Stołeczny Ośrodek
Elektronicznej
Techniki Obliczeniowej

INFORMATYKA
mikrokomputerowa

Wojciech Zientara

Mapa pamięci Atari XL/XE: Procedury wejścia/wyjścia

ATARI

WARSZAWA 1988

Spis treści

PRZEDMOWA

Niewielki byłby pożytek z komputera, gdyby nie można było wprowadzać programów i danych oraz wyprowadzać wyników na ekran lub drukarkę. Każdy komputer na te czynności poświęca znaczną część swej pracy, a w grach zręcznościowych prawie całość.

Komunikacją komputera z urządzeniami zewnętrznymi steruje zespół procedur wejścia/wyjścia. Zajmuje on zwykle ponad połowę obszaru pamięci systemu operacyjnego i dlatego często nazywany jest podsystemem wejścia/wyjścia. Niniejsza książka zawiera kompletny opis całego podsystemu wejścia/wyjścia komputerów Atari XL/XE. Poza standardowymi procedurami obsługi urządzeń zewnętrznych spotykanych w innych komputerach uwzględniono tu także procedury obsługi tzw. "nowych urządzeń", które są specyficznym elementem systemu operacyjnego Atari.

Ponadto opisane zostało szczegółowo działanie specjalizowanych układów wejścia/wyjścia: ANTIC, GTIA, POKEY i PIA. Dwa pierwsze układy zajmują się tworzeniem obrazu, co niewątpliwie jest jednym z najważniejszych zadań wykonywanych w każdym komputerze. Układ POKEY jest odpowiedzialny za tworzenie dźwięku oraz obsługę złącza szeregowego, klawiatury i manipulatorów analogowych. Poza tym zawiera kilka liczników o dużej szybkości zliczania. Manipulatory cyfrowe są natomiast kontrolowane przez układ PIA. Znaczenie jego jest jednak o wiele większe niż tylko użycie w grach, gdyż poprzez gniazda joysticków można prowadzić transmisję dwukierunkową. Dodatkowym zadaniem PIA jest zarządzanie pamięcią komputera.

Książka ta jest przeznaczona przede wszystkim dla użytkowników znających już programowanie w języku maszynowym mikroprocesora 6502. Osoby nie znające tego języka także będą mogły korzystać z zamieszczonych tu opisów procedur, choć w ograniczonym zakresie. Podane adresy rejestrów wykorzystywanych przez system operacyjny oraz procedur systemowych mogą być użyte również w programach napisanych w Basicu poprzez instrukcje PEEK, POKE i USR.

Zawarte w książce informacje mogą stanowić także istotną pomoc dla wszystkich osób, które są zainteresowane wykonywaniem modyfikacji sprzętowych oraz nowych urządzeń peryferyjnych. Zamieszczone procedury obsługi różnych urządzeń mogą być wykorzystane częściowo dla nowozaprojektowanych peryferiów. Ponadto poznanie zasad komunikacji komputera z urządzeniami zewnętrznymi pomoże z pewnością przy opracowywaniu ich konstrukcji.

Osobom nie znającym języka maszynowego polecam książkę Jana Ruszczyca "Asembler 6502" wydaną przez SOETO. Stanowi ona podstawowy podręcznik zarówno dla początkujących, jak i zaawansowanych programistów.

Na końcu książki zamieszczone zostały dodatki ułatwiające korzystanie z niej oraz słownik niektórych użytych terminów i bibliografia pozwalająca na poszerzenie wiedzy o systemie operacyjnym Atari. Osoby, które nie mają wprawy w posługiwaniu się innymi systemami

liczbowymi niż dziesiętny, znajdą tam także tabele przeliczeń pomiędzy systemami dziesiętnym, dwójkowym i szesnastkowym.

Wszystkie zamieszczone w książce procedury zostały napisane w formacie asemblera MAC/65. W przypadku posiadania innego asemblera konieczne będzie dokonanie drobnych poprawek w niektórych słowach kluczowych procedur, aby mogły one działać prawidłowo (wykaz różnic znajduje się w Dodatku F).

WPROWADZENIE

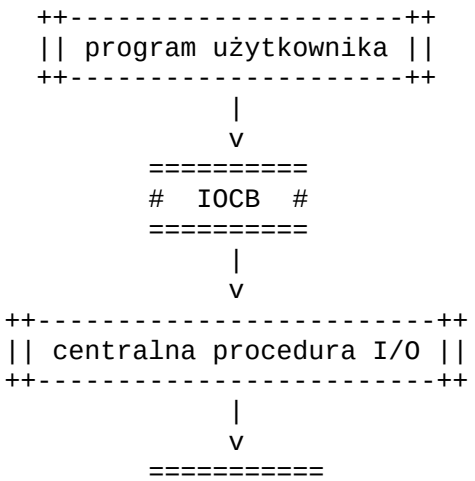
Komputery Atari XL/XE posiadają bardzo rozbudowany podsystem wejścia/wyjścia, który umożliwia proste rozszerzanie systemu oraz łatwą ingerencję programisty w komunikację z urządzeniami peryferyjnymi. Jego podstawową częścią są centralne procedury wejścia/wyjścia (CIO - Central Input/Output routines). Sterują one komunikacją ze wszystkimi urządzeniami peryferyjnymi poprzez szczegółowe procedury obsługi (sterowniki) tych urządzeń.

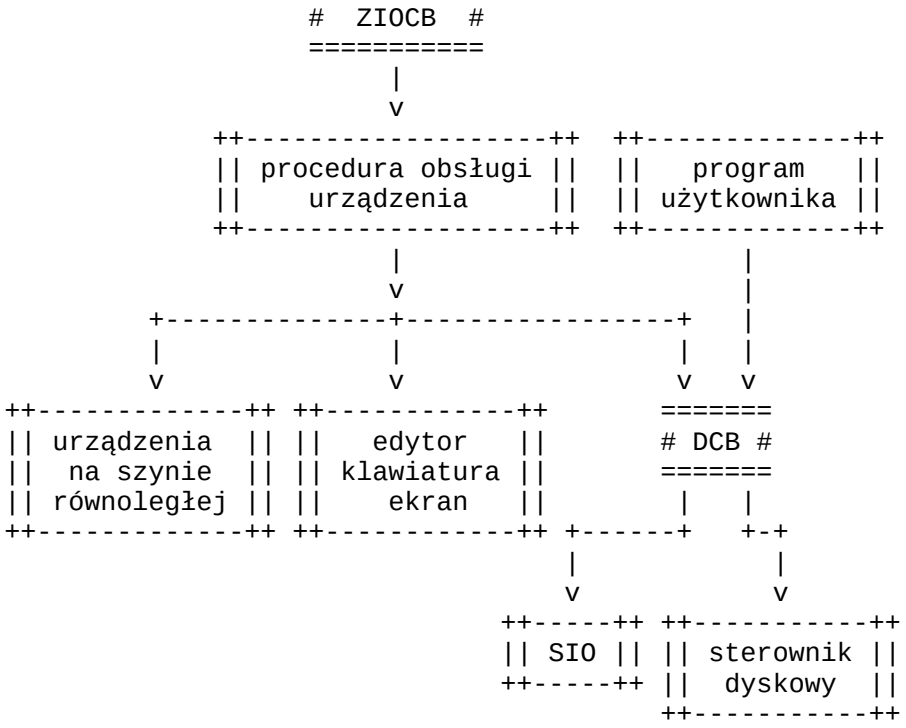
Dostęp CIO do poszczególnych sterowników umożliwiają tabele ich adresów (HATABS - HAndler TABleS). Umieszczenie ich w pamięci RAM pozwala na łatwą ingerencję w przebieg komunikacji oraz proste dołączanie następnych urządzeń. System operacyjny Atari zawiera jedynie procedury obsługi klawiatury, ekranu, drukarki i magnetofonu - pozostałe muszą zostać zainstalowane przed wykorzystaniem.

Sterowniki urządzeń komunikujących się z komputerem za pośrednictwem złącza szeregowego korzystają z kolei z procedur transmisji szeregowej (SIO - Serial Input/Output). Złącze szeregowe jest także wykorzystywane przez rezydujący sterownik stacji dysków (DSKINT - Disk INTerface), nie związany z CIO i dostępny bezpośrednio z poziomu języka maszynowego.

Prawidłowa praca wyżej wymienionych procedur wymaga przechowywania i przekazywania między nimi wielu różnych danych. Do tego celu służą wyodrębnione obszary pamięci RAM zwane blokami kontroli. CIO korzysta z ośmiu bloków IOCB (Input/Output Control Block) umieszczonych na trzeciej stronie pamięci oraz jednego na stronie zerowej (ZIOCB - Zeropage IOCB). SIO i sterownik dyskowy używają bloku DCB (Device Control Block), który znajduje się także na stronie trzeciej. Poza tym liczne rejestry RAM są wykorzystywane przez sterowniki poszczególnych urządzeń.

Obsługa komunikacji z urządzeniami dołączonymi do szyny równoległej odbywa się poprzez procedury umieszczone w pamięci ROM tych urządzeń. Są one również wywoływane poprzez CIO.





Rys.1. Struktura podsystemu wejścia/wyjścia.

Rozdział 1

BLOKI KONTROLI I/O

Do przesyłania informacji pomiędzy programem użytkownika (lub językiem wyższego poziomu) a procedurą CIO służą bloki kontroli wejścia/wyjścia (IOCB - Input/Output Control Block), zwane także kanałami wejścia/wyjścia (Input/Output Channel), Są to szesnastobajtowe obszary pamięci RAM znajdujące się na stronie trzeciej: IOCB0 od adresu $340 (832), IOCB1 od $350 (848), IOCB2 od $360 (864), IOCB3 od $370 (880), IOCB4 od $380 (896), IOCB5 od $390 (912), IOCB6 od $3A0 (928) i IOCB7 od $3B0 (944). Te osiem bloków pozwala na jednoczesną współpracę komputera z ośmioma urządzeniami zewnętrznymi, przy czym IOCB0 jest standardowo wykorzystywany do obsługi edytora.

Dodatkowy dziewiąty blok - ZIOCB (Zeropage IOCB) - znajduje się na stronie zerowej od adresu $20 (32). Jest on wykorzystywany do komunikacji pomiędzy CIO i procedurami obsługi poszczególnych urządzeń. Umieszczenie go na stronie zerowej przyspiesza komunikację z urządzeniami peryferyjnymi, dzięki zastosowaniu zerostronicowego trybu adresowania procesora 6502.

Struktura wszystkich dziewięciu IOCB jest jednakowa. Znaczenie kolejnych rejestrów jest następujące (w nawiasie nazwa rejestru w ZIOCB):

ICCHID (ICHIDZ) - numer identyfikacyjny urządzenia, wskazuje na wpis w tabeli HATABS określający dane urządzenie. Ustawiany przez CIO po odnalezieniu wpisu; gdy IOCB nie jest używany oraz po zamknięciu kanału IOCB, zawiera wartość $FF.

ICDNO (ICDNOZ) - numer urządzenia ustawiany przez CIO według podanej nazwy (np. "D2:NAME.EXT") lub standardowo na 1, gdy dane urządzenie może być tylko jedno lub numer nie został podany.

ICCMD (ICCOMZ) - kod rozkazu (operacji) do wykonania przez CIO, ustawiany przez program użytkownika. Dozwolone są następujące kody operacji:

```
$03 - OPEN
$05 - GET RECORD
$07 - GET BYTE(S)
$09 - PUT RECORD
$0B - PUT BYTE(S)
$0C - CLOSE
$0D - STATUS
```

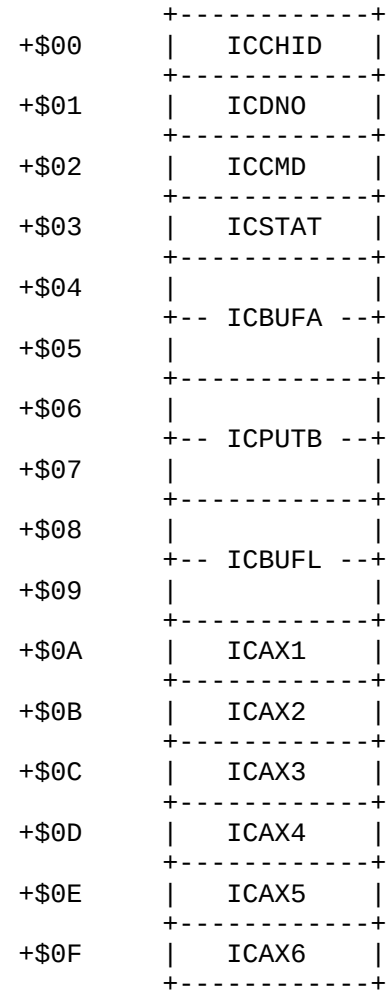
Oprócz tego operacje o kodzie $0E (14) i większym są traktowane jako operacje specjalne i wykonanie ich zależy od rodzaju urządzenia.

ICSTAT (ICSTZ) - status wykonanej operacji IO, ustawiany przez CIO po zakończeniu operacji. Wartość $01 oznacza poprawne wykonanie operacji, a wartość większa od $80 sygnalizuje błąd. Status operacji jest umieszczany również w rejestrze Y procesora przed zakończeniem procedury CIO.

ICBUFA (ICBAZ) - adres bufora danych dla operacji CIO, ustawiany przez program użytkownika. Podczas operacji OPEN, STATUS itp. zawiera adres nazwy urządzenia.

ICPUTB (ICPTZ) - zmniejszony o jeden adres procedury wykonującej żądaną operację, ustawiany przez CIO. Gdy kanał IOCB jest zamknięty, wskazuje procedurę CIONOPN ($E4DC).

ICBUFL (ICBLZ) - długość bufora danych dla operacji CIO, ustawiana przez program użytkownika. Podczas operacji PUT BYTE i GET BYTE wartość $00 oznacza, że przesyłany bajt znajduje się w akumulatorze.



Rys.2. Struktura IOCB.

ICAX1 (ICAX1Z) - rejestr pomocniczy numer 1. Podczas operacji OPEN oznacza rodzaj dostępu do urządzenia (zapis, odczyt i in.).

ICAX2 (ICAX2Z) - rejestr pomocniczy numer 2. Zastosowanie tego i następnych rejestrów pomocniczych jest w przeważającej części zależne od rodzaju urządzenia zewnętrznego (a właściwie od jego procedury obsługi).

ICAX3 (ICAX3Z) - rejestr pomocniczy numer 3. Wraz z rejestrem ICAX4 służy do przekazywania adresów podczas procedury CIO oraz do przekazywania numeru sektora dla instrukcji Basica NOTE i POINT.

ICAX4 (ICAX4Z) - rejestr pomocniczy numer 4 (zob. wyżej).

ICAX5 (ICAX5Z) - rejestr pomocniczy numer 5. Służy do przekazywania numeru bajtu w sektorze dla instrukcji Basica NOTE i POINT oraz do przechowywania numeru IOCB podczas procedury CIO.

ICAX6 (ICAX6Z) - rejestr pomocniczy numer 6. Wykorzystywany przez CIO m. in. do przechowywania transmitowanego bajtu.

Dokładne znaczenie poszczególnych rejestrów IOCB (przede wszystkim ICAX1-6) będzie wyjaśnione przy opisach działania procedur CIO.

