawiatury	OS
Klawisza BREAK	BRKKEY	IRQ	Odczyt klawisza BREAK	OS
Szyny szeregowej	VINTER	IRQ	Obsługa peryferiów	**
Procedury szyny szereg.	VPRCED	IRQ	Obsługa peryferiów	**

* Wektory tych przerwań występują tylko w systemie wersji B

** Niewykorzystane

Rys. 8-4. Wektory przerwań

Rozdział 6 "Operating System Manual" przedstawia szczegółowy opis wszystkich rodzajów przerwań. Przy wykorzystywaniu wszelkiego typu przerwań należy zachować szczególną ostrożność. Na przykład, pomyłkowe zabronienie przerwań IRQ klawiatury spowoduje, że system będzie ignorował całkowicie klawiaturę, za wyjątkiem klawiszy RESET oraz BREAK. Oczywiście, sytuacja taka może być w niektórych przypadkach pożądana, znacznie jednak skomplikuje tworzenie programu.

- Sterownik przerwań IRQ

System operacyjny wyposażony został w specjalny sterownik przerwań, zarządzający wszystkimi przerwaniami IRQ. Sterownik ten posiada wektory, umieszczone w pamięci RAM, dla wszystkich przerwań IRQ (jedynie w pierwotnej wersji "A" OS wektor przerwania klawisza BREAK nie był przepisywany do pamięci RAM). Wartości tych wektorów zapisywane są podczas inicjalizacji systemu, zarówno podczas startu zimnego jak i gorącego. Adresy odpowiednich wektorów IRQ w RAM opisane zostały w podrozdziale o wektorach systemowych.

Funkcje wektorów IRQ są następujące:

VIMIRQ wektor natychmiastowego przerwania IRQ. Wszystkie przerwania IRQ wykonywane są poprzez ten wektor, który normalnie wskazuje na systemowy sterownik przerwań IRQ. Oczywiście, można go zapisać inną wartością i zaprojektować własną procedurę IRQ.

VSEROR wektor żądania szeregowego przesłania danych przez POKEYa. Normalnie wskazuje adres procedury przesyłającej kolejno bajty z buforu wyjściowego do szeregowego portu wyjściowego.

VSERIN wektor gotowości szeregowego czytania danych przez POKEYa. Wskazuje adres procedury przesyłający kolejno bajty z szeregowego portu wejściowego do buforu danych.

VSEROC wektor zakończenia szeregowego przesyłania danych przez POKEYa. Wskazuje adres procedury zapisującej rejestry wskaźnikowe po odczytaniu bajta oznaczającego koniec transmisji danych.

VTIMR1 wektor przerwania licznika 1 układu POKEY. Po inicjalizacji wskazuje sekwencję instrukcji PLA, RTI.

VTIMR2 wektor licznika 2, jak wyżej,

VTIMR4 wektor licznika 4, jak wyżej.

VKEYBD wektor przerwań klawiatury. Naciśnięcie jakiegokolwiek klawisza (oprócz BREAK) wywołuje ten typ IRQ. Wektor ten może być wykorzystany np. do przetworzenia kodów naciśniętych klawiszy, zanim OS dokona konwersji na kod ATASCII. Normalnie wskazuje adres procedury OS obsługującej przerwanie IRQ klawiatury.

BRKKEY wektor klawisza BREAK. Po inicjalizacji wskazuje sekwencję instrukcji PLA, RTI.

VPRCED wektor przerwań procedurowych urządzeń peryferyjnych. Żądanie IRQ przesyłane jest z urządzeń peryferyjnych przez szynę szeregową. Przerwania te nie są wykorzystywane przez aktualny OS, a wektor wskazuje sekwencję instrukcji PLA, RTS.

VINTER wektor przerwań urządzeń peryferyjnych. Żądanie tego typu IRQ może być przesłane do urządzeń również szyną szeregową. Wektor niewykorzystany przez aktualny OS, wskazuje sekwencję instrukcji PLA, RTI.

VBREAK wektor przerwania instrukcji BRK mikroprocesora 6502. Przerwanie to wykonywane jest każdorazowo po otrzymaniu wykonaniu instrukcji BRK przez 6502. Wektor ten może być wykorzystany na przykład do zapisania adresów stopujących

pracę debuggera. Normalnie wskazuje sekwencję instrukcji PLA, RTI.

Cała grupa przerwań IRQ uzależniona jest od bitu rejestru flagowego mikroprocesora 6502 - zezwalającego bądź zabraniającego wykonania przerwania - ustawianego instrukcjami CLI oraz SEI. Ponadto każdy typ IRQ uzależniony jest od analogicznego bitu zezwalającego, znajdującego się w układzie POKEY.

Większość spośród tych bitów znajduje się w rejestrze IRQEN, który jest rejestrem typu tylko zapis. OS cieniuje ten rejestr adresem POKMSK ($0010), jednakże IRQEN nie jest przepisywany wartościami POKIMSK podczas wygaszenia pionowego. Zezwolenie na wykonanie każdego z przerwań wykonuje się poprzez włączenie odpowiedniego bitu w rejestrze IRQEN. Wyzerowanie danego bitu w rejestrze IRQEN stosuje się do zmiany statusu przerwania określonego odpowiadającym bitem rejestru IRQST. Należy zwrócić wagę, że wyjątkiem jest bit 3 - określający status przerwania zakończenia szeregowego przesyłania danych - który nie jest zerowany poprzez wyzerowanie odpowiadającego mu bitu 3 rejestru IRQEN. Bit ten jest jedynie bitem statusowym, odpowiadającym aktualnej wartości rejestru szeregowej transmisji danych.

Do zezwalania na przerwanie oraz testowania statusu przerwań IRQ wykonywanych przez PIA służą rejestry PACTL oraz PBCTL. Bit 0 każdego z tych rejestrów jest bitem zezwalającym na wykonanie przerwania przez poszczególny port, natomiast bit 7 reprezentuje status przerwania. Bit ten jest zerowany każdorazowo, kiedy zawartość rejestrów PACTL oraz PBCTL jest odczytywana.

- Wykorzystanie przerwań IRQ

Dostępność wektorów IRQ, zapisanych w pamięci RAM, oznacza jednocześnie możliwość dowolnego wykorzystania podsystemu I/O w zależności od potrzeb. System operacyjny nie umożliwia wprowadzania innych procedur w trakcie wykonywania operacji I/O. Jednakże poprzez zmianę wartości trzech wektorów przerwań operacji szeregowych I/O oraz przepisanie blokami procedur podsystemu I/O można zaprojektować różnorodne procesy współbieżne.

Trzy przerwania liczników układu POKEY mogą być wykorzystane w dowolnych sytuacjach, wymagających precyzyjnego sterowania czasu. Zwykle wykorzystuje się te liczniki w przypadkach, gdy programowe liczniki (60 Hz) okazują się zbyt wolne. Więcej informacji na ten temat przedstawiono w podrozdziale o programowaniu w czasie rzeczywistym.

W wielu programach aplikacyjnych zachodzi potrzeba zabezpieczenia programu przed błędem użytkownika. Do zabezpieczenia przed błędnym użyciem klawiatury może być wykorzystanych kilka spośród wektorów IRQ. Przykładowy program, przedstawiony na rys. 8-5. wykorzystuje wektor VKEYBD do unieruchomienia klawisza CONTROL oraz wektor VIMIRQ do zamaskowania klawisza BREAK i ignorowaniu jego żądań przerwania. Mimo, iż program został napisany dla pierwotnej wersji A OS (w związku z czym nie wykorzystuje istniejącego w wersji B wektora przerwań klawisza BREAK), w pełni realizuje swoje zadania w wersji B systemu.

```
0010            0010 POMSK      =     $0010
D2G9            0020 KBCODE     =     $D209
0203            0030 VKEYBD     =     $D208
D20E            0040 IRQEN      =     $D20E
D20E            0045 IRQST      =     IRQEN
0216            0046 VMIRQ      =     $0216
0000            0060           *=     $600
0600 78         0080 START SEI         ; WSTRZYMANIE IRQ
0601 AD1602     0090       LDA VMIRQ  ; PRZEPISANIE WEKTORA IRQ
0604 8D4D06     0100       Sta NBRK+1 ; NOWA WARTOSCIA
0607 AD1702     0110       LDA VMIRQ+1; WSZYSTKIE IRQ
060A 8D4E06     0120       STA NBRK+2 ; BEDA KIEROWANE DO NBRK
060D A945       0130       LDA #IRQ&255
060E 8D1602     0140       STA VMIRQ
0612 A906       0150       LDA #IRQ/256
0614 8D1702     0160       STA VMIRQ+1
0617 aD0802     0200       LDA VKEYBD ; ZAPIS WEKTORA IRQ
061B 8D4306     0210       STA JUMP+1 ; KLAWIATURY ADRESEM REP
061E AD902      6220       LDA VKEYBD+1
0621 6D4406     0230       STA JUMP+2
0624 A939       0240       LDA #REP&255 ; WEKTOR IRQ KLAWIATURY
0626 8D0802     0250       STA VKEYBD ; MLODSZY BAJT
0629 A906       0260       LDA #REP/256
062B 8D0902     0270       STA VKEYBD+1
062E 58         0275       CLI  ; ZEZWOLENIE NA IRQ
062F 60         0280       RTS
                0290           *=     5639
0639 AD09D2     0300 REP   LDAK KBCODE ; WSZYSTKIE IRQ KLAWIATURY
063C 2980       0310       AND #$80 ; SPRAWDZAJ KLAWISZ CONTROL
063E F002       0320       BEQ JUMP ; JEZELI NIE WCISNIETY
0640 68         0330       PLA     ; IRQ IGNOROWANE
0641 40         0340       RTI
0642 4C4206     0360 JUMP  JMP JUMP ; WYWOLANIE IRQ KLAWIATURY
0645 48         0375 IRQ   PHA ; POCZATEK PROCEDUR IRQ
0646 AD0ED2     0380       LDA ITQST ; SPRAWDZENIE KLAWISZA BREAK
0649 1004       0390       BPL BREAK ; JEŻELI TAK, WYKONAJ SKOK
064B 68         0405       PLA ; NIE, WYWOLAJ WEKTOR IRQ
064C 4C4C06     0410 NBRK  JMP NBRK ; WYWOLANIE WEKTORA IRQ
064E A97F       0430 BREAK LDA #$7F ; START PROCEDURY BREAK
0651 8D0ED2     0440       STA IRQST ; NIE WYKONUJ IRQ BREAK
0654 A510       0450       LDA POKMSK
0656 8D0ED2     0460       STA IRQEN
0659 68         0462       PLA
065A 40         0464       RTI ; POWROT BEZ WYKONANIA IRQ
065B            0470     *=$02E2
02E2 0006       0480       .WORD START
```

Rys. 8-5. Program zabezpieczający przed błędem użytkownika.

Dwoma wektorami IRQ zarządza układ PIA – są to VPRCED i VINTER. Chociaż wektory te nie są wykorzystywane w aktualnej wersji OS, można posłużyć się nimi do rozszerzenia kontroli nad komunikacją z urządzeniami zewnętrznymi.

- Sterownik przerwań NMI

Przerwania niemaskowalne (NMI) zarządzane są przez specjalny sterownik NMI znajdujący się w OS. W przeciwieństwie do IRQ, NMI nie mogą być zamaskowane (czyli zabronione) na poziomie mikroprocesora 6502. Zezwoleniami oraz zabronieniami wszelkiego typu NMI zarządza układ ANTIC.

Dwa spośród przerwań NMI, przerwania listy displejowej (DLI) oraz wygaszenia pionowego (VBLANK), posiadają swoje wektory w pamięci RAM, które mogą być wykorzystywane przez użytkownika. W rzeczywistości VBLANK może być przerwaniem dwóch rodzajów: natychmiastowym lub opóźnionym. Wektorami NMI są następujące rejestry:

Nazwa przerwania	Wektor
Resetowanie systemu	brak
Przerwania listy displejowej	VDSLST ($0200)
Przerwania wygaszenia pionowego	
Natychmiastowe	VVBLKI ($0222)
Opóźnione	VVBLKD ($0224)

NMI resetowania systemu nie posiada własnego wektora w RAM. Naciśnięcie klawisza RESET powoduje zawsze skok do procedury startu gorącego monitora. Jednakże podczas startu gorącego monitor OS wykorzystuje wektor RAM DOSINI. Ten właśnie wektor może być wykorzystany do ingerencji w przebieg procedury resetowania (patrz podrozdział o monitorze OS).

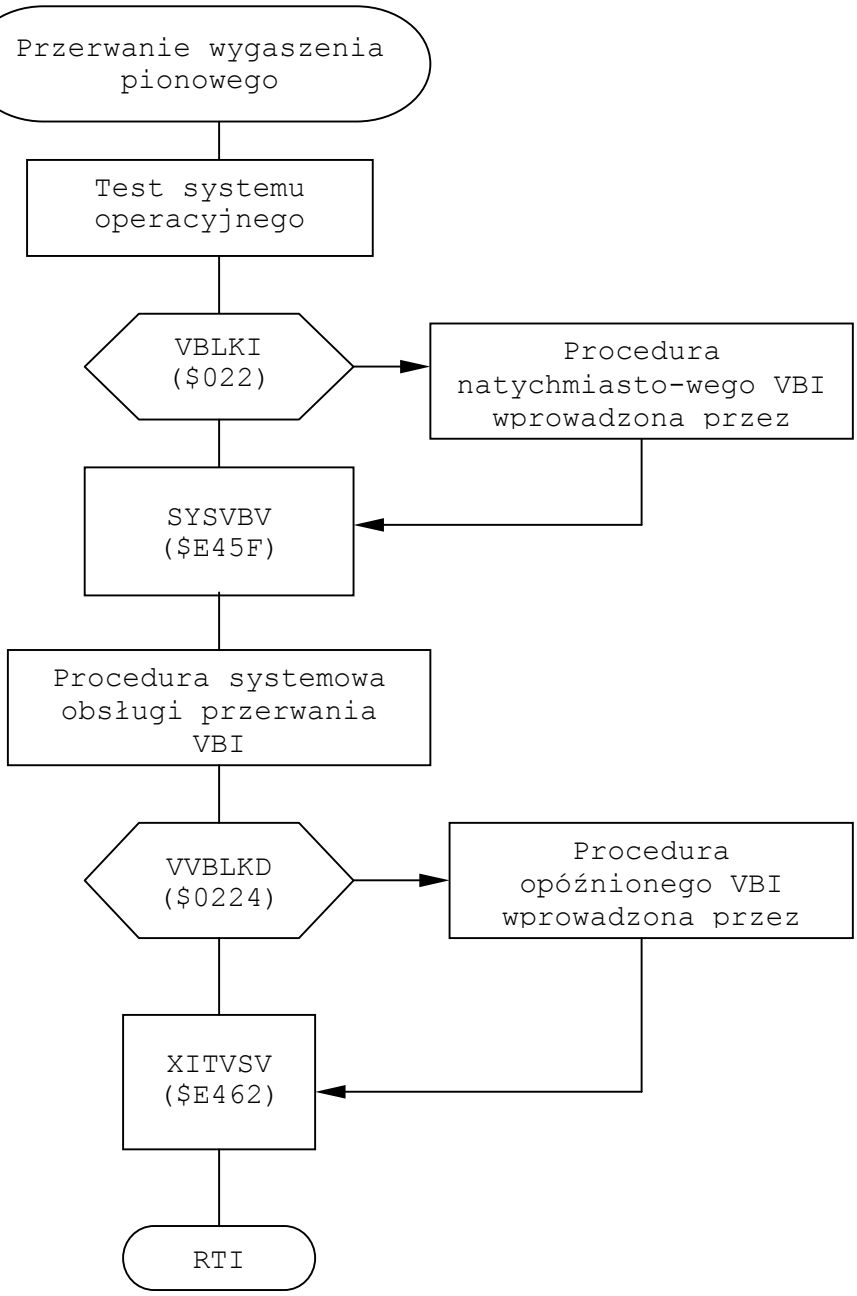
Wektor VDSLST nie jest wykorzystywany przez OS. Dokładny opis procedur przerwań listy displejowej oraz możliwości wykorzystania w programach aplikacyjnych zawarty został w rozdziale 5.

- Przerwania Wygaszenia Pionowego

Możliwość wykorzystania przerwań wygaszenia pionowego jest dla każdego programisty niezwykle wartościową cechą systemu. Są to przerwania niemaskowalne, wykonywane w regularnych odstępach czasowych, uzależnionych od standardu sygnału telewizyjnego (60 razy na sekundę w systemie NTSC, 50 razy na sekundę w systemach PAL oraz SECAM). A co najważniejsze, przerwania te mają miejsce w tym czasie, kiedy obraz na ekranie telewizora pozostaje wygaszony, a więc zmiany dokonywane w tym czasie nie będą bezpośrednio wpływały na obraz. Fakt ten decyduje o bardzo szerokich zastosowaniach przerwań wygaszenia pionowego.

Po otrzymaniu sygnału wykonania przerwania wygaszenia pionowego przez OS, zawartość rejestrów A, X i Y 6502 odkładana jest na stos. System kieruje się następnie do wektora natychmiastowego przerwania

VBLANK (VVBLKI), znajdującego się w rejestrze $0222. Wektor ten normalnie wskazuje na adres procedury obsługi przerwania wygaszenia pionowego OS - $E45F. OS przy pomocy tej procedury zwiększa wskazania zegara czasu rzeczywistego, zmniejsza wartości liczników systemowych, przepisuje wartości rejestrów kolorów, kopiuje pozostałe rejestry cieni oraz odświeża zapisy układów I/O. Procedura ta kończy się skokiem do wektora opóźnionego przerwania VBLANK (VVBLKD), znajdującego się pod adresem $0224. Wektor ten inicjalizowany jest adresem prostej procedury kończącej przerwanie, czyli wartością $E462. Całość procesu przedstawiona została na rys. 8-6.



Rys. 8-6. Proces wykonania przerwania VBI

Oba wektory umieszczone zostały w pamięci RAM, aby umożliwić programistom ingerencję w przebieg procedury przerwania wygaszenia pionowego i wykorzystanie tego przerwania do własnych potrzeb. Wykorzystanie VBLANK jest zabiegiem prostym. Najpierw należy zdecydować, czy własna procedura przerwania wygaszenia pionowego ma być wykonana jako VBI natychmiastowe, czy jako VBI opóźnione. W wielu przypadkach różnica pomiędzy obydwoma VBI jest nieznacząca dla programu. Jednak czasami różnica ta może mieć doniosłe znaczenie. Pierwszy taki przypadek ma miejsce wówczas, kiedy procedura VBI użytkownika odczytuje lub zapisuje rejestry, które są cieniowane przez OS podczas VBI. Czasami może okazać się niezbędne przepisanie zawartości rejestrów sprzętowych już po tym, jak OS przepisał je wartościami rejestrów cieni. Działa tutaj niepisane prawo, że przewagę ma ten, kto ostatni zapisuje rejestry.

Drugi przypadek ma miejsce wówczas, kiedy procedura VBI użytkownika pochłania zbyt wiele czasu mikroprocesora. Wówczas procedura systemowa obsługi VBI może być rozciągnięta poza koniec okresu wygaszenia pionowego. Może to spowodować zmianę zawartości rejestrów graficznych już w trakcie rysowania obrazu na ekranie, co spowoduje niepożądane efekty na obrazie. W takim przypadku procedura użytkownika powinna być wykonywana jako procedura opóźniona VBI. Limit czasu natychmiastowego VBI wynosi około 2000 cykli maszynowych, podczas gdy limit ten dla opóźnionego VBI sięga około 20000 cykli. Jednakże wiele spośród operacji, zawartych w tak szerokim limicie czasowym, będzie wykonywanych już w trakcie rysowania obrazu na ekranie. Wynika stąd wniosek, że długie procedury opóźnionego VBI, wprowadzone przez użytkownika, nie powinny wykonywać żadnych operacji na rejestrach graficznych. Ponadto, czas wykonywania przerwań listy displejowej musi być również wliczony w limit czasowy opóźnionego VBI. Trzeba też pamiętać, że im dłuższy jest czas wykonywania procedury VBI, tym mniej czasu ma 6502 na wykonywanie programu głównego.

Trzeci przypadek ma miejsce wówczas, kiedy wprowadzona przez użytkownika procedura VBI musi być wykonywana w połączeniu z krytyczno-czasowymi operacjami I/O, jak np. komunikacja z magnetofonem czy stacją dysków. Systemowa procedura natychmiastowego VBI dzieli się na dwa etapy: krytyczny pod względem czasowym oraz niekrytyczny. Podczas wykonywania krytyczno-czasowych operacji I/O systemowa procedura natychmiastowa VBI zostaje opóźniona do momentu, aż etap pierwszy dobiegnie końca. Jeżeli zachodzi konieczność wykonywania procedury VBI użytkownika każdorazowo podczas wygaszenia pionowego, musi to być wówczas procedura natychmiastowa VBI. W takich sytuacjach zawsze należy liczyć się z możliwością konfliktu pomiędzy procedurą natychmiastową VBI użytkownika a wykonaniem krytyczno-czasowej operacji I/O.

Po zdecydowaniu, czy dana procedura ma być wykonywana jako natychmiastowa czy jako opóźniona, należy umieścić ją w pamięci komputera, przy czym wygodnym obszarem do tego celu jest strona szósta RAM. Teraz należy dołączyć ją do właściwej procedury systemowej oraz zmodyfikować odpowiedni wektor RAM. Procedura natychmiastowa VBI powinna się kończyć instrukcją JMP $=E45F, procedura opóźniona natomiast instrukcją JMP $E462. Istnieje oczywiście możliwość całkowitego wyeliminowana systemowej procedury VBI (co jednocześnie zaoszczędzi nieco czasu), poprzez zakończenie procedury natychmiastowej VBI użytkownika instrukcją JMP $E462.

Znanym powszechne problemem przy projektowaniu przerwań dla procesorów 8-bitowych są trudności w zmianie adresu wektora przerwań. Wektory są rejestrami dwubajtowymi, a więc zmiana ich zawartości wymaga dwukrotnego użycia instrukcji typu "store". Szansa na to, iż przerwanie będzie miało miejsce już po przepisaniu wartości jednego bajta wektora, ale przed zmianą wartości drugiego bajta, są w zasadzie zerowe. Spowodować może to załamanie się całego systemu. System operacyjny komputerów Atari zawiera rozwiązanie tego problemu w postaci procedury zwanej SETVBV znajdującej się pod adresem $E45C. Wystarczy zapisać rejestr Y 6502 młodszym bajtem adresu, rejestr X starszym bajtem, natomiast akumulator wartością 6 dla procedury natychmiastowej VBI lub 7 dla procedury opóźnionej i wykonać instrukcję JMP SETVBV, by bezpiecznie uruchomić własną procedurę przerwania wygaszenia pionowego, która powinna zacząć działać po upływie 1/60 sekundy.

Przerwania wygaszenia pionowego umożliwiają wykonanie szerokiej gamy operacji. Po pierwsze, wykonanie wszelkiego typu manipulacji rejestrami graficznymi w trakcie VBI daje pewność, iż zmiany nie będą bezpośrednio wpływały na rysowany obraz. Po drugie, VBI umożliwiają wprowadzanie bazo szybkich, regularnych zmian w obrazie. Może to mieć olbrzymia znaczenie przy projektowaniu animacji ekranowej, w której wszelkie zmiany obrazu muszą być wykonane w ścisłym rozgraniczeniu z innymi procesami.

Inną funkcją przerwań VBI jest tworzenie efektów dźwiękowych w czasie rzeczywistym. Rejestry syntezatora dźwięku, w które wyposażone są komputery Atari 400/800, pozwalają na sterowanie częstotliwości, natężenia oraz dystorsji wytwarzanych dźwięków, ale nie czasu ich trwania. Na przykład wybrzmiewanie dźwięku może być sterowane poprzez cykliczną procedurę VBI, która każdorazowo zmniejsza zawartość ustalonego wcześniej rejestru wybrzmiewania. Tą metodą może być uzyskana pełna stera nad głośnością wytwarzanych dźwięków, włączając w to efekty powolnego nabrzmiewania i wybrzmiewania dźwięku oraz pseudo-pogłosu. Rozszerzenie tej techniki pozwala także na bardziej dokładną kontrolę częstotliwości oraz dystorsji dźwięku, co w rezultacie może prowadzić do uzyskania bardzo ciekawych efektów dźwiękowych, Z powodu, iż procedury VBI wykonywane są z częstotliwością 60 Hz, nie są one przydatne jedynie do sterowania dźwięków wytwarzanych techniką regulacji wyłącznie głośności.

Procedury VBI są także przydatne do sterowania operacjami przesyłania i odczytywania danych. Zwykle proces czytania danych, na przykład z klawiatury, wymaga od 6502 przeprowadzenia niewielu operacji, lecz długiego oczekiwania na wprowadzane dane. Procedury VBI umożliwiają taką organizację, w której 6502 podczas każdego VBI sprawdza, czy wprowadzona została dana, po czym wraca do wykonywania innych operacji. Jest to rozwiązanie idealne (choć nieco teoretyczne), będące kompromisem - procesor wykonuje zaprogramowane operacje, nie ignorując jednocześnie użytkownika.

Wreszcie ostatnia idea - wprowadzenie prostej odmiany wielotorowości wykonywania programów, Program uboczny może być wykonywany jako część procesu VBI, podczas gdy program główny wykonywany jest pomiędzy okresami wygaszenia pionowego. Podobnie, jak w przypadku innych procedur przerwań, należy ściśle rozgraniczyć zbiory danych, na których będą operowały oba programy. Realizacja

tej idei wymaga z pewnością dużego nakładu pracy, lecz jej efekty mogą być niewspółmiernie wysokie.

WEKTORY SYSTEMOWE

Jedną z miar prężności operacyjnego systemu komputerowego jest jego zdolność adaptacyjna. Chyba wszyscy wiedzą już, jak proste w przypadku systemu komputerów Atari jest wykorzystanie procedur systemowych do własnych potrzeb bądź zastąpienie czy rozbudowanie istniejących już procedur.

Wydaje się, że system komputerów Atari jest pod tym względem jednym z najbardziej prężnych systemów komputerowych. Praktycznie w każdym przypadku, kiedy zachodzi potrzeba ingerencji w procedury systemowe, OS "otwiera furtkę", przez którą procedury te mogą być wykorzystywane do innych celów lub zastępowane innymi procedurami w zależności od potrzeb.

Tak szerokie możliwości adaptacyjne systemu zapewnione zostały poprzez kombinację kilku różnych mechanizmów. Pierwszym z nich są tablice, umieszczone w pamięci ROM (Rys. 8-7.), zawierająca instrukcje JMP do odpowiednich procedur. W kolejnych wersjach OS adresy tej tablicy z pewnością nie ulegną zmianie, choć mogą się zmienić operandy zawartych w niej instrukcji JMP. Tak więc programy, które wykorzystują procedury systemowe poprzez tę tablicę, będą pracowały niezależnie od wersji OS komputera. Jeżeli natomiast programy nie posługują się tablicą, lecz kierują się bezpośrednio do procedur systemowych własną instrukcją JMP, jest prawie pewne, że nie dadzą się one uruchomić na komputerach wyposażonych w nowsze wersje systemu operacyjnego.

Drugi mechanizm stanowi szereg wektorów adresowych, znajdujących się w pamięci RAM, tworzących drugą tablicę, ułatwiającą wykorzystywanie procedur systemowych. A więc zmiana procedury jakiegokolwiek przerwania na inną, własną procedurę, wymaga zmiany jedynie dwubajtowego wektora danej procedury. OS inicjalizuje wektory podczas procedury startu zimnego. Podobnie, jak w poprzednim przypadku, inicjalizowane wartości mogą ulec zmianie, lecz adresy tablicy wektorów przerwań pozostaną te same, bez względu na odmianę systemu operacyjnego.

Trzeci mechanizm stanowi analogiczna tablica sterowników urządzeń peryferyjnych, której wektory wskazują bezpośrednio na adresy poszczególnych sterowników (np. stacji dysków, drukarki itp.). Dyskusja na temat tej tablicy zawarta została w podrozdziale o scentralizowanych operacjach wejścia/wyjścia.

Ponieważ tablica wektorów w pamięci ROM jest tablicą trzybajtowych instrukcji JMP, przykładem wykorzystania wektora ROM może być:

```
JSR CIOV
```

TABLICE WEKTORÓW W PAMIĘCI RAM

W odróżnieniu od tablicy ROM, zawierającej instrukcje JMP, wektory w pamięci RAM są rejestrami dwubajtowymi. Pełny spis wektorów znajdujących się w RAM przedstawiono w tabl. 8-8. Typowym przykładem

sekwencji instrukcji, wykorzystujących wektory RAM procedur systemowych, może być:

```
JSR CALL
CALL JMP (DOSINI)
```

Nazwa	Adres	Zastosowanie
DISKIV	$E450	Inicjalizacja sterownika dysku
DSKINV	$E453	Wektor sterownika dysku
CIOV	$E456	Wektor procedury scentralizowanego I/O
SIOV	$E459	Wektor procedury szeregowego I/O
SETVBV	$E45C	Wektor procedury ustawienia liczników OS
SYSVBV	$E45F	Wektor procedury wygaszenia pionowego OS
XITVBV	$E462	Wektor procedury zakończenia VBLANK OS
SIOINV	$E465	Wektor inicjalizacji szeregowego I/O
SENDEV	$E468	Wektor zezwolenia przesłania szeregowego
INTINV	$E46B	Wektor sterownika obsługi przerwań
CIOINV	$E46E	Wektor inicjalizacji scentralizowanego I/O
BLKBDV	$E471	Wektor trybu testu pamięci
WARMSV	$E474	Wektor procedury startu gorącego
COLDSV	$E477	Wektor procedury startu zimnego
RBLOKV	$E47A	Wektor procedury odczytu bloku z kasety
CSOPIV	$E47D	Wektor otwarcia kasety do zapisu

Rys. 8-7. Wektory procedur systemowych w pamięci ROM

SCENTRALIZOWANY PODSYSTEM WEJŚCIA/WYJŚCIA

Jednym z najpoważniejszych problemów, stojących przed każdym projektantem systemu operacyjnego, jest sposób organizacji operacji wejścia/wyjścia w komunikacji z różnorodnymi urządzeniami peryferyjnymi, które mogą być dołączane do systemu. Podstawowe założenia każdego podsystemu sterowania I/O są następujące:

- Przesyłanie danych powinno być niezależne od rodzaju urządzenia peryferyjnego.
- Struktura I/O powinna zezwalać zarówno na przesyłanie danych jednobajtowych, wielobajtowych, jak i całkowicie zorganizowanych w blokach rekordów.
- Powinny być jednocześnie dostępne różne urządzenia peryferyjne jak też i różne zbiory danych.
- W przypadku wystąpienia błędów w komunikacji sterowanie powinno być przekazywane konkretnemu urządzeniu.
- Powinna istnieć możliwość zaprojektowania nowych sterowników urządzeń bez konieczności zmiany całego systemu operacyjnego.

Nazwa	Adres	Wartość	Zastosowanie
Wektory strony drugiej			
VDSLST	$0200	$E7B3	Wektor przerwań NMI listy displejowej
VPRCED	$0202	$E7B3	Wektor IRQ procedury pryferyjnej
VINTER	$0204	$E7B3	Wektor IRQ urządzeń peryferyjnych
VBREAK	$0206	$E7B3	Wektor IRQ programowej instrukcji BRK
VKEYBD	$0208	$EFBE	Wektor IRQ klawiatury
VSERIN	$020A	$EB11	Wektor IRQ gotowości wejścia szeregowego
VSEROR	$020C	$EA90	Wektor IRQ gotowości wyjścia szeregowego
VSEROC	$020E	$EAD1	Wektor IRQ zakończenia przesyłania szereg.
VTIMR1	$0210	$E7B3	Wektor IRQ licznika 1 układu POKEY
VTIMR2	$0212	$E7B3	Wektor IRQ licznika 2 układu POKEY
VTIMR4	$0214	$E7B3	Wektor IRQ licznika 4 układu POKEY
VIMIRQ	$0216	$E6F6	Wektor sterownika przerwań IRQ
VVBLKI	$0222	$E7D1	Wektor NMI natychmiastowego VBI
VVBLKD	$0224	$E93E	Wektor NMI opóźnionego VBI
CDTMA1	$0226	$XXXX	Adres JSR licznika systemowego 1
CDTMA2	$0228	$XXXX	Adres JSR licznika systemowego 2
BRKKEY	$0236	$E754	Wektor IRQ klawisza BREAK **
Wektory strony zerowej			
CASINI	$0002	$XXXX	Wektor inicjalizacji bootingu kasetowego
DOSINI	$000C	$XXXX	Wektor inicjalizacji dysku
DOSVEC	$000A	$XXXX	Wektor startu programów dyskowych
RUNVEC	$02E0	$XXXX	Wektor „Load & Run” DUP.SYS dysku
INIVEC	$02E2	$XXXX	Wektor „Load & Initialize” DUP.SYS dysku

XXXX oznacza, że zawartość danego wektora jest zmienna.
** Tylko w OS wersji B.

Rys. 8-8. Tablice wektorów w pamięci RAM.

System operacyjny komputerów Atari 400/800 został zaprojektowany ściśle według wymienionych wyżej wskazówek. Scentralizowany podsystem I/O jest wykorzystywany przez OS metodą tablicową. Z punktu widzenia OS, wszystkie operacje wejścia/wyjścia zgrupowane są wokół IOCB, czyli Bloku Sterującego Wejścia/Wyjścia. IOCB jest standardową tablicą, precyzującą jedną kompletną operację wejścia lub wyjścia. Każda z ośmiu standardowych operacji I/O może być wykonana poprzez IOCB. Poprzez zmianę adresu wejściowego IOCB użytkownik może mieć bezpośredni wpływ na rodzaj każdej operacji I/O, czy nawet na konkretne urządzenie peryferyjne, które staje się podporządkowane IOCB. Tak więc użytkownik może z łatwością wykonać tę samą operację I/O w stosunku do różnych peryferiów, nie przejmując się wcale sprzętowymi szczegółami procesu. Większość

operacji I/O wymaga jedynie prawidłowego wpisania parametrów sterujących IOCB - a następnie przekazania kontroli do podsystemu I/O.
Na podsystem I/O składają się dwa typy elementów: procedury systemowe I/O oraz bloki sterujące systemu I/O. Procedury systemowe I/O zawierają centralną procedurę I/O (CIO). sterowniki urządzeń peryferyjnych (E:, K:, S:, P:, C:, D:, R:) oraz procedurę szeregowego I/O (SIO). Tablica adresowa sterowników (HATABS) jest miejscem centralnym, łączącym CIO z poszczególnymi sterownikami urządzeń. Bloki sterująca systemu I/O zawierają dane sterujące, przenoszone do podsystemu I/O. Działania użytkownika w komunikacji z wszystkimi urządzeniami są w zasadzie takie same - na przykład przekazanie wiersza znaków do drukarki i do edytora przebiega bardzo podobnie.

Zrozumienie struktury podsystemu I/O pozwoli na pełniejsze jego wykorzystanie. Rys. 8-9 ukazuje współzależności pomiędzy procedurami systemowymi I/O a blokami sterującymi systemu I/O.