Rozdział 2

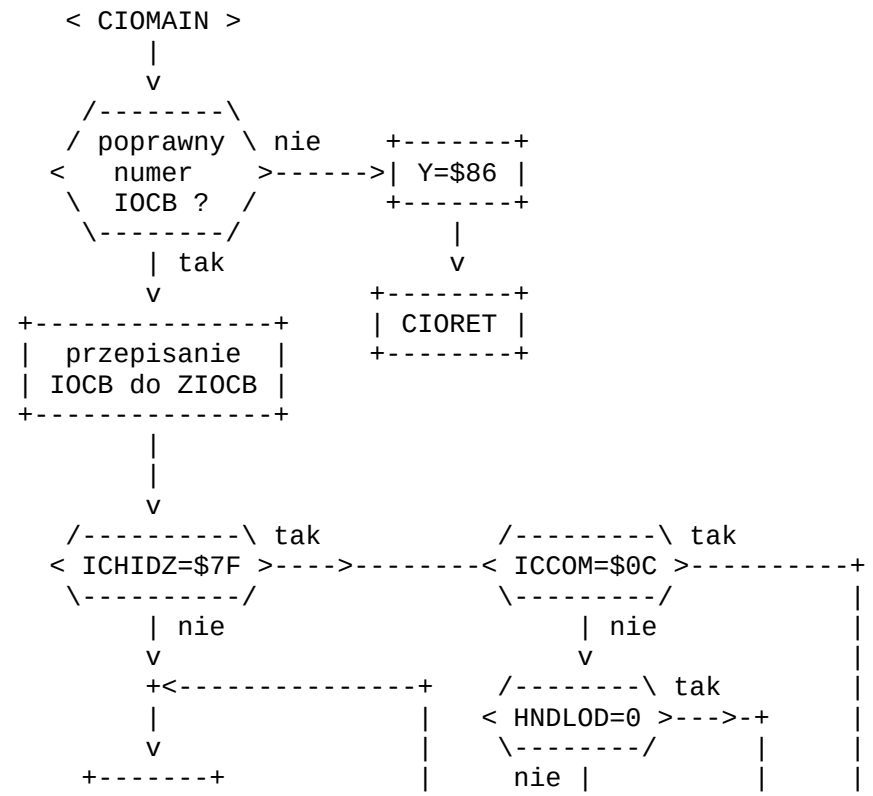
CENTRALNA PROCEDURA I/O

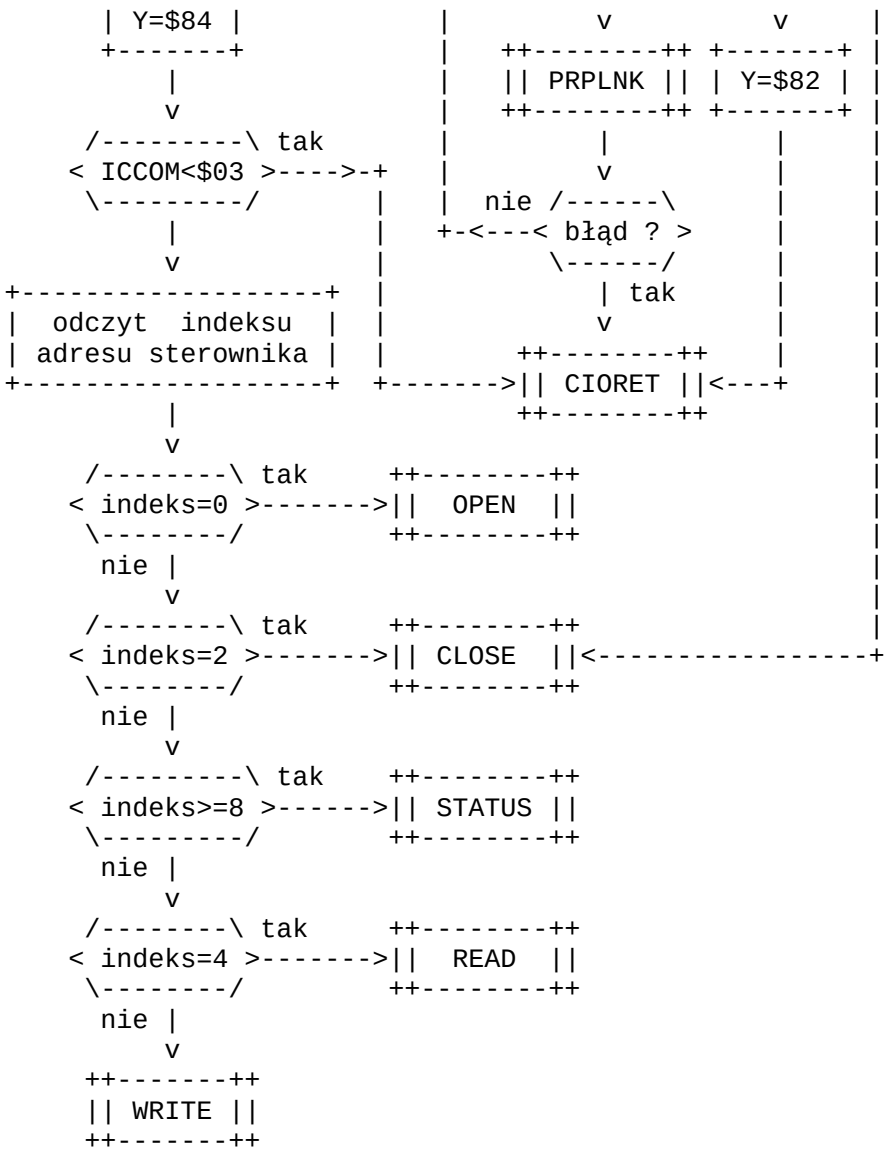
Centralna procedura wejścia/wyjścia (CIO) umożliwia komunikację ze wszystkimi urządzeniami zewnętrznymi, które są przyłączone do systemu komputerowego. Jest ona dostępna bezpośrednio z poziomu języka maszynowego oraz z poziomu języków programowania poprzez instrukcje wejścia/wyjścia (np. w Atari Basic: OPEN, CLOSE, PRINT, GET itd.). Przez urządzenia zewnętrzne należy rozumieć wszystkie urządzenia do komunikacji komputera z otoczeniem w tym również klawiaturę, która fizycznie jest wbudowana do komputera. Dodatkowym, oddzielnym urządzeniem zewnętrznym Atari jest tzw. edytor, który stanowi połączenie klawiatury (wejście) i ekranu w trybie 0 (wyjście).

Przed wywołaniem procedury CIO program użytkownika musi umieścić w odpowiednim IOCB kod operacji do wykonania oraz parametry potrzebne dla żądanej operacji (np. adres bufora danych). Ponadto numer wybranego IOCB pomnożony przez 16 umieszczany jest w rejestrze indeksowym X procesora.

2.1. Główna procedura CIO

Zadaniem głównej procedury CIO jest sprawdzenie poprawności podanego kodu operacji i numeru IOCB. Następnie według odczytanego kodu wywoływana jest podrzędna procedura CIO wykonująca żądaną operację.





Rys.3. Struktura głównej procedury CIO.

Na początku procedury zawartości rejestru X i akumulatora są zapisywane w pomocniczych rejestrach ICAX5Z i ICAX6Z. Następnie sprawdzane jest, czy numer IOCB jest pełną wielokrotnością liczby 16 i czy jest mniejszy od $80. Negatywny wynik powoduje umieszczenie w rejestrze Y wartości $86 (kod błędu BAD IOCB NUMBER - błędny numer IOCB) i przejście do procedury CIORET kończącej operację CIO.

```
0100 ;CIO MAIN routine
0110 ;
0120 CIOCLS = $E57C
0130 CIOOPN = $E53F
0140 CIOREAD = $E5B2
0150 CIORET = $E670
0160 CIOSTSP = $E597
0170 CIOWRT = $E61E
0180 COMTAB = $E72D
0190 HNDLOD = $02E9
0200 ICAX5Z = $2E
```

```
0210 ICAX6Z = $2F
0220 ICCHID = $0340
0230 ICCOMT = $17
0240 ICCOMZ = $22
0250 ICHIDZ = $20
0260 PRPLNK = $CA29
0270 ;
0280     *=  $E4DF
0290 ;
0300     STA ICAX6Z
0310     STX ICAX5Z
0320     TXA
0330     AND #$0F
0340     BNE BNAM
0350     CPX #$80
0360     BCC MOVE
0370 BNAM LDY #$86
0380     JMP CIORET
0390 MOVE LDY #$00
0400 NEXT LDA ICCHID,X
0410     STA ICHIDZ,Y
0420     INX
0430     INY
0440     CPY #$0C
0450     BCC NEXT
0460     LDA ICHIDZ
0470     CMP #$7F
0480     BNE PCMD
0490     LDA ICCOMZ
0500     CMP #$0C
0510     BEQ CIOCLS
0520     LDA HNDLOD
0530     BNE LDH
0540 ;NonEXistent Device ERror
0550 NEXDER LDY #$82
0560 EXIT JMP CIORET
0570 LDH JSR PRPLNK
0580     BMI EXIT
0590 PCMD LDY #$84
0600     LDA ICCOMZ
0610     CMP #$03
0620     BCC $E547   ;do JMP CIORET
0630     TAY
0640     CPY #$0E
0650     BCC CMD
0660     LDY #$0E
0670 CMD STY ICCOMT
0680     LDA COMTAB-3,Y
0690     BEQ CIOOPN
0700     CMP #$02
0710     BEQ CIOCLS
0720     CMP #$08
0730     BCS CIOSTSP
0740     CMP #$04
0750     BEQ CIOREAD
0760     JMP CIOWRT
```

Jeżeli numer IOCB jest poprawny, to jego zawartość jest przepisywana na stronę zerową do ZIOCB. Nie ją jedynie przenoszone dwa ostatnie bajty IOCB, gdyż w odpowiadających im

rejestrach ZIOCB przechowywane są już wartości z akumulatora i rejestru X.

W następnej kolejności sprawdzany jest indeks wpisu w tabeli HATABS - ICHIDZ (IOCB CHannel IDentification). Jego wartość równa $7F oznacza, że operacja dotyczy "nowego urządzenia". W takim przypadku, jeśli kod operacji jest równy $0C (CLOSE), następuje skok do procedury CIOCLS. W przeciwnym razie kontrolowany jest znacznik HNDLOD. Gdy jest on równy zero, to do rejestru Y wpisywana jest wartość $82 (NON EXISTENT DEVICE - urządzenie nie istnieje) i procedura CIOMAIN kończy się skokiem do CIORET. Po stwierdzeniu obecności urządzenia wywoływana jest procedura PRPLNK, która obsługuje nowe urządzenia (zob. rozdział 6.2.).

Wartość ICHIDZ różna od $7F oznacza, że komunikacja z urządzeniem będzie wykonywana przez standardowe procedury CIO. Teraz do rejestru Y wpisywany jest kod błędu INVALID COMMAND ($84 - błędny rozkaz) i sprawdzana jest zawartość rejestru ICCOMZ. Gdy jest ona mniejsza od $03, następuje skok do CIORET. Poprawny kod operacji powoduje (po zapisaniu go do rejestru ICCOMT) pobranie z tabeli COMTAB wartości indeksu wskazującego położenie adresu procedury wykonującej tą operację w tabeli adresowej sterowników. Bliższe informacje na ten temat są zawarte w rozdziale 2.2.

```
0100 ;COMmand TABle
0110 ;
0120     *=  $E72D
0130 ;
0140     .BYTE $00,$04,$04,$04
0150     .BYTE $04,$06,$06,$06
0160     .BYTE $06,$02,$08,$0A
```

Teraz przeprowadzane jest rozpoznanie operacji do wykonania poprzez sprawdzanie wartości odczytanego indeksu. Wartość zero oznacza operację otwarcia kanału IOCB (OPEN) i powoduje przejście do procedury CIOOPN. Wartość dwa oznacza zamknięcie kanału (CLOSE) - wywołuje skok do CIOCLS. Indeks większy od $07 oznacza operację odczytu statusu (STATUS) lub operację specjalną (SPECIAL), specyficzną dla danego urządzenia. W tym przypadku wykonywany jest skok do CIOSTSP. Na końcu sprawdzany jest indeks operacji odczytu ($04 - READ) - skok do CIOREAD. Jeśli żadna z wyżej wymienionych operacji nie została rozpoznana, to znaczy, że chodzi o zapis (WRITE) i następuje skok do CIOWRT.

2.1.1. Zakończenie procedury CIO

Zarówno główna, jak i wszystkie szczegółowe procedury CIO kończą się skokiem do procedury CIORET. Ma ona dwa punkty początkowe: CIORET i CPLCIO. Rozpoczęcie procedury od etykiety CIORET powoduje zapisanie aktualnej zawartości rejestru Y procesora do rejestru statusu ZIOCB (ICSTZ - IOCB STatus, Zeropage). Operacja ta jest omijana przy rozpoczęciu od etykiety CPLCIO.

```
0100 ;CIO RETurn routine
0110 ;
0120 HNDLOD = $02E9
0130 ICAX5Z = $2E
```

```
0140 ICAX6Z = $2F
0150 ICBAZ = $24
0160 ICBUFA = $0344
0170 ICCHID = $0340
0180 ICHIDZ = $20
0190 ICSTZ = $23
0200 ;
0210     *=  $E670
0220 ;
0230     STY ICSTZ
0240 ;ComPLete CIO operation
0250 CPLCIO LDY ICAX5Z
0260     LDA ICBUFA,Y
0270     STA ICBAZ
0280     LDA ICBUFA+1,Y
0290     STA ICBAZ+1
0300     LDX #$00
0310     STX HNDLOD
0320 NEXT LDA ICHIDZ,X
0330     STA ICCHID,Y
0340     INX
0350     INY
0360     CPX #$0C
0370     BCC NEXT
0380     LDA ICAX6Z
0390     LDX ICAX5Z
0400     LDY ICSTZ
0410     RTS
```

Zadaniem CIORET jest przekazanie zawartości ZIOCB do odpowiedniego IOCB. W tym celu najpierw z rejestru ICAX5Z jest pobierany numer właściwego IOCB. Następnie adres bufora w ZIOCB jest odtwarzany według zawartości IOCB i zerowany jest rejestr HNDLOD. Teraz kolejno zawartości rejestrów ZIOCB są przenoszone do IOCB (oprócz dwóch ostatnich bajtów).

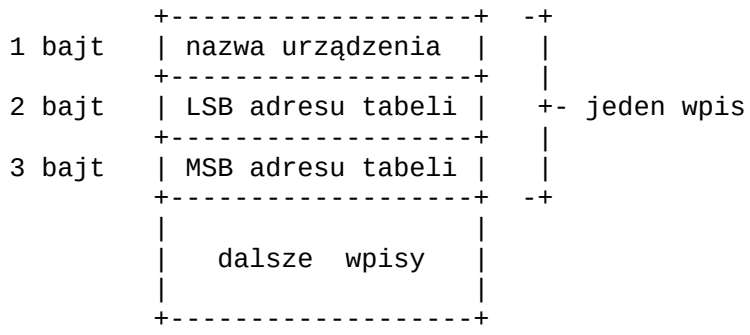
Na początku procedury CIOMAIN w rejestrach ICAX5Z i ICAX6Z zostały zapamiętane zawartości rejestru X i akumulatora. Przed zakończeniem CIO są one odtwarzane i dodatkowo do rejestru Y pobierany jest status wykonanej operacji I/O. Ponieważ jest to ostatnie polecenie przed końcem procedury, to w przypadku błędu (status większy od $7F) po powrocie do miejsca wywołania CIO bit N (negative) w rejestrze statusu procesora jest ustawiony. Pozwala to na bardzo proste wykrycie błędu i przejście do dalszej części programu (np. procedury obsługi błędu) przy użyciu rozkazów BPL (Branch if PLus) lub BMI (Branch if MInus).

2.2. Tabele adresowe sterowników

Kilkakrotnie już wspominane były tabele adresów procedur obsługi urządzeń zewnętrznych. Według umieszczonych w nich adresów CIO rozpoznaje procedury dotyczące poszczególnych urządzeń. W systemie operacyjnym Atari występują dwa rodzaje tabel adresowych: HATABS i tabele wektorów sterowników (procedur obsługi).

Tabela HATABS (Handler Address TABleS) znajduje się na trzeciej stronie pamięci i zawiera

nazwy wszystkich urządzeń i adresy ich tabel wektorów. Każdy wpis w tabeli HATABS składa się z trzech bajtów: nazwa urządzenia, młodszy bajt adresu i starszy bajt adresu:



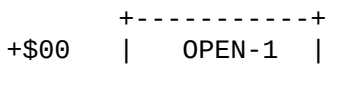
Rys.4. Struktura tabeli HATABS

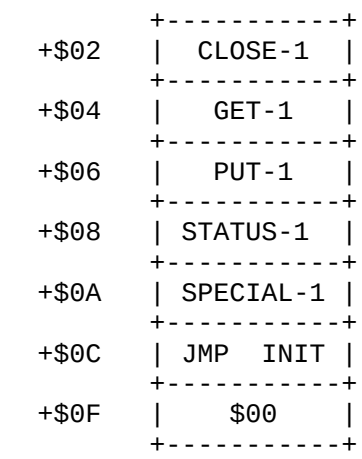
Nazwa urządzenia w HATABS jest zapisaną w kodzie ASCII pierwszą literą nazwy symbolicznej, np. Cassette, Editor, Keyboard, Printer itd. Są to te same nazwy, które występują w Atari Basic w instrukcji OPEN. Początkowo tabela HATABS zawiera pięć wpisów urządzeń (P, C, E, S i K) przeniesionych tu z tabeli INIT31A podczas inicjowania systemu:

```
0100 ;Handler Address TABleS
0110 ;
0120 CASVEC = $E440
0130 EDTVEC = $E400
0140 KBDVEC = $E420
0150 PRTVEC = $E430
0160 SCRVEC = $E410
0170 ;
0180     *=  $031A
0190 ;
0200     .BYTE 'P
0210     .WORD PRTVEC
0220     .BYTE 'C
0230     .WORD CASVEC
0240     .BYTE 'E
0250     .WORD EDTVEC
0260     .BYTE 'S
0270     .WORD SCRVEC
0280     .BYTE 'K
0290     .WORD KBDVEC
```

Dalsze miejsca w HATABS są przeznaczone do zainstalowania innych urządzeń zewnętrznych. Wszystkie niewykorzystywane wpisy mają wartość zero. Razem z wpisami już istniejącymi w tabeli HATABS można umieścić 11 różnych urządzeń zewnętrznych.

Dwa bajty adresowe wpisu w HATABS wskazują na tabelę adresów procedur obsługi poszczególnych operacji wykonywanych przez urządzenie. Należy zwrócić uwagę, że wszystkie adresy są zmniejszone o jeden. Ostatnie cztery bajty zawierają rozkaz skoku do procedury inicjowania oraz bajt zerowy.





Rys.5. Tabela adresowa sterownika

Urządzenia zainstalowane na stałe posiadają tabele adresów zapisaną w pamięci ROM. Można jednak napisać dla nich własne procedury obsługi i po umieszczeniu w dowolnym miejscu RAM tabeli adresów zmienić wektor w HATABS tak, aby wskazywał na nową tabelę adresową. Standardowa tabela zapisana w ROM-ie jest następująca:

```
0100 ;DEVice HAndler Table
0110 ;
0120 CASCLS = $FDCF
0130 CASINIT = $FCDB
0140 CASOPN = $FCE6
0150 CASRDBT = $FD7A
0160 CASSP = $FCE5
0170 CASST = $FDCC
0180 CASWRT = $FDB4
0190 DRAW =  $F9AF
0200 EDOPN = $EF94
0210 EDSP =  $F22D
0220 EGETCH = $F24A
0230 EOUTCH = $F2B0
0240 GETCH = $F180
0250 KBGBYT = $F2FD
0260 KBOPN = $F21E
0270 OUTCH = $F1A4
0280 POWERON = $EF6E
0290 PRCLS = $FF07
0300 PRINIT = $FE99
0310 PROPN = $FEC2
0320 PRRDSP = $FEC1
0330 PRSTAT = $FEA3
0340 PRWRT = $FECB
0350 SCOPN = $EF8E
0360 SCRFIN = $F22E
0370 ;
0380     *=  $E400
0390 ;
0400 EDTVEC .WORD EDOPN-1
0410     .WORD SCRFIN-1
0420     .WORD EGETCH-1
0430     .WORD EOUTCH-1
0440     .WORD KBOPN-1
```

```
0450     .WORD EDSP-1
0460     JMP POWERON
0470     .BYTE $00
0480 SCRVEC .WORD SCOPN-1
0490     .WORD SCRFIN-1
0500     .WORD GETCH-1
0510     .WORD OUTCH-1
0520     .WORD KBOPN-1
0530     .WORD DRAW-1
0540     JMP POWERON
0550     .BYTE $00
0560 KBDVEC .WORD KBOPN-1
0570     .WORD KBOPN-1
0580     .WORD KBGBYT-1
0590     .WORD EDSP-1
0600     .WORD KBOPN-1
0610     .WORD EDSP-1
0620     JMP POWERON
0630     .BYTE $00
0640 PRTVEC .WORD PROPN-1
0650     .WORD PRCLS-1
0660     .WORD PRRDSP-1
0670     .WORD PRWRT-1
0680     .WORD PRSTAT-1
0690     .WORD PRRDSP-1
0700     JMP PRINIT
0710     .BYTE $00
0720 CASVEC .WORD CASOPN-1
0730     .WORD CASCLS-1
0740     .WORD CASDRBT-1
0750     .WORD CASWRT-1
0760     .WORD CASST-1
0770     .WORD CASSP-1
0780     JMP CASINIT
0790     .BYTE $00
```

Z tabeli tej pobierane są przez procedurę CIO adresy procedur obsługi poszczególnych operacji dla konkretnych urządzeń. Sposób wykorzystania tej tabeli przez procedury systemu operacyjnego jest opisany w następnych rozdziałach.