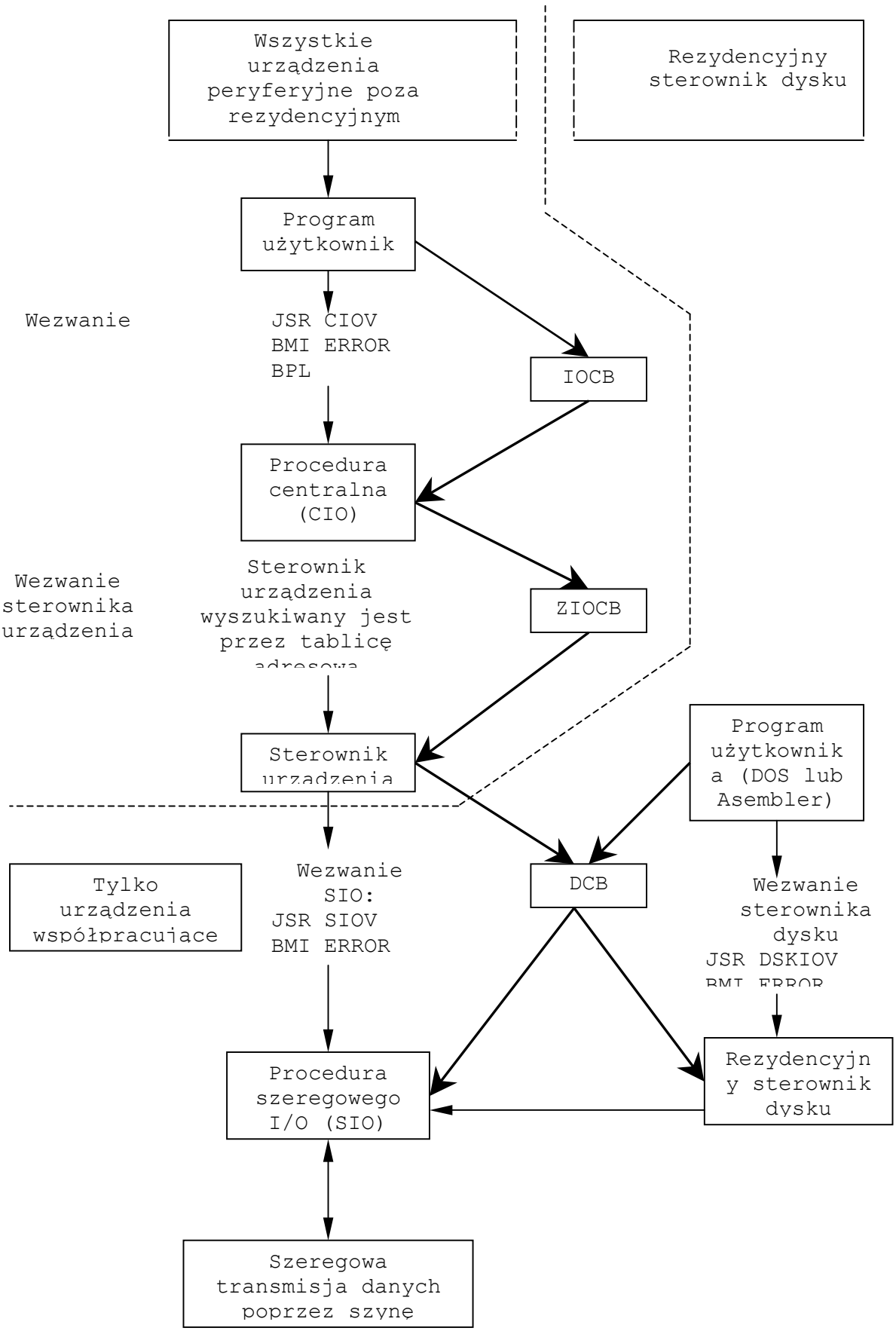
- Bloki sterujące systemu I/O

Wyróżnić można cztery typy bloków sterujących:
- Blok Sterujący Wejścia/Wyjścia (IOCB)
- Blok Sterujący I/O Strony Zerowej (ZIOCB)
- Blok Sterujący Urządzeń (DCB)
- Bufor Ramowy Komend (CFB)

Bloki sterujące systemu I/O wykorzystane są do przesyłania informacji określających funkcję I/O, która ma być wykonana. Bloki te wykonują następnie procedury systemowe I/O w oparciu o parametry sterujące, które prowadzą do realizacji określonej funkcji I/O. Szczegółowe informacje dotyczące struktury wszystkich typów bloków sterujących znaleźć można w "Operating System Manual".

Osiem bloków ZOCB wykorzystywanych jest przez OS do komunikowania programu użytkownika z CIO. Rys. 8-10 ukazuje zawartość IOCB dla niektórych najpowszechniej stosowanych funkcji I/O. Bloki IOCB znajdują się w następującym obszarze pamięci:

Nazwa	Adres, długość
IOCB0	$340, 16
IOCB1	$350, 16
IOCB2	$360, 16
IOCB3	$370, 16
IOCB4	$380, 16
IOCB5	$390, 16
IOCB6	$3A0, 16
IOCB7	$3B0, 16


Wszystkie urządzenia peryferyjne poza rezydencyjnym
Rezydencyjny sterownik dysku
Program użytkownik
Wezwanie
JSR CIOV
BMI ERROR
BPL
IOCB
Procedura centralna (CIO)
Wezwanie sterownika urządzenia
Sterownik urządzenia wyszukiwany jest przez tablicę
ZIOCB
Sterownik
Program użytkownika (DOS lub Asembler)
Tylko urządzenia współpracujące
Wezwanie SIO:
JSR SIOV
BMI ERROR
DCB
Wezwanie sterownika dysku
JSR DSKIOV
Procedura szeregowego I/O (SIO)
Rezydencyjny sterownik dysku
Szeregowa transmisja danych poprzez szynę

Rys. 8-9. Podsystem I/O.

Drugi typ bloku sterującego, ZIOCB ($0020,16), wykorzystywany jest do przekazywania danych pomiędzy CIO oraz sterownikami urządzeń peryferyjnych. Po wywołaniu, CIO wykorzystuje zawartość rejestru X 6502 jako indeks, wskazujący który z ośmiu IOCB ma być wykorzystany. CIO przesyła następnie dane sterujące z danego IOCB do ZIOCB, które dalej wykorzystane będą przez sterownik urządzenia. Bezpośrednie wykorzystanie ZIOCB przez użytkownika ograniczone jest do przypadków projektowania nowych sterowników urządzeń peryferyjnych lub zmiany sterowników już istniejących.

W następnej kolejności odpowiedni sterownik urządzenia, które współpracuje z szyną szeregową, przesyła dane sterujące do DCB, ($0300,12). SIO steruje tę informację, po czym przesyła z powrotem parametry statusowe do wykorzystania przez sterownik urządzenia. Rys. 8-11 ilustruje niektóre powszechne funkcje I/O oraz związaną z nimi zawartość bloku DCB.

Rezydencyjny sterownik dysku nie stosuje tej samej sekwencji użytkownik - CIO - sterownik – SIO. W przeciwieństwie, bezpośrednia komunikacja z rezydencyjnym sterownikiem dysku przebiega tylko przez DCB. W rozdziale 9, dotyczącym dyskowego systemu operacyjnego, zawarto więcej informacji na temat tego sterownika.

Ostatnim typem bloku sterującego, włączonego do podsystemu I/O, jest bufor ramowy komend /CFB/. Ta 4-bajtowa tablica, mieszcząca się pod adresem S023A, wykorzystywana jest przez procedurę SIO w trakcie transmisji danych szyną szeregową. Na cztery bajty składają się kod urządzenia, kod komendy, oraz bajty zapasowe komendy 1 i 2. Bufor danych do transmisji definiuje się przez dwa wektory: BUFRLO ($0032,2) oraz BFENLO ($0034,2). Generalnie nie zaleca się wykorzystywania OS na tym poziomie. Pozostałe parametry mogą być zmieniane, lecz należy wówczas ze szczególną ostrożnością wywoływać procedury systemowe I/O. CIO oraz SIO zostały tak zaprojektowane, by z łatwością można je było wykorzystać w programach użytkownika. Należy z nich korzystać - lecz pod żadnym pozorem nie ingerować w zawartość buforu ramowego komend!

- Procedura centralna systemu I/O

Naczelnym zadaniem CIO jest odczytanie danych sterujących z określonego IOCB, sprawdzenie, czy dotyczą one wyszczególnionego urządzenia peryferyjnego i przekazanie kontroli odpowiedniemu sterownikowi. CIO służy także jako program bazowy dla wszystkich operacji I/O systemu. Większość funkcji I/O OS wykorzystuje CIO jako wejście do danego sterownika, a sterownik jako wejście powrotne do OS. Dla przykładu, CIO będzie wywoływany z BASICu takimi instrukcjami jak OPEN czy GRAPHICS. CIO dopuszcza realizację następujących funkcji:

0PEN	otwiera zbiór bądź urządzenie
CLOSE	zamyka zbiór bądź urządzenie
GET CHARS	czyta N symboli
READ RECORD	czyta następny rekord
PUT CHARS	zapisuje N symboli
WRITE RECORD	zapisuje następny rekord

KARTA IOCB

WEZWANIE	ICHID	ICDNO	ICCOM	ICSTA	ICBAL	ICBAH	ICPTL	ICPTH	ICBLL	ICBLH	ICAX1	ICAX2
OPEN FILE – READ	X	X	3	„1”	$80	06	X	X	X	X	4	0
OPEN FILE – WRITE	X	X	3	„1”	$80	06	X	X	X	X	8	„2”
GET BYTES	X	X	7	„1”	00	06	X	X	$80	00	X	X
PUT BYTES	X	X	$B	„1”	00	06	X	X	$80	00	X	X
GET RECORD	X	X	5	„1”	00	06	X	X	$80	00	X	X
PUT RECORD	X	X	9	„1”	00	06	X	X	$80	00	X	X
CLOSE FILE	X	X	$C	„1”	X	X	X	X	X	X	X	X
STATUS	X	X	$D	„1”	X	X	X	X	X	X	X	X

STATUS określa status urządzenia

„1” – Status komendy I/O zapisywany jest tutaj oraz do rejestru Y 6502 przed powrotem do CIO.
„2” – Bajty dodatkowe IOCB wykorzystywane są przez niektóre sterowniki do określenia specjalnych trybów pracy urządzenia.
X – oznacza wartość ignorowaną, ale nie ulegającą zmianie.

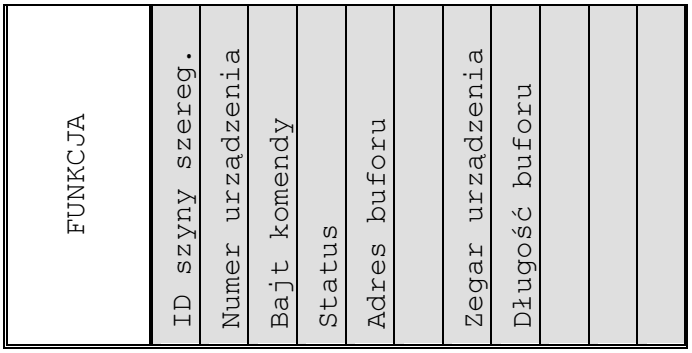
Rys. 8-10. Blok Sterujący Wejścia/ Wyjścia (IOCB).

KARTA DCB

NAZWA	ADRES	STACJA DYSKÓW 810/815						DRUKARKA ATARI 820 WRITE
		READ SECTOR		WRITE SECTOR		PUT	FORMAT	
		810	815	810	815			
DDEVIC	$0300	$30	$30	$30	$30	$30	$30	$40
DUNIT	$301	1-4	1-8	1-4	1-8	1-4	1-4	1
DCOMND	$302	$52	$52	$57	$57	$50	$21	$57
DSTATS	$303	$40	$40	$80	$80	$80	$40	$80
DBUFLO	$304	U	U	U	U	U	U	U
DBUFHI	$305	U	U	U	U	U	U	U
DTIMLO	$306	$30	$30	$30	$30	$31	$130	5
DBYTLO	$308	$80	00	$80	00	$80/00	-	$40
DBYTHI	$309	$00	01	$00	01	$00/01	-	$00
DAUX1	$30A	2*	2*	2*	2*	2*	-	1*
DAUX2	$30B	2*	2*	2*	2*	2*	-	1*

1* - Bajty te określają tryb pracy drukarki
2* - DAUX1 i DAUX2 określają sektory dla komend READ, WRITE (PUT) i WRITE zweryfikowane.
U - Adres wpisany przez użytkownika
- - Bajt ignorowany.

Rys. 8-11. Blok Sterujący Urządzeń Peryferyjnych.



SPECIAL realizuje specyficzne funkcje danego sterownika (np. NOTE dla FMS)

Oczywiście, można zaprojektować własne wezwanie CIO. Sekwencja instrukcji wzywająca CIO jest zastępująca:

```
                    ; REM parametry IOCB muszą być już ustalone
LDX   IOCBWM        ; zapis indeksu IOCB (nr IOCB * 16)
JSR   CIOV          ; wektor procedury systemowej CIO
BMI   ERROR         ; jeżeli wystąpi błąd, wykonywany jest skok,
                    ; podczas którego CIO zapisuje kod błędu
                    ; w rejestrze Y
```

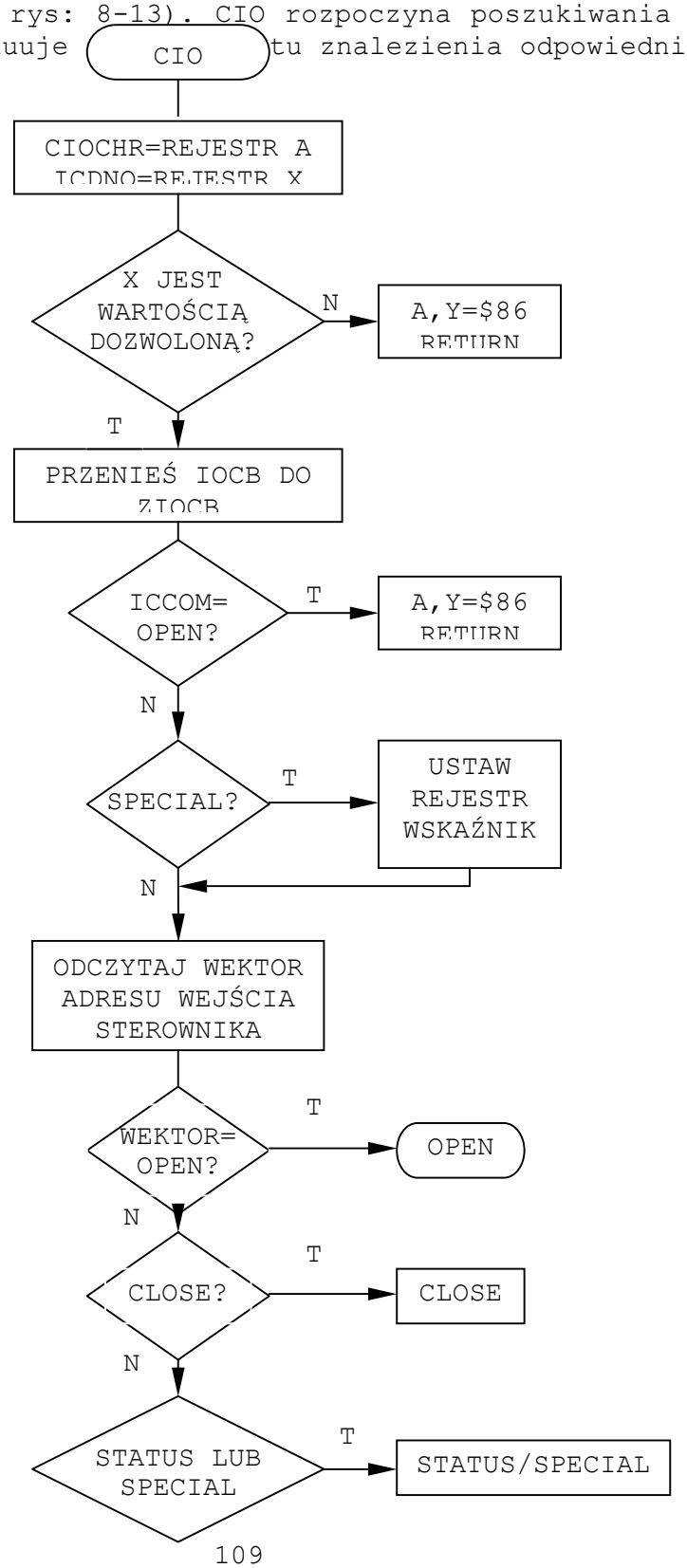
Jak wynika z te j sekwencji, jeden z bloków IOCB wykorzystywany jest do przeniesienia parametrów sterujących do CIO. Do tego celu może być wykorzystany dowolny z ośmiu IOCB. CIO odczytuje numer bloku z rejestru X - musi tam być zapisany numer IOCB pomnożony przez 16 (CIO wykorzystuje rejestr X jako indeks do wykonania skoku do IOCB, a bloki te mają długość 15 bajtów). Przy powrocie bit znakowy (N) rejestru znaczników 6502 jest wykorzystywany jako wskaźnik powodzenia lub błędu danej operacji I/O. Jeżeli bit N równy jest 0, operacja I/O wykonana została z powodzeniem, a w rejestrze Y musi być zapisana 1. Jeżeli bit N równy jest 1 oznacza to, iż podczas wykonywania operacji I/O wystąpił błąd, a rodzaj błędu zakodowany jest w rejestrze Y. Instrukcja BMI, wykonująca skok do procedury błędu w przypadku jego wystąpienia, jest często stosowaną metodą sprawdzenia powodzenia wykonywanej operacji. Odpowiednia wartość błędu lub powodzenia zapisywana jest ponadto w rejestrze ICSTA bloku IOCB. W rozdziale 5 "Operating System Manual" przedstawiono świetny przykład programu, wywołującego CIO do otwarcia (OPEN) zbioru dyskowego, przeczytania (READ) kilku rekordów oraz zamknięcia (CLOSE) zbioru.

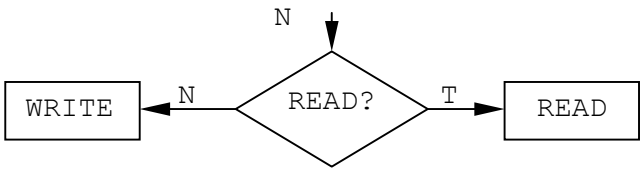
CIO kopiuje parametry sterujące z określonego bloku IOCB na stronę zerową do ZIOCB. CIO wyszukuje adres wejściowy odpowiedniego sterownika i przy pomocy wektora wywołuje żądaną procedurę sterownika. Na rys. 8-12 przedstawiono schemat funkcjonowania procedury systemowej CIO.

- Tablica adresowa sterowników

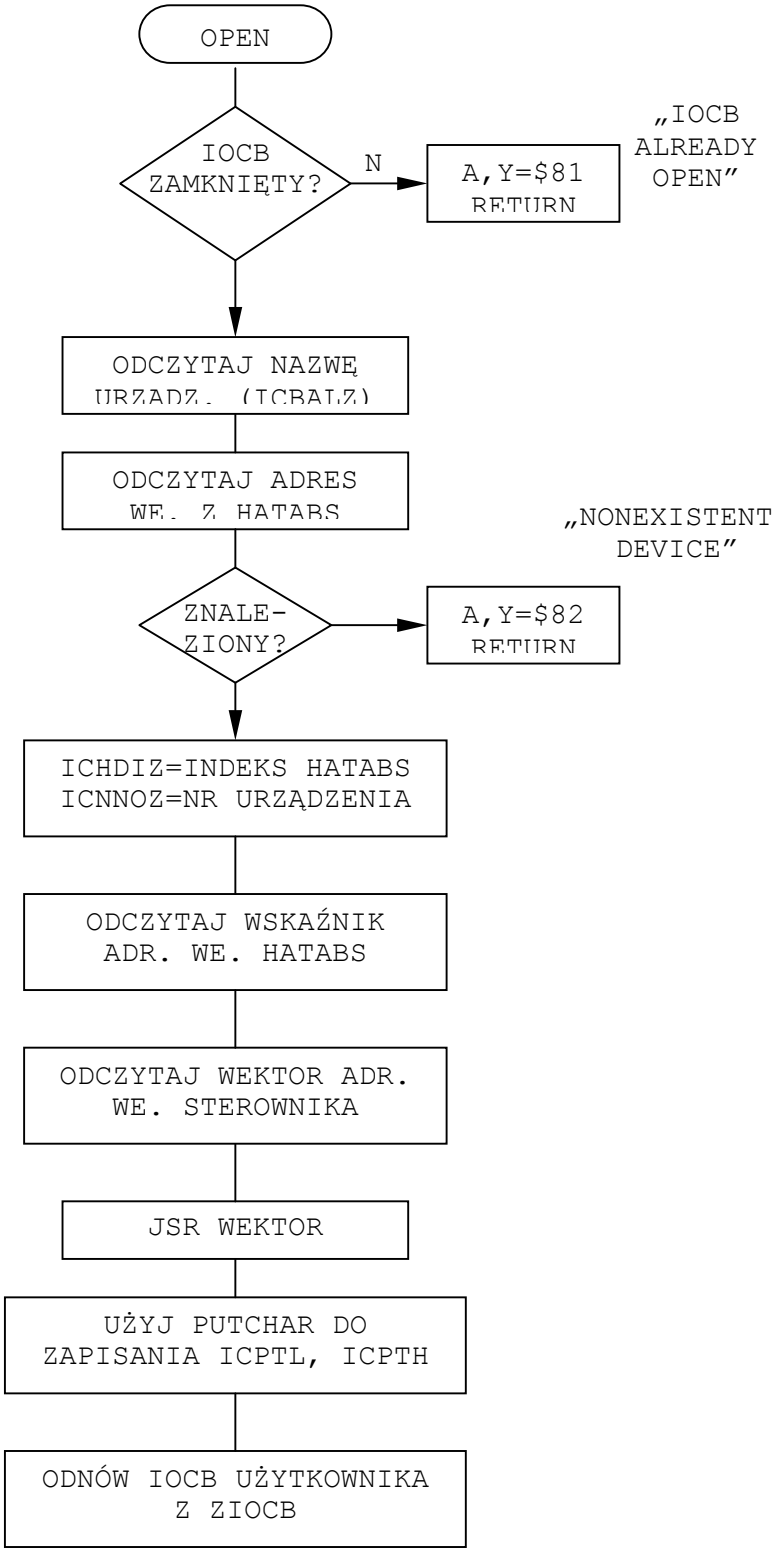
CIO odnajduje adres wejściowy odpowiedniego sterownika urządzenia metodą adresowania pośredniego. Przede wszystkim, wykonanie każdej

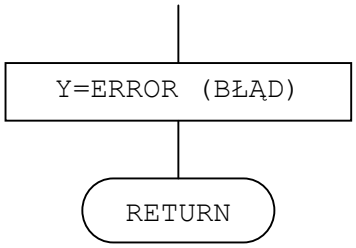
funkcji I/O musi być poprzedzone instrukcją OPEN danego urządzenia. Podczas wykonywania tej komendy CIO odczytuje parametry określające urządzenie, w którym dany zbiór ma być otwarty. Urządzenie charakteryzowane jest poprzez wyrażenie tekstowe w kodzie ATASCII, wskazujące na część adresową buforu bloku IOCB. Pierwszym elementem tego wyrażenia musi być identyfikator urządzenia (tzn. "D" dla stacji dysków, "P" dla drukarki itd.). CIO odnajduje taki sam symbol w tablicy adresów wejściowych sterowników, zwanej HATABS, znajdującej się na obszarze od $031A do $033B (struktura tej tablicy pokazana została na rys: 8-13). CIO rozpoczyna poszukiwania od dołu tej tablicy i kontynuuje tu znalezienia odpowiedniego





Rys. 8-12.1. Procedura CIO.





Rys. 8-12.2. Procedura CIO.

symbolu. Ten kierunek przeszukiwania wybrano dla ułatwienia wprowadzenia do tablicy modyfikacji lub całkowicie nowych elementów. Podczas procedur inicjalizacyjnych tablica HATABS kopiowana jest z ROM do RAM. Jeżeli uprzednio wykonany został booting jakiegoś urządzenia, jak np. stacja dysków lub moduł RS-232, informacje sterujące danego urządzenia dopisywane są do tablicy na jej dole. HATABS zawiera miejsca dla 14 adresów wejściowych, z czego 5 zapisywanych jest podczas procedur inicjalizacyjnych. Jeżeli na dole tabeli zapisane zostały dane sterujące dla np. nowego sterownika drukarki, wówczas CIO odnajdzie je wcześniej, niż dane drukarki przeniesione z ROM. Pozwala to na wprowadzanie adresów nowych sterowników urządzeń peryferyjnych.

Po znalezieniu identyfikatora urządzenia CIO odczytuje następna dwa bajty jako adres wejściowy sterownika danego urządzenia. Jest to tablica adresowa procedur zarządzających każdą z funkcji CIO. Przykładowa tablica adresów wejściowych drukarki przedstawiona została na rys. 8-13.

CIO wykorzystuje następnie rejestr ICCOM, bajt komendy IOCB, jako indeks do odczytania adresu z tablicy adresów wejściowych - czyli traktuje go jako wektor poszukiwania właściwego adresu wejściowego sterownika dla danej komendy. Tablice adresów wejściowych wszystkich rezydencyjnych sterowników urządzeń peryferyjnych mogą być znalezione podczas listingu OS. Pozycja każdego z wektorów we wszystkich tablicach adresów wejściowych jest taka sama. Na przykład, pierwszą pozycją we wszystkich tablicach adresów wejściowych sterowników są wektory procedur OPEN odpowiednie dla każdego sterownika.

Możliwość dołączenia do tablicy HATABS nowych adresów wejściowych stwarza szerokie możliwości rozbudowania funkcji systemu. Rys. 8-15 przedstawia przykładowy program dołączenia do tablicy sterownika zerowego, którego zadania świetnie odzwierciedla jego nazwa: nie robi on nic. Sposób ten może być użyteczny przy disasemblacji programów. Zamiast oczekiwania przez około 50000 odczytów dyskowych na znalezienie konkretnego błędu, informacje wyjściowe mogą być kierowane do sterownika zerowego - pozwoli to na wiele szybsze odnalezienie błędów lub miejsc powodujących błędy w zapisanych na dysku programach.

```
                0001 ; TABLICA ADRESOWA STEROWNIKÓW
E430            0002 PRINTV = $E430
E440            0003 CASETV = $E440
```

```
E400            0004 EDITRV = $E400
E410            0005 SCRENV = $E410
E420            0006 KEYBDV = $E420
                0007 ;
0000            0008       *= $031A
                0009 ;
                0010 HATABS
031A 50         0020       .BYTE "P"       DRUKARKA
031B 30 E4      0030       .WORD PRINTV    ADRES WEJSCIOWY TABLICY
031E 43         0040       .BYTE "C"       KASETA
031E 40 E4      0050       .WORD CASETV    ADRES WEJSCIOWY TABLICY
0320 45         0060       .BYTE "E"       EDYTOR EKRANOWY
0321 00 E4      0070       .WORD EDITRV    ADRES WEJSCIOWY TABLICY
0323 53         0030       .BYTE „S”       EKRAN
0324 10 E4      0090       .WORD SCRENV    ADRES WEJSCIOWY TABLICY
0326 4B         0100       .BYTE "K"       KLAWIATURA
0327 20 E4      0110       .WORD KEYBDV    ADRES WEJSCIOWY TABLICY
0329 00         0120       .BYTE 0         WEKTOR WOLNY 1 (DOS)
032A 00 00      0130       .BYTE 0,0
032C 00         0140       .BYTE 0         WEKTOR WOLNY 2 (850)
032D 00 00      0150       .BYTE 0,0
032F 00         0160       .BYTE 0         WEKTOR WOLNY 3
0330 00 00      0170       .BYTE 0,0
0332 00         0180       .BYTE 0         WEKTOR WOLNY 4
0333 00 00      0190       .BYTE 0,0
0335 00         0200       .BYTE 0         WEKTOR WOLNY 5
0336 00 00      0210       .BYTE 0,0
0338 00         0220       .BYTE 0         WEKTOR WOLNY 6
0339 00 00      0230       .BYTE 0,0
033B 00         0240       .BYTE 0         WEKTOR WOLNY 7
```

Rys. 8-13. Tablica adresowa sterowników (HATABS).

```
                           *=$PRINTV
E430  9E EE      .WORD     PHOPEN-1   OTWÓRZ URZĄDZENIE
E432  DB EE      .WORD     PHCLOS-1   ZAMKNIJ URZĄDZENIE
E434  9D EE      .WORD     BADST-1    CZYTAJ - NIEDOZWOLONE
E436  A6 EE      .WORD     PHWRIT-1   PISZ
E438  80 EE      .WORD     PHSTAT-1   OKRESL STATUS
E43A  9D EE      .WORD     BADST-1    SPECJALNE - NIEDOZWOLONE
E43C  4C 78 EF   JMP  PHINITIT        INICJALIZACJA URZĄDZENIA
```

Rys. 8-14. Tablica adresów wejściowych sterownika drukarki.

```
0000            0010          *= 5600
031A            0020 HATABS     = 3031A
0600 A0 00      0040 START      LDY #0
```

```
0602 B9 1A 03    0050 PETLA      LDA HATABS, Y
0605 09 00       0070            CMP #0 ; CZY JEST WOLNY WEKTOR?
0607 F0 09       0080            BEQ FOUND
0609 C8          0090            INY
060A C8          0100            INY
060B C8          0110            INY
060C C0 22       0120            CPY #34 ; NASTĘPNY WEKTOR HATABS
060E D0 F2       0130            BNE PETLA ; CZY TO KONIEC HATABS?
0610 38          0140            SEC     ; NIE, KONTYNUUJ
0611 60          0150            RTS ; TAK, SYGNALIZUJ BŁĄD
                 0160 ;
0612 A94E        0180 FOUND      LDA #'N' ; NAZWA URZĄDZENIA
0614 99 1A 03    0190            STA HATABS, Y
0617 C8          0200            INY
0618 A9 24       0210            LDA #NULLTAB&255
061A 99 1A 03    0220            STA HATABS,Y ; ADRES STEROWNIKA
061D C8          0230            INY
061E A906        0240            LDA #NULLTAB/256
0620 99 1A 03    0250            STA HATABS, Y
0623 60          0260            RTS
                 0270 ;
0624 32 06       0290 NULLTAB    .WORD RTHAND-1 ; OPEN
0626 32 06       0300            .WORD RTHAND-1 ; CLOSE
0628 34 06       0310            .WORD NOFUNC-1 ; READAD
062A 32 06       0320            .WORD RTHAND-1 ; WRITE
062C 32 06       0330            .WORD RTHAND-1 ; STATUS
062E 34 06       0340            .WORD NOFUNC-1 ; SPECIAL
0630 40 33 06    0350            JMP RTHAND ; INICJALIZACJA
                 0360 ;
0633 A0 01       0380 RTHAND     LDY #1 ; FUNKCJA I/O WYKONANA
0635 60          0400 NOFUNC     RTS ; FUNKCJA NIEDOZWOLONA
```

Rys. 8-15. Sterownik zerowy

- sterowniki urządzeń peryferyjnych

Sterowniki urządzeń peryferyjnych mogą być podzielone na rezydencyjne i nierezydencyjne. Sterowniki rezydencyjne obecne są w systemie operacyjnym ROM i mogą być wywoływane przez CTO, jako że każdy sterownik posiada swój wektor w HATABS. Sterowniki nierezydencyjne muszą być najpierw wpisane do pamięci RAM, a ich wektory winny znaleźć się w HATABS - zanim będą one wywoływane poprzez CIO. Do sterowników rezydencyjnych należą:

(E:) edytor ekranowy
(S:) Ekran
(K:) Klawiatura

(P:) Drukarka
(C:) Kaseta

Chociaż sterownika nierezydencyjne nie są obecne w OS ROM, mogą one być dodane do zestawu sterowników rezydencyjnych podczas procedur inicjalizacyjnych - startu zimnego lub gorącego. Sterownik urządzenia może być wpisany do pamięci nawet już w trakcie wykonywania programu użytkownika. Rys. 8-15 zawiera przykład sposobu wpisywania nowych sterowników do pamięci.

Sterowniki urządzeń peryferyjnych wykorzystują parametry sterujące, przenoszone przez CIO z ZIOCB. Dane w ZIOCB pozwalają na wykonane takich funkcji I/O jak OPEN, CLOSE, PUT oraz GET. Jednakże nie wszystkie urządzenia peryferyjne zezwalają na wykonanie pełnego zestawu funkcji – np. próba wykonania funkcji PUT (UMIEŚĆ SYMBOL)w stosunku do klawiatury da w rezultacie "Error 146 - function not implemented" (funkcja niedozwolona). W rozdziale 5 "Operating System Manual" umieszczono listę wszystkich funkcji, które mogą być realizowane przez poszczególna sterowniki urządzeń, jak też i szczegółowe informacje na temat wykonywania określonych funkcji przez sterowniki urządzeń peryferyjnych.

- Procedura szeregowego I/O

SIO zarządza komunikacją pomiędzy odpowiednimi sterownikami urządzeń peryferyjnych w komputerze a dołączonymi peryferiami za pośrednictwem szyny szeregowej. Po wezwaniu SIO komunikacja przebiega poprzez DCB. Parametry sterujące I/O zapisane w DCB SIO wykorzystuje do wysyłania oraz odbierania komend i danych za pośrednictwem szyny szeregowej. Sekwencja wywołująca SIO ma postać następującą:

```
                ;PARAMETRY I/O MUSZĄ BYC JUŻ ZAPISANE W DCB
     JSR SIOV   ;ODCZYTAJ WEKTOR SYSTEMOWY SIO
     BNI ERROR  ;BIT N JEST WSKAŹNIKEM BŁĘDU OPERACJI I/O
```

DCB zawiera informacje sterujące I/O dla SIO i musi być zapisany jeszcze przed wywołaniem SIO. Rys. 8-11 przedstawia parametry sterujące DCB dla kilku powszechniej stosowanych operacji I/O.