Dodanie nowego urządzenia do tabeli HATABS nie wymaga nawet pisania specjalnej procedury. Do tego celu służy znajdująca się 2już w systemie operacyjnym procedura NEWDEV. Przed jej wywołaniem trzeba jedynie umieścić w rejestrze X nazwę urządzenia, a w rejestrze Y i w akumulatorze odpowiednio młodszy i starszy bajt adresu tabeli adresowej procedur obsługi.

```
0100 ;NEW DEVice
0110 ;
0120 HATABS = $031A
0130 ;
0140     *=  $EEBC
0150 ;
0160     PHA
0170     TYA
0180     PHA
0190     TXA
0200     LDX #$00
0210 NEXT1 CMP HATABS,X
```

```
0220     BEQ FOUND
0230     INX
0240     INX
0250     INX
0260     CPX #$22
0270     BMI NEXT1
0280     LDX #$00
0290     TAY
0300     LDA #$00
0310 NEXT2 CMP HATABS,X
0320     BEQ EMPTY
0330     INX
0340     INX
0350     INX
0360     CPX #$22
0370     BMI NEXT2
0380     PLA
0390     PLA
0400     LDY #$FF
0410     SEC
0420     RTS
0430 FOUND PLA
0440     TAY
0450     PLA
0460     INX
0470     SEC
0480     RTS
0490 EMPTY TYA
0500     STA HATABS,X
0510     PLA
0520     STA HATABS+1,X
0530     PLA
0540     STA HATABS+2,X
0550     CLC
0560     RTS
```

Na początku procedury zawartości akumulatora i rejestru X są umieszczane na stosie, a rejestr X (nazwa urządzenia) przepisywany jest do akumulatora. Następnie wykonywane są dwie pętle przeszukujące HATABS.

Pierwsza pętla sprawdza, czy urządzenie o podanej nazwie znajduje się już w HATABS. Jeśli tak, to po odtworzeniu ze stosu zawartości rejestru Y i akumulatora procedura kończy się z ustawionym bitem Carry statusu procesora. Rejestr X zawiera wtedy indeks młodszego bajtu wektora w HATABS, co pozwala na bezpośrednią zmianę tego wektora poprzez sekwencję rozkazów STY HATABS,X, INX, STA HATABS,X.

Gdy poszukiwane urządzenie nie zostało znalezione, wykonywana jest druga pętla. Jej zadaniem jest wykrycie pierwszego wolnego miejsca na wpis w HATABS. Jeżeli tabela HATABS jest całkowicie zapełniona, to ustawiany jest bit Carry, w rejestrze Y umieszczana jest wartość $FF i procedura się kończy.

Po wykryciu wolnego miejsca w HATABS umieszczana jest w tabeli nazwa urządzenia, a

następnie pobrany ze stosu adres tabeli jego procedur obsługi. W tym przypadku przed zakończeniem procedury bit Carry jest kasowany w celu wskazania pomyślnego przebiegu procedury.

2.2.1. Procedury korzystające z HATABS

Z tabelami adresowymi sterowników jest związanych jeszcze kilka procedur pomocniczych wykorzystywanych przez procedury operacji CIO. Są to procedury DEVNUM, CMPENT i GOHAND oraz pośrednio procedura CIOJMP.

Zadaniem procedury DEVNUM jest ustalenie numeru urządzenia, którego dotyczy operacja I/O; a w drugiej części rozpoczynającej się od etykiety DVSRCH - odnalezienie wpisu urządzenia w HATABS.

```
0100 ;DEVice NUMber
0110 ;
0120 HATABS = $031A
0130 ICBAZ = $24
0140 ICHIDZ = $20
0150 ICDNOZ = $21
0160 ;
0170     *=  $E6FF
0180 ;
0190     SEC
0200     LDY #$01
0210     LDA (ICBAZ),Y
0220     SBC #'1
0230     BMI SET0
0240     CMP #$09
0250     BCC STDN
0260 SET0 LDA #$00
0270 STDN STA ICDNOZ
0280     INC ICDNOZ
0290 ;DeVice SeaRCH
0300 DVSRCH LDY #$00
0310     LDA (ICBAZ),Y
0320 ;Find DeVice HaNDler
0330 FDVHND BEQ ERR
0340     LDY #$21
0350 NEXT CMP HATABS,Y
0360     BEQ SUC
0370     DEY
0380     DEY
0390     DEY
0400     BPL NEXT
0410 ERR LDY #$82
0420     SEC
0430     RTS
0440 SUC TYA
0450     STA ICHIDZ
0460     CLC
0470     RTS
```

Procedura DEVNUM w celu ustalenia numeru urządzenia pobiera najpierw drugi bajt z bufora wskazywanego przez rejestr ICBAZ (bufor musi zawierać nazwę urządzenia). Następnie sprawdza,

czy jego wartość mieści się w dozwolonym zakresie (dopuszczalne są numery urządzeń od 1 do 9). Poprawny numer urządzenia jest umieszczany w rejestrze ICDNOZ, a gdy jest nieprawidłowy, to do ICDNOZ wpisywane jest 1.

Teraz odczytywany jest z bufora pierwszy bajt, który określa nazwę urządzenia i tabela HATABS jest przeszukiwana w celu odnalezienia wpisu tego urządzenia. Po napotkaniu wpisu jego indeks w HATABS jest umieszczany w rejestrze ICHIDZ i bit Carry jest kasowany. Gdy tabela nie zawiera poszukiwanego wpisu, to w rejestrze Y umieszczany jest kod błędu $82 (NON EXISTENT DEVICE) i bit Carry jest ustawiany.

Warto przy tym zauważyć, że tabela HATABS jest przeszukiwana od końca. Jeżeli więc zostanie umieszczony w niej drugi wpis tego samego urządzenia (np. nowa procedura obsługi drukarki), to zostanie on odnaleziony wcześniej niż wpis oryginalny. Umożliwia to wprowadzenie nowego wpisu bez kasowania starego.

Procedura CMPENT służy do ustalenia adresu początkowego procedury obsługi wykonującej żądaną operację I/O. Na początku odczytywany jest z rejestru ICHIDZ indeks urządzenia w HATABS. Jeśli jest on większy od $21, to w rejestrze Y umieszczany jest kod błędu $85 (IOCB NOT OPEN - IOCB nie otwarty) i po ustawieniu bitu Carry procedura się kończy.

Jeśli wartość indeksu jest prawidłowa, to według niej wektor tabeli adresowej odczytywany jest z HATABS i umieszczany w rejestrach ICAX3Z oraz ICAX4Z. Następnie według kodu operacji pobranego z ICCOMT określany jest (przy pomocy tabeli COMTAB) indeks adresu procedury, która wykonuje żądaną operację. Znaleziony adres jest przepisywany z tabeli adresowej do rejestrów ICAX3Z i ICAX4Z. W celu zasygnalizowania poprawnego przebiegu procedury, przed jej zakończeniem kasowany jest jeszcze bit Carry.

```
0100 ;CoMPute ENTry
0110 ;
0120 COMTAB = $E72D
0130 HATABS = $031A
0140 ICAX3Z = $2C
0150 ICAX4Z = $2D
0160 ICCOMT = $17
0170 ICCOMZ = $22
0180 ICHIDZ = $20
0190 ;
0200     *=  $E695
0210 ;
0220     LDY ICHIDZ
0230     CPY #$22
0240     BCC LAD
0250     LDY #$85
0260     BCS END
0270 LAD LDA HATABS+1,Y
0280     STA ICAX3Z
0290     LDA HATABS+2,Y
0300     STA ICAX4Z
0310     LDY ICCOMT
```

```
0320     LDA COMTAB-3,Y
0330     TAY
0340     LDA (ICAX3Z),Y
0350     TAX
0360     INY
0370     LDA (ICAX3Z),Y
0380     STA ICAX4Z
0390     STX ICAX3Z
0400     CLC
0410 END RTS
```

Wywołanie wybranej procedury obsługi urządzenia jest wykonywane przez procedurę GOHAND.

```
0100 ;GO to HANDler
0110 ;
0120 CIOJMP = $E6F4
0130 ICSTZ = $23
0140 ;
0150     *=  $E6EA
0160 ;
0170     LDY #$92
0180     JSR CIOJMP
0190     STY ICSTZ
0200     CPY #$00
0210     RTS
```

Umieszcza ona najpierw w rejestrze Y kod błędu $92 (FUNCTION NOT IMPLEMENTED - operacja niedozwolona), a następnie poprzez procedurę CIOJMP wywołuje sterownik urządzenia zewnętrznego. Po zakończeniu operacji i powrocie do GOHAND zawartość rejestru Y jest przepisywana do ICSTZ, a poprzez rozkaz CPY bit Negative rejestru statusu otrzymuje wartość sygnalizującą powodzenie (lub nie) operacji. Umieszczenie w rejestrze Y kodu błędu $92 przed wywołaniem procedury obsługi żądanej operacji znakomicie upraszcza projektowanie procedur operacji niedozwolonych - wystarczy jako adres takiej operacji podać adres rozkazu RTS. Rozwiązanie to jest często stosowane w OS Atari oraz w sterownikach urządzeń instalowanych po zainicjowaniu systemu.

Procedura CIOJMP służy wyłącznie do wykonania skoku do procedury obsługi urządzenia, której adres jest zawarty w rejestrach ICAX3Z i ICAX4Z. Bardzo interesujący jest natomiast sposób wykonania tego skoku.

```
0100 ;CIO JuMP routine
0110 ;
0120 ICAX3Z = $2C
0130 ICAX4Z = $2D
0140 ICAX5Z = $2E
0150 ;
0160     *=  $E6F4
0170 ;
0180     TAX
0190     LDA ICAX4Z
0200     PHA
0210     LDA ICAX3Z
0220     PHA
```

```
0230     TXA
0240     LDX ICAX5Z
0250     RTS
```

Zastosowana sztuczka polega na tym, że bajty adresu procedury pobrane z rejestrów ICAX3/4Z są umieszczane na stosie. Wykonanie teraz rozkazu RTS powoduje zdjęcie ze stosu dwóch znajdujących się tam bajtów, które normalnie oznaczają stan licznika rozkazów w chwili wykonania rozkazu JSR, zwiększenie ich o jeden i umieszczenie w liczniku rozkazów procesora. W ten sposób dalsze wykonywanie programu odbywa się od początku odpowiedniej procedury obsługi urządzenia. Kończący tą procedurę rozkaz RTS spowoduje powrót do miejsca wywołania CIOJMP, a więc bezpośrednio do procedury GOHAND. Oczywiste jest już także umieszczenie w tabeli adresowej adresów procedur zmniejszonych o jeden.

2.3. Procedura otwarcia IOCB

Rozpoznanie przez CIOMAIN kodu operacji OPEN ($03) powoduje wywołanie procedury CIOOPN. Dla jej poprawnego wykonania rejestr ICBUFA musi zawierać adres nazwy urządzenia, dla którego otwierany jest kanał IOCB (np. S: lub D:NAZWA.EXT). Ponadto w ICAX1 muszą znajdować się informacje o sposobie dostępu do urządzenia (zapis czy odczyt) i ewentualnie inne informacje zależne od rodzaju urządzenia. Niektóre urządzenia wymagają jeszcze dodatkowych danych w rejestrze ICAX2.

```
0100 ;CIO OPeN routine
0110 ;
0120 CIORET = $E670
0130 CMPENT = $E695
0140 CPLCIO = $E672
0150 DEVNUM = $E6FF
0160 DVSTAT = $02EA
0170 GOHAND = $E6EA
0180 HNDLOD = $02E9
0190 ICAX3Z = $2C
0200 ICAX4Z = $2D
0210 ICCOMT = $17
0220 ICHIDZ = $20
0230 ICPTZ = $26
0240 SPCHND = $EEF9
0250 ;
0260     *=  $E53F
0270 ;
0280     LDA ICHIDZ
0290     CMP #$FF
0300     BEQ OPN
0310     LDY #$81
0320 EXIT JMP CIORET
0330 OPN LDA HNDLOD
0340     BNE END
0350     JSR DEVNUM
0360     BCS END
0370     LDA #$00
0380     STA DVSTAT
0390     STA DVSTAT+1
0400 ;INItialize for OPeN
0410 INIOPN JSR CMPENT
```

```
0420     BCS EXIT
0430     JSR GOHAND
0440     LDA #$0B
0450     STA ICCOMT
0460     JSR CMPENT
0470     LDA ICAX3Z
0480     STA ICPTZ
0490     LDA ICAX4Z
0500     STA ICPTZ+1
0510     JMP CPLCIO
0520 END JSR SPCHND
0530     JMP CIORET
```

Przede wszystkim na początku procedury sprawdzany jest indeks wpisu w HATABS. Gdy jest on różny od $FF, to w rejestrze Y umieszczany jest kod błędu $81 (PREVIOUS OPEN - uprzednio otwarty) i następuje skok do CIORET. Ma to na celu zabezpieczenie przed jednoczesnym otwarciem tego samego kanału IOCB dla dwóch różnych urządzeń.

Następnie kontrolowany jest rejestr HNDLOD, który wskazuje użycie nowego urządzenia i wywoływana jest procedura DEVNUM w celu odszukania wpisu urządzenia w HATABS. Negatywny wynik którejkolwiek z tych operacji powoduje wywołanie specjalnej procedury obsługi SPCHND, a następnie skok do CIORET.

Jeśli wpis urządzenia w HATABS został odnaleziony, to procedura CMPENT odczytuje z tabeli adresowej urządzenia adres procedury otwierającej kanał dla tego urządzenia. Następujące po tym sprawdzenie pozytywnego wyniku jest konieczne ze względu na to, że niektóre procedury komunikacji z urządzeniami zewnętrznymi korzystają z CIOOPN od miejsca oznaczonego etykietą INIOPN.

Teraz poprzez GOHAND jest wywoływana właściwa procedura otwarcia urządzenia. Po jej zakończeniu w rejestrze ICSTZ znajduje się status wykonanej operacji informujący o jej pomyślnym (lub nie) przebiegu.

W ostatniej fazie procedury do rejestru ICCOMT wpisywany jest kod rozkazu PUT BYTE i ponownie wywoływana jest procedura CMPENT. Odnaleziony przez nią adres procedury PUT BYTE urządzenia jest przepisywany do rejestru ICPTZ. Procedura CIOOPN kończy się skokiem do CPLCIO.

Po zakończeniu procedury CIOOPN odpowiedni IOCB zawiera: w rejestrze ICCHID - indeks wpisu urządzenia w HATABS, w rejestrze ICDNO - numer urządzenia oraz w rejestrze ICSTAT - wynik (status) operacji OPEN.

2.4. Procedura zamknięcia IOCB

Kod operacji CIO równy $0C powoduje wywołanie procedury CIOCLS, której zadaniem jest zamknięcie odpowiedniego kanału IOCB. Operacja CLOSE nie wymaga żadnych dodatkowych informacji w rejestrach IOCB.

```
0100 ;CIO CLoSe routine
0110 ;
0120 CIONOPN = $E4DC
0130 CPLCIO = $E672
0140 CMPENT = $E695
0150 GOHAND = $E6EA
0160 ICHIDZ = $20
0170 ICPTZ = $26
0180 ICSTZ = $23
0190 ;
0200     *=  $E57C
0210 ;
0220     LDY #$01
0230     STY ICSTZ
0240     JSR CMPENT
0250     BCS CLS
0260     JSR GOHAND
0270 CLS LDA #$FF
0280     STA ICHIDZ
0290     LDA # >[CIONOPN-1]
0300     STA ICPTZ+1
0310     LDA # <[CIONOPN-1]
0320     STA ICPTZ
0330     JMP CPLCIO
```

Od razu na początku procedury status operacji jest ustawiany na $01 (SUCCESS). Następnie przez procedurę CMPENT wyszukiwany jest adres procedury zamykającej komunikację z urządzeniem i procedura ta jest wywoływana poprzez GOHAND.