Przesłanie odpowiednich komend do SIO wymaga zrozumienia struktury DCB, która szczegółowo została przedstawiona w rozdziale 9 "Operating System Manual". Rys. 8-16 przedstawia prosty program w języku assemblera, przesyłający jedną linię danych do drukarki poprzez ustalenie parametrów sterujących DCB oraz wywołanie SIO.

```
0000            0005       *= $3000
                0010 ; PROGRAM PRZESYŁA LINIĘ DANYCH DO DRUKARKI
                0015 ; POPRZEZ WYWOŁANIE SIO
E459            0020 SIOV         = $E459     ; WEKTOR SIO
009B            0030 CR           = $9B       ; EOL
0040            0040 PRINTID      = $40       ; ID SZYNY SZEREG. DRUK.
```

```
004E                0045 MODE         = $4E       ; NORMALNY TRYB DRUKARKI
001C                0050  PTIMOT      = $001C     ; ADRES LCZNIKA CZASU
0300                0060  DDEVIC      = 300       ; IO SZYNY SZEREG. URZ.
0301                0070  DUNIT       = $301      ; NUMER URZĄDZENIA
0302                0080  DCOMND      = $302      ; KOMENDA SIO
0303                0090  DSTATS      = $303      ; KIER. PRZESYŁU DANYCH
0304                0100  DBUFLO      = $304      ; LSB ADRESU BUFORU
0305                0110  DBUFHI      = $305      ; MSB ADRESU BUFORU
0306                0120  DTIMLO      = $306      ; RZEKROCZENIE CZASU SIO
0307                0130  DTIMHI      = $307
0308                0140  DBYTLO      = $308      ; DŁUGOŚĆ BUFORU
0309                0150  DBYTHI      = $309
030A                0160  DAUX1       = $30A      ; AUX1 - RODZAJ DRUKU
030E                0170  DAUX2 =     = $303      ; AUX2 - NIEWYKORZYSTANY
                    0180  ;
3000  45 58 41      0190  MESS         .BYTE ; "PRZYKŁAD 12", CR
3001  4D 50 4C
3005  45 20 31
3009  32 98
                    0200  ;
300B  A9 40         0220               LDA #PRINTID; ZAPIS ID SZYNY
300D  8D 00 03      0230               STA   DDEVIC
3010  A9 01         0240               LDA   #1    ; NUMER URZĄDZENIA
3012  8D 01 03      0250               STA   DUNIT
3015  A9 4E         0260               LDA   #MODE ; NORMALNY TRYB DRUKARKI
3017  8D 0A 03      0270               STA   DAUX1
301A  A9 01         0280               LDA   #1    ; NIEWYKORZYSTANY
301C  8D 0B 03      0290               STA   DAUX2
301E  8D 07 03      0300               STA   DTIMHI; LIMIT CZASU <256 SEK.
3022  A5 1C         0310               LDA   PTIMOT; ZAP. PRZEKR. CZASU SIO
3024  8D 06 03      0320               STA   DTIMLO
3027  A9 00         0330               LDA   #MESS&255
3029  8D 04 03      0340               STA   DBUFLO; ZAPIS MESS JAKO BUFORU
302C  A9 30         0350               LDA   #MESS/256
302E  8D 05 03      0360               STA   DBUFHI
3031  A9 80         0370              LDA    #$80   ; KIER. PRZESYŁU DANYCH
3033  8D 03 03      0380               STA   DSTATS; Z KOMPUTERA DO URZĄDZ.
3036  A9 57         0390               LDA   #"W'" ; SIO KOMENDA WRITE
3038 8D 02 03       0400               StA   DCOMND
303B  20 59 E4      0410               JSR   SIOV  ; WYWOŁANIE SIO
303E  30 01         0420               BMI   ERROR
3040  00            0430  GOOD         BRK
3041  00            0440  ERRORR       BRK
```

Rys. 8-16. wywołanie SIO do wysłania linii danych do drukarki.

- Przerwania SIO

Komunikacja z urządzeniami peryferyjnymi poprzez szynę szeregową sterowana jest przez SIO za pomocą trzech przerwań IRQ:

IRQ	Adres, długość	Funkcja
VSERIR	$020A, 2	Gotowość czytania danych
VSEROR	$020C, 2	Gotowość przesyłania danych
VSEROC	$020E, 2	Zakończenie transmisji

Wykonywanie wszelkiego typu programów jest wstrzymywane podczas komunikowania się z peryferiami szyną szeregową przez SIO. Nawet jeżeli nie zachodzi żaden nadzwyczajny przypadek, operacje I/O zawsze wykonywane są jako przerwanie. Metoda komunikacji pomiędzy SIO a sterownikiem przerwań zwana jest metodą semafora. Wykonywanie programu głównego jest zatrzymywane poprzez wprowadzenie pustej pętli, a system w tym czasie oczekuje na sygnał zakończenia transmisji ze sterownika przerwań. Na przykład, w trakcie operacji wyjścia, SIO umieszcza bajt, który ma być przesłany, w wyjściowym szeregowym rejestrze przesuwnym, znajdującym się w układzie POKEY. Następnie procedura zapętla się, sprawdzając przy każdym powrocie wartość wskaźnika, który zostanie zapisany po wykonaniu żądanej operacji I/O.

W tym czasie POKEY przesyła bity informacji poprzez szynę szeregową. Po przesłaniu całego bajta, generowany jest sygnał żądania IRQ gotowości przesłania danych. IRO, poprzez wektor, wywołuje procedurę, która przepisuje następny bajt z buforu wyjściowego do wyjściowego rejestru przesuwnego. Proces ten trwa do momentu przesłania kompletnej zawartości buforu wyjściowego. Po sprawdzeniu bajta sterującego wykonania transmisji (checksum), sterownik przerwania zapisuje rejestr flagowy zakończenia transmisji. Przez cały ten czas SIO wykonywał zapętloną procedurę, sprawdzając jedynie zawartość danego rejestru flagowego. Po odczytaniu sygnału zakończenia transmisji SIO powraca do programu wywołującego.

Wykonanie operacji wejścia przez SIO przebiega podobnie. POKEY generuje sygnał żądania IRQ (VSERIR), który informuje SIO, że do szeregowego wejściowego rejestru przesuwnego zapisany został bajt informacji. Sterownik przerwania VSERIR przepisuje bit do buforu wejściowego, sprawdzając jednocześnie, czy bufor zapisany został do końca. Jeżeli tak, zapisany zostaje rejestr flagowy zakończenia czytania danych.

Z pewnością wadą tego procesu jest strata czasu, która ma miejsce wówczas, gdy SIO oczekuje na sygnał zakończenia transmisji od układu POKEY. Ponieważ wektory adresowe trzech procedur systemowych IRQ SIO zapisywane są w RAM, mogą one być wykorzystywane do tworzenia własnych procedur obsługi transmisji danych szyną szeregową. W ten właśnie sposób pracuje moduł interfejsu Atari 850. Sterownik modułu zmienia wartości wektorów IRO SIO, zastępując je wektorami adresowymi własnych procedur obsługi transmisji szeregowej danych. Pozwala to na jednoczesną kontynuację głównego programu wywołującego, podczas gdy przesyłem komend oraz danych szyną szeregową zawiaduje moduł interfejsu.

- Wywoływanie CIO z BASICu

Większość funkcji CIO (OPEN, CLOSE itd.) dostępnych jest poprzez bezpośrednie wezwania z BASICu instrukcjami OPEN, GET oraz PUT.

Jednakże BASIC nie pozwala na dostęp do jednego typu operacji I/O - pośredniego przesyłania danych, które nie są zapisywane do RAM, w ilości większej niż jeden bajt jednorazowo - czyli funkcji GET CHRS oraz PUT CHRS.

Zdolność odczytania czy przesłania całego buforu danych jest bardzo cenną możliwością. Na przykład tą metodą procedura w języku asemblera może być bezpośrednio zapisana w odpowiednim obszarze pamięci z rekordu dyskowego, czy też dane dotyczące obrazu ekranowego mogą być bezpośrednio wpisane w obszar pamięci ekranowej. Stosowaną powszechnie metodą, pozwalającą na szybkie wykonanie opisanych funkcji z poziomu BASICu, jest wprowadzenie procedury w języku maszynowym. Niestety, czasem zdarza się, że znalezienie miejsca w pamięci dla takiej procedury maszynowej stanowi poważny problem. Jednym ze stosowanych rozwiązań jest umieszczenie jej w obszarze zarezerwowanym na tablicę zmiennych tekstowych i wywołanie jej z BASICu instrukcją USR o adresie ADR(wyrażenie tekstowe). Ponieważ adres wyrażenia tekstowego w programach BASICu może ulec zmianie, wprowadzona procedura maszynowa powinna być całkowicie relokowalna. Z tego powodu niezmodyfikowane instrukcje adresowe pamięci mogą spowodować załamanie się tej procedury.

Przedstawiony na rys. 8-17 program nie korzysta z tablicy zmiennych tekstowych BASICu, lecz zapisuje procedurę na stronie 6 RAM.

W podobnym przypadku procedura maszynowa nie musi być relokowalna. Instrukcją POKE wpisuje się parametry sterujące do IOCB, a powodują one zapisanie procedury bezpośrednio do RAM pod adresem, gdzie została ona pierwotnie umieszczona. Program BASICu z rys. 8-17 może być takie wykorzystany do bezpośredniego przesłania danych z pamięci po sprecyzowaniu przez użytkownika zarówno adresów jak i długości buforu danych.

```
30 REM program ten ładuje procedurę maszynową bezpośrednio
50 REM na stronę 6 RAM z pliku dyskowego "D:TEST"
100 DIM FILE$(20), CIO$(7): CIO$="hhh*LVd"
105 REM CIO$ oznacza PLA, PLA, PLA, TAX, JMP $E456 (CIOV)
110 FILE$="D:TEST": REM  nazwa zbioru
120 CMD=7: STADR=1536: GOSUB 30000
130 IF ERROR=1 THEN ? "TRANSMISJA ZAKOŃCZONA": STOP
135 ? "ERROR # "; ERROR; " W LINII #";PEEK(186) +256*PEEK(187)
200 END
300 REM
31O REM PODPROGRAM WYWOŁUJACY CIO
320 REM
30000 REM
30001 REM
30002 REM program ten zapisuje program do pamięci bądź odczytuje
30003 REM z pamięci z poziomu BASICU poprzez otwarcie kanału
30004 REM IOCB i bezpośrednie wywołanie procedury CIO
30006 REM
30008 REM na wejściu  CMD=7 oznacza ładuj do pamięci (load)
30009 REM             CMD=11 oznacza  zapisz z pamięci (save)
```

```
30010 REM             STADR= adres startowy po załadowaniu
30011 REM             BYTES= ilość ładunku
30012 REM             IOCB= numer IOCB które ma być użyte
30013 REM             FILE$= nazwa zbioru
30014 REM na wyjściu  ERROR=0 oznacza powodzenie operacji I/O
30018 REM             ERROR=1 oznacza błąd operacji
30019 REM
30020 REM             +++ DANE IOCB +++
30022 REM
30024 IOCBX=IOCB*16: ICCOM=834+IOCBX: ICSTA=835+IOCBX
30026 ICBAL=836+IOCBX: ICBAH=837+IOCBX
30028 ICBLL=840+IOCBX: ICBLH=841+IOCBX
30029 REM
30030 AUX1=4: IF CIMD=11 THEN AUX1=8
30035 TRAP 30900: OPEN#IOCB, AUX1,0,FILE$
30040 TEMP=STADR: GOSUB 30500
30090 POKE ICBAL, LOW: POKE ICBAL, HIGH
30100 TTEMP=BYTES: GOSUB 30500
30130 POKE ICBLL,LOW: POKE ICBLL,HIGH
30140 POKE ICCOM,CMD: ERROR=USR(ADDR(CIO$),IOCBX)
30150 ERROR=PEEK(ICSTA): RETURN
30200 REM
30300 REM obliczanie młodszego i starszego bajta liczby 16-bitowej
30400 REM
30500 HIGH=(TEMP/256): LOW=INT(TEMP-HIGH*256): RETURN
30600 REM
30700 REM podprogram w wypadku wystąpienia błędu operacji I/O
30800 REM
30900 ERROR=PEEK(195)
30920 CLOSE# IOCB: RETURN
```

Rys. 8-17. Baz pośrednie wywołanie CIO z BASICu

PROGRAMOWANIE W CZASIE RZECZYWISTYM

W większości przypadków tworzenia oprogramowania możemy sobie pozwolić na luksus ignorowania zależności czasowych. Zwykle nie ma dla nas znaczenia, jak długo dany program będzie wykonywany, czy też w jaki sposób podprocedury programowe mają wykonywać precyzyjne pomiary czasu. Czasami jednak zależności czasowe odgrywają w wykonywanym programie niepoślednią rolę - w takich przypadkach stykamy się z problemem "programowania w czasie rzeczywistym". Ten typ programowania przysparza kłopotów programistom, pracującym ze wszelkiego typu komputerami. Można jednak stwierdzić, że system Atari pozwala na ominięcie wielu kłopotów spotykanych w innych niewielkich komputerach domowych. Bazowy sygnał czasowy, taktujący wszelkie operacje wewnętrznych obwodów elektronicznych, został tak dobrany, by układy sprzętowe komputera były całkowicie

zsynchronizowane z innym specyficznym sygnałem, jakim jest sygnał telewizyjny.

Aby możliwe było uzyskanie przejrzystych, czytelnych obrazów graficznych oraz efektów specjalnych, system Atari musiał stać się "niewolnikiem" rastrowego sygnału telewizyjnego. Co gorsza, w różnych krajach są używane różne "standardowe" sygnały telewizyjne. W USA standardem jest NTSC: 60 obrazów na sekundę, 262 linie poziome obrazu, 228 punktów ekranu w linii poziomej (pikseli). W rzeczywistości w systemie NTSC rysowanych jest 525 linii na ekranie, w dwóch półobrazach - w dyskusji tej możemy jednak pominąć ów fakt, jako że komputery Atari nie generują półobrazów. W niektórych krajach europejskich standardem jest system PAK 50 obrazów na sekundę, 312 poziomych linii skaningowych. Stąd więc zależności czasowe systemu muszą być odrębne i zależne od standardu telewizyjnego, z jakim komputer ma współpracować. W rozdziale 2, poświęconym układowi ANTIC i liście displejowej, zawarto obszerniejszą dyskusja na temat natury sygnału telewizyjnego. Zależności, omawiane w tym podrozdziale, dotyczą w całości współpracy z systemem NTSC.

- Synchronizacja z sygnałem telewizyjnym

Praca mikroprocesora ó502 została zsynchronizowana z sygnałem telewizyjnym dwoma technikami - zgrubnej i dokładnej synchronizacji. Synchronizacja zgrubna polega na zastosowaniu tej samej częstotliwości, która synchronizuje odbiornik telewizyjny, do wykonywania przerwań systemowych. Sygnał ten nazywany jest wygaszeniem pionowym. W telewizorze wykorzystywany on jest do wygaszenia strumienia elektronów po osiągnięciu prawego dolnego rogu ekranu i przeniesienia go w lewy górny róg ekranu, skąd rozpocznie się rysowanie następnego obrazu (lub półobrazu). Ten sam sygnał występuje w komputerze jako sygnał żądania przerwania niemaskowalnego. Z punktu widzenia programisty oznacza to występowanie regularnego, cyklicznego przerwania, które może zostać wykorzystane dosłownie do wszystkiego - od generacji dźwięku, poprzez taktowanie zegarów, do tworzenia programów działających wielotorowo. Szczegółowa dyskusja na ten temat przedstawiona została w podrozdziale o przerwaniach wygaszenia pionowego.

Jeszcze dokładniejsza synchronizacja operacji wykonywanych przez 6502 z sygnałem telewizyjnym została osiągnięta poprzez taktowanie systemu sygnałem o częstotliwości 1,79 MHz. Objawia się to bezpośrednim uzależnieniem czasu wykonywania pojedynczej instrukcji przez 6502 od czasu przebycia określonej drogi na ekranie kineskopu przez ruchomy strumień elektronów. Dla przykładu, w tym samym czasie, gdy 6502 wykonuje najkrótszą, 2-cykliczną instrukcję, strumień elektronów rysujący linię poziomą przemieszcza się o 4 piksele, co jest odpowiednikiem jednego punktu rastrowego ekranu w trybie graficznym 0 systemu operacyjnego. Ta precyzyjna korelacja czasowa pozwala zaawansowanym programistom na uzyskiwanie niezwykłych efektów ekranowych w trakcie rysowania poziomej linii skaningowej na ekranie. W takich przypadkach należy jednak zachować szczególną ostrożność. Odświeżanie pamięci ekranowej przez ANTIC, którego czas zależny jest od wybranego trybu graficznego a także i od innych czynników, nie pozwala na wykonywanie ścisłych obliczeń zależności czasowych. W praktyce oznacza to, że wprowadzenie

specjalnych efektów ekranowych, związane z precyzyjnymi obliczeniami czasu, musi się zwykle odbywać metodami testowania projektowanego programu.

- Liczniki sprzętowe

W układzie POKEY znajdują się cztery liczniki, odliczające do zera. Wykorzystywane są one jako programowalne, sprzętowe odnośniki czasowe. Najbardziej powszechnym ich zastosowaniem jest użycie liczników w połączeniu z syntezatorem dźwięku do wytwarzania specjalnych efektów dźwiękowych (patrz rozdział 7). Mogą one być także wykorzystane bezpośrednio do odliczania czasu w trakcie generowania przerwań IRQ (patrz podrozdział dotyczący struktury obsługi przerwań). Każdy licznik związany jest z rejestrem częstotliwości, w którym OS zapisuje wartość inicjalizacyjną danego licznika. Wartość ta zapisywana jest do licznika i odliczanie do zera rozpoczyna się w momencie zapisania sprzętowego rejestru STIMER. Gdy w rejestrze licznika pojawi się wartość 0 - koniec odliczania - wygenerowany zostaje sygnał żądania przerwania IRQ. Należy zwrócić uwagę, iż jedynie liczniki 1, 2 i 4 posiadają wektory własnych przerwań IRQ.

W celu wykorzystania danego licznika sprzętowego należy wykonać następujące czynności: ,

1. Wpisać do rejestru AUDCTL (SD208) odpowiednią wartość, sterującą częstotliwością pracy licznika sprzętowego.
2. Natężenie dźwięku, generowanego przez kanał dźwiękowy związany z danym licznikiem, ustawić na 0 - AUDC1, AUDC2, AUDC4 ($D201, $D203, $D207) .
3. Do rejestrów AUDF1, AUDF2, AUDF4 ($D200, $D202, $D206) wpisać żądaną wartość, określającą ilość taktów, które dany licznik ma odmierzyć.
4. Wpisać procedurę obsługi przerwań IRQ licznika sprzętowego.
5. Zmienić wartość wektora obsługi przerwania licznika adresem własnej procedury - VTIMR1, VTIMR2, VTIMR4 ($0210, $0212, $0214). Uwaga - w pierwotnej wersji "A" systemu operacyjnego znalazł się błąd, który powoduje, że obsługa przerwania IRQ licznika 4 nie odbywa się poprzez wektor VTIMR4. Błąd ten usunięty został w wersji "B" OS.
6. Włączyć bity 0,1 lub 2 w rejestrach IRQEN i jego cieniu OS POKMSK (odpowiednio $D20E i $0010) celem zezwolenia na wykonanie przerwań odpowiednio liczników 1, 2 lub 4.
7. Zapisać rejestr STIMER ($D209) dowolną wartością, co spowoduje uruchomienie danego 1icznika.

Jedyną komplikacją w pracy z tymi licznikami może być fakt, iż odpowiedź 6502 na wezwanie przerwania IRQ nie zawsze ze musi być natychmiastowa - wiąże się to z procesami odświeżania pamięci ekranowej przez ANTIC oraz wykonywaniem przez 6502 procedur obsługi przerwań listy displejowej, a także wygaszenia pionowego.

- Liczniki programowe

Wartość wszystkich liczników programowych zmniejszana jest o 1 podczas każdego procesu wygaszenia pionowego. Jeżeli procedury

systemowe obsługi przerwania VBLANK zostaną bądź wstrzymane bądź zastąpione innymi procedurami, praca liczników zostanie wstrzymana.

System wyposażony jest w 6 liczników programowych:

Nazwa	Adres, długość	Wektor lub rejestr flagowy
RTCLOCK	$0012, 3	BRAK
CDTMV1	$0218, 2	CDTMA1 ($0226,2)
CDTMV2	$021A, 2	CDTMA2 ($0228,2)
CDTMV3	$021C, 2	CDTMA3 ($022A,1)
CDTMV4	$021C, 2	CDTMA4 ($022C,1)
CDTMV5	$0220, 2	CDTMA5 ($022E,1)

Zegar czasu rzeczywistego (RTCLK) oraz licznik systemowy 1 (CDTMV1) przepisywane są podczas etapu 1, natychmiastowego przerwania VBLANK. RTCLK jest licznikiem 3-bajtowym, liczącym od zera; kiedy RTCLK osiąga swą maksymalną wartość (16777216), zostaje wyzerowany. Na rys. 8-18 pokazano przykład wykorzystania rejestrów RTCLK jako zegara czasu rzeczywistego.

Ponieważ przepisywanie liczników programowych następuje w trakcie procedur VBLANK, zapis właściwych wartości inicjalizacyjnych liczników musi być wykonany z właściwą ostrożnością. Wykorzystuje się w tym celu systemową procedurę SETVBV ($E45C), wywoływaną w sposób następujący:

```
     rejestr    X - starszy bajt wartości początkowej licznika
                Y - młodszy bajt
                A - numer licznika (1-5)
przykład:
     LDA #1 ; wybór licznika nr 1
     LDY #0
     LDY #2 ; wartość pocz. = $200 okresów VBLANK
     JSR SETVBV ; wywołanie procedury zapisu liczników
```

Liczniki systemowe 1-5 są licznikami 2-bajtowymi. Zapisania wartości początkowej dokonuje się przy pomocy procedury SETVBV. OS następnie podczas każdego wygaszenia pionowego zmniejsza zawartość każdego licznika o 1. Licznik 1 przepisywany jest podczas etapu natychmiastowej procedury VBLANK, liczniki 2-5 przepisywane są podczas etapu 2 natychmiastowej procedury VBLANK. W zależności od tego, którego licznika wartość została odliczona do zera, OS podejmuje różne działania.

Liczniki systemowe 1 i 2 posiadają związane z nimi wektory RAM.

```
1 POKE 752,1
3 ? " ×": REM kasowanie ekranu
4 ? "GODZINA ? ":INPUT HOUR: ? "MINUTA ? ";: INPUT MIN: ? "SEKUNDA ?
";: INPUT SEC
5 CMD=1: GOSU9 45
6 ? " ×";HOUR; " : ";MIN;" : ";SEC:? " ":? " "
7 CMD=2: GOSUB 45
```

```
9 ? " ";HOUR;" : ";MIN;" : ";SEC;”  „ 7
10 REM jest to demonstracja działania zegara czasu rzeczywistego
20 REM program akceptuje wartości początkowe godzin minut i sekund
30 REM w systemie 24-godzinnym, zeruje rejestr RTCLOK, po czym
40 REM aktualną wartość RTCLOK dodaje do wartości początkowych,
42 REM wyświetlając czas aktualny
45 HIGH=1536: MED=1537: LOW=1538
50 REM
60 REM +++++ PUNKT WEJŚCIOWY +++++
65 REM
70 ON CMD GOTO 100,200
95 REM
96 REM +++++ INICJALIZACJA ZEGARA +++++
97 REM
100 POKE 20,0: POKE 19,0: POKE 18,0
105 DIM CLOCK$(50)
106 CLOCK$=" "
110 IHOUR=HOUR: IMIN=MIN: ISEC=SEC: RETURN
197 REM
198 REM +++++ CZYTANIE ZEGARA +++++
199 REM
200 REM
201 A=USR(ADR(CLOCK$))
210 TIME=((((PEEK(HIGH)*256)+PEEK(MED)*256+PEEK(LOW))/ 59.923334
220 HOUR=INT(TIME/3600): TIME=TIME-(HOUR*3600)
230 MIN=INT(TIME/60): SEC=INT(TIME-(MIN*60))
235 SEC=SEC+ISEC: IF SEC>60 THEN SEC=SEC-60: MIN=MIN+1
236 MIN=MIN+IMIN: IF MIN>60 THEN MIN=MIN-60: HOUR=HOUR+1
237 HOUR=HOUR+IHOUR
240 HOUR=HOUR-(INT(HOUR/24))*24
250 RETURN
300 FOR J=1 TO 38: READ Z: CLOCK$(J,J)=CHR$(Z): NEXT J: RETURN
310 DATA 104,165,18,141,0,6,165,19,141,1,6,165
320 DATA 20,141,2,6,165,18,205,0,6,208,234
330 BATA 165,19,205,1,6,208,227,165,20,205,2,6,208,220,96
```

Rys. 8-18. Program zegara czasu rzeczywistego

Kiedy wartość licznika 1 lub 2 osiąga 0, OS wykonuje instrukcję JSR poprzez wektor odpowiedniego licznika. Wektory obu tych liczników przedstawione zostały w tablicy na rys. 8-8.

Liczniki systemowe 3-5 posiadają odpowiadające im rejestry flagowe, które normalnie zapisane są wartościami różnymi od zera. Kiedy wartość jednego z tych liczników osiąga zero, OS zeruje odpowiadający danemu licznikowi rejestr flagowy, Tak więc wykorzystanie tych liczników polega na testowaniu wartości odpowiednich rejestrów flagowych.

Liczniki programowe 1-5 są licznikami ogólnego zastosowania i nogą być wykorzystywane w wielu różnych programach aplikacyjnych. Na

przykład licznik 1 wykorzystywany jest przez SIO do odmierzania czasu operacji szeregowego przesyłania danych. Jeżeli wartość licznika osiągnie zero zanim dana operacja zostanie wykonana, powstaję błąd przekroczenia czasu. Licznik ten zapisywany jest różnymi wartościami początkowymi, w zależności od tego, z jakim urządzeniem peryferyjnym komunikuje się SIO. Oznacza to, że jakkolwiek urządzenie posiada wiele czasu na odpowiedź na żądanie przeprowadzenia danej operacji I/O, to jednak komputer nie będzie czekał w nieskończoność na odpowiedź od nieistniejącego urządzenia, Sterownik magnetofonu kasetowego wykorzystuje z kolei licznik 3 do ustalenia długości czasu odczytu i zapisu kasetowego. Na rys. 8-19 i 8-20 pokazano przykładowe wykorzystanie licznika 2 w programie generacji uderzeń metronomu.

Generalnym zastosowaniem liczników programowych są takie przypadki, kiedy żądana przez użytkownika skala czasu jest większa niż jeden okres VBLANK. w przypadku odliczania krótszych odcinków czasu muszą być wykorzystane bądź liczniki sprzętowe, bądź jakiekolwiek inne techniki zliczające.

```
1 REM jest to program BASICu sterujący częstość uderzeń
2 REM metronomu.
3 REM
5 ? "X": REM kasowanie ekranu
10 X=10: REM wartość początkowa częstości
20 FOR J=1 TO 10: NEXT J: REM pętla opóźniająca
50 IF STICK(0)=14 THEN X=X+1: REM dżojstik do przodu - szybciej
51 IF STICK(0)=13 THEN X=X-1: REM dżojstik do tyłu - wolniej
52 IF X<1 THEN X=1: REM ograniczenie najniższej częstości
53 IF X>255 THEN X=255: REM ograniczenie najwyższej częstości
54 REM umieszczenie na ekranie częstości uderzeń
56 ? "";INT(3600/X);" UDERZEN NA MINUTE "
60 POKE 0,X: REM  wartość częstości zapisywana w rejestrze 0
70 REM tu następuje poniższa procedura maszynowa
```

Rys. 8-19. Program BASICu generacji uderzeń metronomu

```
0040                *=5600
0050  ; PROCEDURA WYKORZYSTUJE REJESTR $0000 DO ZAPISU WARTOSCI
0060  ; CZĘSTOSCI UDERZEŃ METRONOMU
0070  AUDF1      =     $D200      ;REJESTR CZĘSTOTLIWOŚCI DŹWIĘKU
0080  AUDC1      =     $D201      ;REJESTR KONTROLNY DŹWIĘKU
0090  FREQ       =     $08        ;WARTOŚĆ AUDF1
0100  VOLUME     =     $AF        ;WARTOSC AUDC1
0110  OFF        =     $A0        ;GŁOŚNOŚĆ ZEROWA
0120  SETVBV     =     $E45C      ;PROCEDURA ZAPISU LICZNIKA
0130  XITVBV     =     $E462
0140  CDTMV2     =     $021A      ;LICZNIK 2
0150  CDTMA2     =     $0228      ;REJESTR FLAGOWY LICZNIKA
0160  ZTIMER     =     $0000      ;CZĘSTOŚĆ TAKTU LICZNIKA
0170  ;
0180  START      LDA #10
```

```
0190            STA ZTIMER
0200 ; ZAPIS WEKTORA LICZNIKA
0220 ;
0230 INIT       LDA CNTINT&255
0240            STA CDTMA2
0250            LDA CNTINT/256
0260            LDA CDTMA2+1
0270 ;
0280 ; ZAPIS    WARTOŚCI LICZNIKA WEDŁUG WEKTORA
0290 ;
0300            LDY ZTIMER
0310            JSR SETIME
0320            RTS
0340 ; TU ZACZYNA SIĘ PROGRAM GENERACJI UDERZEŃ METRONOMU
0380 ; KANAL    DŹWIĘKOWY GENERUJE KRÓTKIE UDERZENIE
0390 ;
0400 CNTINT     LDA #VOLUME
0410            STA AUDC1
0420            LDA #FREQ
0430            STA AUDF1
0435            LDY #FF ;  OPÓŹNIENIE
0440 DELAY      DEY
0442            BNE DELAY
045             STY AUDC1
0460            JMP INIT
0480 ;
0490 ; PODPROGRAM ZAPISUJĄCY WARTOŚĆ POCZĄTKOWĄ LICZNIKA
0500 ;
0520 SETIME     LDX #0     ; MAX WARTOŚĆ 256 OKRESÓW VBLANK
0530            LDA #2     ; NIMER LICZNIKA
0540            JSR SETVBV
0550            RTS
0560          *=$2E2
0570            .WORD START
0580            .END
```

Rys. 8-20. Procedura maszynowa generacji uderzeń metronomu.

OBSZAR OPERACJI ZMIENNOPRZECINKOWYCH

Obszar operacji zmiennoprzecinkowych (FPP) jest zbiorem integralnych procedur, zapewniających rozszerzenie wykonywania obliczeń matematycznych przez OS. Procedury te zapisane są w wielofunkcyjnym układzie pamięci ROM, wchodzącym w skład 10K systemu operacyjnego Atari 400/800. Obszar ten zajmuje pamięć o adresach od $D800 do $DFFF. Nie uległ on zmianie przy wprowadzeniu nowej wersji "B" systemu operacyjnego. Poniższy rozdział omawia wewnętrzny zapis liczb, dostępne procedury systemowe oraz ich odpowiednie sekwencje

wywołujące. Załączony przykład w języku assemblera ilustruje wykorzystanie FPP w programach użytkownika.

- Wewnętrzni zapis liczb

FPP zapisuje wszystkie liczby w postaci 6-bajtowej. Zapis taki obejmuje 1-bajtową wartość eksponenta oraz 5-bajtową mantysę w zapisie binarnego kodu liczb dziesiętnych (BCD), Zapis taki został wybrany w celu minimalizacji błędów zaokrąglenia, które mogą wystąpić w niektórych obliczeniach arytmetycznych.

Bit znakowy bajta reprezentującego eksponent odpowiada znakowi mantysy - bit równy 0, liczba dodatnia, bit równy 1, liczba ujemna. Najmniej znaczący, bit 7 bajta reprezentującego eksponent zapisuje" wartość eksponenta jako wielokrotność 100, według zapisu "rozszerzonej notacji 64". W notacji tej wartość 64 dodawana jest do wartości eksponenta zanim zostanie on zapisany w bajcie eksponenta. Pozwala to na wprowadzenie pełnej skali eksponentów, tak ujemnych jak i dodatnich, bez konieczności odczytywania dodatkowego bitu znaku eksponenta.

Tak znormalizowany zapis mantysy oznacza, że najbardziej znaczący bajt zapisu liczby nigdy nie jest równy 0. Ponieważ mantysa zapisywana jest w formacie BCD, a eksponent zapisany jest jako wielokrotność 100, a nie potęga 10, pozwala to na osiągnięcie dokładności do 9 lub 10 cyfr znaczących. W zapisie mantysy po pierwszym bajcie OS automatycznie umieszcza przecinek, a więc eksponent mniejszy od 64 ($40) oznacza liczbę mniejszą od 1.

A oto przykłady wewnętrznego zapisu liczb (wartości formatów w systemie heksadecymalnym):

liczba: 0,02 = $2*100^{-1}$
format: 3F 02 00 00 00 00 (eksponent: 40-1)

liczba: -0.02 = $-2*100^{-1}$
format: 02 00 00 00 00 (eksponent: 80+40-1)

liczba: 37.0 = $37*100^{0}$
format: 40 37 00 00 00 00 (eksponent: 40+0)

liczba: -460312 = $-46.0312*100^{2}$
format: C2 46 03 12 00 00 (eksponent: 80+40+2)

Specjalnym zapisem jest zapis liczby 0 – zarówno eksponent jak i mantysa zapisane są wartościami 0. Szukanie liczby równej 0 może się więc ograniczyć do testowania dwóch bajtów w zapisanej liczbie - bajta eksponenta oraz pierwszego bajta mantysy.

Dynamiczny zakres liczb, które mogą być zapisane według przedstawionego schematu, wynosi od 10^{-98} do 10^{+98}.

- Wykorzystanie pamięci

FPP wykorzystuje do swego użytku dwa obszary pamięci. Są to:

$00D4 - $00FF na stronie zerowej

$057E - $05FF na stronie piąte j

Oba te obszary używane są do przechowywania parametrów kontrolnych oraz jako różnorodne rejestry FPP. Dwa pseudo-rejestry o pierwszorzędnym znaczeniu nazywane są FP0 oraz FP1 (adresy $00D4 - $00D9 i $00E0 - $00E5). Każdy z tych pseudo-rejestrów ma długość 6 bajtów i służy do przechowania liczby zapisanej według wewnętrznego schematu. Wektor 2-bajtowy wykorzystywany jest do zapisywania adresu procedury zmiennoprzecinkowej. Nazywany jest on FLPTR i mieści się pod adresem $00FC.

Podczas stosowania zmiennych tekstowych oraz konwersji liczb FPP na zmienne tekstowe ATASCII muszą być zarezerwowane obszary buforowe. Bufor wyjściowy o nazwie LBUFF jest 128-bajtowym blokiem w obszarze $0580 - S05FF. Bufor wejściowy ustalany jest przez 2-bajtowy wskaźnik INBUFF ($00F3). Offsetem dla buforu wejściowego ustalanego przez INBUFF jest 1-bajtowy indeks CIX, mieszczący się pod adresem $00F2.

Typowa sekwencja, wywołująca FPP z programu maszynowego, może przedstawiać się następująco. Po pierwsze zmienna tekstowa ATASCII, reprezentująca jedną z liczb, które mają być wykorzystane w procedurach obliczeń matematycznych, musi być zapisana w buforze w dowolnej części pamięci RAM. Następnie rejestr wskaźnikowy INBUFF musi być zapisany adresem początku danej zmiennej, natomiast wartość indeksu CIX musi zostać ustalona na 0. W ten sposób liczba została przygotowana do konwersji w zapisie wewnętrzny w FPP, a więc może już być wywołana procedura AFP. W jej wyniku liczba w zapisie FPP umieszczona zostaje w FR0, skąd może być wykorzystana przez dowolną procedurę FPP. Po wykonaniu odpowiednich operacji matematycznych wynik ponownie zostaje zapisany jako FPP w FR0. Stąd, poprzez wywołanie procedury FASC, może on być przekształcony w zmienną tekstową ATASCII i umieszczony w LBUFF. Program z rys. 8-22 daje przykład powyższego procesu.