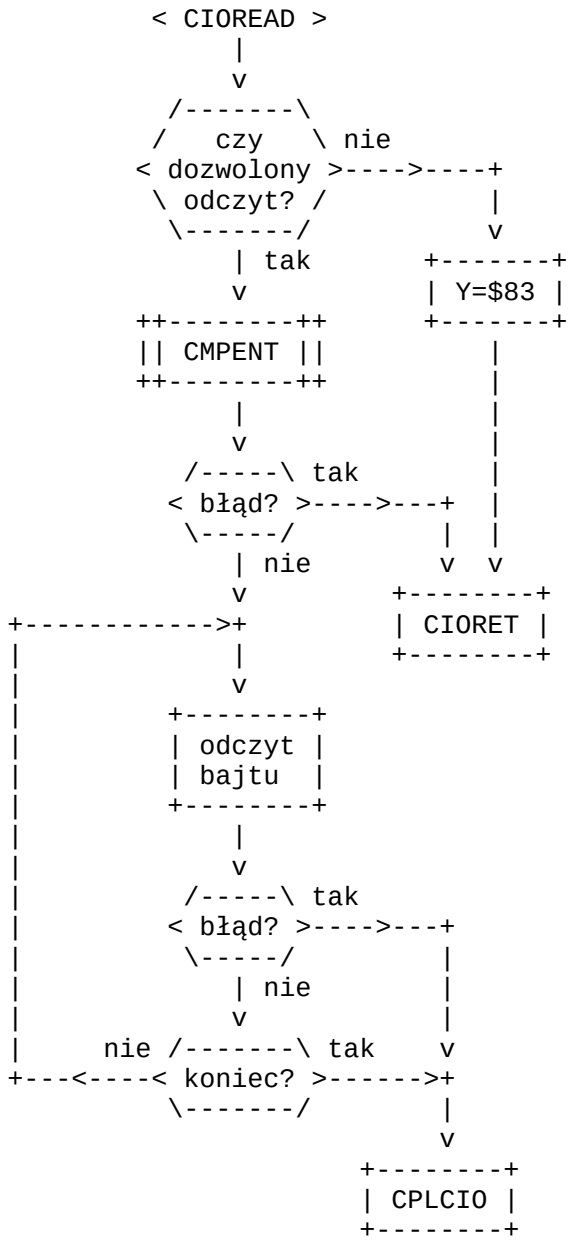
Zarówno w przypadku nieodnalezienia przez CMPENT odpowiedniego wpisu w HATABS, jak i po zakończeniu procedury GOHAND (ewentualny błąd jest przez nią sygnalizowany w rejestrze ICSTZ) dalsze postępowanie jest identyczne. Indeks wpisu urządzenia w HATABS jest ustawiany na wartość $FF, a wektor procedury obsługi w rejestrze ICPTZ wskazuje na procedurę CIONOPN. Procedura ta umieszcza jedynie w rejestrze Y kod błędu $85 (NOT OPEN).

```
0100 ;CIO Not OPeN
0110 ;
0120     *=  $E4DC
0130 ;
0140     LDY #$85
0150     RTS
```

Po zakończeniu operacji CLOSE rejestr ICCHID ma więc wartość $FF, a w rejestrze ICSTAT znajduje się wynik operacji.

2.5. Procedura odczytu z urządzenia

Procedura odczytu CIO jest wywoływana po rozpoznaniu kodu $07 (GET BYTE) lub $05 (GET RECORD). Obie operacje są wykonywane przez tą samą procedurę, która rozpoznaje różnicę według bitu 1 kodu operacji (ustawiony = BYTE, skasowany = RECORD). Przed wywołaniem CIOMAIN konieczne jest jeszcze umieszczenie w rejestrze ICBUFA adresu bufora i w rejestrze ICBUFL jego długości. Jeżeli długość bufora jest równa zero, to odczytany znak umieszczany jest w akumulatorze. Zamieszczony poniżej schemat ukazuje przybliżoną strukturę operacji odczytu.



Rys.6. Struktura operacji READ

Przed rozpoczęciem operacji odczytu kontrolowany jest bit 2 rejestru ICAX1, który sygnalizuje możliwość odczytu z urządzenia. Jeśli jest on skasowany, to do rejestru Y wpisywany jest kod błędu $83 (WRITE ONLY - dozwolony tylko zapis) i operacja kończy się skokiem do CIORET.

Następnie procedura CMPENT odszukuje adres procedury odczytu z urządzenia, a w przypadku niepowodzenia także przechodzi się do CIORET (w takim razie rejestr Y zawiera kod $85 - NOT OPEN).

```
0100 ;CIO READ routine
0110 ;
0120 CIORET = $E670
0130 CPLCIO = $E672
0140 DECBUFL = $E6BB
0150 DECBUFP = $E6C8
0160 CMPENT = $E695
0170 GOHAND = $E6EA
0180 ICAX1Z = $2A
0190 ICAX6Z = $2F
0200 ICBAZ = $24
0210 ICBLZ = $28
0220 ICCOMZ = $22
0230 ICSTZ = $23
0240 INCBUFP = $E6D1
0250 SUBBUFL = $E6D8
0260 ;
0270     *=  $E5B2
0280 ;
0290     LDA ICCOMZ
0300     AND ICAX1Z
0310     BNE PRFM
0320     LDY #$83
0330 EXIT JMP CIORET
0340 PRFM JSR CMPENT
0350     BCS EXIT
0360     LDA ICBLZ
0370     ORA ICBLZ+1
0380     BNE GET1
0390     JSR GOHAND
0400     STA ICAX6Z
0410     JMP CPLCIO
0420 GET1 JSR GOHAND
0430     STA ICAX6Z
0440     BMI END
0450     LDY #$00
0460     STA (ICBAZ),Y
0470     JSR INCBUFP
0480     LDA ICCOMZ
0490     AND #$02
0500     BNE NEXT
0510     LDA ICAX6Z
0520     CMP #$9B
0530     BNE NEXT
0540     JSR DECBUFL
0550     JMP END
0560 NEXT JSR DECBUFL
0570     BNE GET1
0580     LDA ICCOMZ
0590     AND #$02
0600     BNE END
0610 GET2 JSR GOHAND
0620     STA ICAX6Z
0630     BMI ZBF
0640     LDA ICAX6Z
```

```
0650     CMP #$9B
0660     BNE GET2
0670     LDA #$89
0680     STA ICSTZ
0690 ZBF JSR DECBUFP
0700     LDY #$00
0710     LDA #$9B
0720     STA (ICBAZ),Y
0730     JSR INCBUFP
0740 END JSR SUBBUFL
0750     JMP CPLCIO
```

Właściwy odczyt rozpoczyna się od sprawdzenia długości bufora. Gdy jest ona równa zero, to poprzez wywołanie GOHAND odczytywany jest do akumulatora bajt z urządzenia. Po jego przepisaniu do rejestru ICAX6Z procedura kończy się skokiem do CPLCIO.

Przy niezerowej długości bufora postępowanie jest podobne, lecz zamiast zakończenia odczytu następuje sprawdzenie statusu operacji. Ewentualne wystąpienie błędu przerywa odczyt (przed opuszczeniem CIOREAD korygowana jest jeszcze długość bufora).

Prawidłowo odczytany bajt jest przepisywany z akumulatora w miejsce określone zawartością rejestru ICBAZ. Aby następny znak został wpisany w kolejne miejsce pamięci, to zawartość ICBAZ jest zwiększane przez procedurę INCBUFP.

```
0100 ;INCrement BUFfer Pointer
0110 ;
0120 ICBAZ = $24
0130 ;
0140     *=  $E6D1
0150 ;
0160     INC ICBAZ
0170     BNE END
0180     INC ICBAZ+1
0190 END RTS
```

Teraz kontrolowany jest bit 1 kodu operacji, który określa jej rodzaj (zob. wyżej). Gdy odczytywany jest bajt, to CIOREAD kończy się po zmniejszeniu rejestru określającego długość bufora (poprzez procedurę DECBUFL). Dla wybrania odpowiedniego wariantu procedury bit 1 jest kontrolowany potem jeszcze raz. Przy odczytywaniu rekordu trzeba sprawdzić, czy odczytany znak jest kodem końca linii ($9B - RETURN). Jeżeli tak, to po zmniejszeniu długości bufora odczyt również się kończy.

W przeciwnym razie także zmniejszana jest długość bufora i gdy jest większa od zera, to procedura odczytu bajtu jest powtarzana. W celu zmniejszenia wskaźnika długości bufora (ICBLZ) wywoływana jest specjalna procedura DECBUFL. Zwraca ona w akumulatorze wartość zero, jeśli długość bufora jest równa zero lub wartość niezerową w przeciwnym wypadku, co ułatwia wybór wariantu procedury.

```
0100 ;DECrement BUFfer Length
0110 ;
```

```
0120 ICBLZ = $28
0130 ;
0140     *=  $E6BB
0150 ;
0160     LDA ICBLZ
0170     BNE SKIP
0180     DEC ICBLZ+1
0190 SKIP DEC ICBLZ
0200     LDA ICBLZ
0210     ORA ICBLZ+1
0220     RTS
```

Jeżeli cały bufor został już zapełniony (długość bufora równa zero), to procedura GOHAND jest wywoływana w pętli w celu 2odczytania pozostałej części rekordu. Odczytane bajty nie są nigdzie przechowywane i zostają stracone. Pętla jest przerywana po wystąpieniu błędu lub po napotkaniu kodu RETURN ($9B). W tym ostatnim przypadku w rejestrze Y umieszczany jest kod błędu $89 (TRUNCATED RECORD - część rekordu utracona). Ponieważ bufor jest już pełny, to procedura DECBUFP zmniejsza jego adres o jeden.

```
0100 ;DECrement BUFfer Pointer
0110 ;
0120 ICBAZ = $24
0130 ;
0140     *=  $E6C8
0150 ;
0160     LDA ICBAZ
0170     BNE END
0180     DEC ICBAZ+1
0190 END DEC ICBAZ
0200     RTS
```

W uzyskane w ten sposób miejsce wpisywany jest znak RETURN i adres bufora jest ponownie zwiększany przez procedurę INCBUFP.

Po wszystkich opisanych wyżej wariantach zakończenia odczytu, przed opuszczeniem CIOREAD jest jeszcze wywoływana procedura SUBBUFL, której zadaniem jest uaktualnienie długości bufora dla operacji CIO. Wykorzystuje ona do rozpoznania właściwego IOCB jego numer zapisany w rejestrze ICAX5Z.

```
0100 ;SUBtract BUFfer Length
0110 ;
0120 ICAX5Z = $2E
0130 ICBLZ = $28
0140 ICBUFL = $0348
0150 ;
0160     *=  $E6D8
0170 ;
0180     LDX ICAX5Z
0190     SEC
0200     LDA ICBUFL,X
0210     SBC ICBLZ
0220     STA ICBLZ
0230     LDA ICBUFL+1,X
0240     SBC ICBLZ+1
```

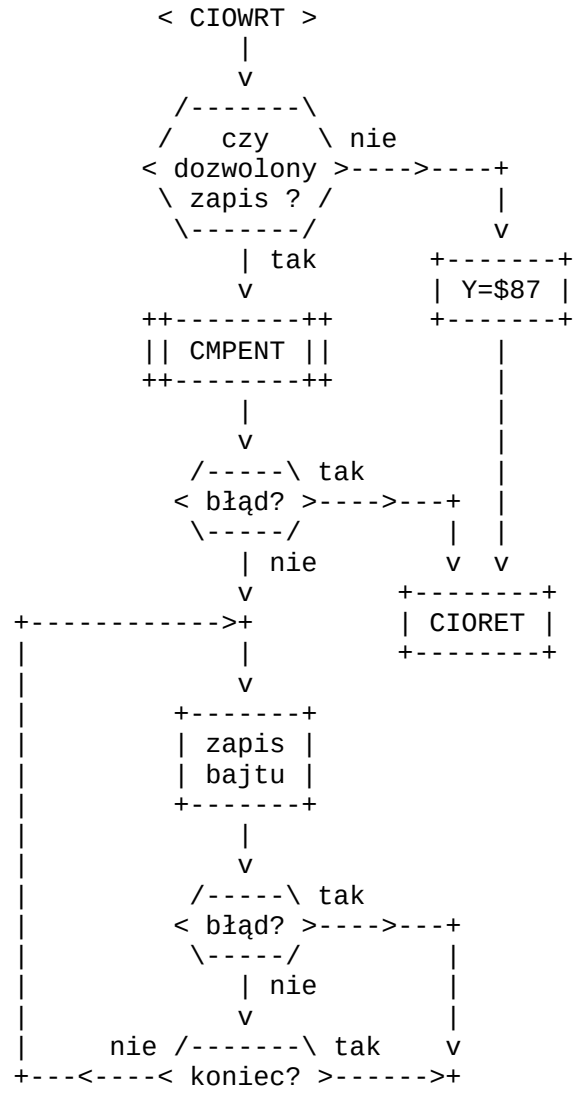
```
0250     STA ICBLZ+1
0260     RTS
```

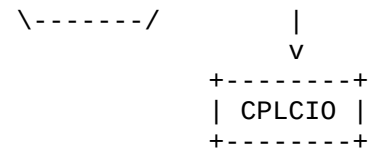
Procedura odejmuje od początkowej długości bufora wpisanej w ICBUFL długość jego części zajętej przez odczytane dane (zawiera ją rejestr ICBLZ). W wyniku otrzymujemy długość pozostałej jeszcze (pustej) części bufora.

Po zapoznaniu się z przebiegiem CIOREAD można łatwo określić różnicę pomiędzy operacjami GET BYTE i GET RECORD. Przy odczycie rekordu znak RETURN ($9B) przerywa odczyt, a jego brak powoduje przepełnienie bufora i utratę części informacji. Operacja GET BYTE kończy się natomiast po zapełnieniu bufora, a znak o kodzie $9B jest traktowany tak, jak wszystkie pozostałe.

2.6. Procedura zapisu na urządzenie.

Kody operacji $09 (PUT RECORD) i $0B (PUT BYTE) powodują wywołanie z CIOMAIN procedury CIOWRT. Przebiega ona w sposób bardzo zbliżony do procedury CIOREAD, a wymagane dla niej parametry są identyczne (ICBUFA i ICBUFL). Struktura procedury CIOWRT jest pokazana na rysunku 7.





Rys.7. Struktura operacji WRITE

Podobnie jak przy odczycie na początku kontrolowany jest bit 3 rejestru ICAX1, który sygnalizuje możliwość zapisu na urządzenie. Jeśli jest on skasowany, to do rejestru Y wpisywany

```
0100 ;CIO WRiTe routine
0110 ;
0120 CIORET = $E670
0130 CPLCIO = $E672
0140 CMPENT = $E695
0150 DECBUFL = $E6BB
0160 GOHAND = $E6EA
0170 ICAX1Z = $2A
0180 ICAX6Z = $2F
0190 ICBAZ = $24
0200 ICBLZ = $28
0210 ICCOMZ = $22
0220 INCBUFP = $E6D1
0230 SUBBUFL = $E6D8
0240 ;
0250     *=  $E61E
0260 ;
0270     LDA ICCOMZ
0280     AND ICAX1Z
0290     BNE PRFM
0300     LDY #$87
0310 EXIT JMP CIORET
0320 PRFM JSR CMPENT
0330     BCS EXIT
0340     LDA ICBLZ
0350     ORA ICBLZ+1
0360     BNE LBF
0370     LDA ICAX6Z
0380     INC ICBLZ
0390     BNE PUT
0400 LBF LDY #$00
0410     LDA (ICBAZ),Y
0420     STA ICAX6Z
0430 PUT JSR GOHAND
0440     PHP
0450     JSR INCBUFP
0460     JSR DECBUFL
0470     PLP
0480     BMI END
0490     LDA ICCOMZ
0500     AND #$02
0510     BNE NEXT
0520     LDA ICAX6Z
0530     CMP #$9B
0540     BEQ END
0550 NEXT LDA ICBLZ
0560     ORA ICBLZ+1
0570     BNE LBF
```

```
0580     LDA ICCOMZ
0590     AND #$02
0600     BNE END
0610     LDA #$9B
0620     JSR GOHAND
0630 END JSR SUBBUFL
0640     JMP CPLCIO
```

jest kod błędu $87 (READ ONLY - dozwolony tylko odczyt) i operacja kończy się skokiem do CIORET. Następnie przez CMPENT wyszukiwany jest adres procedury zapisu (w przypadku niepowodzenia skok do CIORET z kodem $85 w rejestrze Y).

Poprzez sprawdzenie długości bufora określane jest miejsce pobrania znaku do wysłania. Jeżeli bufor ma zerową długość, to znak pobierany jest z akumulatora, w przeciwnym razie wysyłany jest pierwszy znak z bufora. Samo wysłanie bajtu, który zostaje przepisany do ICAX6Z, jest wykonywane przez wywołanie procedury obsługi urządzenia poprzez GOHAND. Następnie przez wywołanie INCBUFP i DECBUFL zwiększany jest adres bufora i zmniejszana jego długość.

Teraz wykonywana jest seria sprawdzeń różnych warunków. Procedura zapisu jest przerywana, jeśli wystąpił błąd przy zapisie, jeśli zapisywany był bajt (PUT BYTE) i długość bufora jest równa zero lub jeśli przy zapisywaniu rekordu został wysłany znak RETURN. Jeżeli przy zapisie bajtów długość bufora jest różna od zera, to następny bajt z bufora jest przepisywany do ICAX6Z i zapis jest powtarzany. Zerowa długość bufora w przypadku zapisu rekordu powoduje jeszcze przed zakończeniem operacji zapisanie bajtu $9B (RETURN).

We wszystkich wymienionych przypadkach zakończenia operacji zapisu, przed opuszczeniem CIOWRT jest jeszcze uaktualniana długość bufora (przy pomocy SUBBUFL). Cała operacja zapisu kończy się skokiem do CPLCIO.

Dokładniejsze informacje o procedurach DECBUFL, INCBUFP i SUBBUFL znajdują się w rozdziale 2.5. (Procedura odczytu z urządzenia).

2.7. Odczyt statusu i procedury specjalne

Wszystkie kody operacji o wartości równej lub większej od $0D powodują wywołanie procedury CIOSTSP. Kod $0D oznacza operację odczytu statusu urządzenia, zaś wyższe kody są rozkazami operacji specjalnych, których przebieg zależy od rodzaju urządzenia.

Procedura ta, jako jedyna z procedur CIO, umożliwia wykonanie operacji BEZ uprzedniego otwierania urządzenia operacją OPEN. Najpierw sprawdzany jest bowiem indeks wpisu w HATABS (zawarty w rejestrze ICHIDZ). Jeżeli ma on wartość $FF, to wywoływana jest procedura DEVNUM, która wyszukuje prawidłowy indeks (wystąpienie przy tym błędu przerywa operację skokiem do CIORET).

```
0100 ;CIO STatus/SPecial routine
```

```
0110 ;
0120 CPLCIO = $E672
0130 CMPENT = $E695
0140 DEVNUM = $E6FF
0150 GOHAND = $E6EA
0160 ICAX5Z = $2E
0170 ICCHID = $0340
0180 ICHIDZ = $20
0190 ;
0200     *=  $E597
0210 ;
0220     LDA ICHIDZ
0230     CMP #$FF
0240     BNE PRFM
0250     JSR DEVNUM
0260     BCS $E547     ;do JMP CIORET
0270 PRFM JSR CMPENT
0280     JSR GOHAND
0290     LDX ICAX5Z
0300     LDA ICCHID,X
0310     STA ICHIDZ
0320     JMP CPLCIO
```

Dalszy przebieg procedury CIOSTSP jest podobny do opisanych wcześniej. Adres sterownika jest odszukiwany przez wywołanie CMPENT, a sama procedura obsługi wywoływana jest poprzez GOHAND. Ponieważ podczas procedury CIOMAIN zmieniane są kody operacji większe od $0D, to przed zakończeniem CIOSTSP prawidłowy kod jest odtwarzany według zawartości właściwego IOCB. Procedura kończy się skokiem do CPLCIO.