W celu wykorzystania przez FPP wartości 16-bitowych, należy umieścić dwa bajty liczby w dwóch pierwszych adresach FR0 ($D4 i $D5), po czym wykonać JSR IFP, która to procedura wykona konwersję do zapisu wewnętrznego FPP, umieszczając wynik w FR0. Procedura FPI wykonuje konwersję w drugą stronę.

Tablica umieszczona na następnej stronie przedstawia dostępne funkcje FPP, ich adresy w ROM, zapisywane pseudo-rejestry oraz przybliżony czas wykonywania procedur.

PROCEDURY FPP

Nazwa	Adres	Funkcja	Wynik	Maksym. czas (µsek)
AFP	$D800	Konwersja ATASCII na FPP	FR0	3500
FASC	$D8E6	Konwersja FPP na ATASCII	LBUFF	950
IFP	$D9AA	Konwersja liczby na FPP	FR0	1330
FPI	$D9D2	Konwersja FPP na liczbę	FR0	2400
FSUB	$DA60	Odejmowanie FR0-FR1	FR0	740
FADD	$DA66	Dodawanie FR0+FR1	FR0	710
FMUL	$DAD8	Mnożenie FR0*FR1	FR0	12000
FDIV	$DB28	Dzielenie FR0/FR1	FR0	10000
FLDOR	$DD89	Ładowanie FPP do rejestrów X, Y	FR0	70
FLDOP	$DD8D	Ładowanie FPP przez FLPTR	FR0	60
FLD1R	$DD98	Ładowanie FPP do rejestrów X, Y	FR1	70
FLD1P	$DD9C	Ładowanie FPP przez FLPTR	FR1	60
FSTOR	$DDA7	Zapis FPP z rejestrów X, Y	FR0	70
FSTOP	$DDA8	Zapis FPP przez FLPTR	FR0	70
FMOVE	$DDB6	Przeniesienie FR0	FR1	60
PLYEVL	$DD40	Losowanie przypadkowe	FR0	88300
EXP	$DDC0	Potęgowanie – e^{FR0}	FR0	115900
EXP10	$DDCC	Potęgowanie – 10^{FR0}	FR0	108800
LOG	$DECD	Logarytm naturalny	FR0	136000
LOG10	$DED1	Logarytm dziesiętny	FR0	125400
ZFR0	$DA44	Zerowanie	FR0	80
AF1	$DA46	Zerowanie rejestru w rejestrze X	-	80

Rys. 8-21. Procedury FPP.

```
0000            0020          * = 54000 ; ARBITRALNY ADRES STARTOWY
DDB6            0030 FMOVE      = $DDb6
DA60            0040 FSUB       = $dA60
0482            0050 FTEMP      = $0482
DDA7            0060 FSTOR      = $DDA7
D8E6            0070 FASC       = $d8E6
00F3            0080 INBUFF     = $00F3
D800            0085 AFP        = $D800
00F2            0090 CIX        = $00F2
0580            0100 LBUFF      = $0580
009b            0120 CR         = $009B
0009            0130 PUTREC     = $0009
```

```
0005            0140 GETREC     = $0005
E456            0150 CIOV       = $E456
0342            0160 ICCM       = $0342
0344            0170 ICBAL      = $0344
0348            0180 ICBLL      = $0348
                0190 ;
                0200 ; DEMONSTRACJA DZIAŁANIA FPP
                0210 ; CZYTA DANE LICZBY Z EDYTORA EKRANOWEGO
                0215 ; PRZEPROWADZA KONWERSJĘ W ZAPIS FPP
                0220 ; ODEJMUJE PIERWSZĄ LICZBĘ OD DRUGIEJ
                0225 ; WYNIK UMIESZCZA W FTEMP, KTÓRY JEST
                0230 ; REJESTREM FPP, PO CZYM WYŚWIETLA
                0240 ; REZULTAT
4000 20 53 40   0260 START      JSR GETNUM ; CZYTA PIERWSZĄ LICZBĘ
4003 20 B6 DD   0270           JSR FMOVE  ; PRZENOSI Z FR0 DO FR1
4006 20 53 40   0280           JSR GETNUM ; CZYTA DRUGĄ LICZBĘ
4009 20 60 DA   0290           JSR FSUB   ; FR0-FR1 DO FR0
400C 90 0A      0300           BCC NOERR  ; SKOK JESLI BEZ BŁĘDU
400E A9 81      0340           LDA #ERRMSG&255; PODAJ KOD BŁĘDU
4010 8D 44 03   0350           STA ICBAL
4013 A9 40      0360           LDA #ERRMSG/256
4015 40 39 40   0370           JMP CONTIN
4018 A2 82      0390 NOERR     LDX #FTEMP&255;ZAPIS WYNIKU W FTEMP
401A A0 04      0400           LDY #FTEMP/256
401C 20 A7 DD   0410           JSR FSTOR
                0420 ;
                0430 ; KONWERSJA LICZBY DO TEKSTOWEJ ATASCII
                0440 ; SZUKANIE KOŃCA ZIENNEJ TEKSTOWEJ
                0445 ; ZAMIANA LICZBY UJEMNEJ W DODATNIĄ
                0450 ; ORAZ DODANIE POŻYCZKI
401F 20 E6 D8   0470           JSR FASC ; KONWERSJA W ATASCII
4022 A0 FF      0480           LDY #$FF
4024 C8         0490 MLOOP     INY
4025 B1 F3      0500           LDA (INBUFF),Y; ŁADUJ NASTĘPNY BAJT
4027 10 FB      0510           BPL MLOOP ; GDY DODATNI - KONTYNUUJ
4029 29 7E      0520           AND #$7F ; JEŚLI NIE MASKUJ MSBIT
402B 91 F3      0530           STA (INBUFF),Y
402D C8         0540           INY
402E A9 9B      0550           LDA #CR ; ZAPISZ POŻYCZKĘ
4030 91 F3      0550           STA (INBUFF),Y
                0570 ;
                0580 ; WYŚWIETLENIE REZULTATU
4032 A5 F3      0600           LDA INBUFF ; ŁADUJDUJ ADRES BUFORU
4034 8D 44 03   0610           STA ICBAL
4037 A5 F4      0620           LDA INBUFF+1
4039 8D 45 03   0630 CONTIN    STA ICBAL+1
4030 A9 09      0640           LDA #PUTREC ; KOMENDA PUT RECORD
403E 8D 42 03   0650           STA ICCOM
```

```
4041 A9 28      0660            LDA #$40 ; DŁUGOSC BUFORU $40
4043 8D 48 03   0670            STA ICBLL
4046 A9 00      0690            L7A #$00
4048 8D 49 03   0700            STA ICBLL+1
4048 A2 00      0710            LDX #$00 ; IOCB #0 EDYTORA EKRAN.
404D 20 56 E4   0720            JSR CIOV ; WYWOŁANIE CIO
4050 4C 00 40   0730            JMP START ; POWTÓRZ OPERACJĘ
                0750 ; CZYTANIE ZMIENNEJ TEKSTOWEJ ATASCII Z E:
                0755 ; KONWERSJA W ZAPIS FPP I ZAPISANIE W FR0
4053 A9 05      0780 GETNUM     LDA #GETREC ; GET RECORD
4055 8D 42 03   0790            STA  ICCOM
4058 A9 80      0800            LDA #LBUFF&255;  ADRES BUFORU=LBUFF
405A 8D 44 03   0810            STA ICBAL
405D A9 05      0820            LDA #LBUFF/256  _
405E 8D 45 03   0830            STA ICBAL+1
4062 A928       0840            LDA #$40 ; DŁUGOŚĆ BUFORU $40
4064 8D 48 03   0850            STA ICBLL
4067 A9 00      0860            LDA #$00
4069 8D 49 03   0870            STA ICBLL+1
406C A2 00      0880            LDX #$00 ; IOCB #0 EDYTORA EKRANOW.
406E 20 56 E4   0890            JSR CIOV ; WYWOŁANIE CIO
4071 A9 80      0900            LDA #LBUFF&255;  ADRES BUFORU=INBUFF
4073 85 F3      0910            STA INBUFF
4075 A9 05      0920            LDA #LBUFF/256
4077 85 F4      0930            STA INBUFF+1
4079 A9 00      0940            LDA #$00 ; INDEKS BUFORU = 0
407B 85 F2      0950            STA CIX
407D 20 00 D8   0950            JSR AFP ;  KONWERSJA ATASCII W FPP
4080 60         0970            RTS
4081 45         0980            .BYTE "ERROR",CR
4082 52
4083 52
4084 4F
4085 52
4086 93
                1000 ;     START PROGRAMU
4087            1020            *=    $2E0
02E0 00 40      1030            .WORD START
02F2            1040            .END
```

Rys. 8-22. Przykład wykorzystania procedur FPP.

9. DYSKOWY SYSTEM OPERACYJNY

Dyskowy System Operacyjny (DOS) jest rozszerzeniem systemu operacyjnego komputerów domowych Atari 400/800, który pozwala na komunikowanie się komputera ze stacją dysków w taki sam sposób, jak z innymi urządzeniami peryferyjnymi. DOS składa się z trzech podstawowych części: rezydencyjnego sterownika dyskowego, systemu gospodarowania plikami (File Management System – FMS) oraz zbioru procedur sterowania stacją dysków (Disk Utility Package – DUP). Rezydencyjny sterownik dysku jest jedyną częścią DOS, która znajduje się w pamięci R0M systemu operacyjnego. Zarówno FMS jak i DUP zapisane są na dyskietce, z czego FMS zostaje przepisany do pamięci RAM komputera ("zabootowany") z dyskietki w chwili włączenie zasilania i wykonywania procedur inicjalizacyjnych. DUP nie jest automatycznie przepisywany do pamięci, 1ecz wymaga specjalnego wywołania przez użytkownika lub program aplikacyjny. Jeżeli na przykład podczas inicjalizacji uruchamiany jest cartridge BASICu, DUP przepisuje się do pamięci RAM instrukcją DOS z BASICu. Rozdział niniejszy omawia szczegółowo wszystkie części składowe DOS oraz przedstawia pewne techniki, które powinny być pomocne przy efektywnym wykorzystywaniu wszystkich możliwości, jakie oferuje system dyskowy Atari. Należy zwrócić uwagę, że wszystkie dane odnoszą się do systemu DOS II wersja 2.OS, który dość nieznacznie różni się od innych wersji systemu DOS II, natomiast w znaczący sposób różni się od wcześniejszych wersji systemu DOS I Atari. Zainteresowanych szczegółami technicznymi DOSu odsyłamy do "Operating System Manual" oraz "Disk Operating System II Manual".

- Rezydencyjny Sterownik Dysku

Rezydencyjny sterownik dysku jest najprostszą częścią DOSu. Sterownik ten nie wykorzystuje normalnej sekwencji wywołującej CIO, jaka używana jesz przez inne sterowniki urządzeń peryferyjnych, a która opisana została w rozdziale 8, w części dotyczącej centralnego podsystemu wejścia/wyjścia. W systemie DOS 2.OS rezydencyjny sterownik dysku wykorzystywany jest jedynie podczas inicjalizacyjnego procesu bootingowego. Od tego momentu komunikacja pomiędzy komputerem a stacją dysków przebiega za pośrednictwem FMS. Miejsce sterownika dysku w podsystemie I/O pokazane zostało na rys. 8-9.

Do komunikowania się ze sterownikiem dysku wykorzystywany jest blok kontrolny peryferiów (DCB). Na rys. 8-11 ukazana została struktura DCB. Sekwencja wywołująca sterownik dysku ma postać następującą:

```
                    ; parametry kontrolne DCB zostały już ustalone
    JSR DSKINV      ; wywołanie sterownika przez wektor DSKINV
    BPL OKAY        ; wykonaj skok gdy nie ma błądu, rej. Y=1
                    ; jeśli błąd, rej. Y zawiera status błędu
                    ; kod błędu zapisany jest także w rejestrze ;
                    ; DC9STA
```

Sterownik dysku fest podprogramem, umożliwiającym fizyczny transfer danych pomiędzy mikroprocesorem 6502 w komputerze, a innym mikroprocesorem, zawiadującym stacją dysków. Dane przesyłane są szeregową szyną wejścia/wyjścia. Rezydencyjny sterownik dysku, znajdujący się w OS, może wykonać cztery funkcje:

FORMAT	przesłanie komendy FORMAT do kontrolera dysku
READD SECTOR	czytane danych wyszczególnionego sektora
WRITE/VERIFY SECTOR	zapis sektora, sprawdzenie czy sektor nie jest już zapisany
STATUS	wezwanie kontrolera dysku do podania statusu

Komenda FORMAT kasuje wszelkie zapisy na dyskietce, po czym formatuje ją - czyli zapisuje adresy poszczególnych sektorów w specjalnej części zapisu dyskowego. Komenda ta nie umieszcza na dyskietce żadnego pliku, nawet systemowego. Część zawierająca dane w każdym sektorze zostaje wyzerowana, po czym do Tablicy Zawartości Dysku oraz do Rejestru Plików wpisane zostają wartości inicjalizacyjne. Więcej danych na temat fizycznej struktury zapisu dyskowego można znaleźć w "Operating System Manual" oraz w podrozdziale "Dyskowy Zapis FMS" niniejszej książki.

Należy zwrócić uwagę, że wszelkie operacje I/O wykonywane są przez sterownik dysku drogą adresowania sektorów na dyskietce. Bezpośrednie komendy I/O sterownika mogą być wykorzystane do czytania danych bądź zapisywania danych w wyszczególnionym sektorze dyskietki. Przy ich pomocy można na przykład zaprojektować własną strukturę każdego pliku. W rozdziale 10 "Operating System Manual" przedstawiono świetny przykład wykorzystywania bezpośrednich komend sterownika dysku do stworzenia niestandardowego pliku bootingowego na dyskietce.

Funkcja STATUS wykorzystywana jest do określania statusu stacji dysków w systemie. Komenda ta powoduje, że ze stacji przesyłane są do komputera cztery bajty kontrolne, określające aktualny status stacji. Bajty te zapisywane są w rejestrze DVSTAT ($02EA,4). Pierwszy bajt jest bajtem statutowym komendy, a znaczenie jego poszczególnych bitów jest następujące:

bit 0 = 1 :	przekazana została niewłaściwa komenda
bit 1 = 1 :	przekazane zostały niewłaściwe dane
bit 2 = 1 :	operacja PUT zakończona niepowodzeniem
bit 3 = 1 :	dyskietka jest zabezpieczona przed zapisem
bit 4 = 1 :	stacja oczekuje na instrukcje

Drugi bajt jest sprzętowym bajtem statutowym i zawiera kopię rejestru statutowego układu 1 NS 1771-1, będącego kontrolerem stacji dysków elastycznych. Trzeci bajt określa maksymalny czas (w sekundach), jaki kontroler dysku przeznacza sterownikowi rezydencyjnemu na pomyślne wykonanie określonej operacji. Czwarty bajt jest niewykorzystany. Komenda STATUS posiada niewielkie znaczenie dla użytkownika. Ponieważ czas przeznaczony na wykonanie komendy STATUS jest znacznie krótszy od czasów przeznaczonych na wykonanie innych komend, może ona być wykorzystana do sprawdzania, czy specyficzna stacja dysków została dołączona do systemu. Jeżeli

sterownik dysku zasygnalizuje przekroczenie czasu dostępnego na wykonanie komendy, będzie to oznaczało, że staja nie jest dołączona do systemu.

- System gospodarowania plikami (FMS)

FMS jest nierezydencyjnym sterownikiem urządzenia peryferyjnego, który wykorzystuje normalną drogę komunikowania się z CIO podobnie jak inne sterowniki. FMS nie jest obecny w pamięci ROM systemu operacyjnego. Zostaje on przepisany do pamięci RAM podczas procedur inicjalizacyjnych (start zimny), o ile dyskietka zawierająca DOS znajduje się w stacji.

FMS, podobnie jak inne sterowniki peryferiów, odczytuje dane kontrolne operacji I/O z CIO, po czym wykorzystuje rezydencyjny sterownik dysku do przesyłania danych z i do dyskietki. Wprowadzenie dodatkowego sterownika dysku, jaki między innymi zawiera FMS, miało na celu uniknięcie błędu, jaki często powodowany był przez OS. Polegał on na niewłaściwym interpretowaniu przez OS 16-bitowych wskaźników buforu danych podczas operacji SIO. Błąd ten stawał się błędem systemowym, jeżeli bufor danych kończył się na granicy strony pamięci. Ponieważ rezultatem naprawienia tego błędu stało się umieszczenie sterownika dysku w pamięci RAM, pozwala to na szczególne jego wykorzystanie przez użytkownika. Na przykład układy sprzętowe stacji dysków zdolne są do wykonania jednej funkcji, której nie obejmuje rezydencyjny sterownik dysku. Funkcją tą jest WRITE SECTOR WITHOUT VERIFY - czyli zapis sektora bez sprawdzania czy jest on zapisany. Pomimo, iż wykonanie tej funkcji wiąże się z pewnym ryzykiem, zapis dyskowy wykonywany jest szybciej. Można dokonać zmiany w FMS, która na stałe wprowadzi tę funkcję zamiast komendy FORMAT/VERIFY. Z BASICu uzyskuje się to przez:

```
POKE 1913, 80
```

Aby powrócić do normalnego zapisu dyskowego należy wykonać

```
POKE 1913, 87
```

FMS wywoływany jest przez odpowiednie otwarcie kanału IOCB, a następnie wywołanie procedury CIO. FMS może wykonywać kilka dodatkowych funkcji, których CIO nie dopuszcza w przypadku innych sterowników urządzeń peryferyjnych. Są to:

FORMAT	FMS wywołuje sterownik dysku do wykonania funkcji FORMAT sterownika. Po jej pomyślnym zakończeniu. FMS zapisuje na dyskietce pewne pliki strukturalne.
NOTE	FMS odczytuje bieżącą wartość wskaźnika pliku.
POINT	FMS zapisuje wskaźnik pliku żądaną wartością.

Wykorzystanie instrukcji NOTE i POINT zostało omówione w podrozdziale pt. Random Access:

- Dyskowe operacje I/O

W przypadku dyskowych operacji I/O mogą być wykorzystywane wszystkie standardowe funkcje I/O, wykonywane poprzez wywołanie CIO. Dla BASICu oznacza to używanie instrukcji I/O takich jak: OPEN, CLOSE, GET, PUT oraz XIO. W języku asemblera zachodzi konieczność własnoręcznego ustalenia parametrów kanału IOCB i wywołania CIO.

Każda dyskowa operacja I/O wymaga wcześniejszego otwarcia zbioru na dyskietce. W BASICu wykorzystuje się do tego celu instrukcję OPEN, posiadającą format:

```
OPEN#IOCB, ICAX1, 0, ”D:MYPROG.BAS”
```

Numer IOCB określa jeden z ośmiu dostępnych kanałów IOCB (patrz rozdział 8, podrozdział omawiający CIO). Należy unikać wykorzystywania IOCB #0,6 oraz 7, ponieważ używane są one przez OS oraz BASIC do różnych celów. ICAX1 jest parametrem, kodującym typ instrukcji OPEN. Poszczególne bity tego bajta oznaczają:

```
bit:  7 6 5 4 3 2 1 0
      x x x x W R D A
```

gdzie:
- A – (Append) - Łączenie zapisów
- D – (Directory) - Wyszukiwanie zapisów
- R – (Read) - Czytanie zapisów
- W – (Write) - Tworzenie zapisów
- x - niewykorzystany

Różne wartości parametru ICAX1 dyskutowane są szeroko w rozdziale 5 "Operating System Manual". Najpowszechniej wykorzystywanymi typami funkcji OPEN są:

ICAX1 = 6 Czytanie katalogu plików na dyskietce. Czytane rekordy są wejściami katalogowymi plików.

ICAX1 = 4 Czytanie rekordów.

ICAX1 = 8 Zapisywanie rekordów. Jeżeli na dyskietce obecny jest plik o tej samej nazwie, zostaje on wykasowany, a pierwsze zapisywane bajty będą umieszczone w części początkowej pliku.

ICAX1 = 9 Zapisywanie i łączenie rekordów. Plik o podanej nazwie pozostaje nienaruszony, a pierwsze zapisywane bajty będą umieszczone poza istniejącym zapisem.

ICAX1 = 12 Weryfikacja zapisu. Tryb ten umożliwia zarówno czytanie jak i zapisywanie rekordów. Obie operacje rozpoczynają się od pierwszych rekordów pliku.

ICAX1 = 13 Kod niedozwolony.

Istnieją dwa typy operacji I/O obejmujących transmisję danych pomiędzy programem a rekordami na dyskietce - przenoszenie rekordów oraz przeroszenie symboli.

I/O symboli oznacza, że dane w pliku stanowią sekwencję bajtów wyrażenia tekstowego. DOS interpretuje kolejne bajty jako dane; żaden z nich nie jest interpretowany jako parametr kontrolny. Przykładem ciągu danych w rekordzie może być:

```
00 23 4F 55 FF 34 21
```

I/O rekordów oznacza, że dane w pliku stanowią zbiór rekordów. Rekord oznacza w tym przypadku ciąg bajtów, kończący się symbolem EOL (End Of Line - koniec linii) o wartości $9B. Przykładem zapisu rekordów może być:

```
  00 23 4F 55 FF 34 93 21 34 44 93
|     rekord 1       | rekord 2   |
```

Zarówno I/O symboli jak i rekordów dopuszcza wykonywanie wszelkich omówionych powyżej operacji I/O. Dane zapisane jako rekordy mogą być czytane jako pojedyncze symbole i odwrotnie. Jedyną różnicą pomiędzy zapisem symboli i rekordów jest fakt, iż każdy rekord musi kończyć się symbolem EOL, $9B. Przy wykorzystaniu I/O symboli EOL traktowany będzie na równi z innymi bajtami rekordu.

BASIC bezpośrednio może wykorzystywać operacje I/O rekordów. Do zapisu oraz czytania kolejnych rekordów z pliku służą instrukcje PRINT oraz INPUT. Natomiast w ograniczonym stopniu możliwe jest wykonywanie I/O symboli z BASICu. Instrukcje GET i PUT pozwalają na czytanie i zapisywanie tylko jednego bajta. Mimo iż OS posiada możliwość czytania i zapisywania jednorazowo całych bloków symboli, BASIC nie umożliwia jej wykorzystania. Przy wykorzystaniu procedury systemowej konieczne jest sprecyzowanie długości oraz części adresowej transmitowanych bloków symboli. Przenoszenie bloków symboli przy pomocy procedury OS może być także wykorzystane przy posługiwaniu się BASICiem – należy w tym celu stworzyć krótki program w języku maszynowym, który będzie wywoływany instrukcją USR z BASICu. Przykład takiego programu maszynowego przedstawiony został na rys: 8-17.

Instrukcja XIO w BASICu jest generalnym rozkazem wykonania operacji I/O i komunikuje ona bezpośrednio BASIC z CIO. Znaczenie tej instrukcji zostało dokładniej omówione w dalszej części tego rozdziału.

- Procedury sterowania stacją dysków (DUP)

DUP jest zbiorem procedur, umożliwiających użytkownikowi bezpośrednie gospodarowanie plikami na dyskietce. Opcje DUP nazywane są zwykle menu DOSu. Komendy DUP wykonywane są poprzez odwołanie się do FMS i wywołanie CIO. Komendami tymi (w DOS II 2.OS) są:

A. DIRECTORY	Katalog plików
B. RUN CARTRIDGE	Przekazanie kontroli kartridżowi
C. COPY FILES	Kopiowanie plików
D. DELETE FILES	Kasowanie plików
E. RENAME FILES	Zmiana nazwy pliku

F. LOCK FILES	Zabezpieczenie pliku
G. UNLOCK FILES	Odbezpieczenie pliku
H. WRITE DOS FILES	Zapisanie plików systemowych
I. FORMAT DISK	Formatowanie dyskietki
J. DUPLICATE DISK	Duplikowanie dyskietki
K. SAVE BINARY FILES	Zapis pliku binarnego
L. LOAD BINARY FILES	Ładowanie pliku binarnego
M. RUN AT ADDRESS	Przekazanie kontroli programowi
N. WRITE MEMORY SAVE FILES	Tworzenie pliku wymiennego z DOS
O. DUPLICTE FILE	Duplikacja pliku

Każda z tych funkcji została szerzej opisana w następnych podrozdziałach. Szczegółowy opis działania każdej z nich znaleźć można w „Disk Operating System II Manual".

- Wild Cards

Większość komend DUP wymaga sprecyzowania nazwy pliku. DOS dopuszcza jednak stosowanie dwóch typów "wild cards", którymi można zastępować wszelkie symbole znajdujące się w nazwach plików. Są one zarezerwowane przez symbole specjalne: znak zapytania (?) oraz gwiazdkę (*).

Symbole te mogą być stosowane w nazwach plików wszędzie tam, gdzie - z jakichkolwiek przyczyn - występują jakieś nieścisłości. Na przykład wówczas, gdy użytkownikowi nie jest znany ekstender (rozszerzenie) nazwy pliku. Innym przykładem może być żądanie skopiowania tylko tych plików, które posiadają charakterystyczny ekstender, np. .OBJ.

Znak zapytania może być stosowany w zastępstwie jednego symbolu w nazwie pliku. Gwiazdka natomiast zastępować może każdy dozwolony ciąg symboli o dowolnej długości w nazwie pliku. Poniższe przykłady ilustrują wykorzystanie symboli specjalnych przy czytaniu katalogu plików komendą DIRECTORY:

*.BAS	spowoduje wylistowanie wszystkich nazw plików znajdujących się na dyskietce D1, które posiadają ekstender .BAS
D2:*.*	spowoduje wylistowanie wszystkich plików znajdujących się na dyskietce w D2
PRO*.BAS	spowoduje wylistowanie wszystkich nazw plików znajdujących się na dyskietce w D1, które zaczynają się od liter PRO i posiadają ekstender .BAS
TEST??	spowoduje wylistowanie wszystkich nazw plików znajdujących się na dyskietce w D1, które po literach TEST zawierają jeszcze dwa inne dowolne symbole.

- Disk Directory (A)

Funkcja ta powoduje wylistowanie plików znajdujących się na dyskietce. Na ekranie wyświetlane są nazwy plików, ekstendery, oraz ilość sektorów, które zajmuje dany plik na dyskietce. Częściową

lista nazw plików można wywołać przez podanie specyficznych parametrów nazw plików, w których mogą być także stosowane "wild cards".

- Run Cartridge (B)

Wykonanie tej komendy powoduje przekazanie przez DOS kontroli nad systemem dowolnemu kartridżowi dołączonemu do układu. Wszelkie dalsze czynności zależą od rodzaju dołączonego kartridża. Na przykład BASIC zgłosi się napisem READY na ekranie.
Jeżeli dyskietka znajdująca się w stacji nie została zmieniona po załadowaniu DUP do pamięci i na dyskietce tej obecny jest plik typu MEM.SAV, zawartość tego pliku zostanie skopiowana z powrotem do pamięci RAM w miejsce DUP przed przekazaniem kontroli do kartridża. Pliki typu MEM.SAV tworzy się zwykle do przechowania tymczasowego zawartości tej części pamięci RAM, do której załadowany zostanie DUP. Jednakże plik taki musi się już znajdować na dyskietce w momencie wywołania ładowania DUP. Zanim DUP zostanie przepisany do pamięci RAM, nawar rość tej części pamięci powinna być utrwalona na dyskietce w postaci pliku MEM.SAV do późniejszego jej wykorzystania. Pliki MEM.SAv oraz DUP.SYS należy traktować jako wymieniające nawzajem swoje miejsca pomiędzy dyskietką a pamięcią RAM.

- Copy File (C)

Komenda ta używana jest do bezpośredniego kopiowania pliku z dyskietki znajdującej się w jednej stacji na dyskietkę znajdującą się w drugiej stacji. Po ukazaniu się pytania COPY-FROM,TO należy sprecyzować nazwy plików. Pierwsza nazwa pliku źródłowego może zawierać "wild cards" i w ten sposób oznaczać całą serię plików do skopiowania. Drugi parametr jest generalnie również nazwą pliku, lecz może to być również oznaczenie innego urządzenia peryferyjnego, jak np. E: (ekran) czy P: (drukarka). Po parametrze tym może występować opcja "/A", oznaczająca, że plik o nazwie wymienionej w drugiej kolejności ma zostać dołączony do pliku o pierwszej nazwie. Opcja ta nie powinna tyć używana w stosunku do plików zawierających stokenizowane wersje programów BASICu.

- Delete File (D)

Funkcja ta pozwala na skasowanie jednego lub więcej plików na dyskietce. Przy wyszczególnianiu nazw plików mogą być używane "wild cards". Program prosi o potwierdzenie polecenia skasowania danego pliku przez wpisanie "Y". W tym miejscu można nie potwierdzić polecenia, wpisując "N". '

- Rename File (E)

Komenda ta pozwala zmienić nazwę pliku znajdującego się już na dyskietce. Program prosi o wyszczególnienie starej oraz nowej nazwy pliku (GIVE OLD NAME, NEW). Pierwszy parametr musi być dokładnym określeniem danego pliku, drugi natomiast dowolną żądaną nazwą.
W obu parametrach mogą występować "wild cards". Jeżeli nie wyszczególnia się numeru stacji, której komenda dotyczy, program

traktuje ją jako D1. Błąd programowy powstaje wówczas, kiedy wyszczególnionego pliku nie ma na dyskietce, kiedy plik ten jest zabezpieczony, bądź dyskietka jest zabezpieczona przed zapisaniem.

- LOCK FILE (F)

Komenda ta jest wykorzystywana do zabezpieczenia danego pliku przed przypadkowym skasowanie lub modyfikacją. Podczas katalogowania plików pliki zabezpieczone oznaczane są gwiazdką (*) przed nazwą danego pliku. Należy pamiętać, że zabezpieczenie pliku jest całkowicie ignorowane podczas wykonywania komendy FORMAT oraz DUPLICATE DISK.

- Unlock File (G)

Komenda ta służy do odbezpieczenia pliku, uprzednio zabezpieczonego funkcją LOCK FILE. Obie te komendy dopuszczają stosowanie "wild cards" w nazwach plików.

- Write DOS File (H)

Funkcja ta służy do skopiowania plików DOSu na sformatowanej uprzednio dyskietce, jako że pliki systemowe nie mogą być skopiowane przy pomocy funkcji COPY FILE.

- Format Diskette (I)

Funkcja ta służy do formatowania dyskietek, czyli do całkowitego wykasowania dyskietki (jeżeli była ona zapisana) oraz stworzenia na niej zarysów sektorów i zapisania informacji systemowych, które będą następnie wykorzystane przez DOS. Jeżeli dyskietka zawiera uszkodzone sektory ("bad sectors"), DOS nie sformatuje jej. Przy wykorzystywaniu tej funkcji należy zachować szczególną ostrożność, jako że wykasowuje ona wszelkie istniejące na dyskietce zapisy.

- Duplicate Disk (J)

Funkcja ta pozwala na stworzenie dokładnego duplikatu dyskietki, która została zagospodarowana i zapisana przez DOS. Może ona być stosowana zarówno w systemach zawierających jedną stację dysków, jak i w systemach o wielu stacjach. Operowanie tą funkcją przy jednej stacji dysków wiąże się z wielokrotnym przekładaniem dyskietek źródłowej i docelowej.

Proces duplikacji polega na kolejnym przepisywaniu sektorów, Kopiowane są jednakże tylko te sektory, których wykorzystanie zostało zaznaczone w Tablicy Zawartości Dyskietki.

Przy wykorzystywaniu tej funkcji należy również zachować ostrożność, gdyż podobnie jak funkcja FORMAT, kasuje ona wszelkie istniejące na docelowej dyskietce zapisy. Dobrą praktyką jest zabezpieczanie przed zapisem dyskietek źródłowych, co pozwoli na wykluczenie katastrofalnego w skutkach błędu pomylenia dyskietek źródłowej i

docelowej podczas wielokrotnego ich przekładania w pojedynczej stacji dysków.

- Binary Save (K)

Komenda ta używana jest do utrwalenia na dyskietce zawartości rejestrów pamięci RAM w formacie pliku binarnego. Format ten jest używany także przez kartridż Atari Assembler Editor. Jest on oznakowany przez 6 pierwszych bajtów czołowych zapisu. Dwa pierwsze muszą mieć wartość $FF, dwa następne stanowią adres startowy procesu ładowania, dwa kolejne adres końcowy procesu ładowania. Pozostała część pliku traktowana jest jako dane. Funkcja ta wymaga od użytkownika podania nazwy pliku, oraz adresów startowego i końcowego rejestrów pamięci, z których plik ma zostać utworzony (i do których będzie załadowany funkcją LOAD). Istnieją ponadto dwa dodatkowe adresy, które mogą być zapisywane na życzenie użytkownika.
W takim przypadku odpowiednie wartości (adresy) zostaną zapisane w procesie ładowania pliku do pamięci w rejestrach zwanych INIT ($02E0,2) oraz RUN ($02E2,2). Zapisanie tych adresów w procesie ładowania spowoduje, że po wykonaniu ładowania (INIT) lub już w momencie zapisu rejestru (RUN) kontrola przekazana zostanie prog-ramowi, którego adres startowy zostanie wpisany do odpowiedniego rejestru.

- Binary Load (L)

Komenda ta wykorzystywana jest do ładowania plików zapisanych w formacie binarnym z dyskietki do pamięci RAM. Jeżeli plik ten podczas ładowania zapisuje wartości rejestrów adresowych INIT i/lub RUN, wówczas funkcja ta jest typu "ładuj i ruszaj".

- Run at Address (M)

Komenda ta jest używana do przekazania kontroli nad systemem programowi w kodzie maszynowym, rozpoczynającym się pod nakazanym adresem pamięci RAM. Stosuje się ją głównie do uruchamiania programów, które w trakcie ładowania nie zapisują adresów startowych w rejestrach INIT i/lub RUN.

- Create MEM.SAV (N)

Funkcja ta tworzy plik typu MEM.SAV na dyskietce. Tworzenie tego typu plików ma na celu utrwalenie na dyskietce zawartości tej części pamięci RAM, do której załadowany zostanie DUP.SYS. W rezultacie MEM.SAV oraz DUP.SYS zamieniają się miejscami w pamięci RAM i na dyskietce. Należy zwrócić uwagę, że plik typu MEM.SAV musi się już znajdować na dyskietce w stacji 1 przed wywołaniem ładowania DUP.SYS. Procedura przepisywania MEM.SAV oraz DUP.SYS zajmuje około 20 sekund.

- Duplicate File (O)

Funkcja ta służy do tworzenia duplikacji danego pliku (kopiowania go) z jednej dyskietki na drugą przy użyciu tylko jednej stacji dysków. Jest to więc operacja analogiczna do funkcji COPY, nie wymagająca jednakże użycia dwóch stacji dysków. Funkcja ta może także kopiować pliki utworzone pod kontrolą systemu DOS I.

- Zastosowanie funkcji z menu DUP instrukcją XIO BASICu

W BASICu istnieje instrukcja XIO, będąca generalną komendą wykonania operacji I/O, wykorzystującą bezpośrednie wywołanie centralnego podsystemu wejścia/wyjścia. Format komendy XIO jest następujący:

XIO numer komendy, numer IOCB, parametr1, parametr2, nazwa pliku

Instrukcja XIO może być wykorzystywana do przeprowadzenia funkcji, które normalnie wykonywane są przez DUP i wymagają jego obecności w pamięci RAM. Numer komendy instrukcji XIO dla poszczególnych funkcji DUP jest następujący:

Numer komendy	Funkcja
3	OPEN
5	GET Record
7	GET Characters
9	PUT Record
11	PUT Characters
12	CLOSE
13	STATUS
32	RENAME
33	DELETE
35	LOCK File
36	UNLOCK File

- Random Access

Jednym z najszerzej wykorzystywanych zastosowań dyskietek jest tworzenie rekordów, które w każdej chwili mogą być użyte przez pogram aplikacyjny. Technika ta znana jest powszechnie jako "Random Access". Wykorzystanie instrukcji I/O w połączeniu z instrukcjami specjalnymi NOTE oraz POINT pozwala na tworzenie własnych rekordów typu Random Access z poziomu BASICu.

DOS wykorzystuje rejestr, zwany wskaźnikiem pliku, do przechowywania adresu kolejnego bajta pliku (uprzednio otwartego komendą OPEN), który ma być akceptowany (czytany bądź zapisywany). Instrukcje NOTE oraz POINT pozwalają bezpośrednio kontrolować wartości zawarte w tym rejestrze - odczytywać je (NOTE) lub zapisywać (POINT). Wskaźnik pliku jest rejestrem dwubajtowym, zawierającym numer sektora oraz numer kolejny bajta w danym sektorze dyskietki. Numer sektora jest wartością od 1 do 719, wskazującą

systemowi DOS, w którym sektorze znajdują się dane, które mają być kolejno akceptowane.

Numer bajta w sektorze (0 – 127) wskazuje, który kolejny bajt danego sektora będzie w następnej kolejności akceptowany. Na rys. 9-2 pokazane zostały wskazania tego rejestru dla różnych bajtów, składających się na plik (wszystkie wartości w systemie heksadecymalnym).

Plik, wskazany na rysunku 9-1, utworzony został przez następujący pogram BASICu:

```
10 OPEN #1,8,0,"D:FILE"
20 ? #1; "ABC"
30 ? #1; "DEF”
40 ? #1; "GHIJK"
     :
     REM zapisanie pozostałej części sektora
     :
100 ? #1;"AB" :REM ten rekord przekracza granicę sektorów
150 CLOSE #1
```

Sektor numer 50 został arbitralnie wybrany dla potrzeb niniejszego przykładu; Numer sektora zmienił się w pewnym momencie na 51, ponieważ plik jest dłuższy od pojemności pojedynczego sektora.

FMS zaczął zapisywać dalszą część pliku w kolejnym dostępnym sektorze, to znaczy w sektorze 51. Granica sektorów przekroczona została przez rekord "AB", zapisywany w linii 100 przykładowego programu.

				EOL				EOL						EOL			EOL
	A	B	C	L	D	E	F	L	G	H	I	J	K	L	A	B	L
Plik	41	42	43	9B	44	45	46	9B	47	48	49	4A	4B	9B	...	41	42	9B
Wskaźnik pliku:																	
Numer sektora	50	50	50	50	50	50	50	50	50	50	50	50	50	50	...	50	51	51
Numer bajta	00	01	02	03	04	05	05	07	08	09	0A	0B	0C	0D	...	7C	00	01

Rys. 9-1. Wartości instrukcji NOTE oraz POINT

Numer bajta wskaźnika pliku zawiera początkowo wartość 0, po czym zwiększa się przy każdej operacji, aż do wskazania końca sektora, $7C (dziesiętnie 125). Trzy ostatnie bajty każdego sektora DOS rezerwuje na dane kontrolne danego sektora oraz pliku. Tak więc maksymalnym numerem bajta w sektorze jest 124 (0-124 = w sumie 125 bajtów). Jeżeli plik przekracza granicę sektora, numer sektora wskaźnika zwiększany jest o 1 (najczęściej), natomiast numer bajta startuje ponownie od 0.

Po otrzymaniu instrukcji POINT, żądającej przepisania wartości wskaźnika pliku, DOS sprawdza, czy wskazywany sektor należy do pliku, który został otwarty instrukcją OPEN. Jeżeli zaistnieje jakaś kolizja, instrukcja POINT nie zostanie wykonana.

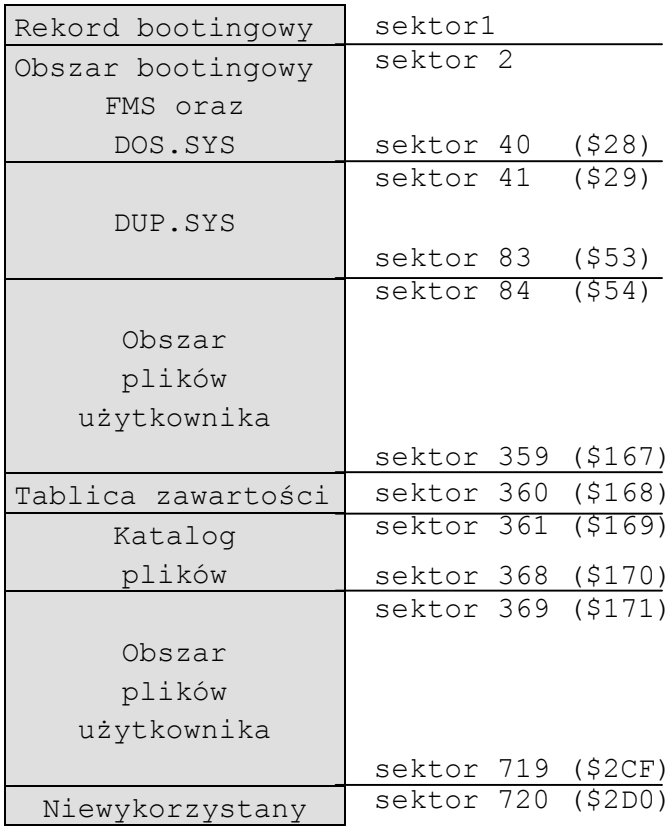
Rys. 9-2 ukazuje program, który może być wykorzystany do zapisania na dyskietce rekordów oraz udostępnienia ich w technice "Random Access":

```
1000 REM program ten tworzy i udostępnia pliki typu Random
1001 REM Access o zmiennej długości rekordów.
1002 REM
1003 REM ... Komendami są
1004 REM   CMD=1 zapisywanie n-tego rekordu
1005 REM   CMD=2 czytanie n-tego rekordu
1006 REM   CMD=3 udostępnianie n-tego rekordu
1007 REM
1008 REM RECORD$ jest rekordem wejścia/wyjścia
1009 RBM N jest kolejnym numerem rekordu
1010 REM INDEX jest dwuwymiarową zmienną
1020 REM w INDEX przechowywane są wartości NOTE wszystkich rekordów
1025 REM program zakłada, że plik #1 został otwarty dla I/O
1100 REM
1120 REM program zaczyna się w linii 1200
1130 REM
1200 ON CMD GOTO 2000,3000,4000
2000 REM ..................
2100 REM Zapis n-tego rekordu
2200 NOTE #1, X, Y
2300 INDEX(SEC, N)=X: INDEX(BYTE, N)=Y
2400 ? #1;RECORD$: RETURN
3000 REM ..................
3010 REM Czytanie n-tego rekordu
3030 X=INDEX(SEC, N): Y=INDEX(BYTE, N)
3040 POINT #1, X, Y
3C50 INPUT #1; RCORD$
3050 RETURN
4000 REM ..................
4010 REM Udostępnianie n-tego rekordu
4020 X=INDEX(SEC, N): Y=INDEX(BYTE, N)
4030 POINT #1, X, Y
4040 ? #1; RECORD$
4050 RETURN
```

Rys. 9-2. Przykład wykorzystania instrukcji NOTE i POINT

- Zagospodarowanie dyskietki przez FMS __

Przedstawiona poniżej mapa ukazuje schemat zagospodarowania dyskietki przez DOS w przypadku standardowej dyskietki mieszczącej 720 sektorów.



Rys. 9-3. Zagospodarowanie dyskietki przez DOS.

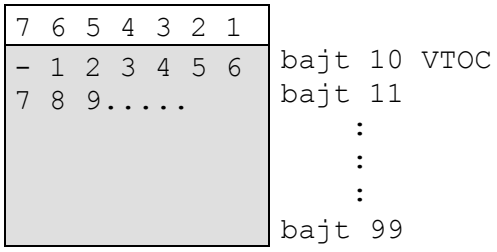
- Rekord bootingowy FMS

Pierwszy sektor na dyskietce zarezerwowany jest dla obszaru bootingowego FMS. Rekord ten zawiera informacje, związane z konfiguracją systemu FMS, oraz wskazuje, czy na dyskietce znajduje się plik systemowy DOSu. Jeżeli pliki DOSu znajdują się na dysku, zajmują one sektory od 2 do 81.

- Tablica Zawartości Dysku

FMS rezerwuje sektor 60 na tablicę zawartości dyskietki (VTOC). Tablica ta zawiera mapę bitów ukazującą, który sektor dysku jest wolny, a który zapisany. Na użytek tej tablicy zarezerwowany został sektor 360, ponieważ jest on zapisywany przed umieszczeniem na dyskietce jakiegokolwiek innego zapisu; znajduje się on dokładnie pośrodku dyskietki, a więc w miejscu, skąd dostęp do każdego innego sektora dyskietki jest najprostszy. Mapa bitów rozpoczyna się od bajta 10 tego sektora i rozciąga do bajta 99 - a więc każdy bajt sektora zawiera informację dotyczącą 8 sektorów dyskietki. Bit równy

0 na określonej pozycji oznacza, że odpowiadający mu sektor jest wykorzystany; bit równy 1 odpowiada sektorowi wolnemu. Siódmy bit bajta 10 jest niewykorzystany, gdyż na dyskietce nie ma sektora o numerze 0.



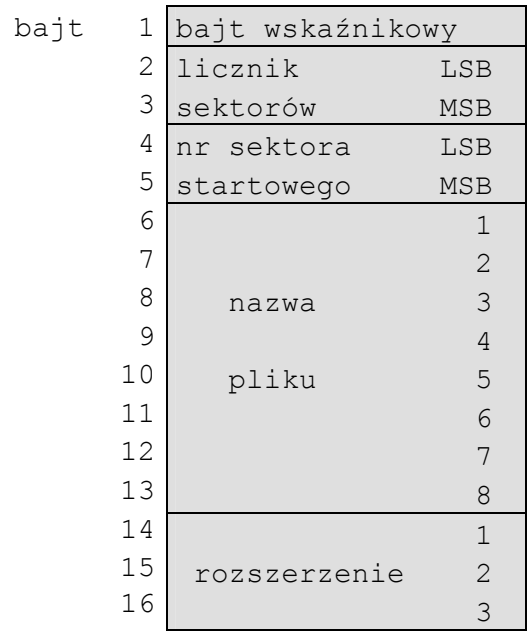
Rys. 9-4. Organizacja mapy bitów VTOC.

- Format katalogu plików

Osiem sektorów (361-356) zarezerwowanych jest na katalog plików znajdujących się na dysku. Każdy sektor zawiera informacje dotyczące ośmiu różnych plików. Wynika stąd, że maksymalna ilość plików, jakie mogą być umieszczone na dyskietce - niezależnie od ich długości - wynosi 64.

Dane dotyczące każdego pliku na dyskietce składają się z 16 bajtów, których mapa przedstawiona została na rys. 9-5. Znaczenie poszczególnych bitów bajta wskaźnikowego jest następujące:

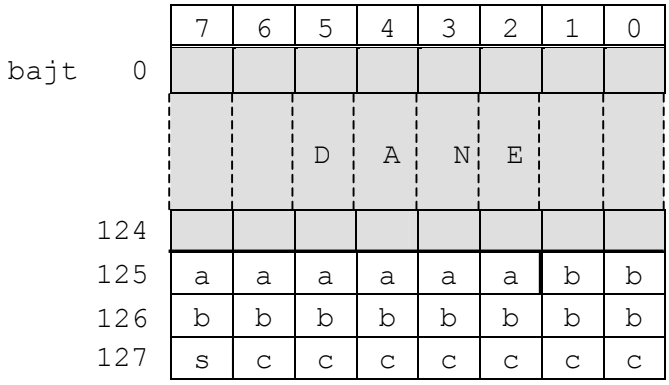
bit 7 = 1 oznacza, że plik został skasowany
bit 6 = 1 oznacza, że plik jest wykorzystywany
bit 5 = 1 oznacza, że plik jest zabezpieczony
bit 0 = 1 oznacza, że plik jest otwarty do czytania



Rys. 9-5. Mapa danych pliku

- Format sektora danych zapisanych przez FMS

Format jednego sektora pliku utrwalonego na dyskietce przedstawiony został na poniższym rysunku:



a – bity numeru zbioru
b – wskaźnik kolejności sektora
s – bit zapełnienia sektora
c – licznik ilości danych

Rys. 9-6. Format sektora FMS

Numer kolejny pliku na dyskietce jest informacją, wykorzystywaną przez FMS do kontrolowania integralności zapisu. Numer ten jest identyczny z numerem tegoż pliku znajdującym się w katalogu plików. Jeżeli istnieje jakakolwiek sprzeczność pomiędzy numerem kolejnym danego pliku znajdującym się w katalogu, a numerem znajdującym się w zapisie bajta 125 każdego sektora danego pliku, FMS zasygnalizuje błąd 164 i nie wykona żadnej operacji w stosunku do tegoż pliku.

Wskaźnik kolejności jest wskaźnikiem 10-bitowym, zawierającym numer następnego sektora, będącego częścią tego samego pliku. Wskaźnik ten jest czytany przez FMS podczas czytania danego pliku i wskazuje on, który sektor ma być czytany następny. Wskaźnik kolejności sektorów ostatniego sektora pliku jest równy 0.

Bit S wskazuje na zapełnienie sektora. Jeżeli bit ten jest włączony (równy 1) oznacza, iż sektor zawiera mniej niż 125 bajtów danych i jest to tzw. "krótki sektor".

9-bitowy licznik jest zapisywany ilością danych danego sektora. Jeżeli sektor zapisany jest w całości, licznik zawiera wartość 125 ($7D).

PLIKI TYPU AUTORUN.SYS

System DOS zawiera opcję, umożliwiającą tworzenie plików, które będą automatycznie ładowane do pamięci RAM komputera podczas startu zimnego. Może to być plik, zawierający dane służące do wstępnego zaprogramowania komputera w momencie załączania, jak np. ustawienie marginesów ekranowych czy zapisanie rejestrów kolorów. Może to być

także inny dowolny program w języku maszynowym, którego wykonanie wymagane jest przed standardowym procesem bootingu systemu DOS.

Pliki tego typu posiadają pewne ograniczenia: Muszą to być zbiory binarne o nazwie AUTORUN.SYS. Jeżeli mają to być programy samoczynnie się uruchamiające, muszą podczas ładowania zapisywać swój adres startowy w rejestrach INIT ($02E0,2) bądź RUN ($02E2,2). Różnica pomiędzy adresowaniem obu tych rejestrów polega na tym, że kontrola nad systemem przekazana zostanie pod adres zapisany w INIT natychmiast po zapisaniu tego rejestru; kontrola przekazana zostanie pod adres zapisany w RUN, dopiero po całkowitym zakończeniu ładowania programu do pamięci RAM. Po wykonaniu programu zapisanego jako AUTORUN.SYS kontrolą może być ponownie przekazana systemowi DOS - w takim wypadku AUTORUN.SYS musi się kończyć instrukcją RTS.

Pliki tego typu mają szerokie zastosowanie przy projektowaniu zbiorów typu "ładuj i ruszaj". Umożliwiają owe ponadto wszelką dopuszczalną ingerencję w system operacyjny, jak np. przepisywanie wartości wektorów systemowych, zanim jeszcze nastąpi standardowa inicjalizacja systemu DOS. Z drugiej strony, zapisy typu AUTORUN.SYS mogą być wykorzystywane do kluczowania i szyfrowania zapisów znajdujących się na dyskietce. Przykład takiego pliku przedstawiony został na rys. 8-3, w programie przepisującym wartość wskaźnika MEMLO przed standardową inicjalizacją systemu.

10. ATARI BASIC

CZYM JEST ATARI BASIC ?

Atari BASIC jest interpretacyjnym językiem programowania. Oznacza to, że programy napisane w Atari BASIC i wprowadzone do komputera mogą być wykonywane bez pośrednich etapów kompilacji oraz sprzęgania. Interpreter języka Atari BASIC znajduje się w 8K kartridżu ROM, traktowanym jako kartridże "B", albo inaczej "lewy":• Zajmuje on obszar pamięci od adresu $A000 do $BFFF. A więc minimum 8K pamięci ROM jest wymagane do pracy w języku BASIC.

Język ten w porównaniu z innymi językami programowania posiada wiele zalet, lecz nie jest także pozbawiony wad - warto się więc z nim zapoznać, co może być ułatwieniem w efektywnym wykorzystywaniu tego języka.

Do zalet języka Atari BASIC należy zaliczyć:

- świetna współpraca z tą częścią systemu operacyjnego, która odpowiedzialna jest za tworzenie obrazu na ekranie telewizora. Większość możliwości graficznych OS może być wykorzystana przez wykonanie prostych instrukcji BASICu.
- łatwa współpraca z innymi układami sprzętowymi - poprzez takie instrukcje jak SOUND, STICK czy PADDLE możliwa jest bezpośrednia komunikacja z układami interfejsowymi I/O.
- obecność instrukcji USR pozwala na bezpośrednie akceptowanie przez BASIC procedur czy też całych programów napisanych w języku maszynowym komputera.
- obecność interpretera języka w pamięci ROM stwarza możliwość wpisywania go da pamięci na wyraźne żądanie użytkownika, i zapobiega jednocześnie pomyłkowej modyfikacji interpretera przez program użytkownika.
- Atari BASIC posiada możliwość bezpośredniego komunikowania się z systemem DOS - poprzez takie instrukcje jak DOS, NOTE czy POINT - co upraszcza tworzenie i czytanie zapisów dyskowych bezpośrednio z języka BASIC.
- możliwość bezpośredniego komunikowania się z innymi urządzeniami peryferyjnymi, takimi jak drukarka, magnetofon kasetowy i inne.

Natomiast z najpoważniejszych wad tego języka należy wymienić:

- nieakceptowanie w sposób bezpośredni liczb całkowitych - wszystkie wartości liczbowe w Atari BASIC zapisywane są w 6-bajtowym kodzie BCD.
- zapis wszystkich liczb w 6-bajtowym kodzie BCD jest również przyczyną bardzo powolnego wykonywana wszystkich operacji matematycznych.
- Atari BASIC nie dopuszcza także stosowania indeksowanych zmiennych tekstowych, wszystkie zmienne tekstowe traktowane są jako jednowymiarowe.

JAK DZIAŁA ATARI BASIC ?

Praca interpretera języka BASIC może być w skrócie przedstawiona następującym schematem:

- BASIC odczytuje linię programu wprowadzonego przez użytkownika, sprawdza jej poprawność syntaktyczną, po czym przeprowadza konwersję zapisu w specjalny kod "tokenów", czyli w tzw. "stokenizowaną" wersję programu.
- zapisuje stokenizowaną wersję programu kolejno w pamięci RAM komputera.
- przystępuje do wykonywania programu.

Szczegóły wymienionych wyżej operacji przedstawione zostały szerzej w następujących podrozdziałach:

- Proces tokenizacji
- Struktura zapisu tokenów
- Proces wykonywania programu
- Współpraca z systemem

PROCES TOKENIZACJI

W uproszczeniu proces tokenizacji wprowadzonej jednej linii programu w języku Atari BASIC można przedstawić następująco:

1. BASIC odczytuje linię programu.
2. Sprawdza jej poprawność syntaktyczną.
3. W tym samym czasie zamienia koi ATASCII na tokeny.
4. Stokenizowana linia programu zostaje umieszczona w pamięci i ewentualnie dołączona do poprzednio wprowadzonych linii.
5. Jeżeli linia ta wpisana została w trybie bezpośrednim, BASIC przystępuje do jej natychmiastowego wykonania.

Zdefiniujmy teraz niektóre terminy, które pomogą nam zrozumieć proces tokenizacji.

Token - bajt kodu, interpretowanego w sposób szczególny.

Instrukcja - kompletny ciąg tokenów, oznaczający wykonanie jakiejś operacji przez Atari BASIC. Podczas listowania programów poszczególne instrukcje oddzielone są dwukropkami.

Linia - jedna lub więcej instrukcji, połączonych numerem linii z przedziału od 0 do 32767, bądź też linia trybu bezpośredniego, nie zawierająca numeru linii, wykonywana natychmiast po wczytaniu.

Komenda - pierwszy token instrukcji, która ma być wykonywana, mówiący BASICowi w jaki sposób interpretować następujące po niej inne tokeny.

Zmienna - token, wskazujący bezpośrednio wartość samej zmiennej. Wartość zmiennej może być zmieniona bez konieczności zmiany tokenu oznaczającego zmienną.

Stała - wartość w 6-bajtowym kodzie BCD, poprzedzona specjalnym tokenem. Oznacza wartość, która pozostanie niezmieniona podczas wykonywania programu BASICu.

Operator - jeden z listy 46 tokenów, które powodują określoną zmianę czy modyfikację wartości następującej po nim.

Funkcja - token, którego wykonanie wiąże się z wprowadzeniem do programu określonej wartości.

EOL - koniec linii. Znacznik końca linii programu o wartości $9B.

BCD - binarny kod zapisu liczb dziesiętnych. Kod ten wykorzystuje dziesiętny tryb pracy mikroprocesora 6502.

BASIC rozpoczyna proces tokenizacji od odczytania linii wprowadzonej przez użytkownika - ściślej mówiąc, linia ta może być wprowadzona przez którykolwiek ze sterowników urządzeń peryferyjnych systemu operacyjnego. Normalnie odczytywana jest ona ze sterownika edytora ekranowego; jednakże wykonanie instrukcji ENTFR pozwala na wprowadzenie linii programu z dowolnego urządzenia wejściowego. BASIC wykorzystuje komendę GET RECORD, po czym odczytuje ciąg danych w kodzie ATASCII, zakończony znacznikiem EOL. Dane te przepisywane są z urządzenia wejściowego przez CIO do bufora wejściowego BASICu, mieszczącego się w RAM pod adresami od $580 do $5FF.

Po odczytaniu rekordu rozpoczyna się proces sprawdzania poprawności syntaktycznej oraz tokenizacji. Po pierwsze BASIC szuka numeru kolejnego linii. Jeżeli takowy istnieje, zostaje on zamieniony w 2-bajtową liczbę całkowitą. Jeżeli rekord nie zawiera numeru linii, BASIC traktuje rekord jako zapis w trybie bezpośrednim i nadaje mu numer linii $8000. W ten sposób tworzone są pierwsze dwa tokeny linii programu. Tokeny pozostałej części rekordu zapisywane są początkowo w buforze wyjściowym tokenów, który posiada długość 256 bajtów i znajduje się na końcu pamięci RAM rezerwowanej na użytek systemu operacyjnego.

Następny token pozostaje chwilowo pusty i jest on przeznaczony na licznik bajtów, oznaczający długość danej linii programu, od jej początku do początku kolejnej linii. W drugiej kolejności rezerwowany jest jeszcze jeden pusty token, do którego później zostanie wpisana ilość bajtów od początku danej linii do początku kolejnej instrukcji - w tej czy też w następnej linii programu. Wartości tych tokenów zostaną zapisane przez BASIC po całkowitym zakończeniu tokenizacji bądź jednej linii bądź jednej instrukcji programu. Dyskusja na temat wykorzystania tych bajtów przedstawiona została w podrozdziale omawiającym proces wykonywania programu.

Teraz BASIC poszukuje komendy pierwszej instrukcji danej linii programu. Po jej odnalezieniu następuje sprawdzenie – porównanie z listą legalnych komend, znajdującą się w pamięci ROM. Jeżeli komenda danej instrukcji jest komendą legalną, jako następny token zapisywany jest kod danej komendy, skopiowany z listy komend w ROM. Jeżeli występuje niezgodność samej komendy z listą komend, BASIC oznacza bajt, w którym znaleziony został błąd syntaktyczny, po czym

przerywa proces tokenizacji, kopiuje zawartość buforu wejściowego do buforu wyjściowego w formacie ATASCII, po czym umieszcza na ekranie daną linię wraz z sygnałem wystąpienia błędu syntaktycznego.

Zakładając, że wprowadzona została poprawna komenda, może po niej występować siedem różnych elementów: zmienna liczbowa, stała liczbowa, operator, znak funkcji, cudzysłów, inna instrukcja lub znak EOL. BASIC najpierw sprawdza, czy pierwszym symbolem zapisu jest cyfra. Jeżeli nie, symbol ten oraz następujące po nim porównywane są z zapisami w tablicy nazw zmiennych. Załóżmy, że jest to pierwsza wprowadzana linia programu, a więc tablica nazw zmiennych jest jeszcze pusta, czyli porównanie to daje wynik negatywny. Teraz następuje porównanie tego zapisu z tablicami legalnych operatorów oraz funkcji, znajdujących się w ROM. Jeżeli i to porównanie daje wynik negatywny, BASIC stwierdza, że zapis ten jest kolejną nazwą zmiennej liczbowej – ponieważ jest to pierwsza wprowadzona nazwa, umieszczona zostaje na pierwszym miejscu w tablicy nazw zmiennych. Symbole są kopiowane bezpośrednio z buforu wejściowego do tablicy nazw zmiennych przy czym włączony zostaje najstarszy bit ostatniego bajta nazwy. Teraz BASIC rezerwuje 8 bajtów w tablicy wartości zmiennych na wartość zmiennej o wprowadzonej nazwie (patrz struktura zapisu tokenów).

Kolejnym tokenem zapisywanym w stokenizowanej wersji programu będzie numer porządkowy zmiennej wpisanej do tablicy nazw zmiennych, pomniejszony o jedność i z włączonym najstarszym bitem. A więc token odpowiadający pierwszej zmiennej z tablicy nazw będzie miał wartość $80, następny $81 i tak dalej, aż do $FF, co pozwala na umieszczenie w tablicy 128 różnych nazw zmiennych.

Jeżeli zapis ten odpowiada którejkolwiek nazwie funkcji z tablicy legalnych funkcji ROM, wówczas jako kolejny token zapisywany jest kod danej funkcji, zaczerpnięty z tablicy. Zapis funkcji wymaga stosowania określonych sekwencji parametrów; jeżeli zapis ten jest niezgodny z wzorcami znajdującymi się w tablicach poprawności syntaktycznej, sygnalizowany jest błąd syntaktyczny.

Jeżeli zapis ten odpowiada nazwie operatora, znajdującej się w tablicy legalnych operatorów, wówczas kolejnym tokenem będzie kod danego operatora. Operatory mogą występować obok siebie w dość długich sekwencjach (jak na przykład wielokrotnie stosowany nawias), tak więc sprawdzanie ich poprawności syntaktycznej jest procesem nieco bardziej skomplikowanym.

W przypadku napotkania cudzysłowu BASIC traktuje występujące po nim symbole jako wyrażenie tekstowe i jako token zapisuje wartość $0F, rezerwując kolejny bajt na oznaczenie długości wyrażenia tekstowego. Po odnalezieniu drugiego cudzysłowu i zapisaniu tokenu oznaczającego długość wyrażenia, całość zapisu przenoszona jest z buforu wejściowego do wyjściowego.

Jeżeli pierwszy porównywany symbol okaże się cyfrą, wówczas w buforze wyjściowym jako kolejny token zapisana zostanie wartość S0E, oznaczająca stałą liczbową, a wartość liczbowa zostanie przepisana w 6-bajtowym kodzie BCD i również przeniesiona do buforu.

Po odnalezieniu dwukropka zapisana zostanie wartość $14, oznaczająca koniec instrukcji, po czym zapisany zostanie bajt, zarezerwowany uprzednio na oznaczenie długości danej instrukcji od początku linii do początku nowej instrukcji. W dalszej kolejności

rezerwowany jest następny pusty token, gdzie zapisana zostanie długość następnej instrukcji, po czym rozpoczyna się taki sam proces, począwszy od poszukiwania następnej komendy.

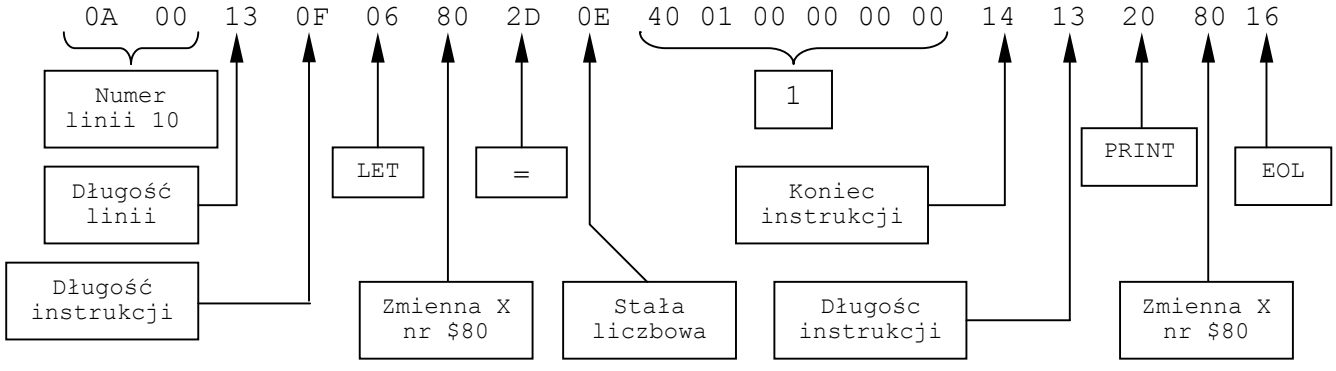
Po odczytaniu symbolu EOL BASIC umieszcza token $16, oznaczający koniec linii, po czym zapisuje bajt zarezerwowany uprzednio na oznaczenie długości danej linii. W tym momencie tokenizacja jednej linii programu dobiega końca I linia ta jest przenoszona z buforu wyjściowego do pamięci RAM i ewentualnie dołączana do zapisanych tam uprzednio linii programu. Przede wszystkim BASIC odczytuje wszystkie numery linii zapisanego uprzednio programu. Jeżeli znajduje linię o numerze takim samym, jak linia znajdująca się jeszcze w buforze wyjściowym, wówczas zastępuje starą linię programu linią nową z buforu. Jeżeli nie znajduje linii o takim samym numerze, umieszcza linię z buforu w pamięci w taki sposób, by zachować kolejność numerów linii. W obu przypadkach ciąg tokenów w pamięci może być przesuwany w górę bądź w dół pamięci tak, by kolejne linie programu o rosnących numerach były zapisane w pamięci kolejno.

Teraz BASIC sprawdza, czy wprowadzona linia jest w trybie bezpośrednim. Jeżeli tak, przystępuje natychmiast do jej wykonywania, zgodnie z metodami opisanymi przy omawianiu procesów interpretacji stokenizowanego programu. Jeżeli nie, BASIC przystępuje do czytania następnej linii programu wprowadzonej przez użytkownika.

Jeżeli w którymkolwiek momencie procesu tokenizacji przekroczona zostanie pojemność buforu wyjściowego, to znaczy długość stokenizowanej linii programu przekroczy 256 bajtów, BASIC zasygnalizuje błąd 14, zbyt długa linia programu; po czym przystąpi do odczytywania następnej linii.

Przykład wprowadzonej linii programu oraz jej stokenizowanej wersji można przedstawić następująco:

```
10 LET X=1: PRINT X
```



Rys. 10-1. Przykład stokenizowanej wersji linii programu

K O M E N D Y											
HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa
00	0	REM	0E	14	BYE	1C	28	POINT	2A	42	PUT
01	1	DATA	0F	15	CONT	1D	29	XIO	2B	43	GRAPHICS
02	2	INPUT	10	16	COM	1E	30	ON	2C	44	PLOT
03	3	COLOR	11	17	CLOSE	1F	31	POKE	2D	45	POSITION
04	4	LIST	12	18	CLR	20	32	PRINT	2E	46	DOS
05	5	ENTER	13	19	DEG	21	33	RAD	2F	47	DRAWTO
06	6	LET	14	20	DIM	22	34	READ	30	48	SETCOLOR
07	7	IF	15	21	END	23	35	RESTORE	31	49	LOCATE
08	8	FOR	16	22	NEW	24	36	RETURN	32	50	SOUND
09	9	NEXT	17	23	OPEN	25	37	RUN	33	51	LPRINT
0A	10	GOTO	18	24	LOAD	26	38	STOP	34	52	CSAVE
0B	11	GO TO	19	25	SAVE	27	39	POP	35	53	CLOAD
0C	12	GOSUB	1A	26	STATUS	28	40	?	36	54	pomin.LET
0D	13	TRAP	1B	27	NOTE	29	41	GET	37	55	ERROR syn

HX=HEX DC=DEC pomin.LET – gdy nie użyto LET

Rys. 10-2.1. Tokeny komend.

O P E R A T O R Y											
HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa
0E	14	stała num	1A	26	STEP	26	38	-	32	50	<(txt)
0F	15	stała txt	1B	27	THEN	27	39	/	33	51	>(txt)
10	16	niewykorz	1C	28	#	28	40	NOT	34	52	=(txt)
11	17	niewykorz	1D	29	<= (num)	29	41	OR	35	53	+(znak))
12	18	,	1E	30	<>(num)	2A	42	AND	36	54	-(znak)
13	19	$	1F	31	>=(num)	2B	43	(	37	55	(txt
14	20	:	20	32	<(num)	2C	44	)	38	56	(z.i.
15	21	;	21	33	>(num)	2D	45	= (ar)	39	57	(w.z.i.
16	22	EOL	22	34	=	2E	46	=(w.txt)	3A	58	(w.f.
17	23	GOTO	23	35	^	2F	47	<= (txt)	3B	59	(w.z.t.
18	24	GOSUB	24	36	*	30	48	<>(txt)	3C	60	, (in.zm)
19	25	TO	25	37	+	31	49	>=(txt)	----------		

w.txt – wyrażenie tekstowe z.i. – zmienna ineksowana
w.z.i. - wymiar zmiennej ind. w.z.t. – wymiar zmiennej txt.
w.f. – wyrażenie funkcyjne in.zm – w zapisach indeksów zm.
ar – równość arytmetyczna num – liczbowe

Rys. 10-2.2. Tokeny operatorów.

F U N K C J E											
HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa	HX	DC	Nazwa
3D	61	STR$	43	67	ADR	49	73	FRE	4F	79	ABS
3E	62	CHR$	44	68	ATN	4A	74	EXP	50	80	INT
3F	63	USR	45	69	COS	4B	75	LOG	51	81	PADDLE
40	64	ASC	46	70	PEEK	4C	76	CLOG	52	82	STICK
41	65	VAL	47	71	SIN	4D	77	SQR	53	83	PTRIG
42	66	LEN	48	72	RND	4E	78	SGN	54	84	STRIG

Rys. 10-2.3. Tokeny funkcji.