Rozdział 3

OBSŁUGA URZĄDZEŃ ZEWNĘTRZNYCH

System operacyjny Atari zawiera procedury obsługi pięciu urządzeń zewnętrznych. Są to: edytor (E: - editor), ekran (S: - screen), klawiatura (K: - keyboard), magnetofon (C: - cassette) i drukarka (P: - printer). Bliższego wyjaśnienia wymaga jedynie edytor. Jest to urządzenie zewnętrzne składające się z ekranu w trybie 0 i klawiatury.

Procedury obsługujące komunikację poprzez wyżej wymienione urządzenia oraz ich wektory znajdują się w pamięci ROM komputera. Można je więc wykorzystać we własnych programach. Korzystając z podanych opisów można również ułożyć własne procedury obsługi tych urządzeń i wykorzystywać je poprzez zmianę wpisu w HATABS.

Opis trzech pierwszych urządzeń (E, S i K) jest dość skomplikowany, ponieważ ich procedury wzajemnie się wywołują. Sprawia to wrażenie pewnego bałaganu, lecz jest to bałagan pozorny. Zastosowany system pozwala na efektywną pracę wszystkich urządzeń i jednocześnie oszczędza znaczny obszar pamięci. W związku z tym opis został ułożony w kolejności ułatwiającej zrozumienie, lecz niezupełnie dostosowanej do struktury CIO.

3.1. Procedury obsługi klawiatury

Klawiatura jest urządzeniem wejścia, to znaczy, że można z niej tylko odczytywać dane. Ponieważ klawiatura jest fizycznie wbudowana do komputera, to część procedur ma znaczenie prawie symboliczne i służy jedynie do zapewnienia poprawnego przebiegu operacji CIO. Odpowiednie wektory w tabeli adresowej KBDVEC wskazują adresy następujących procedur:

```
OPEN    - KBOPN
CLOSE   - KBOPN
GET     - KBGBYT
PUT     - EDSP
STATUS  - KBOPN
SPECIAL - EDSP
```

System operacyjny nie przewiduje dla klawiatury operacji zapisu i operacji specjalnych. Etykietą EDSP jest oznaczony rozkaz RTS w procedurze KBOPN. Po wywołaniu jednej z tych operacji powoduje to uzyskanie w wyniku błędu $92 (FUNCTION NOT IMPLEMENTED). Zostało to wyjaśnione przy opisie GOHAND.

3.1.1. Odczyt z klawiatury

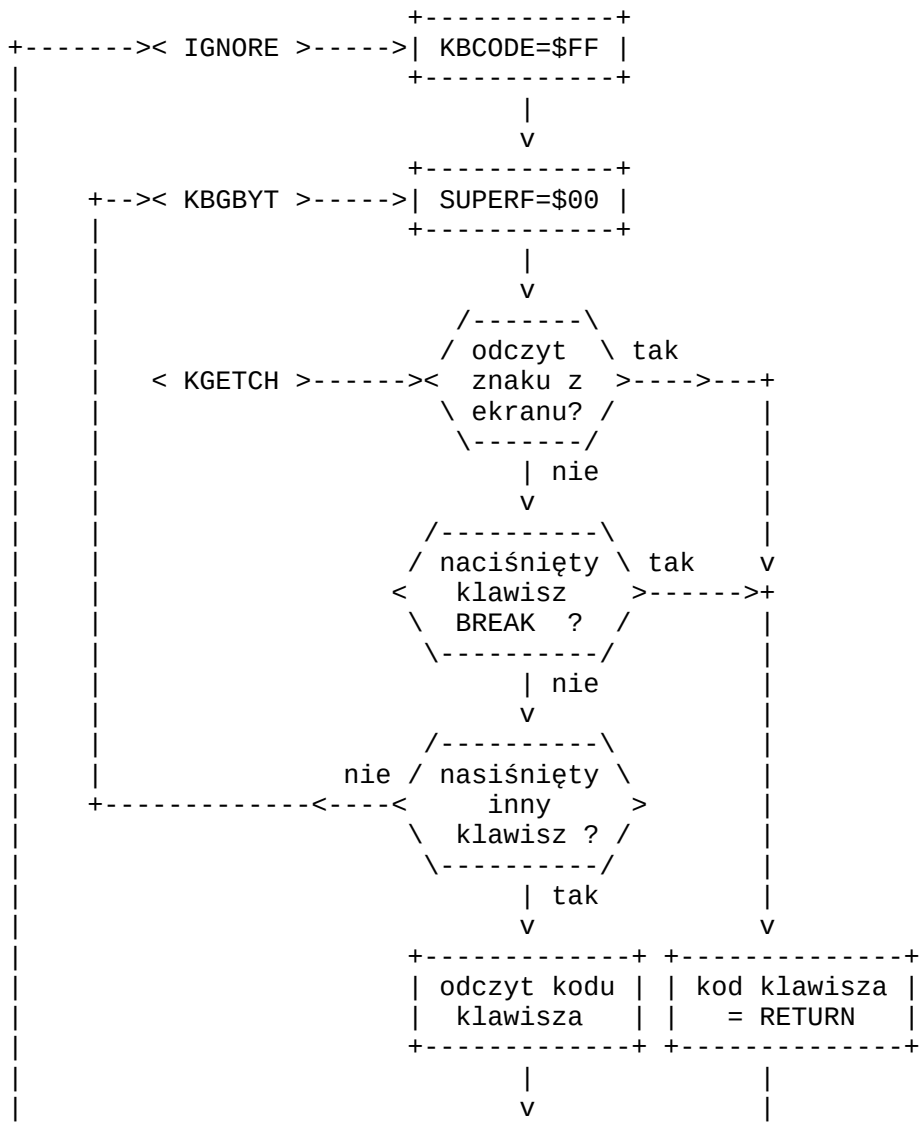
Procedura odczytu z klawiatury służy do odczytu znaku w kodzie ATASCII (ATari ASCII). Oprócz pobrania znaku z odpowiedniego rejestru musi więc jeszcze rozpoznać specjalne kombinacje klawiszy, które nie mają swoich odpowiedników w tym kodzie. Schematyczny przebieg procedury jest przedstawiony na rysunku 8 (str. 42).

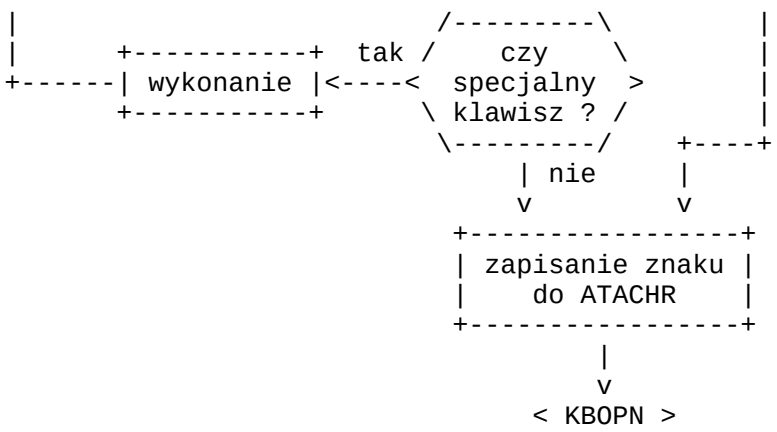
Rozpoczyna się ona nie od fizycznego początku (IGNORE), lecz od etykiety KBGBYT. Najpierw zerowany jest znacznik SUPERF (SUPER Flag) i sprawdzany bit 0 w ICAX1Z. Gdy jest on ustawiony, to znaczy, że odczyt wykonywany jest z ekranu. Do rejestru ATACHR (ATASCII CHaRacter) wpisywany jest więc znak RETURN ($9B), co wywołuje potem odczyt z edytora i procedura się kończy. Jeśli znak ma być odczytany z klawiatury, to sprawdzany jest rejestr KBCODES (KeyBoard CODE Shadow register). Wartość $FF w tym rejestrze oznacza, że żaden klawisz nie został naciśnięty. W takim przypadku następuje skok do KBGBYT i opisane wyżej operacje są powtarzane, aż do naciśnięcia klawisza.

```
0100 ATACHR = $02FB
0110 DSTAT = $4C
0120 FKDEFP = $60
0130 HOLDCH = $7C
0140 ICAX1Z = $2A
0150 INVFLG = $02B6
0160 IRQSTAT = $11
0170 KBCODES = $02FC
0180 KBOPN = $F21E
0190 KEYCLK = $F983
0200 KEYDEFP = $79
0210 NOCLIK = $02DB
0220 SHFLOK = $02BE
0230 SUPERF = $03E8
0240 TSTCNT = $F93C
0250 ;
0260     *=  $F2F8
0270 ;
0280 IGNORE LDA #$FF
0290     STA KBCODES
0300 ;
0310 ;KeyBoard Get BYTe
0320 ;
0330 KBGBYT LDA #$00
0340     STA SUPERF
0350 ;
0360 ;Keyboard GET CHaracter
0370 ;
0380 KGETCH LDA ICAX1Z
0390     LSR A
0400     BCS RET
0410     LDA #$80
0420     LDX IRQSTAT
0430     BEQ STAT
0440     LDA KBCODES
0450     CMP #$FF
0460     BEQ KBGBYT
0470     STA HOLDCH
0480     LDX #$FF
0490     STX KBCODES
0500     LDX NOCLIK
0510     BNE KEY
0520     JSR KEYCLK
0540 KEY TAY
0550     CPY #$C0
0560     BCS IGNORE
```

```
0570     LDA (KEYDEFP),Y
0590 SAC STA ATACHR
0600     TAX
0610     BMI INV
0620     JMP CTR
0630 INV CMP #$80
0640     BEQ IGNORE
0650     CMP #$81
0660     BNE SHF
0670     LDA INVFLG
0680     EOR #$80
0690     STA INVFLG
0700     BCS IGNORE
0710 SHF CMP #$82
0720     BNE CPL
0730     LDA SHFLOK
0740     BEQ LOW
0750     LDA #$00
0760     STA SHFLOK
0770     BEQ IGNORE
0780 CPL CMP #$83
0790     BNE CTL
0800 LOW LDA #$40
0810     STA SHFLOK
0820     BNE IGNORE
0830 CTL CMP #$84
0840     BNE EOF
0850     LDA #$80
0860     STA SHFLOK
0870     JMP IGNORE
0880 EOF CMP #$85
0890     BNE KCL
0900     LDA #$88
0910 STAT STA DSTAT
0920     STA IRQSTAT
0930 RET LDA #$9B
0940     JMP FIN
0950 KCL CMP #$89
0960     BNE FNC
0970     LDA NOCLIK
0980     EOR #$FF
0990     STA NOCLIK
1000     BNE IGN
1010     JSR KEYCLK
1020 IGN JMP IGNORE
1030 FNC CMP #$BE
1040     BCS CFK
1050     CMP #$8A
1060     BCC IGN
1070     SBC #$8A
1080     ASL HOLDCH
1090     BPL FKY
1100     ORA #$04
1110 FKY TAY
1120     LDA (FKDEFP),Y
1130     JMP SAC
1140 CFK CMP #$92
1150     BCS CTR
1160     CMP #$8E
1170     BCC IGN
```

```
            1180     SBC #$72
            1190     INC SUPERF
            1200     BNE FIN
            1220 CTR LDA HOLDCH
            1230     CMP #$40
            1240     BCS CTRL
            1250     LDA ATACHR
            1260     CMP #$61
            1270     BCC CTRL
            1280     CMP #$7B
            1290     BCS CTRL
            1300     LDA SHFLOK
            1310     BEQ CTRL
            1320     ORA HOLDCH
            1330     JMP KEY
            1340 CTRL JSR TSTCNT
            1350     BEQ EXIT
            1360     LDA ATACHR
            1370     EOR INVFLG
            1390 FIN STA ATACHR
            1400 EXIT JMP KBOPN
```





Rys.8. Odczyt znaku z klawiatury

Po naciśnięciu klawisza jego kod jest przepisywany do rejestru HOLDCH (CHOLD CHaracter), a rejestr KBCODE otrzymuje wartość $FF, aby umożliwić następny odczyt. Jeśli zawartość rejestru NOCLIK (NO key CLIcK) jest równa zero, to wywoływana jest procedura KEYCLK. Generuje ona dźwięk naciśnięcia klawisza przez zapisywanie 127 razy rejestru CONSOL (jego bit 3 steruje dźwiękiem).

```
0100 ;KEY CLicK
0110 ;
0120 CONSOL = $D01F
0130 VCOUNT = $D40B
0140 ;
0150     *=  $F983
0160 ;
0170     LDX #$7E
0180     PHA
0190 NEXT STX CONSOL
0200     LDA VCOUNT
0210 WAIT CMP VCOUNT
0220     BEQ WAIT
0230     DEX
0240     DEX
0250     BPL NEXT
0260     PLA
0270     RTS
```

Teraz kod naciśniętego klawisza jest zamieniany na kod ATASCII według wartości umieszczonych w tabeli definicji klawiszy (KEYDEF). Ponieważ w tabeli wpisane są tylko 192 kody, to przedtem kody wyższe od $BF są eliminowane przez skok do etykiety IGNORE. Tabela definicji klawiszy znajduje się wprawdzie w pamięci ROM, lecz odwołanie do niej następuje według wektora KEYDEFP (KEY DEFinitions Pointer). Przez utworzenie nowej tabeli i zmianę tego wektora można więc zmienić znaczenie prawie wszystkich klawiszy, co zresztą jest stosowane w wielu programach użytkowych.

Na przykład, zdefiniowanie nowego zestawu znaków i przedefiniowanie klawiatury umożliwia stworzenie bazy danych posiadającej polskie litery i sortującej dane z ich uwzględnieniem. Oczywiście pojawia się wtedy problem drukowania takiego tekstu, lecz można go rozwiązać przez

wpisanie nowej procedury obsługi drukarki, która normalnym znakom przywróci standardowy kod ASCII, a polskie litery wydrukuje w trybie graficznym lub jako złożenie dwóch znaków standardowych (np. "ł" = "l" + "/"). System operacyjny Atari daje tu programiście szerokie pole do popisu.

Po dokładnym przejrzeniu tabeli KEYDEF łatwo zauważyć, że kilka klawiszy naciśniętych wraz z CONTROL nie posiada żadnego znaczenia. Poprzez ich przedefiniowanie można więc uzyskać dodatkowe funkcje. Umieszczone w tabeli kody klawiszy F1-F4 dotyczą jedynie modelu 1200XL, który posiada te klawisze. Ich definicje można zmienić tylko w zakresie występującym w tabeli, ponieważ w kombinacji z CONTROL mają one specjalne funkcje, które są wykonywane już podczas przerwania IRQ wywołanego naciśnięciem klawisza.

```
0100 ;KEYboard DEFinitions table
0110 ;
0120 BL  =   $FD     ;bell
0130 BS  =   $7E     ;backspace
0140 CD  =   $1D     ;cursor down
0150 CK  =   $84     ;CTRL key lock
0160 CL  =   $1E     ;cursol left
0170 CP  =   $82     ;caps
0180 CR  =   $1F     ;cursor right
0190 CS  =   $7D     ;clear screen
0200 CT  =   $9E     ;clear TAB
0210 CU  =   $1C     ;cursor up
0220 DC  =   $FE     ;delete char
0230 DL  =   $9C     ;delete line
0240 EF  =   $85     ;end of file
0250 ES  =   $1B     ;escape
0260 F1  =   $8A     ;F1
0270 F2  =   $8B     ;F2
0280 F3  =   $8C     ;F3
0290 F4  =   $8D     ;F4
0300 IC  =   $FF     ;insert char
0310 IL  =   $9D     ;insert line
0320 IN  =   $81     ;inverse
0330 KC  =   $89     ;key click
0340 NU  =   $80     ;not used
0350 RT  =   $9B     ;return
0360 ST  =   $9F     ;set TAB
0370 TB  =   $7F     ;tabulate
0380 ;
0390     *=  $FB51
0400 ;
0410     .BYTE 'l,'j,';,F1
0420     .BYTE F2,'k,'+,'*
0430     .BYTE 'o,NU,'p,'u
0440     .BYTE RT,'i,'-,'=
0450     .BYTE 'v,NU,'c,F3
0460     .BYTE F4,'b,'x,'z
0470     .BYTE '4,NU,'3,'6
0480     .BYTE ES,'5,'2,'1
0490     .BYTE ',,' ,'.,'n
0500     .BYTE NU,'m,'/,IN
0510     .BYTE 'r,NU,'e,'y
0520     .BYTE TB,'t,'w,'q
0530     .BYTE '9,NU,'0,'7
```

```
0540     .BYTE BS,'8,'<,'>
0550     .BYTE 'f,'h,'d,NU
0560     .BYTE CP,'g,'s,'a
0570 ;with SHIFT
0580     .BYTE 'L,'J,':,F1
0590     .BYTE F2,'K,'\,'^
0600     .BYTE 'O,NU,'P,'U
0610     .BYTE RT,'I,'_,'|
0620     .BYTE 'V,NU,'C,F3
0630     .BYTE F4,'B,'X,'Z
0640     .BYTE '$,NU,'#,'&
0650     .BYTE ES,'%,'",'!
0660     .BYTE '[,' ,'],'N
0670     .BYTE NU,'M,'?,IN
0680     .BYTE 'R,NU,'E,'Y
0690     .BYTE ST,'T,'W,'Q
0700     .BYTE '(,NU,'),''
0710     .BYTE DL,'@,CS,IL
0720     .BYTE 'F,'H,'D,NU
0730     .BYTE CP+1,'G,'S,'A
0740 ;with CONTROL
0750     .BYTE $0C,$0A,$7B,NU
0760     .BYTE NU,$0B,CL,CR
0770     .BYTE $0F,NU,$10,$15
0780     .BYTE RT,$09,CU,CD
0790     .BYTE $16,NU,$03,KC
0800     .BYTE NU,$02,$18,$1A
0810     .BYTE NU,NU,EF,NU
0820     .BYTE ES,NU,BL,NU
0830     .BYTE $00,' ,$60,$0E
0840     .BYTE NU,$0D,NU,IN
0850     .BYTE $12,NU,$05,$19
0860     .BYTE CT,$14,$17,$11
0870     .BYTE NU,NU,NU,NU
0880     .BYTE DC,NU,CS,IC
0890     .BYTE $06,$08,$04,NU
0900     .BYTE CK,$07,$13,$01
```

Kod znaku odczytany z tabeli KEYDEF jest umieszczany w rejestrze ATACHR. Wartość kodu większa lub równa $80 oznacza specjalny klawisz (lub kombinację klawiszy). W takim przypadku procedura rozpoznaje jego znaczenie, ewentualnie wykonuje konieczną operację i ponownie oczekuje na naciśnięcie klawisza.

Kod $80 oznacza zabronioną kombinację klawiszy i jest po prostu ignorowany, a procedura przechodzi do początkowej etykiety IGNORE.

Kod $81 oznacza klawisz "inverse" (zwany także w starszych publikacjach Atari Logo Key). Powoduje on wykonanie operacji EOR #$FF na zawartości rejestru INVFLG (INVerse FLaG) przełączając tryb wyprowadzania znaków z normalnych na negatywy lub odwrotnie.

Kod $82 oznacza klawisz CAPS i powoduje przełączenie klawiatury z małych liter na duże lub odwrotnie. W pierwszym przypadku w rejestrze SHFLOK jest umieszczana wartość $40, a w drugim - $00.

Kod $83 oznacza kombinację SHIFT-CAPS i powoduje ustawienie klawiatury na pisanie dużymi literami przez wpisanie wartości $40 do rejestru SHFLOK.

Kod $84 oznacza kombinację CONTROL-CAPS i powoduje, przez wpisanie wartości $80 do rejestru SHFLOK, uzyskiwanie znaków pseudograficznych bez naciskania klawisza CONTROL. Tryb ten jest kasowany zarówno przez klawisz CAPS, jak i przez kombinację SHIFT-CAPS.

Kod $85 oznacza znak EOF (End Of File), który jest uzyskiwany przez naciśnięcie CONTROL-3. W tym przypadku kod błędu EOF ($88) jest wpisywany do rejestru DSTAT (Display STATus) oraz do IRQSTAT (Interrupt ReQuest STATus), a w ATACHR umieszczany jest kod $9B (RETURN). Teraz, zamiast oczekiwania na następny znak, procedura jest przerywana skokiem do KBOPN.

Kod $89 oznacza naciśnięcie klawiszy CONTROL-F3 i powoduje włączenie lub wyłączenie dźwięku klawiatury. W tym celu na zawartości rejestru NOCLIK jest wykonywana operacja EOR #$FF. Jeśli w jej wyniku NOCLIK ma wartość zero, to poprzednio ominięta procedura KEYCLK jest teraz wywoływana.

Kody od $8A do $8D oznaczają klawisze funkcyjne F1-F4 (1200XL), zarówno naciśnięte oddzielnie, jak i razem z SHIFT. W celu ich odróżnienia właściwy kod jest odczytywany z tabeli FKDEF z uwzględnieniem kodu klawiatury (z rejestru HOLDCH). Po odczycie kodu następuje powrót do etapu rozpoznawania znaku.

```
0100 ;Function Key DEFinitions table
0110 ;
0120 CB  =   $8F     ;cursor bottom
0130 CD  =   $1D     ;cursor down
0140 CH  =   $8E     ;cursor home
0150 CL  =   $1E     ;cursor left
0160 CR  =   $1F     ;cursor right
0170 CU  =   $1C     ;cursor up
0180 LM  =   $90     ;cursor to left margin
0190 RM  =   $91     ;cursor to right margin
0200 ;
0210     *=  $FC11
0220 ;
0230     .BYTE CU,CD,CL,CR
0240     .BYTE CH,CB,LM,LR
```

Tabela FKDEF jest dostępna przez wektor FKDEFP, podobnie jak KEYDEF. Zmiana tego wektora stosowana jest jednak bardzo sporadycznie, ze względu na brak klawiszy funkcyjnych we wszyskich modelach XL/XE poza 1200XL.

Kody z zakresu od $8E do $91 oznaczają kombinację klawisza SHIFT z jednym z klawiszy funkcyjnych (F1-F4). Rozpoznanie takiego kodu powoduje zmniejszenie go o $72, zwiększenie znacznika SUPERF i po zapisaniu kodu w rejestrze ATACHR skok do procedury KBOPN.

```
0100 ;TeST CoNTrols
0110 ;
0120 ATACHR = $02FB
0130 CNTRLS = $FB0D
0140 ;
0150     *=  $F93C
0160 ;
0170     LDX #$2D
0180 NEXT LDA CNTRLS,X
0190     CMP ATACHR
0200     BEQ EXIT
0210     DEX
0220     DEX
0230     DEX
0240     BPL NEXT
0250 EXIT RTS
```

```
0100 ;CoNTRoLS table
0110 ;
0120 BELL =  $F556
0130 CLRSCR = $F420
0140 CRSBS = $F450
0150 CRSCTB = $F49A
0160 CRSDWN = $F3F3
0170 CRSLFT = $F400
0180 CRSRGT = $F411
0190 CRSSTB = $F495
0200 CRSTAB = $F47A
0210 CRSUP = $F3E6
0220 DELCHR = $F4D5
0230 DELLIN = $F520
0240 ESCAPE = $F3E0
0250 INSCHR = $F49F
0260 INSLIN = $F50C
0270 RTWSCR = $F661
0280 ;
0290     *=  $FB0D
0300 ;
0310     .BYTE $1B
0320     .WORD ESCAPE
0330     .BYTE $1C
0340     .WORD CRSUP
0350     .BYTE $1D
0360     .WORD CRSDWN
0370     .BYTE $1E
0380     .WORD CRSLFT
0390     .BYTE $1F
0400     .WORD CRSRGT
0410     .BYTE $7D
0420     .WORD CLRSCR
0430     .BYTE $7E
0440     .WORD CRSBS
0450     .BYTE $7F
0460     .WORD CRSTAB
0470     .BYTE $9B
0480     .WORD RTWSCR
0490     .BYTE $9C
0500     .WORD DELLIN
0510     .BYTE $9D
0520     .WORD INSLIN
```

```
0530     .BYTE $9E
0540     .WORD CRSCTB
0550     .BYTE $9F
0560     .WORD CRSSTB
0570     .BYTE $FD
0580     .WORD BELL
0590     .BYTE $FE
0600     .WORD DELCHR
0610     .BYTE $FF
0620     .WORD INSCHR

0630     .BYTE $1C
0640     .WORD CRSHOM
0650     .BYTE $1D
0660     .WORD BTMLIN
0670     .BYTE $1E
0680     .WORD CRSLMR
0690     .BYTE $1F
0700     .WORD CRSRMR
```

Jeżeli dotąd nie został rozpoznany żaden ze znaków specjalnych, to znaczy, że został naciśnięty normalny klawisz (lub kombinacja). Teraz następuje ostatnia seria sprawdzeń. Jeśli kod klawisza jest większy lub równy $40 (naciśnięty dowolny klawisz razem z CONTROL lub SHIFT) albo kod ATASCII jest mniejszy od $61 lub większy od $7A (znak nie jest małą literą z zakresu od "a" do "z") albo znacznik SHFLOK ma wartość zero (tryb pisania małymi literami), to wywoływana jest procedura TSTCNT, która rozpoczyna końcową fazę KBGBYT. W przeciwnym razie po ustawieniu w rejestrze HOLDCH bitów, które są ustawione w SHFLOK, procedura powraca do odczytu z tabeli KEYDEF (do etykiety KEY).

Procedura TSTCNT przeszukuje tabelę znaków kontrolnych edytora porównując zawarte w niej wpisy z zawartością rejestru ATACHR. Ponieważ poszukiwanie znaku odbywa się od końca, to gdy nie zostanie znaleziony, bit Zero jest skasowany. Znalezienie znaku powoduje ustawienie bitu Zero przez rozkaz CMP.

Jeśli po zakończeniu TSTCNT bit zero jest ustawiony, to procedura odczytu kończy się bezpośrednio skokiem do KBOPN. W przeciwnym razie, przed opuszczeniem procedury KBGBYT, najstarszy bit znaku w rejestrze ATACHR jest ustawiany według zawartości znacznika INVFLG.

3.1.2. Pozostałe procedury klawiatury

Wektory operacji OPEN, CLOSE i STATUS dla klawiatury wskazują na procedurę KBOPN. Nie jest ona samodzielnym elementem systemu, lecz stanowi część procedury RETURM. W tej części do rejestru Y pobierany jest z DSTAT status edytora i wpisywana jest tam wartość 1 (SUCCESS), a do akumulatora przenoszona jest zawartość rejestru ATACHR. Dokładny opis procedury RETURM znajduje się w rozdziale 3.2.1. (Procedura otwarcia ekranu - str. 58).

UWAGA! System operacyjny zawiera dwie procedury o bardzo zbliżonych nazwach: RETUNM i RETURN. Nie należy ich ze sobą mylić.

3.2. Procedury obsługi ekranu

Ekran jest w zasadzie urządzeniem wyjścia, lecz możliwy jest również odczyt z niego. Ponieważ ekran w trybie 0 jest częścią edytora, to procedury obsługujące te urządzenia są ze sobą mocno powiązane. W tym rozdziale opisane zostały wszystkie procedury dotyczące ekranu, a ich elementy dotyczące edytora zostały wyjaśnione w następnym rozdziale (3.3.).

Tabela adresowa ekranu SCRVEC zawiera następujące wektory:

```
OPEN     - SCOPN
CLOSE    - SCRFIN
GET      - GETCH
PUT      - OUTCH
STATUS   - KBOPN
SPECIAL  - DRAW
```

W komputerach serii XL/XE tworzeniem obrazu zajmuje sie drugi, dodatkowy mikroprocesor (tzw. ANTIC - zob. rozdział 7). Posiada on własną listę rozkazów i własny program. ANTIC umożliwia uzyskanie 14 trybów graficznych, z których jeden (tryb 3 ANTIC-a) jest niedostępny dla systemu operacyjnego. Dodatkowe trzy tryby tworzy specjalny układ graficzny GTIA. Razem mamy więc 16 trybów (a z kombinacjami 64). Tryby te są inaczej numerowane przez ANTIC, a inaczej przez OS. W tym i następnym rozdziale używana jest numeracja trybów stosowana przez system operacyjny (identyczna jak w Atari Basic), a ewentualne odstępstwa od tej zasady są odpowiednio zaznaczone.

3.2.1. Procedura otwarcia ekranu

Procedura otwarcia jest wspólna dla ekranu i edytora. Różnica występuje jedynie na początku, gdyż ekran może być otwarty w dowolnym trybie graficznym. Przy otwieraniu ekranu w trybie 0 przebieg ich jest identyczny.

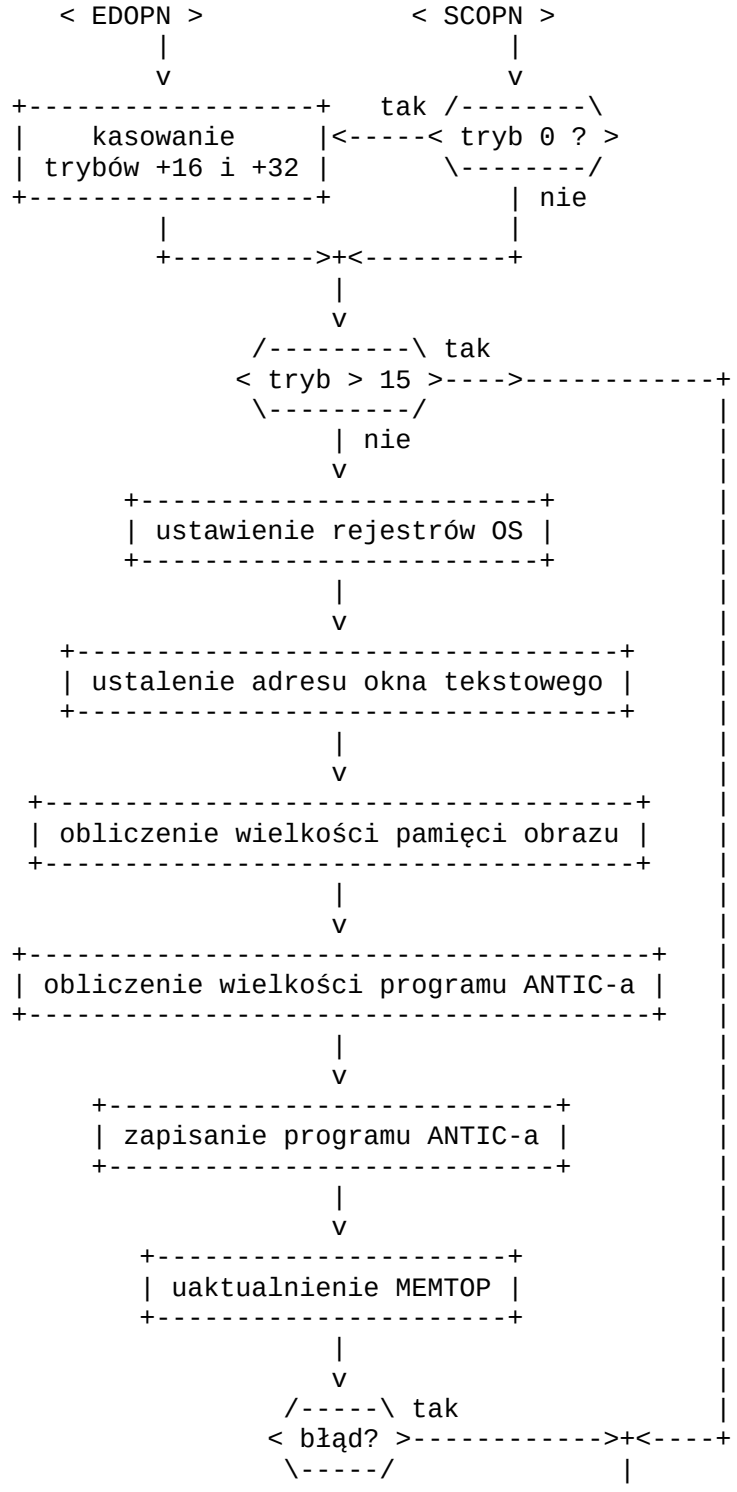
Znaczna długość procedury wynika z dwóch powodów. Seria XL/XE obejmuje komputery o różnej wielkości pamięci RAM, a od tego zależy położenie pamięci obrazu, ponadto zajmowany przez nią obszar zależy od wybranego trybu. Konieczne jest więc obliczenie wielkości i adresu pamięci obrazu. Zastosowanie do tworzenia obrazu drugiego procesora wymaga ułożenia dla niego programu. Ze względu na dużą liczbę trybów programy dla nich zajęłyby kilka kilobajtów pamięci, nie mogą więc znajdować się w ROM-ie. Program ANTIC-a jest każdorazowo tworzony od nowa podczas procedury otwarcia i zapisywany w pamięci RAM bezpośrednio przed obszarem pamięci obrazu. Czynność ta zajmuje dużą część procedury.

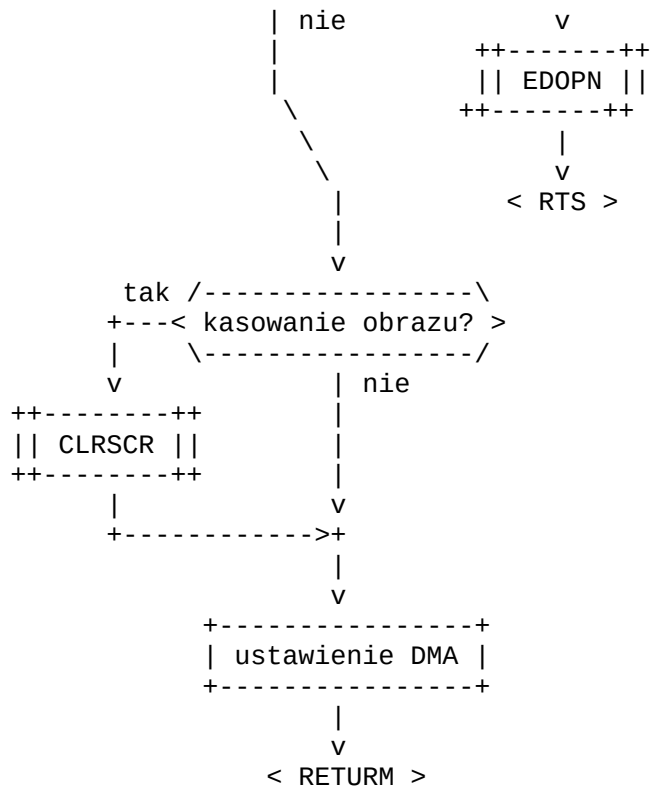
Przed wywołaniem procedury CIO w celu otwarcia ekranu należy w rejestrach ICAX1 i ICAX2

umieścić parametry określające sposób dostępu, tryb graficzny i inne elementy obrazu. W rejestrze ICAX1 znaczenie mają tylko cztery bity (pozostałe są niewykorzystane):

```
bit 2 - odczyt
bit 3 - zapis
bit 4 - brak okna tekstowego
bit 5 - zawartość pamięci obrazu bez zmian
```

Wymienione opcje są aktywne, gdy odpowiadające im bity są ustawione.