STRUKTURA ZAPISU TOKENÓW

Zapis programu w wersji stokenizowanej składa się z dwóch podstawowych części:

1. grupy rejestrów wskaźnikowych ze strony zerowej RAM, odnoszących się do stokenizowanego programu w pamięci RAM, oraz
2. z samego programu w wersji stokenizowanej. Rejestry strony zerowej są wskaźnikami 2-bajtowymi, w których zapisane zostają adresy różnych sekcji stokenizowanego programu. Tych 9 2-bajtowych rejestrów znajduje się pod adresami od $80 do $91. A oto opis rejestrów wskaźnikowych oraz części programów BASICu, do których one się odnoszą.

LOMEM ($80, 81) - Wyjściowy bufor tokenizacji

Jest to adres 256bajtowego buforu, wykorzystywanego przez BASIC w procesie tokenizacji jednej linii programu. Znajduje się on na końcu pamięci RM rezerwowanej na użytek systemu operacyjnego.

VNTP ($82, 83) - Tablica nazw zmiennych

Adres listy wszystkich nazw zmiennych wprowadzonych w programie. Zmienne zapisywane są w kodzie ATASCII w takiej kolejności, w jakiej zostały wprowadzone do programu. Występują trzy typy nazw:

1. Zmienne skalarne - oznaczane przez włączenie najstarszego bitu ostatniego bajta nazwy.
2. Zmienne tekstowe - oznaczane przez włączenie najstarszego bitu ostatniego bajta nazwy, którym musi być symbol $.
3. Zmienne indeksowane - oznaczane przez włączanie najstarszego bitu ostatniego bajta nazwy, którym musi być symbol (.

VNTD ($84, 85) Koniec tablicy nazw zmiennych

Adres wykorzystywany przez BASIC do oznaczenia końca tablicy nazw zmiennych. Jeżeli w tablicy znajduje się mniej niż 128 nazw zmiennych, wówczas wektor ten wskazuje przypadkowy bajt zerowy. Jeżeli tablica zawiera 128 nazw zmiennych, rejestr zapisany jest adresem ostatniego bajta ostatniej w tablicy nazwy zmiennej.

VVTP ($86, 87) Tablica wartości zmiennych

Tablica ta zawiera bieżące informacje dotyczące wartości każdej zmiennej programu. Dla każdej zmiennej znajdującej się w tablicy nazw rezerwowanych jest 8 bajtów w tablicy wartości zmiennych. Informacje zawarte w tych ośmiu bajtach dla każdego typu zmiennej są następujące:

Typ zmiennej	Numer bajta				
	1	2	3 4	5 6	7 8
Zmienna skalarna	00	#	6 bajtów kodu BCD		
Zmienna indeksowana wymiarowana niewymiarowana	41 40	#	wartość STARP	pierwszy wymiar +1	drugi wymiar +1
Zmienna tekstowa indeksowana nieindeksowana	81 80	#	wartość STARP	długość	wymiar

- numer kolejny zmiennej w tablicy nazw

Rys. 10-3. Format opisu zmiennych w VVTP.

Zmienna skalarna zawiera wartość liczbową. Przykładem jest X=1. Skalar X posiada wartość 1, zapisywaną w tablicy wartości w postaci 6 bajtów kodu BCD. Zmienna indeksowana (albo matrycowa) jest kompozycją elementów liczbowych zawartych w tablicy lub matrycy; w tablicy wartości zmiennych posiada tylko jeden zapis, niezależnie od wielkości matrycy. Zmienna tekstowa, składająca się z symboli zawartych w tablicy, posiada także tylko jeden zapis w tablicy wartości.

Pierwszy bajt każdego zapisu w tablicy wartości zmiennych oznacza jej typ: 00 oznacza skalar, 40 zmienną matrycową, 80 natomiast tekstową. Jeżeli z zmienna matrycowa lub tekstowa posiada określony przez użytkownika wymiar (indeks), wówczas w bajcie tym włączony zostaje najmłodszy bit.

Drugi bajt zawiera numer zmiennej. Pierwsza zmienna w tablicy nazw będzie posiadała numer 0, ostatnia (o ile występuje 128 nazw) numer $FF.

W przypadku zmiennych skalarnych bajty od 3 do 8 zawierają bieżącą wartość danej zmiennej, zapisaną w 6-bajtowym kodzie BCD. Dla zmiennych matrycowych oraz tekstowych bajty 3 i 4 zawierają wartość przepisaną ze STARP (poniżej), określającą odległość H pamięci pomiędzy miejscem w tablicy wartości zmiennej a początkiem danych dla tej zmiennej w obszarze danych matrycowych.

Dla zmiennych matrycowych bajty 5 i 6 zawierają pierwszy indeks zmiennej, natomiast bajty 7 i 8 drugi indeks. Zapisane tu wartości są o 1 wyższe od wartości wprowadzonych przez użytkownika. 5 i 6 bajt zapisu zmiennej tekstowej stanowią 16-bitowy zapis liczby całkowitej, oznaczającej długość zmiennej. Bajty 7 i 8 zawierają wprowadzony przez użytkownika indeks - maksymalnie do 32767 bajtów wielkości.

STMTAB ($88, 89) Tablica instrukcji

Ten blok danych zawiera kody wszystkich linii wprowadzonych przez użytkownika i stokenizowanych przez BASIC, włącznie z kodami linii wykonanych w trybie bezpośrednim. Format poszczególnych zapisów w tej tablicy omówiony został w poprzednim podrozdziale.

STMCUR ($8A, 8B) Wskaźnik bieżącej instrukcji

Wartość tego rejestru wykorzystywana jest przez BASIC do oznaczania bieżąco kolejnych instrukcji zapisanych w tablicy instrukcji. Jeżeli BASIC oczekuje na wprowadzenie danych, rejestr ten wskazuje początek linii trybu bezpośredniego.

STARP ($8C, 8D) Obszar matrycowy

Ten blok zawiera wszystkie dane zmiennych indeksowanych oraz tekstowych. Symbole imiennych tekstowych zapisywane są tutaj w kodzie ATASCII, a więc zmienna o 20 symbolach zajmie 20 bajtów w pamięci. Zmienne matrycowe zapisywane są w 6-bajtowym kodzie BCD dla każdego elementu matrycy lub tablicy, a więc 10-elementowra zmienna zajmie w pamięci 60 bajtów.

Obszar ten jest umieszczany w różnych rejonach RAM, a jego wielkość zależy od ilości i jakości wprowadzonych zmiennych. Wykonanie instrukcji DIM wiąże się z zarezerwowaniem w tym obszarze odpowiedniej ilości pamięci - w przypadku zmiennych tekstowych równej długości zgłoszonej zmiennej, natomiast w przypadku zmiennych matrycowych równej sześciokrotnej wartości indeksu zmiennej.

RUNSTK ($8E, 8F) Stos bieżący

Na ten stos programowy odkładane są adresy powrotne przy komendach GOSUB oraz FOR/NEXT. Przy wykonywaniu komendy GOSUB na stos odkładane są 4 bajty - pierwszy równy 0 oznacza, iż wykonywana jest komenda GOSUB, w dwóch kolejnych bajtach, zapisywany jest numer linii, w której wystąpiła ta komenda, w ostatnich natomiast odległość w pamięci pozwa1ająca na prawidłowe wykonanie komendy RETURN.

W przypadku pętli FOR/NEXT na stos odkładanych jest 16 bajtów. W pierwszym zapisywany jest limit, do którego może być przepisywany licznik pętli, drugi zawiera krok licznika, określony przez użytkownika operatorem STEP; obie powyższe wartości zapisywane są w 6-bajtowym kodzie BCD. Bajt trzynasty zawiera numer kolejny zmiennej, będącej bazą pętli; bajty 14 i 15 zapisywane są numerem linii; natomiast bajt 16 zawiera odległość w pamięci od instrukcji NEXT.

MEMMTOP ($90, 91) Szczyt zajętej pamięci RAM

Wskaźnik ten zawiera adres kontrolny stokenizowanego programu. Program może być rozbudowywany w pamięci aż da końca dostępnej RAM, określonej przez adres listy displejowej. Funkcja FRE daje w rezultacie ilość wolnej pamięci RAM, powstałą przez odjęcie wartości MEMTOP od HIMEM ($2E5, 22E6). Należy zwrócić uwagę, że wskaźnik MEMTOP BASICu jest czymś innym niż zmienna systemu operacyjnego o nazwie MEMTOP.

PROCES WYKONYWANIA PROGRAMU

Wykonanie linii kodu jest procesem rozpoczynającym się od odczytania tokenów, zapisanych w procesie tokenizacji. Każdy token posiada swoje szczególne znaczenie, w związku z czym BASIC musi wykonać ciąg określonych operacji. Metoda ta zmusza BASIC do jednorazowego odczytywania tylko jednego tokenu z całego programu i wykonania danych operacji. Token jest wykorzystywany jako indeks skoku do tablicy rutynowych procedur. Tak więc token oznaczający komendę PRINT będzie wskazywa3 bezpośrednio adres początkowy procedury wykonującej tę komendę. Po zakończeniu jednego procesu BASIC odczytuje następny token. Rejestrem wskazującym, który z tokenów ma być czytany w następnej kolejności, jest STMCUR.

Pierwszą linią wykonywanego kodu programu jest linia wprowadzona w trybie bezpośrednim. Zwykle jest to instrukcja RUN lub GOTO. W przypadku instrukcji RUN, BASIC odczytuje pierwszą linię tokenów z tablicy instrukcji (stokenizowana wersja programu), po czym wykonuje ją. Jeżeli program w tej linii nie zawiera pętli, kody tej linii wykonywane są kolejno.

W przypadku instrukcji GOTO BASIC musi odnaleźć linię o numerze wskazywanym tą instrukcją. W tablicy instrukcji linie programu zapisane są w kolejności rosnących numerów. Aby znaleźć linię znajdującą sit gdziekolwiek w środku tej tablicy, BASIC musi rozpocząć poszukiwania od początku tablicy.

Adres pierwszej linii programu zapisywany jest we wskaźniku STMTAB. Adres ten przepisywany jest do rejestru tymczasowego. Pierwsze dwa bajty pierwszej linii programu stanowią numer linii, który teraz jest porównywany z numerem występującym w instrukcji GOTO. Jeżeli numer ten jest mniejszy, BASIC odczytuje numer następnej linii programu - wykonując to poprzez dodanie do rejestru tymczasowego trzeciego bajta sprawdzonej linii, gzie zapisana jest długość tej linii, co daje w efekcie adres linii następnej. Ponownie dwa pierwsze bajty nowej linii porównywane są z numerem zawartym w instrukcji GOTO. Proces ten powtarzany jest tak długo, aż zostanie znaleziona żądana linia programu. Wówczas zawartość rejestru tymczasowego przepisywana jest do rejestru STMCUR, po czym BASIC zaczyna odczytywać kolejne tokeny danej linii. Jeżeli dana linia programu nie zostanie znaleziona, sygnalizowany jest błąd 12.

Wykonanie instrukcji GOSUB jest procesem nieco bardziej skomplikowanym. Wyszukiwanie odpowiedniej linii programu przebiega analogicznie, lecz przed przystąpieniem do jej wykonania BASIC na bieżący stos odkłada bajty parametrów, które szczegółowo opisane były w poprzednim podrozdziale. Dopiero teraz następuje wykonanie

żądanej linii programu. Po odczytaniu komendy RETURN, BASIC odczytuje ze stosu adres powrotny, po czym przystępuje do wykonywania kolejnych kodów linii, w której wystąpiła komenda GOSUB.

W analogiczny sposób przy wykonywaniu pętli FOR/NEXT na stos odkładanych jest aż 16 bajtów, z czego 12 stanowią zapisy limitu oraz kroku licznika, zapisanych w 6-bajtowym kodzie BCD. Po odczytaniu komendy NEXT, BASIC odczytuje ostatni zapis ze stosu. następuje tutaj sprawdzenie, czy zmienna zapisana w instrukcji NEXTT ma tę samą wartość, co zmienna odłożona na stos podczas wykonywania komendy FOR; następnie BASIC sprawdza, czy wartość licznika osiągnęła wartość limitu licznika, odłożoną na stosie. Jeżeli nie, BASIC powraca do linii, w której zawarta była komenda NEXT i kontynuuje wykonywanie kodu. Jeżeli limit licznika został osiągnięty, zapis na stosie zostaje skasowany, a wykonywanie programu kontynuowane jest po ominięciu komendy NEXT.

Podczas wykonywania obliczeń matematycznych operatory odkładane są na stos operatorów, a całe wyrażenie matematyczne czytane jest jednocześnie. Kolejność wykonywania operatorów może być nieokreślona - w takim przypadku BASIC sprawdza priorytet wykonywania operatorów, odwołując się do tablicy w ROM - bądź też określona przez użytkownika poprzez użycie nawiasów.

Naciśnięcie klawisza BREAK w dowolnym momencie wykonywania programu powoduje, że system operacyjny zapisuje odpowiednią wartość w rejestrze flagowy BREAK. BASIC sprawdza ten rejestr po wykonaniu każde j operacji stokenizowanego programu. Jeżeli rejestr flagowy wykaże, że klawisz został naciśnięty, BASIC utrwala numer linii, w trakcie wykonywania której się znajdował, umieszcza na ekranie wiadomość "STOPPED AT LINE XX”, przywraca pierwotną wartość rejestru flagowego BREAK i przechodzi do oczekiwania na kolejną instrukcję użytkownika. Jeżeli w takim momencie wprowadzona zostanie instrukcja CONT, BASIC powróci do wykonywania stokenizowanego programu, poczynając od następnej linii.

WSPÓŁPRACA Z SYSTEMEM

Komunikacja BASICu z systemem operacyjnym ogranicza się w większości przypadków do wezwań I/O Centralnego Podsystemu Wejścia/wyjścia. Na rys. 10-4 przedstawiamy listę dostępnych w BASICu wezwań I/O oraz schemat operacji zapisywania parametrów IOCB.

SAVE/LOAD: Podczas zapisywania stokenizowanej wersji programu BASICu w urządzeniu peryferyjnym zapisywane są dwa bloki informacji: Pierwszy stanowi siedem spośród dziewięciu rejestrów wskaźnikowych BASICu ze strony zerowej pamięci - począwszy od LOMEM ($80, 81) do STARP ($8C, 8D). Podczas zapisywania dokonywana jest jedna zmiana - wartość LOMEM odejmowana jest od wszystkich wskaźników - i tak zmodyfikowane wartości rejestrów wskaźnikowych zapisywane są w urządzeniu peryferyjnym. A więc pierwsze 2 bajty zapisu będą w każdym przypadku równe 0, 0.

BASIC	OS
OPEN #1, 12, 0, "E: "	IOCB = 1 Komenda = 3 (OPEN) Aux1 = 12 (wejście/wyjście) Aux2 = 0 Adres buforu = ADR(E:)
GET #1, X	IOCB = 1 Komenda = 7 (GET Character) Długość buforu = 0 Symbol zapisywany w rej. A
PUT #1, X	IOCB = 1 Komenda = 11 (PUT Character) Długość buforu = 0 Symbol przepisywany z rej. A
INPUT #1, A$	IOCB = 1 Komenda = 5 (GET Record) Dł. buforu = dł. A$ (max. 120 bajtów) Adres buforu = Adr. buf. wej. BASICu
PRINT #1, A$	IOCB = 1 BASIC wykorzystuje specjalny wektor "PUT BYTE" w IOCB do bezpośredniej komunikacji ze sterownikiem urządzenia.
XIO 18,#6,12,0, "S:"	IOCB = 6 Komenda = 18 (SPECIAL = FILL) AUX1 = 12 AUX2 = 0

Rys. 10-4. Lista wezwań I/O w BASICu i schemat zapisu parametrów IOCB

Drugi blok informacji składa się z następujących części składowych:

1. tablica nazw zmiennych
2. tablica wartości zmiennych
3. kody stokenizowanego programu
4. linie trybu bezpośredniego

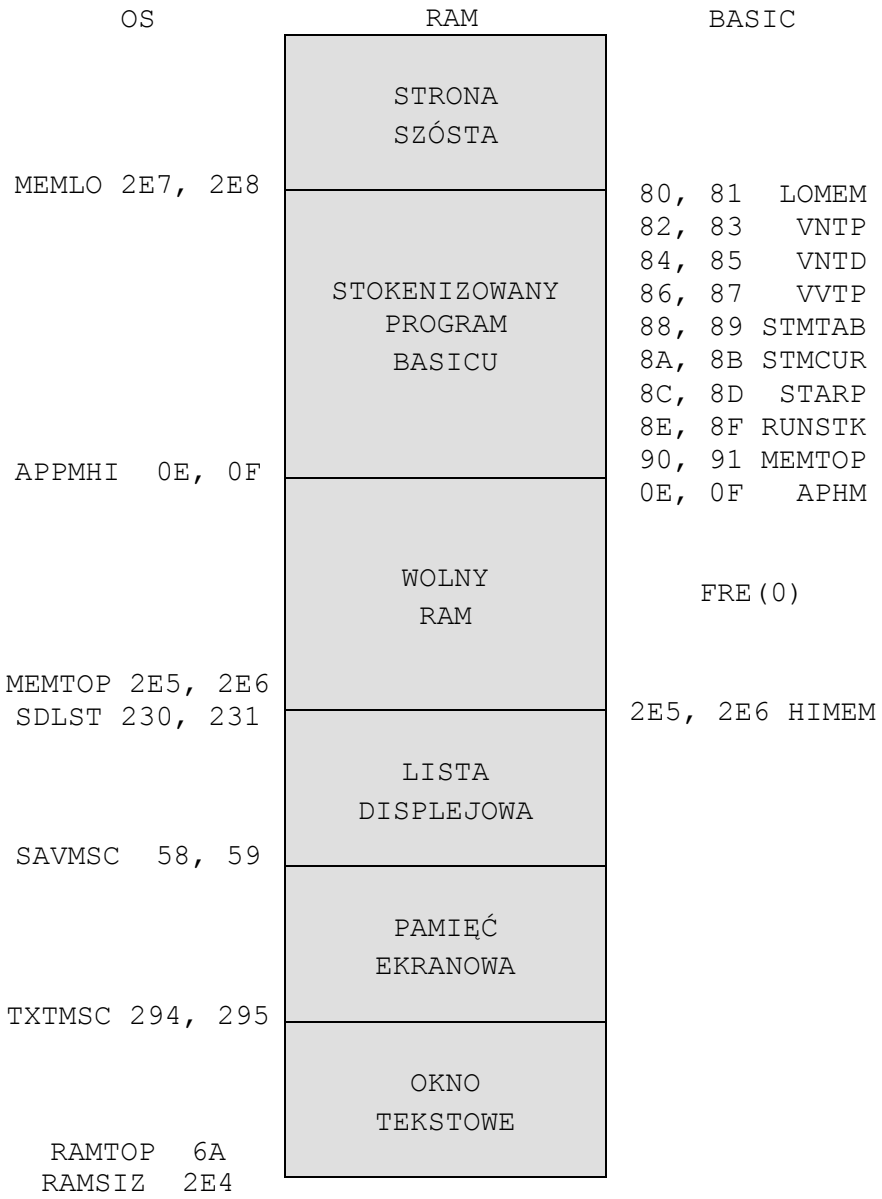
Podczas ładowania takiego programu do pamięci, BASIC odczytuje wartość rejestru wskaźnikowego OS, MEMLO ($2E7, 2E8), po czym dodaje ją do każdego z dwubajtowych wskaźników BASICu odczytanych z zapisu. Wartości te umieszczane są ponownie w pamięci na stronie zerowej, po czym rejestry RUNSTK oraz MEMTOP zostają zapisane wartością znajdującą się w rejestrze STARP.

W następnej kolejności BASIC rezerwuje obszar 256-bajtów powyżej adresu wskazywanego przez MEMLO z przeznaczeniem na bufor wyjściowy tokenów, po czym odczytuje zapisany program, umieszczając bloki

kolejno powyżej buforu wyjściowego począwszy od tablicy nazw zmiennych, a skończywszy na liniach trybu bezpośredniego programu:

EFEKTYWNE WYKORZYSTANIE ATARI BASIC

Poprawianie programów BASICu powinno mieć dwa podstawowe cele: Możliwe przyspieszenie wykonywania programu oraz oszczędne gospodarowanie pamięcią komputera. Poniżej przedstawiono generalne uwagi, które powinny pomóc w realizacji obu wymienionych celów. Ukazane sposoby wymienione zostały według malejącej efektywności - oznacza to, że uwzględnienie uwag znajdujących się na początku każdej z list da o wiele lepsze rezultaty od tych, umieszczonych na końcu listy.



Rys. 10-5. Rejestry wskaźnikowe OS i BASICu (bez DOSu).

- Przyspieszone wykonywanie programów BASICu

1. Konstrukcja algorytmu - Ponieważ BASIC nie jest językiem strukturalnym, kod programu ma tendencję do "bycia mało wydajnym"; po wielu przekonstruowaniach język ten wydaje się być jeszcze mniej sprawnym niż w pierwotnej wersji. Stąd czas poświęcony na konstruowanie algorytmu przynosi zwykle znaczące efekty.
2. Logika ogólna algorytmu - należy się upewnić, że program, który ma być wykonywany, jest tak prosty i tak zwarty jak tylko to możliwe.
3. Często wywoływane podprogramy oraz pętle programowe FOR/NEXT należy umieszczać w początkowych liniach programu - BASIC każdorazowo poszukuje danej linii programu poczynając od linii pierwszej, a więc umieszczenie podprogramów lub pętli na końcu wydłuży czas poświęcony na odnalezienie odpowiedniej linii.
4. Często wykonywane operacje należy umieszczać kolejno w pętli, a nie w oddzielnych podprogramach - zaoszczędzi to nie tylko czasu wyszukiwania żądanej linii podprogramu, ale także odkładania i odzyskiwania informacji ze stosu bieżącego.
5. Pętle, w których dokonywana jest największa ilość zmian powinny być wykonywane jako pierwsze - zmniejszy to ilość kolejnych odłożeń, odczytywań i kasowań parametrów na stosie bieżącym.
6. Operacje matematyczne wykonywane wewnątrz pętli powinny być w miarę możliwości proste. Jeżeli na przykład jakiś parametr otrzymuje się przez pomnożenie stałej przez wartość licznika, znacznym zaoszczędzeniem czasu będzie kolejne dodawanie do stałej wartości licznika.
7. Pętle powinny być w miarę możliwości zapisywane jako ciąg instrukcji w jednej linii programu - zaoszczędzi to czasu BASICu na odczytywanie kolejnych linii programu w ramach tej samej pętli.
8. Wygaszenie ekranu - Jeżeli przez jakiś czas informacje ekranowe nie są ważne podczas obliczeń, wykonanie instrukcji POKE 559,0 może zaoszczędzić nawet do 30% czasu.
9. Należy wykorzystywać prostsze tryby graficzne oraz krótsze listy displejowe - jeżeli nie jest wykorzystywany cały ekran do przekazywania informacji, można zaoszczędzić nawet do 25% czasu.
10. Należy szeroko wykorzystywać podprogramy maszynowe - każda pętla lub podprogram może być wywołany instrukcją USR.

- Oszczędna gospodarka pamięcią w programach BASICu

1. Przekonstruowanie algorytmu programu, jak wspomniano wcześniej, może być źródłem nie tylko zaoszczędzenia czasu, ale również pamięci komputera.
2. Usunięcie wszystkich opisów w programie, które przechowywane są w pamięci w kodzie ATASCII, pozwoli na zaoszczędzenie RAM.
3. Wszystkie stałe, występujące wielokrotnie w programie, należy zapisywać jako zmienne - BASIC rezerwuje dla każdej stałej

cyfrowej 7 bajtów pamięci, podczas gdy zmienna zapisywana jest jako 1 bajt, oznaczający jej numer kolejny w tablicach nazw oraz wartości zmiennych.

4. Należy wpisywać wartości zmiennych przy pomocy instrukcji READ. Dane po instrukcji DATA przechowywane są w kodzie ATASCII, gdzie 1 bajt odpowiada jednej cyfrze, natomiast wprowadzanie wartości zmiennej instrukcją LET pochłania w pamięci 7 bajtów.
5. Należy starać się w miarę możliwości zapisywać stałe w postaci kombinacji wprowadzonych wcześniej zmiennych. Jeżeli np. zdefiniowane zostało Z1 = 1 oraz Z2 = 2, a wymagana jest w programie wprowadzenie stałej równej 3, wykorzystajmy Z1 + Z2.
6. Często wykorzystywane wywołania numerów linii (instrukcjami GOTO oraz GOSUB) należy zapisywać przy pomocy wartości zmiennych. Jeżeli np. odnosimy się 50 razy do linii 100, możemy zaoszczędzić ok. 300 bajtów, definiując wartość zmiennej Z100 i odwołując się np. GOSUB Z100.
7. Należy wystrzegać się wprowadzania nadmiernej ilości zmiennych -każda nowo wprowadzona zmienna wymaga rezerwacji 8 bajtów pamięci w tablicy wartości zmiennych oraz kilka bajtów w tablicy nazw zmiennych.
8. Należy wykasowywać zbędne miejsca w tablicach nazw oraz wartości zmiennych. Tablice te nie są uaktualniane nawet wtedy, gdy odpowiednie zmienne zostały usunięte z programu. Wykasowanie tablic można wykonać poprzez utrwalenie programu instrukcją LIST, wykonanie instrukcji NEW, po czym wczytanie programu instrukcją ENTER.
9. Należy wystrzegać się długich nazw zmiennych. Każda nazwa zmiennej zapisywana jest w tablicy nazw w kodzie ATASCII, a więc im krótsze nazwy zmiennych, tym krótsza tablica nazw.
10. Należy w miarę możności często używane komunikaty tekstowe zastępować wyrażeniami tekstowymi, po czym odwoływać się do odpowiednich zmiennych tekstowych.
11. Zmienne tekstowe należy wpisywać przy pomocy instrukcji LET. Ten sposób nadawania wartości zmiennym, nawet zawierającym dane liczbowe w cudzysłowie, wymaga mniej pamięci niż wykorzystanie instrukcji READ czy funkcji CHR$.
12. Linie programu powinny zawierać - tam, gdzie to możliwe - jak najwięcej instrukcji. Zamiana dwóch linii o jednej instrukcji każda w jedną linię o dwóch instrukcjach oszczędza 3 bajty RAM.
13. Błędem jest stosowanie pętli programowych oraz podprogramów wykorzystywanych tylko raz – instrukcje GOSUB oraz RETURN zajmują niepotrzebnie miejsce w pamięci jeżeli wykorzystywane są w programie tylko raz.
14. Należy starać się zastępować zmienne indeksowane zmiennymi tekstowymi (o ile ich długość nie przekracza 255 bajtów). Zapis zmiennej matrycowej w tablicy zajmuje 6 bajtów, podczas gdy zapis zmiennej tekstowej tylko 1 bajt.
15. Należy stosować komendy przesuwania kursora zamiast instrukcji POSITION.. Zapis parametrów X i Y instrukcji POSITION zajmuje w pamięci 15 bajtów, podczas gdy przesunięcie kursora o jedną pozycję tylko 1 bajt.

16. Instrukcje SETCOLOR należy zastępować odpowiednimi instrukcjami POKE - zaoszczędzi to 8 bajtów RAM.
17. Należy stosować kasowanie linii programu - patrz podrozdział "Zaawansowane techniki programowania".
18. Wczytywanie zdefiniowanych wcześniej danych należy dokonywać poprzez zmianę wskaźnika obszaru matrycowego - przepisanie wartości rejestru STARP pozwoli na zaoszczędzenie miejsca w tej części RAM.
19. Szeroko powinny być wykorzystywane krótkie procedury maszynowe, wywoływane z BASICu funkcją USR.
20. Należy stosować programy łączone w łańcuch - przykładem może być procedura inicjalizacyjną, która po wczytaniu do pamięci i uruchomieniu, sama powoduje odczytanie i uruchomienie kolejnej części programu.

ZAAWANSOWANE TECHNIKI PROGRAMOWANIA

Zrozumienie podstaw Atari BASIC stwarza możliwości kreowania wielu programów aplikacyjnych. Mogą się one opierać wyłącznie na operacjach BASICu, bądź też wykorzystywać szeroko procedury systemowe OS.

Przykład 1

Inicjalizacja zmiennych tekstowych. Program ten nadaje wszystkim symbolom zmiennej tekstowej o dowolne j długości tę samą wartość. BASIC kopiuje pierwszy bajt tekstu źródłowego do pierwszego bajta tekstu przeznaczenia, to samo czyni z bajtem drugim, trzecim itd. Jeżeli tekst przeznaczenia umieścimy w drugim bajcie tekstu źródłowego, wówczas całe wyrażenie tekstowe zapisane zostanie tymi samymi symbolami.

```
10 REM  Inicjalizacja wyrażenia tekstowego
20 DIM A$(1000)
30 A$(1)=”A”: A$(1000)=”A”
40 A$(2)=A$
```

Przykład 2.

Kasowanie linii kodu. Poprzez wykorzystanie procedur systemu operacyjnego, program może wykasować, bądź zmodyfikować linie własnego kodu. Edytor ekranowy może być zaprogramowany na akceptowanie danych z ekranu bez potrzeb wprowadzania ich przez użytkownika. Tak wiec po umieszczeniu odpowiednich instrukcji na ekranie, umieszczeniu kursora na początku tej listy i zastopowaniu programu, BASIC będzie odczytywał komendy uprzednio umieszczone na ekranie.

```
10 REM Przykład kasowania linii
20 GRAPHICS 0: POSITION 2, 4
30 ? 70: ? 80: ? 90: ? CONT
40 POSITION 2, 0
```

```
50 POKE 842,13: STOP
60 POKE 842,12
70 REM Te linie
80 REM zostaną
90 REM wykasowane
```

Przykład 3

Tworzenie grafiki PMG przy pomocy zmiennych tekstowych. Przykład ten ilustruje jedną z możliwości szybkiego zapisywania danych dotyczących grafiki PMG. Tekstowa zmienna indeksowana posiada zmieniony wskaźnik własnego obszaru matrycowego, wskazujący obszar pamięci grafiki PMG. Zapis wartości tej zmiennej, na przykład instrukcją LET, spowoduje umieszczenie tych danych obszarze grafiki ekranowej PMG z szybkością programu maszynowego.

```
100 REM Przykład zapisu grafiki PMG
110 DIM A$(512),B$(20)
120 X=X+1: READ A: IF A<>-1 THEN B$(X,X)=CHR$(A): GOTO 120
130 DATA 0,255,129,129,129,129,129,129,129,129,255,0,-1
2000 POKE 559,62: POKE 704,88
2020 I=PEEK(106)-16: POKE 54279,I
2030 POKE 53277,3: POKE 710,224
2040 VTAB=PEEK(134)+PEEK(135)*256
2050 ATAB=PEEK(140)+PEEK(141)*256
2060 OFFS=I*256+1024-ATAB
2070 HI=INT(OFFS/256): LO=OFFS-hi*256
2090 POKE VTAB+2, LO: POKE VTAB+3, HI
3000 I=60: Z=100: V=1: H=1
4000 A$(Y,Y+11)=B$: POKE 53248,Z
4010 Y=Y+V: Z=Z+H
4020 IF Y>213 OR Y<33 THEN V=-V
4030 IF Z>206 OR Z<49 THEN H=-H
4400 GOTO 4000
```

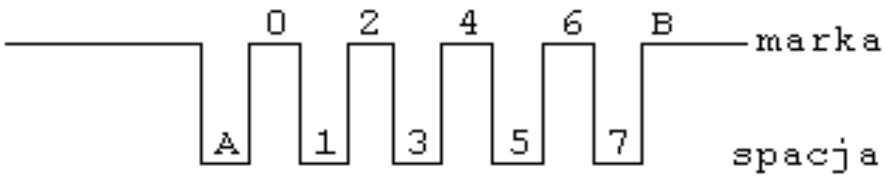
11. ZAPIS KASETOWY

STRUKTURA REKORDU

- Definicja bajta

System operacyjny zapisuje zbiory na taśmie magnetofonowej w blokach o zmiennej długości z szybkością 600 bodów (600 bitów na sekundę). Przesyłanie danych pomiędzy komputerem Atari 400/800 a magnetofonem kasetowym (tzw. Program Recorder) odbywa się metodą asynchronicznej transmisji szeregowej. Układ POKEY odczytuje lub zapisuje każdy bajt w następującej sekwencji: bit startowy (spacja), 8 bitów danych (0=spacja, 1=marka), bit końcowy (marka). Każdy bajt jest transmitowany w kolejności od najmłodszego do najstarszego bitu.

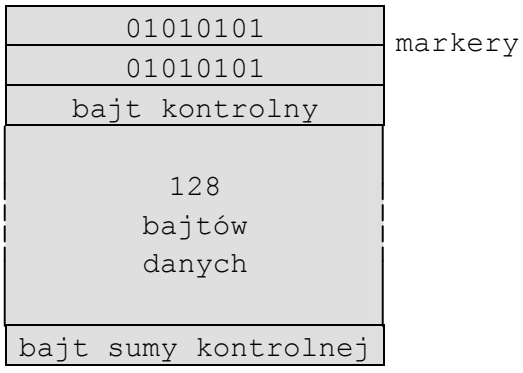
Marka tak utworzonego ciągu impulsów reprezentowana jest częstotliwością 5327 Hz, spacja natomiast częstotliwości 3995 Hz. Format jednego bajta danych jest więc następujący:



A = bit startowy
0-7 = bity danych
B = bit końcowy

- Definicja rekordu

Rekord ma długość 132 bajtów. Składają się na niego: dwa markery, służce do określenia szybkości przesuwu taśmy, bajt kontrolny, 128 bajtów danych oraz bajt sumy kontrolnej. Format rekordu jest następujący:



- Pierwszy i drugi marker

Każdy z markerów stanowi bajt o wartości S55 - a włączając do tego bity początkowy i końcowy, ma on długość 10 bitów. W warunkach idealnych pomiędzy dwoma bajtami markerów a pozostałymi bajtami rekordu nie powinna występować żadna przerwa w zapisie.
Markery służą do pomiaru szybkości przesuwania się taśmy.

- Pomiar szybkości

Nominalna szybkość zapisu kasetowego wynosi 600 bodów. Jednakże procedura systemowa SIO została zaplanowana w ten sposób, by mogła ona uwzględnić nierównomierności pracy silnika magnetofonu, rozciągliwość taśmy itp. Po skalkulowaniu właściwej szybkości przesuwu taśmy dokonane zostają poprawki w procedurach sprzętowych. Przy użyciu tej metody SIO może - teoretycznie - odczytywać zapis kasetowy przy szybkości od 318 do 1407 bodów.

Pomiar szybkości przesuwu taśmy odbywa się w następujący sposób: procedura systemu operacyjnego sprawdza w sposób ciągły wartość bitu odczytu szeregowego układu POKEY. Po odczytaniu wartości 0 - bit startowy, oznaczający początek rekordu - OS zapisuje aktualną wartość licznika ramki telewizyjnej, czyli rejestru 'VCOUNT układu ANTIC. Wartość bitu odczytu szeregowego kontrolowana jest w dalszym ciągu, aż do momentu odczytania 20-tego bitu, będącego końcowym bitem drugiego markera. Teraz OS, ponownie wykorzystując rejestr VCOUNT, określa ilość czasu, który upłynął pomiędzy obydwoma bitami, a na podstawie tej wartości obliczana jest właściwa szybkość przesuwu taśmy. Pomiar taki wykonywany jest każdorazowo przed odczytaniem rekordu.

- Bajt kontrolny

Bajt kontrolny może zawierać jedną z trzech wartości:

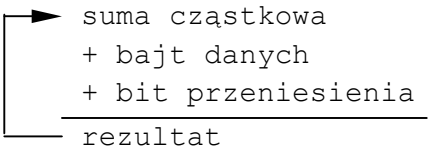
- $FC oznacza rekord zapełniony danymi (128 bajtów)
- $FA oznacza rekord krótszy, składający się z mniejszej ilości niż 128 bajtów. Sytuacja taka może wystąpić tylko wtedy, jeżeli rekord ten poprzedza rekord oznaczający koniec zbioru (EOF - End ot File). Ilość danych, zawartych w krótkim rekordzie, zapisywana jest jako ostatni bajt danych, bezpośrednio poprzedzający bajt sumy kontrolnej.
- $FE oznacza rekord będący końcem zbioru (EOF) i składający się ze 128 bajtów zerowych.

- Suma kontrolna bajtów

Suma kontrolna jest generowana oraz sprawdzana przez procedurę SIO, jednakże bajt ten nie jest zapisywany w buforze I/O sterownika kasety - CASBUF ($03FD).

Suma kontrolna stanowi jednobajtową sumę wszystkich bajtów danego rekordu, włączając w to dwa markery. Wartość każdego bajta rekordu

dodawana jest do sumy kontrolnej z uwzględnieniem bitu przeniesienia:



ZALEŻNOŚCI CZASOWE

- Przerwy międzyrekordowe (IRG)

Jak zaznaczono wyżej, każdy rekord kasetowy składa się ze 132 bajtów, włączając w to dwa markery, bajt kontrolny oraz sumę kontrolną bajtów. Poszczególne rekordy oddzielone są na taśmie - generowanymi przez sterownik kasety - przedrekordowym tonem podstawowym (PRWT) i przerwą porekordową (PRG). Zarówno PRWT jak i PRG stanowią czysty ton o wysokości marki (5327 Hz). Tak więc na przerwę międzyrekordową (IRG) pomiędzy dowolnymi dwoma rekordami na taśmie składa się PRWT mającego nastąpić rekordu oraz PRG rekordu zakończonego. Układ ten ilustruje poniższy schemat:

PRWT	MARKER	DANE	PRG	PRWT	MARKER	DANE
R E K O R D 1				R E K O R D 2		

- Normalne IRG i krótkie IRG

Długość zarówno PRWT jak i PRG zależy od parametrów zapisu, określonych bajtami AUX1 i AUX2 komendy OPEN. Rozróżnia się dwa typy przerw międzyrekordowych: normalne IRG oraz krótkie IRG.

Podczas otwierania danego zbioru komendą OPEN, typ przerw międzyrekordowych określa najstarszy bit parametru AUDC2. Wartość tego bitu określa sposób wykonywania przez sterownik kasety komend READ oraz WRITE odpowiednio podczas operacji wejścia i wyjścia.

Bit równy 1 oznacza, że kaseta ma być odczytywana lub zapisywana metoda krótkich IRG (zapis ciągły).

Bit równy 0 oznacza tryb normalnych IRG.

- Normalne IRG

Ten tryb pracy wykorzystywany jest np. podczas czytania kasety połączonego z wykonywaniem procesów określonych przez każdy rekord. Oznacza to, te taśma jest zatrzymywana po odczytaniu każdego rekordu. Oczywiście, o ile po zatrzymaniu taśmy wykonanie danego procesu nie zajmuje komputerowi zbyt wiele czasu, odczyt następnego rekordu następuje tak szybko, te zatrzymanie taśmy jest niezauważalne.

- Krótkie IRG

W tym trybie taśma nie jest zatrzymywana pomiędzy poszczególnymi rekordami, niezależnie czy jest ona zapisywana czy odczytywana. Podczas odczytu taśmy program musi sprecyzować parametry komendy READ tak szybko, by mogły być one zapisane w trakcie trwania krótkiej przerwy międzyrekordowej. Najpowszechniej stosowanym wykorzystaniem tego trybu pracy jest zapis oraz odczyt stokenizowanych programów BASICu - gdzie, podczas odczytu, BASIC nie musi pomiędzy poszczególnymi rekordami wykonywać żadnych innych procesów poza zapisywaniem w pamięci RAM odbieranych danych.

Instrukcje BASICu CSAVE oraz CLOAD narzucają właśnie ten tryb pracy sterownika kasety.

W tym trybie pracy występuje jednak potencjalne niebezpieczeństwo. Program zapisujący dane na kasetę, musi stworzyć na tyle długie przerwy międzyrekordowe, by podczas odczytu sterownik kasety nie miał trudności ze znalezieniem początków poszczególnych rekordów.

- Struktura czasowa zapisu

Struktura czasowa przerw międzyrekordowych jest następująca:

PRWT –	normalne IRG = 3 sek. tonu podstawowego
	krótkie IRG = 0.25 sek. tonu podstawowego
PRG -	normalne IRG = do 1 sekundy nieokreślonego tonu
	krótkie IRG = od 0 do N sekund tonu nieokreślonego, gdzie N jest uzależnione od programu zapisującego dane na kasecie.

Każdy rekord zapisywany jest w następującej sekwencji operacji: po uruchomieniu silnika magnetofonu zapisywany jest ton podstawowy PRWT, a czas jego zapisu uzależniony jest od trybu pracy, co podano powyżej. W dalszej kolejności zapisywany jest rekord danych, po czym ton podstawowy PRG. W trybie normalnych IRG w tym momencie silnik jest zatrzymywany, natomiast w trybie krótkich IRG natychmiast zapisywany jest ton podstawowy PRWT następnego rekordu.

Należy zwrócić uwagę, że w trybie normalnych IR zatrzymywanie i ponowne uruchamianie silnika magnetofonu stwarza niebezpieczeństwo wystąpienia na taśmie ciągu nieokreślonych danych, pochodzących z poprzedniego zapisu na tej samej taśmie. Bezwładność silnika podczas zatrzymywania i ponownego uruchamiania może - w zależności od typu magnetofonu - doprowadzić do powstania na taśmie luk zapisu o długości nawet do 1 sekundy. Stąd wynika konieczność całkowitego wykasowania taśmy przed wprowadzeniem na nią nowego zapisu.

- Dźwiękowa kontrola I/O

Dźwiękowa kontrola kasetowych operacji I/O jest znacznym udogodnieniem, pozwalającym na szybkie rozpoznanie właściwego bądź niewłaściwego odczytywania kasety - co ma szczególne znaczenie przy wykorzystywaniu instrukcji CLOAD BASICu. Różnica częstotliwości spacji i marki jest na tyle duża, iż użytkownik powinien szybko nauczyć się rozpoznawać właściwe i niewłaściwe brzmienie tonów wytwarzanych przez OS, a towarzyszących kasetowym operacjom I/O.

STRUKTURA ZAPISU KASETOWEGO

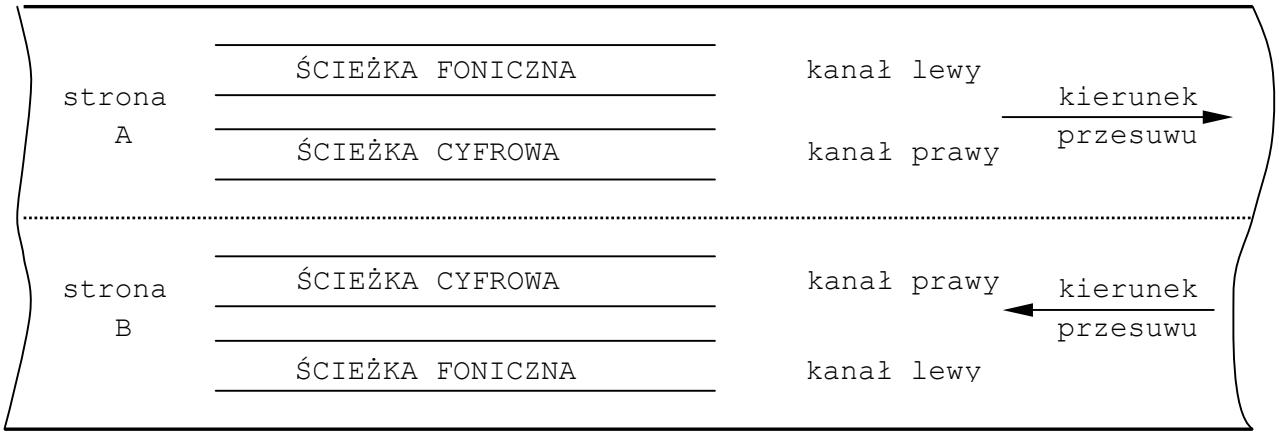
Każdy zbiór zapisany na kasecie musi się składać z następujących trzech elementów:

- Otwierające 20 sekund tonu podstawowego
- Pewna ilość rekordów danych
- Rekord oznaczający koniec zbioru

Po otwarciu danego zbioru podczas operacji wyjścia, OS zapisuje 20 sekund tonu podstawowego, po czym powraca do programu wywołującego, pozostawiając jednakże magnetofon nadal zapisujący ton podstawowy. Jeszcze przed powrotem do programu wywołującego OS ustawia licznik przekroczenia czasu operacji I/O na wartość odpowiadającą ok. 35 sekundom. Jeżeli czas ten upłynie przed zakończeniem zapisywania pierwszego rekordu, wówczas OS zatrzymuje taśmę, przez co tworzy się luka w zapisie pomiędzy otwierającym tonem podstawowym a tonem podstawowym PRWT pierwszego rekordu.

STRUKTURA TAŚMY MAGNETOFONOWEJ

Taśma kasetowa posiada dwie strony, oznaczane A i B. Na każdej stronie znajdują się dwie ścieżki dźwiękowe - odpowiadające ścieżkom lewego i prawego kanału w zapisie stereofonicznym - z których jedna przeznaczona jest na zapis informacji fonicznych, druga natomiast na zapis informacji cyfrowych. Układ ścieżek na taśmie przedstawia się następująco:



Tak więc taśma zapisywana jest w formacie 1/4 zapisu stereofonicznego przy standardowej szybkości 4,76 cm/sek. Stąd zapis kasetowy komputerów Atari 400/800 wymaga magnetofonu o stereofonicznej głowicy uniwersalnej.

BOOTING KASETOWY

Booting autobootingowego programu kasetowego może być wykonany podczas załączania komputera, jako część procedury inicjalizacyjnej systemu operacyjnego. Inicjalizacja systemu obejmuje takie procesy, jak wyzerowanie wszystkich rejestrów sprzętowych, wyzerowanie pamięci RAM, ustawienie rejestrów wskaźnikowych itp.

Po przepisaniu wszystkich rezydencyjnych sterowników urządzeń peryferyjnych - o ile podczas załączania wciśnięty został klawisz START - włączony zostaje rejestr flagowy żądania bootingu kasety, CKEY ($4A). OS, po odczytaniu wartości tego rejestru flagowego, rozpoczyna odczytywanie autobootingowego programu z kasety.

W celu wykonania bootingu kasetowego muszą zostać wykonane następujące czynności:

1. Podczas załączania komputera powinien zostać wciśnięty klawisz START.
2. Do magnetofonu powinna zostać włożona kaseta, zawierająca zbiór o właściwym formacie bootingowym, po czym naciśnięty klawisz PLAY magnetofonu.
3. Zbiór znajdujący się na kasecie musi być zapisany w trybie krótkich IRG.
4. Po odebraniu sygnału dźwiękowego operator musi nacisnąć dowolny klawisz na klawiaturze komputera.

Jeżeli wszystkie te warunki zostaną spełnione, system operacyjny odczyta i zapisze w pamięci RAM program bootingowy z kasety, po czym przekaże kontrolę do odczytanego programu. Proces bootingu kasetowego można przedstawić w następujących punktach:

1. Pierwszy rekord zapisu kasetowego jest odczytywany i zapisywany w buforze sterownika kasety CASBUF.
2. Odczytywane są informacje zawarte w 6 pierwszych bajtach tego rekordu. Tych pierwszych 6 bajtów bootingowego programu kasetowego (analogicznie jak czołowych 6 bajtów bootingowego programu dyskowego) sformatowanych jest w sposób następujący:

BAJT	1	IGNOROWANY	
	2	LICZBA REKORDÓW	
	3	ADRES STARTOWY ŁADOWANIA	LSB
	4		MSB
	5	ADRES INICJALIZACJI	LSB
	6		MSB

- Pierwszy bajt nie jest wykorzystywany w procesie bootingu kasetowego.
- Drugi bajt zawiera liczbę 128-bajtowych rekordów kasetowych, które mają być odczytane w ramach procesu bootingu (włączając w to pierwszy rekord, zawierający te informacje). Liczba ta może się wahać od 1 do 255, przy czym 0 oznacza 256 rekordów.
- Bajty trzeci i czwarty zawierają adres, od którego ma się rozpocząć zapisywanie w pamięci RAM tegoż programu.
- Bajty piąty i szósty zawierają adres inicjalizacji, do którego zostanie przekazana kontrola po zakończeniu procesu bootingu. Naciśnięcie klawisza RESET także spowoduje przekazanie kontroli pod zapisany tu adres, pod warunkiem, że proces bootingu został zakończony.

Po zakończeniu kroku 2 utrwalone więc zostają:

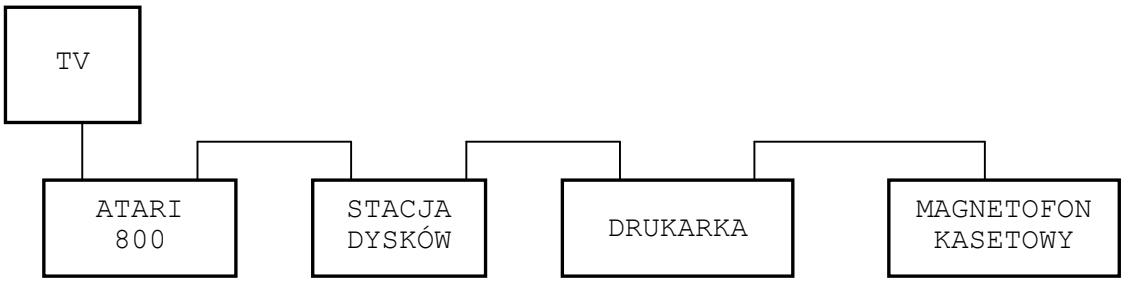
- ilość rekordów bootingowych, które mają być odczytane,
- adres początkowy ładowania programu,
- adres inicjalizacyjny programu - CASINI ($02,03).

3. Odczytany rekord zostaje przepisany z buforu do RAM, począwszy od wskazanego adresu początku ładowania.
4. Odczytywana jest wskazana ilość rekordów bootingowych, które również - poprzez bufor kasetowy - zostają przepisane do pamięci RAM bezpośrednio za rekordem pierwszym.
5. Wykonywany jest skok JSR do adresu o 6 bajtów wyższego od adresu ładowania, gdzie może znajdować się program wykonujący booting, wieloetapowy. Powodzenie tej operacji sygnalizowane jest przez bit przeniesienia © rejestru flagowego 6502 podczas wykonywania RTS (C=1 oznacza błąd).
6. Wykonywany jest skok JSR, pośrednio poprzez rejestr CASINI, inicjalizujący program aplikacyjny. Program aplikacyjny powinien zapisywać swój adres startowy w rejestrze DOSVEC ($0A,0B) podczas inicjalizacji, po czym wykonywać RTS.
7. Wykonywany jest skok JMP pośrednio poprzez DOSVEC, przekazujący kontrolę programowi aplikacyjnemu.

Po zakończeniu bootingu programu aplikacyjnego, naciśnięcie klawisza RESET spowoduje ponowne wykonanie operacji zawartych w punktach 6 i 7.

JAK DOŁĄCZYĆ MAGNETOFON DO SYSTEMU ATARI

Większość urządzeń peryferyjnych, komunikujących się z komputerem za pośrednictwem szyny szeregowej wyposażonych jest w dwa identyczne gniazda, z których jedno służy do podłączenia urządzenia do komputera, drugie natomiast przeznaczone jest do rozbudowy systemu. Pozwala to na łączenie urządzeń w ciągi poprzez szeregowe dołączanie kolejnych peryferiów, jak na przedstawionym przykładzie:



Jednakże magnetofon kasetowy Atari 410 nie może być dołączony pomiędzy inne urządzenia peryferyjne - musi on być ostatnim urządzeniem w tak utworzonym szeregu, jako że nie posiada drugiego równoległego gniazda komunikacji szeregowej tak jak inne urządzenia. Fakt ten świadczy, iż do komputerów Atari 400/800 nigdy nie może być dołączonych więcej niż jeden magnetofon kasetowy. Magnetofon nie jest urządzeniem "inteligentnym" - w odróżnieniu od innych peryferiów - a więc system nie ma możliwości sprawdzenia, czy magnetofon jest dołączony do szeregu czy też nie jest. Stąd też magnetofon może być dołączany i odłączany od systemu w zależności od potrzeb, bez żadnych ograniczeń.

Zawsze podczas otwierania ładunku kasetowego, niezależnie czy do czytania czy do zapisu, muszą być wykonane następujące czynności:

- Wejście (dane z magnetofonu do komputera). Po otwarciu urządzenia C:- magnetofonu kasetowego - generowany jest pojedynczy sygnał dźwiękowy. Jeżeli kaseta jest przygotowana (magnetofon włączony, połączony z komputerem, kaseta włożona), użytkownik musi wcisnąć klawisz PLAY magnetofonu, a następnie dowolny klawisz z klawiatury Atari 400/800 (za wyjątkiem klawisza BREAK), co spowoduje rozpoczęcie odczytywania kasety.
- Wyjście (dane z komputera do magnetofonu). Po otwarciu urządzenia C: generowane są dwa krótkie sygnały dźwiękowe. Jeżeli kaseta jest przygotowana (jak wyżej), użytkownik musi jednocześnie wcisnąć klawisze PLAY i RECORD magnetofonu, a następnie dowolny klawisz z klawiatury Atari 400/800 (za wyjątkiem klawisza BREAK), co spowoduje rozpoczęcie zapisywania danych na kasecie.

ZAPISYWANIE I ŁADOWANIE PROGRAMÓW BINARNYCH

Poniżej opisano techniki zapisywania danych cyfrowych bezpośrednio z pamięci komputera, poprzez port I/O, na kasecie znajdującej się w firmowym magnetofonie kasetowym.

z BASICu:

Format: CSAVE
100 CSAVE

Instrukcja ta, używana głównie w trybie bezpośrednim, wykorzystywana jest do zapisania znajdującego się w pamięci RAM

programu na taśmie. CSAVE zapisuje na kasecie stokenizowaną formę programu BASICu.

Format: CLOAD
100 CLOAD

Instrukcja ta, używana tak w trybie bezpośrednim jak i programowym, powoduje odczytanie stokenizowanej wersji programu BASICu z kasety i zapisanie jej w pamięci RAM.

z języka ASEMBLERA

program źródłowy
Format: LIST #C:[XX,YY]

Instrukcja ta używana jest do zapisywania kodu programu źródłowego asemblera na kasecie. Parametry, przedstawione w nawiasach kwadratowych, mogą być wykorzystane do zapisania na taśmie jedynie wierszy od XX do YY programu. Jeżeli parametry te nie zostaną wyszczególnione, zapisany zostanie cały program źródłowy.

Format: ENTER #C:

Instrukcja ta odczytuje kod programu źródłowego z kasety.

Program wynikowy
Format: SAVE #C:<XXXX,YYYY>

Na taśmie zapisywana jest zawartość bloku pamięci pomiędzy adresami XXXX i YYYY.

Format: LOAD #C:

Instrukcja ta powoduje odczytania utrwalonego uprzednio kodu programu wynikowego. Adresy, pomiędzy którymi program ten zostanie zapisany w pamięci RAM, są te same, jakie użyte zostały podczas zapisu programu na taśmie instrukcją SAVE.

ZAPISYWANIE PROGRAMÓW Z PODKŁADEM FONICZNYM

Technika ta nie oznacza, oczywiście, możliwości wykorzystania ścieżki fonicznej taśmy do rejestracji programów cyfrowych. Ścieżka foniczna może być wykorzystywana wyłącznie jako podkład, np. w celu urozmaicenia oczekiwania na zakończenie monotonnego procesu ładowania programu z kasety.

Krok 1: Nasz przykład tworzenia fonicznej oprawy programu cyfrowego dotyczy rejestracji kasety na technicznym, laboratoryjnym urządzeniu Atari, które w znakomity sposób nadaje się do postawionego przed nami zadania. Niestety, na obecnym etapie rozwoju sprzętu Atari stworzenie bardziej rozbudowanej oprawy fonicznej kaset przy wykorzystaniu magnetofonu firmowego bezpośrednio przez użytkownika jest niemożliwe. Użytkownik może jedynie - przy pomocy zwykłego magnetofonu stereofonicznego - nagrać na lewym kanale kasety zawierającej program prosty podkład muzyczny.

Stworzenie podkładu fonicznego wymaga w pierwszej kolejności całkowitego zapisania na kasecie programu cyfrowego.

Krok 2: Po utrwaleniu programu cyfrowego, należy przełączyć urządzenie na zapisywanie lewego kanału i tu nagrać oprawę foniczną.

PROGRAMY BINARNE, PODKŁAD FONICZNY, ZNAKI SYNCHRONICZNE ORAZ GOSPODARKA EKRANEM

Program odczytujący cyfrowe dane z kasety, nie posiada żadnej kontroli nad zapisem i odczytem ścieżki fonicznej. Problem synchronizacji obu ścieżek został rozwiązany poprzez wprowadzenie znaków synchronicznych, informujących program o zakończeniu odtwarzania danego segmentu fonicznego (będącego czy to podkładem muzycznym, czy zbiorem słownych instrukcji, w zależności od potrzeb).

Ściślej - mimo iż zapis foniczny nie ma w żadnym stopniu struktury rekordu - znak synchroniczny, zapisany na ścieżce cyfrowej, możemy przyrównać do znaku określającego koniec rekordu, a odnoszącego się do lewej ścieżki fonicznej kasety. Dla przykładu, program odczytujący znak synchroniczny może w zależności od jego wystąpienia wykonywać różne operacje, czy to zatrzymać silnik magnetofonu, czy to odtworzyć następny blok informacji fonicznych.

A oto przykład:

Krok 1: Programista opracował scenariusz podkładu fonicznego do programu ŻABKA. Wygląda on następująco:

(MUZYKA) Dziś mam zamiar opowiedzieć wam zabawną historię pod tytułem "Księżniczka i Żabka". To naprawdę zabawna opowieść, warto jej posłuchać. §

(MUZYKA) Zanim jednak zacznę, chciałbym wiedzieć z kim rozmawiam. Jak masz na imię? Proszę, napisz swoje imię i wciśnij RETURN. (PAUZA)

(MUZYKA) Zacznijmy więc opowieść: Kiedyś, dawno temu, żyła w pewnym zamku przepiękna księżniczka, a na imię miała YYYY. §

(MUZYKA) Pewnego słonecznego dnia księżniczka przechadzała się właśnie... §

- § - oznacza, że program w tym miejscu oczekuje na znak synchroniczny. Najlepiej, jeżeli dalsza część informacji fonicznych po odczytaniu tegoż znaku będzie odtwarzana z krótkim, powiedzmy półsekundowym opóźnieniem.

- PAUZA - oznacza, że lektor musi w tym miejscu wykonać dłuższą przerwę, uwzględniającą konieczność zatrzymania oraz ponownego uruchomienia silnika magnetofonu. Każdy segment informacji fonicznych powinien mieć długość od 10 do 30 sekund, jako że zbyt często umieszczone znaki synchroniczne mogą spowodować wystąpienie błędów w programie.

Krok 2: Wskazane jest, by programista przed rozpoczęciem kodowania stworzył generalny plan programu, uwzględniający w szczególności korelację pomiędzy informacjami fonicznymi a obrazami pojawiającymi się na ekranie.

Przykład takiego planu pokazany został na rys. 11.1.

Krok 3: Teraz można przystąpić do rozpoczęcia kodowania. Załóżmy, że program ma postać następującą:

BLOK	AUDIO	EKRAN	CZEKANIE NA ZNAK SYNCHRO	SILNIK
1				
2	Dziś mam zamiar...	Tytuł, grafika		
3			Tak	
4	Zanim jednak...	Tytuł, grafika „Podaj swoje imię”		
5			Tak	
6				Stop
7		Wpisanie imienia		
8				
9		Kasowanie ekranu		Start
10	Zacznijmy więc...	Grafika 1		
11			Tak	
12	Pewnego dnia...	Grafika 2		
13	...	...	...	...

Rys. 11-1. Przykład planu korelacji ścieżki fonicznej i ekranu.

```
10 REM program "żabka”, demonstrujący synchronizację
20 REM ścieżek fonicznej i cyfrowej na kasecie
30 REM
40 DIM IN$(20)
50 POKE; 54018,52: REM włączenie silnika magnetofonu
60 GRAPHICS 1
70 PRINT #6; "KSIĘŻNICZKA I ŻABKA": PRINT #6;... : REM tytułowy obraz graficzny
80 GOSUB 1000: REM oczekiwanie na znak synchroniczny, czyli na zakończenie pierwszego bloku audio
100 POSITION X,Y: PRINT #6; "PODAJ SWOJE IMIĘ"
110 POKE 54018,60: REM wyłączenie silnika, oczekiwanie na
115 REM wprowadzenie danych z klawiatury
120 INPUT IN$: REM wczytanie imienia użytkownika
130 POKE 54018,52
```

```
135 PRINT #6:CHR$(125): REM kasowanie ekranu
140 POSTTION X,Y: PRINT #6;IN$: PRINT #6;... : REM imię użytkownika
oraz grafika na ekranie
150 GOSUB 1000: REM oczekiwanie na zakończenie kolejnego bloku audio
160 PRINT #6;... : REM tworzenie nowego obrazu graficznego
```

PROGRAM OCZEKIWANIA NA ZNAK SYYNCHRONICZNY: ton podstawowy , na taśmie reprezentowany jest częstotliwością marki, podczas gdy znak synchroniczny jest krótkim impulsem spacji; program ten w sposób ciągły kontroluje wszystkie bajty odczytywane przez port szeregowy komputera, sprawdzając 5-ty bit każdego bajta. Bit ten równy 0 oznacza znak synchroniczny:

```
1000 IF INT(PEEK(53775)/32+0.5)=INT(PEK(53775)/32 THEN RETURN
1010 GOTO 1000
```

PROGRAM STEROWANIA SILNIKA MAGNETOFONU: włączenie bądź wyłączenie silnika można uzyskać Przez zapisanie rejestru 54018 (POKE) wartościami:

```
POKE 54018,52 włącza silnik
POKE 54018,60 wyłącza silnik
```

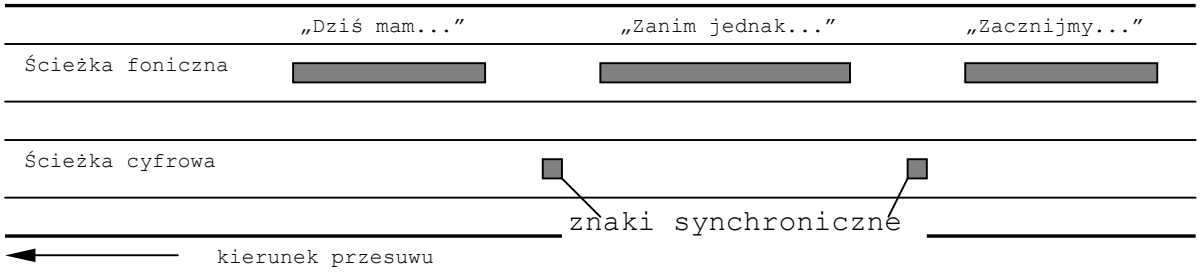
Krok 4 Po opracowaniu dokładnego scenariusza, tak podkładu fonicznego jak i programu, należy obliczyć czas oraz ilość taśmy potrzebną do wykonania zamierzonego celu. Jeżeli czas ten okaże się zbyt długi, należy tak przeorganizować scenariusz, by cały program mógł zmieścić się na jednej, niezbyt długiej kasecie.

Krok 5: Program należy utrwalić na taśmie wzorcowej.

Krok 6: Podkład foniczny powinien zostać zarejestrowany na oddzielnej, fonicznej taśmie wzorcowej.

Krok 7: Po utrwaleniu dwóch taśm wzorcowych należy dokonać zgrywania obu ścieżek, tworząc zapis źródłowy. Zapis taki należy tworzyć nagrywając najpierw program cyfrowy, a następnie dogrywając i synchronizując podkład foniczny; etap ten wymaga dysponowania trzema wysokiej klasy magnetofonami laboratoryjnymi.

Krok 8: Do komputera należy załadować program tworzący znaki synchroniczne na ścieżce cyfrowej. Znaki te powinny zostać zarejestrowane w tych miejscach taśmy, które - zgodnie ze scenariuszem - oddzielają od siebie kolejne bloki informacji fonicznych. Prawidłowo umieszczone znaki synchroniczne pokazane zostały na rysunku 11-2.



Rys. 11-2. Rozmieszczenie znaków synchronicznych.

Program zapisujący znaki synchroniczne może mieć następującą postać:

```
10 REM znak zapisywany jest przez naciśnięcie klawisza
20 REM START w odpowiednim miejscu taśmy
0 REM
40 REM
50 I0=53760: CONSOLE=53279: CASS=54018
100 FOR I=0 TO 8
110 READ J: POKE I0+I,J
120 NEXT I
125 REM pętle zapisujące rejestry generacji dźwięku
130 DATA 5,160,7,160,5,160,7,160,0
140 REM
150 REM parametry I/O już ustawione, teraz zapis
160 POKE CASS,52
200 POKE CONSOLE,8
210 IF PEEK(CONSOLE)<>7 THEN 230:
215 REM CONSOLE=7 oznacza zapisanie znaku, naciśnięty klawisz START
220 POKE I0+15,11: GOTO 200: REM klawisz START nie wciśnięty
230 POKE I0+15,128+11; GOTO 200: REM zapisanie znaku
```

Krok 9: W dwu niezależnych magnetofonach należy umieścić obie taśmy wzorcowe i przewinąć je obie do początku. Jeden z magnetofonów musi być podłączony do komputera Atari 800, na którym wczytany jest program zapisujący znaki synchroniczne. Drugi magnetofon będzie równolegle odtwarzał ścieżkę foniczną.

Krok 10: Teraz trzeba uruchomić program, ustawiając jednocześnie pierwszy magnetofon na zapisywania znaków na ścieżce cyfrowej, drugi zaś na odtwarzanie ścieżki fonicznej. Programista musi zapisywać znaki synchroniczne (przez naciśnięcie klawisza START) w tych miejscach taśmy, gdzie zapis foniczny zawiera wymagane scenariuszem pauzy.

Krok 11: Tak przygotowana taśma, zawierająca zarówno program zapisany na ścieżce cyfrowej, jak i informacje foniczne, zapisane na ścieżce fonicznej, może być wzorcem do masowego powielania stworzonego programu.

ZABRONIENIA KLAWISZA BREAK

W wielu przypadkach dogodnie jest wprowadzić zabronienie wykonania przerwania klawisza BREAK. Pozwoli to na uniknięcie występującego często błędu, jaki ma miejsce podczas przypadkowego naciśnięcia klawisze BREAK w trakcie zapisywania bądź odczytywania programów z kasety. System operacyjny nie stwarza możliwości powrotu do momentu, w którym nastąpiło przerwanie, a więc cała operacja musi być wykonywana od początku.

Program zabraniający wykonania przerwania po naciśnięciu klawisza BREAK może mieć następującą postać:

```
4000 X=PEEK(1): IF PEEK (16)<128 THEN 4020
4010 POKE 16,X-128: POKE 53774, X-128
4020 RETURN
```

Program ten powinien być wywoływany instrukcją GOSUB 4000 każdorazowo podczas zmiany trybu graficznego oraz podczas otwierania ekranu komendą OPEN.

12. ARTEFAKTY TELEWIZYJNE

W rozdziale tym przedstawiono sposób uzyskania na ekranie wielu kolorów w trybach graficznych jednokolorowych poprzez wykorzystanie artefaktów telewizyjnych.

Metoda ta może być stosowana w przypadku trybów graficznych ANTICu 2, 3 i 15. Tryb 2 ANTICu odpowiada trybowi 0 BASICu, tryb 15 ANTICu to tryb graficzny 8 BASICu, natomiast tryb tekstowy 3 ANTICu nie posiada odpowiednika wywoływanego z BASICu. Wielkość piksela w każdym z tych trybów jest identyczna i wynosi połowę punktu ekranowego na jedną linię skaningową. Generalnie tryby te operują jednym kolorem o dostępnych dwóch różnych wartościach luminancji. Przy wykorzystaniu artefaktów telewizyjnych we wszystkich tych trybach jednocześnie na ekranie mogą być wyświetlane piksele w czterech różnych kolorach.

Termin "artefakt telewizyjny" używany jest do określenia punktu ekranowego (czy też piksela), którego kolor jest inny, niż teoretycznie być powinien.

Prostym przykładem wykorzystania artefaktów przez komputery Atari400/800 mogą być następujące linie BASICu:

```
GRAPHICS 8:
COLOR 1
POKE 710, 0:
PLOT 60,60:
PLOT 63,60
```

Powyższe instrukcje spowodują wyświetlenie na czarnym tle ekranu dwóch punktów w odmiennych kolorach - mimo iż tryb graficzny 8 dysponuje teoretycznie tylko jednym kolorem.

Aby zrozumieć przyczyny powstawania różnokolorowych punktów na ekranie należy najpierw dokładnie zapoznać się z metodą przesyłania z komputera do telewizora wielu informacji przy pomocy jednego modulowanego sygnału.

Dwa podstawowe składniki takiego sygnału stanowią luminancja, czyli jasność punktu, oraz kolor, czyli jego barwa. Sygnałem podstawowym jest sygnał luminancji - zawiera on nie tylko informacje o jasności punktów, lecz także wskazuje, które punkty mają być wyświetlone, a które wygaszone. Sygnał koloru przenosi informacje o kolorze poszczególnych punktów i jest on kombinowany, czyli zmodulowany z podstawową falą luminancji.

Jasność każdego piksela na ekranie jest wprost proporcjonalna do amplitudy sygnału luminancji występującej w określonym momencie jej przebiegu. Im wyższa jest amplituda sygnału, tym jaśniejszy punkt jest wyświetlany na ekranie. Informacja o kolorze punktu jest sygnałem przesuniętym w fazie.