Rys.9. Otwarcie ekranu i edytora

```
0100 ADRESS = $64
0110 BOTSCR = $02BF
0120 CHACT = $02F3
0130 CHARSET1 = $E000
0140 CHARSET2 = $CC00
0150 CHBAS = $02F4
0160 CHSPTR = $026B
0170 CLRSCR = $F420
0180 COLBAKS = $02C8
0190 COLPF0S = $02C4
0200 COLTAB = $FB08
0210 CRSINH = $02F0
0220 DBDEC = $F565
0230 DCUSAC = $F57A
0240 DERRF = $03EC
0250 DINDEX = $57
0260 DLIV =  $0200
0270 DLPTRS = $0230
0280 DMACTLS = $022F
0290 DSTAT = $4C
0300 FINE =  $026E
0310 FSDL =  $FCC4
0320 GTICTLS = $026F
0330 HOLD1 = $51
0340 ICAX1Z = $2A
0350 ICAX2Z = $2B
0360 IRQENS = $10
0370 IRQST = $D20E
0380 LMARGN = $52
0390 MEMTOP = $02E5
0400 MLTTMP = $66
0410 NMIEN = $D40E
0420 RAMTOP = $6A
0430 RETURM = $F20B
0440 SAVADR = $68
0450 SAVMSC = $58
0460 SGDEC = $F578
0470 SSDLE = $F5A0
0480 STDDSP = $F570
0490 STDFS = $F569
0500 SWPFLG = $7B
0510 TABMAP = $02A3
0520 TAGRM = $EE4D
0530 TDLEC = $EE2D
0540 TDLVL = $EE5D
0550 TINDEX = $0293
0560 TSMAL = $EE1D
0570 TXTCOL = $0291
0580 TXTMSC = $0294
0590 TXTROW = $0290
0600 VCOUNT = $D40B
0610 ;
0620     *=  $EF8E
0630 ;
0640 ;SCreen OPeN
0650 ;
0660 SCOPN LDA ICAX2Z
0670     AND #$0F
```

```
0680     BNE SMD
0690 ;
0700 ;EDitor OPeN
0710 ;
0720 EDOPN LDA ICAX1Z
0730     AND #$0F
0740     STA ICAX1Z
0750     LDA #$00
0760 SMD STA DINDEX
0770     CMP #$10
0780     BCC OPN
0790     LDA #$91
0800     JMP DERR
0810 OPN LDA # >CHARSET1
0820     STA CHBAS
0830     LDA # >CHARSET2
0840     STA CHSPTR
0850     LDA #$02
0860     STA CHACT
0870     STA DMACTLS
0880     LDA #$01
0890     STA DSTAT
0900     LDA #$C0
0910     ORA IRQENS
0920     STA IRQENS
0930     STA IRQST
0940     LDA #$40
0950     STA NMIEN
0960     BIT FINE
0970     BPL NFS
0980     LDA # <FSDL
0990     STA DLIV
1000     LDA # >FSDL
1010     STA DLIV+1
1020     LDA #$C0
1030 NFS STA NMIEN
1040     LDA #$00
1050     STA TINDEX
1060     STA ADRESS
1070     STA SWPFLG
1080     STA CRSINH
1090     LDY #$0E
1100     LDA #$01
1110 LOOP1 STA TABMAP,Y
1120     DEY
1130     BPL LOOP1
1140     LDX #$04
1150 LOOP2 LDA COLTAB,X
1160     STA COLPF0S,X
1170     DEX
1180     BPL LOOP2
1190     LDY RAMTOP
1200     DEY
1210     STY TXTMSC+1
1220     LDA #$60
1230     STA TXTMSC

1800     STA BOTSCR
1810     LDA DINDEX
1820     CMP #$0C
1240     LDX DINDEX
1250     LDA TAGRM,X
1260     STA HOLD1
1270     LDA RAMTOP
1280     STA ADRESS+1
1290     LDY TSMAL,X
1300 LOOP3 LDA #$28
1310     JSR DCUSAC
1320     DEY
1330     BNE LOOP3
1340     LDA GTICTLS
1350     AND #$3F
1360     STA MLTTMP+1
1370     TAY
1380     CPX #$08
1390     BCC SMS
1400     CPX #$0F
1410     BEQ LSM
1420     CPX #$0C
1430     BCS SMS
1440     TXA
1450     ROR A
1460     ROR A
1470     ROR A
1480     AND #$C0
1490     ORA MLTTMP+1
1500     TAY
1510 LSM LDA #$10
1520     JSR DCUSAC
1530     CPX #$0B
1540     BNE SMS
1550     LDA #$06
1560     STA COLBAKS
1570 SMS STY GTICTLS
1580     LDA ADRESS
1590     STA SAVMSC
1600     LDA ADRESS+1
1610     STA SAVMSC+1
1620 WAIT LDA VCOUNT
1630     CMP #$7A
1640     BNE WAIT
1650     JSR SGDEC
1660     LDA TDLVL,X
1670     BEQ SKIP
1680     LDA #$FF
1690     STA ADRESS
1700     DEC ADRESS+1
1710 SKIP JSR DBDEC
1720     LDA ADRESS
1730     STA SAVADR
1740     LDA ADRESS+1
1750     STA SAVADR+1
1760     LDA #$41
1770     JSR STDDSP
1780     STX MLTTMP
1790     LDA #$18

2360     SBC #$10
2370     JSR STDDSP
2380     LDA #$00
```

```
1830     BCS DL              2390     JSR STDDSP
1840     CMP #$09            2400     LDA HOLD1
1850     BCS SDL             2410     ORA #$40
1860 DL  LDA ICAX1Z          2420     JSR STDDSP
1870     AND #$10            2430 LOOP4 LDA HOLD1
1880     BEQ SDL             2440     JSR STDDSP
1890     LDA #$04            2450     DEX
1900     STA BOTSCR          2460     BNE LOOP4
1910     LDX #$02            2470 CDL LDA SAVMSC+1
1920     LDA FINE            2480     JSR STDDSP
1930     BEQ EDL             2490     LDA SAVMSC
1940     JSR SSDLE           2500     JSR STDDSP
1950 EDL LDA #$02            2510     LDA HOLD1
1960     JSR STDFS           2520     ORA #$40
1970     DEX                 2530     JSR STDDSP
1980     BPL EDL             2540     LDA #$70
1990     LDY RAMTOP          2550     JSR STDDSP
2000     DEY                 2560     LDA #$70
2010     TYA                 2570     JSR STDDSP
2020     JSR STDDSP          2580     LDA ADRESS
2030     LDA #$60            2590     STA DLPTRS
2040     JSR STDDSP          2600     LDA ADRESS+1
2050     LDA #$42            2610     STA DLPTRS+1
2060     JSR STDFS           2620     LDA #$70
2070     CLC                 2630     JSR STDDSP
2080     LDA #$10            2640     LDA ADRESS
2090     ADC MLTTMP          2650     STA MEMTOP
2100     TAY                 2660     LDA ADRESS+1
2110     LDX TDLEC,Y         2670     STA MEMTOP+1
2120     BNE ACMD            2680     LDY #$01
2130 SDL LDY MLTTMP          2690     LDA DLPTRS
2140     LDX TDLEC,Y         2700     STA (SAVADR),Y
2150     LDA DINDEX          2710     INY
2160     BNE ACMD            2720     LDA DLPTRS+1
2170     LDA FINE            2730     STA (SAVADR),Y
2180     BEQ ACMD            2740     LDA DSTAT
2190     JSR SSDLE           2750     BPL NERR
2200     LDA #$22            2760 DERR STA DERRF
2210     STA HOLD1           2770     JSR EDOPN
2220 ACMD LDA HOLD1          2780     LDA DERRF
2230     JSR STDDSP          2790     LDY #$00
2240     DEX                 2800     STY DERRF
2250     BNE ACMD            2810     TAY
2260     LDA DINDEX          2820     RTS
2270     CMP #$08            2830 NERR LDA ICAX1Z
2280     BCC CDL             2840     AND #$20
2290     CMP #$0F            2850     BNE END
2300     BEQ DL2             2860     JSR CLRSCR
2310     CMP #$0C            2870     STA TXTROW
2320     BCS CDL             2880     LDA LMARGN
2330 DL2 LDX #$5D            2890     STA TXTCOL
2340     LDA RAMTOP          2900 END LDA #$22
2350     SEC                 2910     ORA DMACTLS

2920     STA DMACTLS         2930     JMP RETURM
```

Do rejestru ICAX2 należy natomiast wpisać numer trybu graficznego, w którym ma zostać otwarty ekran. Przy otwieraniu edytora wartość ta zostanie zignorowana. Ponieważ informacja o oknie tekstowym i kasowaniu zawartości ekranu są już zawarte w rejestrze ICAX1, to wpisana tu

wartość określa tylko tryb bez wyżej wymienionych wariantów.

Procedura otwarcia ekranu najpierw sprawdza wybrany tryb graficzny i zapisuje go w rejestrze DINDEX (Display INDEX). W trybie zero kasowane są bity 4 i 5 rejestru ICAX1, gdyż w tym trybie nie ma okna tekstowego, a obraz musi być wyczyszczony. Jeśli numer trybu jest większy od $0F, to po umieszczeniu w rejestrze Y kodu błędu $91 (BAD SCREEN MODE - zły tryb graficzny) następuje skok do końcowej części procedury (DERR).

Drugim etapem jest ustalenie wartości zmiennych systemowych w rejestrach RAM. Starsze bajty adresów zestawów znaków są umieszczane w wektorach CHBAS i CHSPTR. Rejestry sterujące wyglądem znaków (CHACT - CHAracter ConTrol) i dostępem do pamięci (DMACTLS - DMA ConTroL Shadow register) otrzymują wartość 2, a status ekranu (DSTAT - Display STATus) wartość 1. Następnie ustawiane są rejestry zezwolenia przerwań IRQEN i NMIEN oraz sprawdzany jest znacznik delikatnego przesuwu obrazu FINE. Zawartość $FF oznacza włączenie delikatnego przesuwu i powoduje ustawienie wektora DLIV (Display List Interrupt Vector) na adres procedury przerwania FSDL (zob. "Mapa pamięci Atari. Podstawowe procedury systemu", str. 50).

Teraz zerowane są rejestry TINDEX (Text window INDEX), SWPFLG (SWap FLaG) i CRSINH (CuRSor INHibition) oraz rejestr pomocniczy ADRESS. Mapa pozycji tabulacji (TABMAP) jest wypełniana wartościami $01, a rejestry koloru otrzymują wartości z tabeli COLTAB.

```
0100 ;COLor TABle
0110 ;
0120     *=  $FB08
0130 ;
0140     .BYTE $28,$CA,$94,$46,$00
```

Na podstawie wartości RAMTOP ustalany jest adres obszaru pamięci dla okna tekstowego (TXTMSC - TeXT Memory SCreen). Ma on zawsze takie samo położenie: starszy bajt mniejszy o jeden od RAMTOP, a młodszy równy $60.

Teraz rozpoczyna się obliczanie wielkości pamięci obrazu w zależności od wybranego trybu. Najpierw numer trybu ANTIC-a jest odczytywany z tabeli TAGRM i umieszczany w pomocniczym rejestrze HOLD1.
0100 ;Table: ANTIC GRaphics Modes 0110 ; 0120 *= $EE4D 0130 ; 0140 .BYTE $02,$06,$07,$08 0150 .BYTE $09,$0A,$0B,$0D 0160 .BYTE $0F,$0F,$0F,$0F 0170 .BYTE $04,$05,$0C,$0E
Następnie wartość RAMTOP jest przepisywana do starszego bajtu pomocniczego rejestru ADRESS, a z tabeli TSMAL odczytywana jest wielkość przemieszczenia pamięci obrazu.

```
0100 ;Table: Screen Memory ALlocation
0110 ;
0120     *=  $EE1D
0130 ;
0140     .BYTE $18,$10,$0A,$0A
0150     .BYTE $10,$1C,$34,$64
0160     .BYTE $C4,$C4,$C4,$C4
```

```
0170     .BYTE $1C,$10,$64,$C4
```

Służy ona jako licznik pętli obliczającej rzeczywisty adres pamięci obrazu. Obliczenie to jest dokonywane przez procedurę DCUSAC, która stanowi część większej procedury obliczeniowej. Zostanie ona w całości opisana teraz, ponieważ jest szeroko wykorzystywana w procedurze SCOPN. Posiada ona pięć punktów początkowych (DBDEC, STDFSC, STDDSP, SGDEC i DCUSAC), co umożliwia uzyskanie wielu różnych funkcji.

```
0100 ADRESS = $64
0110 APPMHI = $0E
0120 DSTAT = $4C
0130 FINE =  $026E
0140 SUBTMP = $029E
0150 ;
0160     *=  $F565
0170 ;
0180 ;DouBle DECreasing
0190 ;
0200     LDA #$02
0210     BNE DCUSAC
0220 ;
0230 ;STore Data for Fine Scroll
0240 ;
0250     LDY FINE
0260     BEQ STDDSP
0270     ORA #$20
0280 ;
0290 ;STore Data for DiSPlay
0300 ;
0310 STDDSP LDY DSTAT
0320     BMI EXIT
0330     LDY #$00
0340     STA (ADRESS),Y
0350 ;
0360 ;SinGle DECreasing
0370 ;
0380     LDA #$01
0390 ;
0400 ;DeCrease USing ACumulator
0410 ;
0420 DCUSAC STA SUBTMP
0430     LDA DSTAT
0440     BMI EXIT
0450     LDA ADRESS
0460     SEC
0470     SBC SUBTMP
0480     STA ADRESS
0490     BCS BPS
0500     DEC ADRESS+1
0510 BPS LDA APPMHI+1
0520     CMP ADRESS+1
0530     BCC EXIT
0540     BNE ERR
0550     LDA APPMHI
0560     CMP ADRESS
0570     BCC EXIT
0580 ERR LDA #$93
0590     STA DSTAT
```

```
0600 EXIT RTS
```

Po wejściu od etykiety STDFSC sprawdzany jest znacznik FINE i, gdy jest różny od zera, w akumulatorze ustawiany jest bit 5. Następnie (od STDDSP) sprawdzany jest status ekranu (DSTAT) i, gdy wskazuje błąd, procedura się kończy. W przeciwnym razie zawartość akumulatora jest umieszczana w miejscu wskazanym wektorem ADRESS i następuje przejście do SGDEC.

Pozostałe części procedury służą do zmniejszenia wartości wektora ADRESS. Rozpoczęcie procedury od DBDEC zmniejsza ADRESS o 2, od SGDEC ADRESS jest zmniejszany o jeden, a od DCUSAC - o aktualną zawartość akumulatora. Podczas operacji zmniejszania sprawdzany jest jeszcze status ekranu (DSTAT), a po niej otrzymany wynik jest porównywany z wektorem APPMHI (APPlication Memory HIgh), który wskazuje najwyższy adres zajęty przez program użytkownika. Jeżeli ADRESS jest mniejszy od APPMHI, to w rejestrze DSTAT umieszczany jest kod błędu $93 (INSUFFICIENT SCREEN MEMORY - niewystarczająca pamięć obrazu).

Ponieważ trzy tryby uzyskiwane są poprzez układ GTIA, to po ich rozpoznaniu odpowiednia wartość jest wpisywana do dwóch najstarszych bitów rejestru GTICTLS (GTIA ConTroL Shadow register). Ponadto tryby $09, $0A, $0B i $0F wymagają jeszcze zwiększenia obszaru pamięci obrazu, więc ponownie wywoływana jest procedura DCUSAC.

Ostatecznie obliczony adres początkowy pamięci obrazu jest przepisywany z rejestru pomocniczego ADRESS do właściwego wektora SAVMSC (SAVe Memory SCreen). Wywołanie procedury SGDEC daje nam teraz w rezultacie adres końcowy programu ANTIC-a. Po pobraniu z tabeli TDLVL wartości określającej podział programu ANTIC-a (zob. rozdział 7) obliczany jest adres jego ostatniej instrukcji i przez wywołanie STDDSP zostaje ona umieszczona na właściwym miejscu.

```
0100 ;Table: Display List VuLnerability
0110 ;
0120     *=  $EE5D
0130 ;
0140     .BYTE $00,$00,$00,$00
0150     .BYTE $00,$00,$00,$01
0160     .BYTE $01,$01,$01,$01
0170     .BYTE $00,$00,$01,$01
```

Kolejna faza procedury służy do określenia parametrów okna tekstowego. Najpierw do rejestru BOTSCR (BOTtom SCreen Rows) jest wpisywana liczba wierszy w trybie 0 ($18). Jeśli wybrany został jeden z trybów 9-11 lub bit 4 w ICAX1 jest skasowany, to etap ten jest pomijany. W przeciwnym razie w BOTSCR umieszczana jest wartość $04, która określa liczbę wierszy w oknie tekstowym. Gdy znacznik FINE zezwala na delikatny przesuw obrazu, to wywoływana jest procedura SSDLE.

```
0100 ;Set Scrolling DL Entry
0110 ;
0120 STDDSP = $F570
0130 ;
```

```
0140     *=  $F5A0
0150 ;
0160     LDA #$02
0170     JSR STDDSP
0180     LDA #$A2
0190     JSR STDDSP
0200     DEX
0210     RTS
```

Wpisuje ona w przedostatnim rozkazie tworzenia linii obrazu wartość, która powoduje wywołanie procedury przerwania FSDL. Pozostałe rozkazy tworzenia linii obrazu są wpisywane przez wywołanie STDFS. Ostatnim rozkazem programu ANTIC-a ustalanym w tej fazie jest rozkaz ładowania licznika obrazu (w AMTIC-u) wartością adresu okna tekstowego.

Pozostała część programu ANTIC-a jest wypełniana rozkazami tworzenia linii obrazu w wybranym trybie. Długość programu jest pobierana z tabeli TDLEC, przy czym pierwsze 16 pozycji w tej tabeli dotyczy trybów bez okna tekstowego, a reszta - trybów z oknem. Przy wpisywaniu programu jest także uwzględniany delikatny przesuw obrazu w trybie 0.

```
0100 ;Table: Display List Entry Counts
0110 ;
0120     *=  $EE2D
0130 ;
0140     .BYTE $17,$17,$0B,$17
0150     .BYTE $2F,$2F,$5F,$5F
0160     .BYTE $61,$61,$61,$61
0170     .BYTE $17,$0B,$BF,$61
0180     .BYTE $13,$13,$09,$13
0190     .BYTE $27,$27,$4F,$4F
0200     .BYTE $41,$41,$41,$41
0210     .BYTE $13,$09,$9F,$41
```

Zapisywanie programu ANTIC-a kończy się po wpisaniu rozkazu ładującego licznik obrazu i rozkazów tworzących puste linie w górnej części ekranu. Teraz rejestr pomocniczy ADRESS zawiera adres ostatniej wolnej komórki pamięci RAM. Adres tej jest przepisywany do rejestru MEMTOP w celu zabezpieczenia obrazu przed zniszczeniem przez program użytkownika. Adres programu ANTIC-a jest natomiast umieszczany w rejestrze DLPTRS (Display List PoinTeR Shadow register) oraz po rozkazie skoku kończącym ten program (adres tego rozkazu jest przechowywany w rejestrze SAVADR).