Sygnał przesunięty w fazie jest falą, oscylującą ciągle, przesuniętą o pewien okres czasu w stosunku do sygnału odniesienia. To właśnie, przesunięcie czasowe, jego wielkość, jest informacją o kolorze danego punktu. Sygnał koloru oscyluje ze stałą częstością wokół częstotliwości 3,579 MHz, która to częstotliwość jednocześnie definiuje maksymalną rozdzielczość jednej linii skaningowej na

ekranie. Określa ona maksymalną ilość punktów ekranowych jednej linii na 160 (nie licząc punktów wygaszonych, sygnałów synchronizacji oraz punktów wyświetlanych poza granicami ekranu).

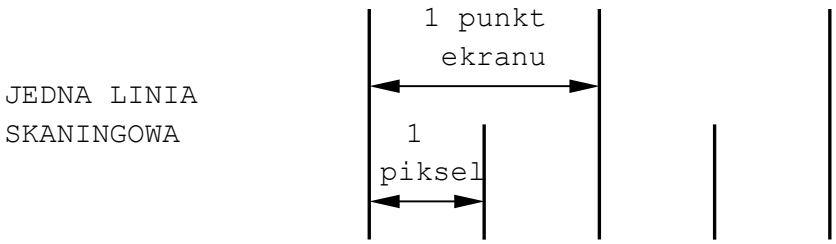
Termin "punkt ekranowy" odpowiada więc jednemu okresowi sygnału koloru i jest powszechnie stosowany jako miara odległości poziomej w jednej linii skaningowej ekranu. Tryb graficzny 7 jest przykładem trybu o rozdzielczości jednego piksela, gdzie każdy piksel - równy punktowi ekranowemu - może mieć odmienny kolor (przynajmniej teoretycznie, nie uwzględniając ograniczeń mikroprocesora).

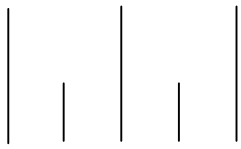
Atari udostępnia ponadto tryby o większej rozdzielczości, np. tryb graficzny 8, które wyświetlają 320 pikseli w jednej linii skaningowej. Uzyskuje się to poprzez zmiany amplitudy sygnału luminancji wokół średniej częstotliwości 7.16 MHz, co jest częstotliwością dwukrotnie wyższą od częstości sygnału koloru.

Ponieważ oba te sygnały są - przynajmniej teoretycznie - niezależne od siebie, proste jest niezależne przesłanie informacji o kolorze całego ekranu, czyli tła, a następnie odpowiednie ukształtowanie fali luminancji tak, by punkt za punktem przesyłała informacje o jasności poszczególnych pikseli. W ten właśnie sposób działa tryb graficzny 8, który informację o kolorze tła pobiera z rejestru kolorów pola 2, natomiast informacje o jasności poszczególnych punktów czerpane są z rejestrów kolorów pola 1 (punkty wyświetlane) i 2 (punkty tła).

Problem polega na tym, że w praktyce sygnały luminancji i koloru nie są całkowicie od siebie niezależne. Oba są częścią składową zmodulowanego sygnału, który w telewizorze ulega demodulacji na czynniki składowe. Ponieważ sygnał luminancji jest sygnałem podstawowym, każda zmiana tego sygnału pociąga za sobą także zmiany w przesunięciu fazowym sygnału koloru. Jest to niezauważalne, jeżeli wielkość piksela jest równa lub większa od jednego punktu ekranowego - zmiana przesunięcia fazowego sygnału koloru przy zmianie sygnału luminancji informującej o konieczności wyświetlenia danego punktu, będzie taka sama w przypadku wszystkich punktów ekranowych. Inaczej sprawa przedstawia się, jeżeli zmiana sygnału luminancji następuje w momencie odpowiadającym wyświetleniu połowy punktu ekranowego - spowoduje ona pewne zachwianie w stałym przesunięciu fazowym sygnału koloru. Co więcej powstały w ten sposób kolor jest całkowicie niezależny od urządzenia kontrolującego sygnał koloru, a więc od komputera Atari 400/800.

Zmiana sygnału luminancji w połowie wyświetlania punktu ekranowego narzuca możliwość powstania dwóch rodzajów "fałszywych" kolorów, czyli dwóch rodzajów pikseli artefaktowanych. Ponieważ piksele te mogą być ze sobą kombinowane w kolejne dwa piksele - tym razem o wielkości punktu ekranowego - możliwe jest uzyskanie jednocześnie czterech różnych "fałszywych" kolorów. Ilustruje to poniższy schemat:





0	1	0	0	Piksel o wielkości ½ punktu ekr. kolor A
1	0	0	0	Piksel o wielkości ½ punktu ekr. kolor B
1	1	0	0	Piksel o wielkości 1 punktu ekr. kolor C
0	1	1	0	Piksel o wielkości 1 punktu ekr. kolor D

Luminancja: 0 – punkt wygaszony 1 – punkt zapalony

Należy jeszcze zaznaczyć, że uzyskiwane w ten sposób na ekranie różnokolorowe piksele wymagają, by oddzielone były na odległość co najmniej 1 punktu ekranowego między sobą.

Kolory od A do D są różne dla różnych typów telewizorów, i zależą głównie od rodzaju i barwy poszczególnych luminoforów, jak i od rodzaju maski ekranowej. Stąd nie mogą one być opisane jako kolory absolutne - na przykład A = czerwony; jedyne co można powiedzieć, to fakt, iż są to cztery pastelowe, silnie różniące się między sobą barwy.

Aby zilustrować najprostsze zastosowania artefaktowania, odwołajmy się jeszcze raz do przedstawionego niżej programu. Rysuje on linie w każdym z czterech artefaktowanych kolorów, po czym wypełnia ograniczony obszar trzema spośród czterech kolorów, Trzeba tu jeszcze zaznaczyć, że wyświetlenie zbyt dużej ilości pikseli w kolorze C lub D w bezpośrednim sąsiedztwie spowoduje powstanie linii o stałej luminancji i kolorze tła.

Instrukcja POKE 87,7 powoduje, te system operacyjny traktuje ten tryb graficzny jako tryb 7 i używa maski 2-bitowej do organizacji pamięci ekranowej. Kolor A możemy generować instrukcją COLOR 1, kolor B wykorzystuje COLOR 2, kolor D natomiast COLOR 3. Kolor D powstaje, jeżeli z prawej strony piksela w kolorze COLOR 1 znajdzie się piksel w kolorze COLOR 2.

```
10 GRAPHICS 8: POKE 87,7: POKE 710,0: POKE 709,14
20 COLOR 1: PLOT 10,5: DRAWTO 10,70
30 PLOT 40,5: DRAWTO 40, 70
40 COLOR 2: PLOT 20,5: DRAWTO 20, 70
50 PLOT 41,5 DRAWTO 41,70
60 COLOR 3: PLOT 30,5: DRAWTO 30,70
70 FOR X=1 TO 3: COLOR X: POKE 765,X
80 PLOT X*25+60,5: DRAWTO X*25+60,70
90 DRAWTO X*25+40,70: POSITION X*25+40,5
100 XIO 18,#6,12,0,"S:”
110 NEXT X
```

13. GTIA

GTIA to nowy układ scalonego interfejsu telewizyjnego, który w najbliższym czasie zastąpi używany w komputerach Atari 400/800 układ CTIA. GTIA niewiele się różni od CTIA, stwarza jedynie kilka nowych możliwości - są to trzy nowe tryby graficzne, oparte na trzech nowych sposobach interpretowania informacji przesyłanych przez układ ANTIC. A więc ANTIC nie wykorzystuje żadnego nowego sposobu komunikowania się z GTIA, jedynie tryb ANTICu może być przez GTIA interpretowany na trzy różne sposoby. Poza tym GTIA jest w pełni kompatybilny z CTIA. W poprzednich rozdziałach opisane były możliwości układu CTIA, poniżej przedstawiamy więc tylko różnice pomiędzy GTIA a CTIA.

CTIA przeznaczony został do tworzenia sygnału, rysującego obraz na ekranie telewizyjnym. Może on wyświetlać pola ekranowe, niezależne obiekty ekranowe (Player/Missile), oraz wykrywać nakładanie się na siebie i kolizje obiektów ekranowych. CTIA interpretuje dane, przesyłane przez ANTIC, w rozbiciu na sześć różnych trybów tekstowych i osiem trybów graficznych. Na obrazie statycznym dane pochodząca z ANTICu służą do wyświetlania punktów o barwie i jasności zdefiniowanej przez jeden z czterech rejestrów kolorów. GTIA rozszerza te możliwości, dysponując wszystkimi dziewięcioma rejestrami kolorów (tak pola jak i obiektów PMG), bądź wyświetlając punkty o 16 różnych barwach jednakowej jasności lub o 16 różnych jasnościach jednej barwy - oczywiście, w przypadku obrazu statycznego.

Trzy nowe tryby graficzne GTIA są niczym innym, jak trzema nowymi interpretacjami trybu graficznego ANTICu o najwyższej rozdzielczości, trybu $F. Wszystkie trzy tryby dotyczą wyłącznie pola ekranowego. Możliwości wyświetlania obiektów ekranowych, uzyskiwania za ich pomocą nowych kolorów, możliwości zamiany zawartości rejestrów kolorów "w locie" przy wykorzystaniu przerwań listy displejowej - to wszystko pozostaje bez zmian również w tych trzech trybach graficznych. GTIA odczytuje cztery bity danych, dotyczących każdego piksela, pochodzące z układu ANTIC. Każdy piksel ma szerokość dwóch punktów ekranowych, natomiast wysokość jednej linii skaningowej; a więc piksele są cztery razy szersze od wysokości jednej linii. Obraz posiada rozdzielczość 80 pikseli w poziomie i 192 piksele w pionie. Każda linia skaningowa wymaga więc 320 bitów, czyli 40 bajtów pamięci ekranowej - tyle samo, co w trybie $F ANTICu. Stąd program, wykorzystujący jeden z trybów graficznych GTIA musi dysponować co najmniej 8K pamięci RAM na pamięć ekranową.

Tryby graficzne GTIA wybierane są przy pomocy rejestru priorytetów ekranowych PRIOR. Rejestr ten znajduje się pod adresem $D01B w układzie GTIA i jest cieniowany przez OS rejestrem $26F. Bitami kontrolnymi są bity D6 oraz D7. Jeżeli żaden z tych bitów nie jest włączony, wówczas GTIA zachowuje się identycznie jak CTIA. Jeżeli bit D7 jest równy 0, natomiast bit D6,równy 1, wówczas ekran jest zagospodarowywany w trybie graficznym 9, dysponującym 1 kolorem o 16 różnych wartościach luminancji. Ponieważ ANTIC rezerwuje w pamięci ekranowej 4 bity dla każdego piksela ekranu, stąd istnieje 16 różnych wartości, a więc 16 różnych luminancji jednego koloru. Oczywiście, dodatkowe wartości luminancji mogą być uzyskane przez

zastosowania grafiki PMG. Jeżeli bit D7 równy jest 1, natomiast bit D6 ma wartość 0, wówczas wyświetlany jest tryb graficzny 10. Tryb ten dysponuje 9 różnymi kolorami, wykorzystując 4 rejestry kolorów pola, 4 rejestry kolorów obiektów PMG oraz rejestr koloru tła. Jeżeli w trybie tym wykorzystywane są obiekty PMG, analogicznie wyświetlane są one w kolorze własnych rejestrów kolorów, tak samo jak w innych trybach graficznych. Jeżeli oba bity, D7 i D6, mają wartość 1, wówczas wyświetlany jest tryb graficzny 11. Tryb ten dysponuje 16 różnymi barwami punktów, przy jednej tylko wartości luminancji (w analogii do trybu 9, 4 bity danych mogą przyjmować 16 różnych wartości). Dodatkowe wartości luminancji można w tym trybie uzyskać poprzez wykorzystanie obiektów PMG.

rejestr PRIOR

D7	D6	..	..	..	..	..	..

D7	D6	Rodzaj pracy	Tryb
0	0	Tryby GTIA wyłączone (CTIA)	0 – 8
0	1	1 barwa, 16 luminancji	9
1	0	9 barw/luminancji	10
1	1	16 barw, 1 luminancja	11

Rys. 13-1. Tryby graficzne GTIA włączane bitami D7 i D6

Wykorzystanie nowych trybów graficznych, udostępnionych przez GTIA, jest równie proste, jak przełączanie trybów w przypadku układu CTIA. Wywołanie tych trybów z BASICu uzyskuje się przez stosowanie instrukcji GRAPHICS 9, GRAPHICS 10 oraz GRAPHICS 11 odpowiednio w stosunku do trybów 9, 10 i 11. W języku asemblera wywołanie tych trybów jest identyczne, jak otwieranie ekranu w przypadku wszystkich innych trybów graficznych. Jeżeli natomiast mamy zamiar posłużyć się własną listą displejową, wybór tych trybów graficznych musi być dokonany przez bezpośrednie włączenie odpowiednich bitów rejestru PRIOR, jak podano na rys. 13-1.

Tryb 9 udostępnia maksymalnie do 16 różnych wartości luminancji tej samej barwy. ANTIC odczytuje dane ekranowe przy pomocy maski cztero-bitowej, co zapewnia możliwość wyboru jednej z 16 wartości luminancji danego punktu, natomiast barwa punktów odczytywana jest z rejestru kolorów tła. Przy pomocy BASICu odpowiednią barwę w rejestrze kolorów tła zapisujemy instrukcją SETCOLOR, zmieniając wartość jedynie starszego nibla tego rejestru. Format instrukcji jest następujący:

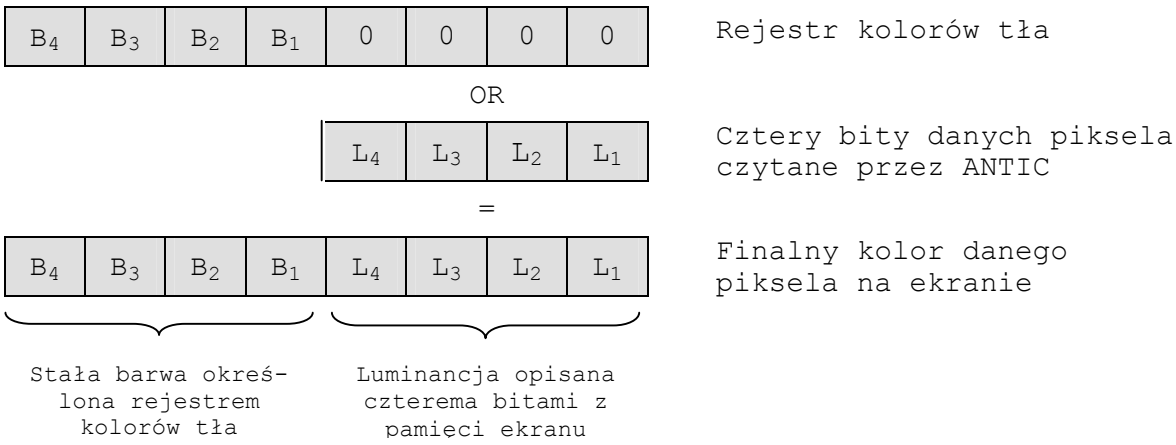
```
SETCOLOR 4,kod barwy,0
```

gdzie 4 określa rejestr koloru tła, kod barwy może być zapisany przez użytkownika żądaną wartością z zakresu od 0 do 15, 0 natomiast zapisuje jasność tła zerową, Tło musi być w tym przypadku całkowicie wygaszone, ponieważ ANTIC, odczytując każde cztery bity z pamięci ekranowej, określa jasność danego punktu wykonując operację logiczną

OR tych czterech bitów z czterema bitami określającymi luminancję w rejestrze kolorów tła. Wybór jasności poszczególnych punktów, podczas rysowania obrazu na ekranie, wykonuje się następnie instrukcją COLOR, po której mogą występować parametry z zakresu od 0 do 15, określające jasność punktów. A oto krótki program BASICu, demonstrujący tryb graficzny 9:

```
10 GRAPHICS 9 : REM wywołanie trybu graficznego 9
20 SETCOLOR 4,12,0: REM  zapis zielonej barwy rejestru tła
30 FOR I=0 TO 15
40 COLOR I: REM wykorzystanie instrukcji COLOR
50: PLOT 4,I+10: REM do zmiany jasności rysowanych punków
6o NEXT I
```

W języku Assemblera należy żądaną wartość barwy zapisać (od $0 do $F) w górnym niblu cienia OS rejestru kolorów tła, znajdującego się pod adresem S2C8. Jeżeli dane ekranowe zapisujemy przy pomocy wywoływania procedury CIO, odpowiednie wartości danych poszczególnych pikseli należy wpisywać do rejestru systemowego ATACHR, znajdującego się pod adresem $2FB. Zapisana tu wartość będzie określała luminancję piksela w zakresie. od $0 do $F. Jeżeli natomiast w sposób bezpośredni chcemy tworzyć obraz ekranowy, dane poszczególnych pikseli należy bezpośrednio wpisywać do obszaru pamięci ekranowej, umieszczając dane w górnym bądź w dolnym niblu każdego bajta RAM.



Rys. 13-2. Sposób odczytywania koloru punktów poprzez zastosowanie funkcji OR do wartości rejestru kolorów tła oraz danych z pamięci ekranowej w trybie graficznym 9.

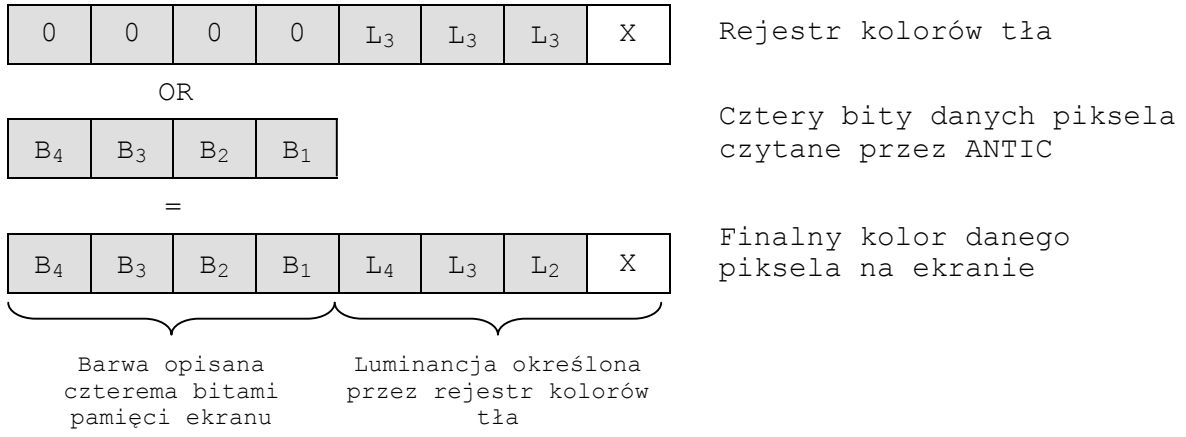
Tryb graficzny 11 jest bardzo zbliżony do trybu 9, z tym, że udostępnia 16 różnych barw o jednakowej jasności. Analogicznie ANTIC odczytuje dane z pamięci ekranowej, określające barwę poszczególnych punktów; podobnie jak poprzednio, należy wykorzystać instrukcję BASICu SETCOLOR do zapisania wybranej wartości luminancji w młodszym niblu rejestru kolorów tła, górny nibel, określający barwę, zapisując wartością 0. Format tej instrukcji jest następujący:

SETCOLDR 4,O,wartość luminancji

gdzie 4 precyzuje rejestr kolorów tła, 0 określa czarną barwę tła, natomiast wartość luminancji powinna zostać zapisana przez użytkownika wybraną wartością z przedziału od 0 do 15. Podobnie jak w innych trybach graficznych (poza trybem 9) najmłodszy bit nibla luminancji jest ignorowane, a więc rejestr ten powinien być zapisywany tylko parzystymi liczbami określającymi luminancję, co daje w efekcie 8 różnych wartości luminancji. Wybór odpowiedniej barwy dokonuje się przy użyciu instrukcji COLOR, której parametry mogą wahać się od 0 do 15. Analogicznie jak w trybie 9, finalna barwa punktu określana jest przez ANTIC z wykorzystaniem funkcji logicznej OR w stosunku do zapisu rejestru kolorów tła oraz danych z pamięci ekranowej. Poniżej przedstawiamy prosty program BASICu demonstrujący tryb graficzny 11.

```
10   GRAPHICS 11
20   SETCOLOR 4,0,12
30   FOR I-0 TO 15
40   COLOR I
50   PLOT 4,I+10
```

W języku Asemblera należy bezpośrednio wpisać żądaną wartość z przedziału od $0 do $F w młodszym niblu rejestru kolorów tła, którego cień mieści się pod adresem $2C8. Jeżeli wykorzystywane są wywołania procedury CIO, dane należy przepisywać poprzez rejestr ATACHR pod adresem $2FB. W ten sposób można wybrać barwę punktów wartościami z zakresu od $O do $F. Jeżeli obraz ekranowy zapisywany jest bezpośrednio w pamięci ekranowej, dane dotyczące barwy poszczególnych pikseli należy zapisywać a dolnych lub górnych niblach każdego z bajtów pamięci ekranowej RAM.



Rys. 13-3. Odczytywanie koloru punktów w trybie graficznym 11.

Tryb 10 pozwala na wykorzystanie wszystkich 9 rejestrów kolorów w stosunku do pola ekranowego jednocześnie. Każdy z tych rejestrów musi być, oczywiście, zapisany wcześniej odpowiednią wartością, będącą kombinacją barwy i luminancji. Dane, odczytywane przez ANTIC z pamięci ekranowej, służą do określenia rejestru kolorów, z którego czerpane będą dane dla danego piksela ekranu. Z BASICu, zgodnie z opisem znajdującym się w "Basic Reference Manual", do zapisania rejestrów kolorów tła oraz czterech rejestrów pola może być wykorzystana instrukcja SETCOLOR: Można to uczynić także instrukcją POKE, zapisując adresy od 708 do 712, gdzie znajdują się cienie OS rejestrów kolorów. Cztery rejestry kolorów obiektów PMG, znajdujące się pod adresami 704 - 707, mogą być zapisane wyłącznie przy pomocy instrukcji POKE. Podczas rysowania obrazu na ekranie, kolor piksela wybierany jest instrukcją COLOR. Jednakże przy pomocy tej instrukcji można wybrać tylko jeden z 9 rejestrów kolorów - jej argumentem może być liczba z przedziału od 0 do 8.

ANTIC odczytuje z pamięci ekranowej, podobnie jak w trybach 9 i 11, cztery bity danych dla każdego piksela - co pozwalałoby na wybór jednego z 16 rejestrów kolorów. Jednakże system sprzętowy ustala jedynie 9 rejestrów kolorów, stąd niedozwolone argumenty instrukcji COLOR, liczby od 9 do 15, spowodują wybór jednego z niższych rejestrów. Program BASICu, wywołujący tryb graficzny 10, musi zawierać:

instrukcję GRAPHICS 10

zbiór instrukcji POKE (lub instrukcji POKE oraz instrukcji SETCOLOR), zapisujących rejestry kolorów wybranymi wartościami barwy i jasności poszczególnych pikseli.

instrukcje COLOR, wybierające jeden z rejestrów kolorów do rysowania określonych pikseli na ekranie.

W języku Assemblera dane ekranowe powinny być zapisywane bądź przy pomocy rejestru ATACHR, bądź bezpośrednio w obszarze pamięci ekranowej, jak w przypadku trybów 9 i 11. Dane ekranowe nogą być wybrane z przedziału od 0 do 8, co precyzuje wybór następujących rejestrów:

Parametr instrukcji COLOR	Rejestr kolorów	Cień OS
1	$D012	$2C0
2	$D013	$2C1
3	$D014	$2C2
4	$D015	$2C3
5	$D016	$2C4
6	$D017	$2C5
7	$D018	$2C6
8	$D019	$2C6
9	$D01A	$2C8

Niezwykle ważnym zagadnieniem jest sprzętowa kompatybilność GTIA ze stosowanym dotąd układom CTIA. Układ GTIA jest w pełni kompatybilny z CTIA, a więc wszystkie programy, stworzone dla systemów opartych na CTIA, będą działały w ten sam sposób w systemach opartych na GTIA. GTIA również stwarza możliwość wykorzystywania niezależnych obiektów ekranowych, detekcji ich kolizji i nakładania się na siebie oraz stosowania przerwań listy displejowej. Nowe tryby graficzne GTIA są w pełni adaptowane przez system operacyjny, ponadto wszystkie możliwości graficzne, jakie udostępniał układ CTIA, są także do wykorzystania w systemach pracujących z GTIA.

GTIA udostępnia większą ilość kolorów, które mogą być równocześnie wyświetlano na ekranie. Dzięki GTIA w jednej linii skaningowej może nastąpić aż 16 zmian kolorów, całkowicie niezależnie od ograniczeń mikroprocesora. Jest to więcej, niż z układem CTIA można było osiągnąć nawet przy pomocy przerwań listy displejowej, gdzie można było wprowadzić jedynie 12 zmian. Tryby graficzne GTIA, 9, 10 i 11, pozwalają na uzyskanie większej głębi obrazu i delikatnego konturowania, co oznacza możliwość stosunkowo realistycznego tworzenia obrazów trójwymiarowych.

Z drugiej strony wprowadzenie nowego interfejsu stwarza również pewne niedogodności. Wszystkie nowe tryby GTIA są trybami graficznymi, a nie tekstowymi. Wyświetlenie w tych trybach tekstu jest więc możliwe tylko poprzez ingerencję w systemową listę displejową trybu. Piksele wszystkich trybów GTIA są długie i wąskie (stosunek 4:1), nie sprzyjają więc tworzeniu obrazów o zaokrąglonych liniach. Ponieważ dane ekranowe każdego piksela stanowi 1/2 bajta (4 bity), wykorzystanie tych trybów wymaga poświęcenia na obszar pamięci ekranowej aż 8K pamięci RAM. Poza tym, chociaż GTIA jest kompatybilny z CTIA, relacja ta nie jest odwrotna - stąd programy, napisane dla systemów pracujących z układem GTIA, nie będą dawały właściwego obrazu w systemach pracujących z CTIA; obraz ten będzie czytelny, becz pozbawiony wielu kolorów. Ponadto sam system nie ma możliwości sprawdzenia, czy współpracuje z CTIA czy z GTIA - tak jak może sprawdzić, czy interfejs będzie tworzył obraz w systemie NTSC, czy w PAL. Wreszcie wykorzystania artefaktów telewizyjnych przy pomocy układu GTIA spowoduje powstanie kolorów różnych od tych, jakie produkowane były przez CTIA – nawet jeżeli wykorzystany zostanie ten sam odbiornik telewizyjny.

14. GOSPODAROWANIE PAMIĘCIĄ

Wykorzystanie pamięci w komputerach domowych Atari może być nieco utrudnione, jako że stosunkowo duże obszary pamięci są bądź to zajmowane bądź rezerwowane przez system operacyjny, rezydujący cartridge (np. interpreter języka BASIC) oraz przez dyskowy system operacyjny. Pod bezpośrednią kontrolą programisty pozostaje stosunkowo niewielki obszar RAM, co oczywiście nakłada pewne ograniczenia w swobodnym dysponowaniu pamięcią RAM dla programu użytkowego. Przy właściwym gospodarowaniu pamięcią nie powinno to jednak stanowić poważnego problemu.

STRONA ZEROWA

Najważniejszą częścią pamięci dla programów napisanych w języku asemblera jest strona zerowa RAM. Na stronie zerowej umieszczone zostały jedynie najważniejsze wskaźniki systemu, a kod znajdujący się w tej części RAM jest łatwiej dostępny i szybciej wykonywany przez mikroprocesor. Zwykle każdy programista chciałby wiedzieć dokładnie, jak wiele rejestrów strony zerowej jest pozostawionych do jego dyspozycji. A wiec rozdział ten nie będzie omawiał dokładnie, w jaki sposób mogą być wykorzystane wskaźniki systemowe, znajdujące się na stronie zerowej, jako że te informacje zawarte są w wielu publikacjach firmowych. Postaramy się jedynie omówić tę część strony zerowej, która nie jest wykorzystywana przez system sprzętowy.

Dolna połowa strony zerowej, adresy od S00 do $80, zarezerwowana jest dla użytku systemu operacyjnego. Tych 128 bajtów przeznaczonych jest na wektory wielu operacji systemowych, wykonywanych przez OS. Wiele programów będzie z pewnością wykorzystywało jedynie część tych wektorów, jako że nie będą używane wszystkie procedury systemowe, a więc część rejestrów nie będzie potrzebna systemowi. W szczególności może to dotyczyć 43 bajtów, od $50 do $7A, które używane są przez edytor ekranowy oraz sterownik ekranu. Wiele programów wprowadza własne listy displejowe, oparte na własnym sterowniku ekranu, co jednocześnie uwalnia owe 43 bajty i oddaje je do dyspozycji programowi. Analogicznie można oczekiwać, że wiele rejestrów, przeznaczonych dla specjalnych procedur systemowych, także pozostanie niewykorzystanych, uwalniając jeszcze więcej miejsca na stronie zerowej RAM.

Niestety, rozumowanie takie może się okazać całkowicie błędne. Wydział programowy koncernu Atari bezustannie rozszerza możliwości systemu operacyjnego Atari, dokonując zmian w kodzie OS. Obecnie na rynku znajdują się dwie wersje systemu operacyjnego - wersja starsza "A" oraz nowa wersja "B". Obie wersja są niemalże identyczne i większość - lecz nie całość - oprogramowania stworzonego dla wersji "A" będzie działała poprawnie w przypadku wersji "B". Planuje się jednak wprowadzenie kolejnych zmian w systemie, a więc jest bardzo prawdopodobne, że owych 43 lub więcej bajtów ze strony zerowej zostanie inaczej zagospodarowanych w przyszłych wersjach systemu operacyjnego. Stąd też jakikolwiek program wykorzystujący te 43 bajty, czy też inne rejestry z dolnej połowy strony zerowej, może nie pracować poprawnie w systemach wyposażonych w nowe wersje systemu operacyjnego Atari 400/800. Tak więc programiści powinni

unikać wykorzystywania na szerszą skalę rejestrów, rezerwowanych na stronie zerowej dla użytku systemu operacyjnego.

Wolnych rejestrów należy szukać przede wszystkim w górnej połowie strony zerowej. Tych 128 bajtów zarezerwowanych zostało na użytek kartridża. Jeżeli kartridż nie jest używany, cały ten obszar pozostaje wolny. Jeżeli natomiast wykorzystujemy kartridż, do użytku programisty pozostanie zaledwie kilka bajtów, BASIC pozostawia niewykorzystanych jedynie 7 bajtów strony zerowej - adresy od $CB do $D1. W tym przypadku programista, któremu potrzebny jest większy obszar strony zerowej, ma tylko jedno wyjście: wykorzystanie bajtów używanych przez operacje zmiennoprzecinkowe, adresy od SD4 do $FF. Jest to możliwe tylko w tym przypadku; kiedy program nie wykorzystuje operacji zmiennoprzecinkowych, oraz gdy procedury systemowe, wywoływane przez ten program, nie wymagają operacji zmiennoprzecinkowych. Nawet w tym przypadku wykorzystanie owych 44 bajtów nie jest całkowicie dowolne - w obszarze tym nie mogą być zapisywane żadne programy obsługi przerwań, jako że może doprowadzić to do załamania się systemu podczas wywoływania przez BASIC jednej z procedur zmiennoprzecinkowych. Poza tym żadne inne rejestry ze strony zerowej nie mogą być wykorzystywane przez programistę,

Z drugiej strony programista, pracujący z interpreterem BASICu, nie jest zbytnio zainteresowany szybkością operacji oraz łatwością dostępu, jaką oferują rejestry strony zerowej. Jeżeli natomiast wymagana jest łatwość dostępu do pewnej liczby wskaźników oraz szybkość ich używania, w żadnym wypadku nie należy decydować się na pracę z bardzo powolnym interpreterem BASICu. Najbardziej zainteresowani szerokim dostępem do rejestrów strony zerowej są programiści pracujący w języku Asemblera. Bezpośrednia praca w języku asemblera nie wymaga instalacji jakiegokolwiek kartridża, a więc udostępnia wszystkich 128 rejestrów górnej połowy strony zerowej. W praktyce nie jest to aż tak proste, ponieważ program w języku asemblera musi być zdebuggowany, a to z kolei wymaga obecności programu debuggera - bądź to w pamięci RAM, bądź w postaci kartridża - a ten z kolei wykorzystuje wiele spośród rejestrów górnej połowy strony zerowej. Program ten pozostawia użytkownikowi zaledwie 32 bajty strony zerowej, adresy od $B0 do $CF. Oczywiście jest to ilość niewystarczająca jak na potrzeby dużego, skomplikowanego programu użytkowego. W górnej połowie strony zerowej debugger pozostawia wolne jeszcze inne, rozproszone rejestry. A są to adresy: $A4-A5, $AD-AE, $DB-E5, $EA-F1, $F5-F6, SF9-FB, $FE-FF. Tych 30 bajtów nie jest wykorzystywanych przez debugger, jeżeli jednak zamierzamy wykorzystać je w swoim programie, musimy wystrzegać się wywoływania czy to asemblera, czy edytora, które zajmują część spośród tych adresów. Ponadto nie należy w tym przypadku korzystać z Mini-Assemblera.

Jeżeli chcemy używać innych programów interpretacyjnych - czy to kartridżowych, czy wczytywanych do RAM z dysku - musimy niestety we własnym zakresie rozpoznać, które z rejestrów strony zerowej pozostają niewykorzystane przez ten program. Należy jednak założyć, że większość programów interpretacyjnych zajmuje do swoich celów dużą część rejestrów strony zerowej.

ABSOLUTNY RAM

Następny problem rodzi się wówczas, kiedy użytkownik wykorzystuje procedury DOSu. Nie jest proste stworzenie programu, który może pracować zarówno w 16K systemie kasetowym, jak i w 48K systemie dyskowym. Niestety, programy DOSu w znaczący sposób zmniejszają obszar dostępnej pamięci RAM – DOS łącznie z procedurami DUP niemal całkowicie wypełniają pamięć RAM systemów 16K. Istnieje kilka rozwiązań tegoż problemu. Najprostszym jest stworzenie dwóch wersji programu - wersji kasetowej oraz wersji dyskowej. Obie wersje programu mogą wówczas zajmować różne obszary w pamięci. Oznacza to równocześnie, że użytkownik pracujący z systemem kasetowym nie będzie w stanie przenieść swoich programów na dyskietki w przypadku rozbudowy posiadanego systemu.

Jest jeszcze inna metoda. Wszystkie systemy rezerwują stronę szóstą pamięci dla użytkownika. Zmienne oraz wektory programowe mogą być umieszczane przez program na stronie szóstej i wykorzystywane zarówno przez system kasetowy jak i dyskowy. W takim przypadku program może przy pomocy wektorów ze strony szóstej wykonywać skoki do procedur znajdujących się gdziekolwiek w pamięci RAM. Technika ta jest dość trudna w praktycznych zastosowaniach – z pewnością niezbyt się nadaje do obszernych projektów w asemblerze. Może mieć natomiast szerokie zastosowanie w przypadku programów o średniej objętości, rzędu 1 – 2K, a już z pewnością do podprogramów maszynowych wywoływanych z BASICu lub innego języka programowania.