W tym momencie ekran jest już otwarty i pozostaje tylko sprawdzenie, czy podczas SCOPN nie wystąpił żaden błąd. Informację o tym zawiera rejestr DSTAT. W przypadku wykrycia błędu wykonywana jest procedura EDOPN, która otwiera edytor, aby umożliwić systemowi zasygnalizowanie błędu.

Jeśli otwarcie przebiegło bezbłędnie, to sprawdzany jest bit 5 rejestru ICAX1. Gdy jest on skasowany, to zawartość pamięci obrazu jest kasowana przez wywołanie procedury CLRSCR (zob. str. 86), a kursor jest ustawiany w lewym, górnym rogu ekranu (z uwzględnieniem marginesu).

Ostatnią czynnością jest ustawienie w rejestrze DMACTLS bitów 1 (normalna szerokość obrazu) i 5 (bezpośredni dostęp do pamięci dla programu ANTIC-a). Procedura SCOPN nie kończy się rozkazem RTS, lecz skokiem do RETURM.

Procedura RETURM zawiera w sobie (od etykiety KBOPN) procedurę otwarcia i zamknięcia klawiatury oraz procedurę statusu dla klawiatury, ekranu i edytora.

```
0100 ;RETURn from Monitor
0110 ;
0120 ATACHR = $02FB
0130 CRSINH = $02F0
0140 DINDEX = $57
0150 DISPLY = $F1E9
0160 DSTAT = $4C
0170 GETPLT = $F18F
0180 OLDCHR = $5D
0190 ;
0200     *=  $F20B
0210 ;
0220     JSR GETPLT
0230     STA OLDCHR
0240     LDX DINDEX
0250     BNE KBOPN
0260     LDX CRSINH
0270     BNE KBOPN
0280     EOR #$80
0290     JSR DISPLY
0300 ;
0310 ;KeyBoard OPeN
0320 ;
0330 KBOPN LDY DSTAT
0340     JMP SKIP
0350 ;
0360     *=  $F226
0370 ;
0380 SKIP LDA #$01
0390     STA DSTAT
0400     LDA ATACHR
0410 ;
0420 ;EDitor SPecial
0430 ;
0440     RTS
```

Znajdujący się w środku RETURM skok służy do ominięcia instrukcji skoku do procedury testującej (TESTROM). Jest to pozostałość systemu Atari 400/800, który korzystał z tego adresu.

RETURM najpierw odczytuje znak znajdujący się na pozycji kursora poprzez wywołanie procedury GETPLT. Jeżeli obraz jest w trybie 0 i kursor jest widoczny, to zamienia najstarszy bit tego znaku, co zmienia znak z normalnego na negatywowy lub odwrotnie. Następnie znak ten poprzez procedurę DISPLY jest ponownie umieszczany na ekranie.

Dalszy przebieg procedury (od etykiety KBOPN) był już opisywany. Przypomnijmy, że w rejestrze Y umieszczany jest status ekranu, a w akumulatorze ostatnio użyty znak ASCII i status

(DSTAT) otrzymuje wartość 1. KBOPN stanowi także procedurę odczytu statusu ekranu i edytora, zaś na jej ostatnią instrukcję (oznaczoną EDSP) wskazują wektory procedur specjalnych klawiatury i edytora oraz zapisu na klawiaturę.

3.2.2. Procedura zamknięcia ekranu

Procedura zamykająca jest także wspólna dla ekranu i edytora. Służy ona właściwie jedynie do wyłączenia delikatnego przesuwu obrazu. Na początku procedury jest bowiem sprawdzany znacznik FINE i, gdy wskazuje on wyłączenie delikatnego przesuwu, następuje skok do procedury KBOPN.

```
0100 ;SCRolling FINe
0110 ;
0120 DLIV =  $0200
0130 EDOPN = $EF94
0140 FINE =  $026E
0150 KBOPN = $F21E
0160 NMIEN = $D40E
0170 RTI =   $C0CE
0180 ;
0190     *=  $F22E
0200 ;
0210     BIT FINE
0220     BPL KBOPN
0230     LDA #$40
0240     STA NMIEN
0250     LDA #$00
0260     STA FINE
0270     LDA # <RTI
0280     STA DLIV
0290     LDA # >RTI
0300     STA DLIV+1
0310     JMP EDOPN
```

Jeżeli delikatny przesuw jest włączony, to najpierw blokowane są w rejestrze NMIEN przerwania wywoływane przez program ANTIC-a. Następnie znacznik FINE jest zerowany, a wektor DLIV otrzymuje wartość wskazującą na instrukcję RTI (powrót z przerwania). Ponieważ wyjście na ekran musi być stale czynne, to procedura kończy się bezpośrednio skokiem do EDOPN.

3.2.3. Zapis na ekranie

znak przeznaczony do zapisania na ekranie jest umieszczany w akumulatorze i poprzez GOHAND wywoływana jest procedura OUTCH. Na jej początku znak jest przepisywany z akumulatora do rejestru ATACHR, a następnie dokonywane jest sprawdzenie jego kodu.

```
0100 ;OUTput CHaracter
0110 ;
0120 ATACHR = $02FB
0130 CLRSCR = $F420
0140 EOLSUB = $F60E
0150 OUTPLT = $F1CA
0160 RANGE = $F6CA
0170 RETURM = $F20B
0180 RTWSCR = $F661
```

```
0190 ;
0200     *=  $F1A4
0210 ;
0220     STA ATACHR
0230     CMP #$7D
0240     BNE PT2
0250     JSR CLRSCR
0260     JMP RETURM
0270 PT2 JSR RANGE
0280 ;TeST for RETurn
0290 TSTRET LDA ATACHR
0300     CMP $9B
0310     BNE PT3
0320     JSR RTWSCR
0330     JMP RETURM
0340 PT3 JSR OUTPLT
0350     JSR EOLSUB
0360     JMP RETURM
```

Jeżeli znak do zapisu oznacza czyszczenie ekranu ($7D), to po wywołaniu procedury CLRSCR, która przeprowadza tę operację, wykonywany jest skok do RETURM. W innym przypadku wywoływana jest procedura RANGE, której zadaniem jest sprawdzenie, czy kursor mieści się na ekranie.

Jeżeli aktualna pozycja kursora mieści się w dozwolonym zakresie, to kod znaku jest porównywany z $9B (RETURN). Pozytywny wynik powoduje wywołanie procedury RTWSCR, a następnie przejście do RETURM. W przeciwnym razie wywoływane są procedury OUTPLT i SUBEND umieszczające znak na ekranie, a dopiero po tym następuje skok do RETURM.

RANGE jest punktem wejścia do nieco większej procedury ERANGE, która służy do obsługi ekranu i edytora. Jeżeli ekran jest otwarty w trybie 0 lub posiada okno tekstowe, to obie procedury są identyczne. W przeciwnym razie najpierw otwierany jest edytor (przez EDOPN) i dopiero dalszy przebieg procedur (od RANGE) jest wspólny. Przy rozpoczęciu od RANGE pierwsza faza kontroli jest pomijana.

```
0100 ;Editor RANGE
0110 ;
0120 BOTSCR = $02BF
0130 COLCRS = $55
0140 CRSHOM = $F440
0150 DINDEX = $57
0160 DSTAT = $4C
0170 EDOPN = $EF94
0180 IRQSTAT = $11
0190 KBOPN = $F21E
0200 RMARGN = $53
0210 ROWCRS = $54
0220 SWAP =  $F962
0230 SWPFLG = $7B
0240 TMCCN = $EE7D
0250 TMRCN = $EE8D
0260 ;
0270     *=  $F6BC
```

```
0280 ;
0290     LDA BOTSCR
0300     CMP #$04
0310     BEQ RANGE
0320     LDA DINDEX
0330     BEQ RANGE
0340     JSR EDOPN
0350 RANGE LDA #$27
0360     CMP RMARGN
0370     BCS ROW
0380     STA RMARGN
0390 ROW LDX DINDEX
0400     LDA TMRCN,X
0410     CMP ROWCRS
0420     BCC ERR
0430     BEQ ERR
0440     CPX #$08
0450     BNE HCC
0460     LDA COLCRS+1
0470     BEQ SUC
0480     CMP #$01
0490     BNE ERR
0500     BEQ COL
0510 HCC LDA COLCRS+1
0520     BNE ERR
0530 COL LDA TMCCN,X
0540     CMP COLCRS
0550     BCC ERR
0560     BEQ ERR
0570 SUC LDA #$01
0580     STA DSTAT
0590     LDA #$80
0600     LDX IRQSTAT
0610     STA IRQSTAT
0620     BEQ SWP
0630     RTS
0640 ERR JSR CRSHOM
0650     LDA #$8D
0660 SWP STA DSTAT
0670     PLA
0680     PLA
0690     LDA SWPFLG
0700     BPL KOP
0710     JMP SWAP
0720 KOP JMP KBOPN
```

Najpierw sprawdzany jest prawy margines ekranu. Jeżeli przekracza on $27, to jest sprowadzany do tej wartości. Następnie według zawartości DINDEX jest pobierana z tabeli TMRCN dopuszczalna liczba wierszy w danym trybie graficznym. Gdy aktualna pionowa pozycja kursora (w rejestrze ROWCRS - ROW CuRSor) przekracza tą liczbę, sygnalizowany jest błąd.

```
0100 ;Table: Mode Row CouNts
0110 ;
0120     *=  $EE8D
0130 ;
0140     .BYTE $18,$18,$0C,$18
0150     .BYTE $30,$30,$60,$60
0160     .BYTE $C0,$C0,$C0,$C0
```

```
0170     .BYTE $18,$0C,$C0,$C0
```

Przed sprawdzeniem poziomego położenia kursora kontrolowany jest tryb graficzny. Ponieważ tryb 8 ma rozdzielczość poziomą 320 punktów, to w tym przypadku wystarczy sprawdzić, czy starszy bajt rejestru COLCRS (COLumn CuRSor) jest równy 0 lub 1. Inna wartość powoduje błąd. Błąd wystąpi także wtedy, gdy w pozostałych trybach starszy bajt COLCRS jest różny od zera. Ostateczna kontrola następuje po pobraniu z tabeli TMCCN dopuszczalnej liczby kolumn w danym trybie i porównaniu jej z młodszym bajtem COLCRS.

```
0100 ;Table: Mode Column CouNts
0110 ;
0120     *=  $EE7D
0130 ;
0140     .BYTE $28,$14,$14,$28
0150     .BYTE $50,$50,$A0,$A0
0160     .BYTE $40,$50,$50,$50
0170     .BYTE $28,$28,$A0,$A0
```

Wykrycie błędu powoduje umieszczenie kursora w lewym, górnym rogu ekranu (przez CRSHOM), wpisanie do rejestru DSTAT kodu $8D (CURSOR OVERRANGE - kursor poza zakresem) oraz zdjęcie ze stosu adresu powrotnego do procedury bezpośrednio wywołującej RANGE (lub ERANGE). Przy poprawnym przebiegu procedury ta faza jest omijana, a rejestr DSTAT otrzymuje wartość 1.

Sposób zakończenia procedury zależy od wartości znacznika SWPFLG (SWaP FLaG). Zero oznacza, że kursor znajduje się w oknie tekstowym i procedura kończy się skokiem do KBOPN. Odczytanie wartości $FF powoduje natomiast skok do procedury SWAP.
0100 ;SWAP cursor 0110 ; 0120 BOTSCR = $02BF 0130 KBOPN = $F21E 0140 ROWCRS = $54 0150 SWPFLG = $7B 0160 TXTROW = $0290 0170 ; 0180 *= $F962 0190 ; 0200 LDA BOTSCR 0210 CMP #$18 0220 BEQ EXIT 0230 LDX #$0B 0240 NEXT LDA ROWCRS,X 0250 PHA 0260 LDA TXTROW,X 0270 STA ROWCRS,X 0280 PLA 0290 STA TXTROW,X 0300 DEX 0310 BPL NEXT 0320 LDA SWPFLG 0330 EOR #$FF 0340 STA SWPFLG 0350 EXIT JMP KBOPN
Procedura SWAP przenosi kursor pomiędzy dwoma obszarami: ekranu i okna tekstowego. Przedtem jednak sprawdzany jest rejestr BOTSCR. Zapisana w nim wartość $18 oznacza, że ekran nie jest podzielony i przerywa procedurę.

Każda inna wartość powoduje wymianę zawartości między dwoma obszarami po 12 bajtów rozpoczynającymi się od ROWCRS i TXTROW. Następnie znacznik SWPFLG jest zmieniany z $FF na zero lub odwrotnie i procedura jest opuszczana skokiem do KBOPN.

Procedura RTWSCR wywoływana po rozpoznaniu znaku RETURN ma za zadanie umieszczenie kursora w następnej linii obrazu. Jeśli spowoduje to przekroczenie dopuszczalnego zakresu pozycji kursora, wykonywane jest przesunięcie zawartości ekranu w górę o jeden wiersz logiczny.

```
0100 BUFSTR = $6C
0110 COLCRS = $55
```

```
0120 COMLOG = $F88E
0130 DINDEX = $57
0140 DOSCR = $F7F7
0150 HOLD3 = $029D
0160 INSDAT = $7D
0170 LOGMAP = $02B2
0180 ROWCRS = $54
0190 SCLED = $F997
0200 SCRFLG = $02BB
0210 SWPFLG = $7B
0220 TMRCN = $EE8D
0230 ;
0240     *=  $F661
0250 ;
0260 ;ReTurn With SCRoll
0270 ;
0280 RTWSCR LDA #$9B
0290     STA INSDAT
0300 ;
0310 ;RETURN routine
0320 ;
0330 RETURN JSR SCLED
0340     LDA #$00
0350     STA COLCRS+1
0360     INC ROWCRS
0370     LDX DINDEX
0380     LDY #$18
0390     BIT SWPFLG
0400     BPL ROW
0410     LDY #$04
0420     TYA
0430     BNE CRS
0440 ROW LDA TMRCN,X
0450 CRS CMP ROWCRS
0460     BNE EXIT
0470     STY HOLD3
0480     TXA
0490     BNE EXIT
0500     LDA INSDAT
0510     BEQ EXIT
0520     CMP #$9B
0530     BEQ SCR
0540     CLC
0550 SCR JSR DOSCR
0560     INC SCRFLG
0570     DEC BUFSTR
0580     BPL BPS
0590     INC BUFSTR
0600 BPS DEC HOLD3
0610     LDA LOGMAP
0620     SEC
0630     BPL SCR
0640     LDA HOLD3
0650     STA ROWCRS
0660 EXIT JMP COMLOG
```

Na początku kod znaku RETURN ($9B) jest umieszczany w pomocniczym rejestrze INSDAT. Następnie kursor jest przesuwany na lewą krawędź obrazu przy pomocy procedury SCLED. Dla

trybu 0 oraz dla okna tekstowego oznacza to ustawienie kursora na pozycji określonej przez zawartość rejestru LMARGN (Left MARGiN), w pozostałych przypadkach na pozycji 0.

```
0100 ;Set Cursor at Left EDge
0110 ;
0120 COLCRS = $55
0130 DINDEX = $57
0140 LMARGN = $52
0150 SWPFLG = $7B
0160 ;
0170     *=  $F997
0180 ;
0190     LDA #$00
0200     LDX SWPFLG
0210     BNE MRG
0220     LDX DINDEX
0230     BNE STR
0240 MRG LDA LMARGN
0250 STR STA COLCRS
0260     RTS
```

Teraz numer wiersza zawierającego kursor jest porównywany z dopuszczalną ich liczbą pobraną z tabeli TMRCN (dla okna tekstowego przyjmowana jest wartość 4). Jeśli liczby te są różne, tryb jest inny niż 0 lub w INSDAT jest wartość zero, to procedura się kończy. W przeciwnym razie wywoływana jest procedura DOSCR, która przesuwa obraz. Wywołanie to następuje w pętli, która jest przerywana po usunięciu z górnej części ekranu całego wiersza logicznego. Dopiero wtedy do rejestru ROWCRS wpisywana jest pionowa pozycja kursora i wykonywany jest skok do procedury COMLOG.

Zadaniem procedury DOSCR jest przemieszczenie danych w pamięci obrazu w ten sposób, aby zawartość ekranu przesunęła się o jedną linię w górę. Choć czynność ta jest prosta, to procedura ma znaczną długość, gdyż konieczna jest duża ilość obliczeń.

```
0100 ;DO SCRolling
0110 ;
0120 ADRESS = $64
0130 BITROL = $F732
0140 BOTSCR = $02BF
0150 COLCRS = $55
0160 COUNTR = $7E
0170 FINE =  $026E
0180 HOLD1 = $51
0190 LGET2 = $F75A
0200 LOGCOL = $63
0210 PUTMSC = $F9A6
0220 RAMTOP = $6A
0230 ROWCRS = $54
0240 VCOUNT = $D40B
0250 VSFLAG = $026C
0260 ;
0270     *=  $F7F7
0280 ;
0290     JSR BITROL
0300     LDA FINE
0310     BEQ NSC
```

```
0320 WT1 LDA VSFLAG
0330     BNE WT1
0340     LDA #$08
0350     STA VSFLAG
0360 WT2 LDA VSFLAG
0370     CMP #$01
0380     BNE WT2
0390 WT3 LDA VCOUNT
0400     CMP #$40
0410     BCS WT3
0420     LDX #$0D
0430     LDA BOTSCR
0440     CMP #$04
0450     BNE WT4
0460     LDX #$70
0470 WT4 CPX VCOUNT
0480     BCS WT4
0490 NSC JSR PUTMSC
0500 ;
0510 ;COMpute ADdRess
0520 ;
0530 COMADR LDA ADRESS
0540     LDX ADRESS+1
0550 LOOP1 INX
0560     CPX RAMTOP
0570     BEQ NLN
0580     SEC
0590     SBC #$10
0600     JMP LOOP1
0610 NLN ADC #$27
0620     BNE SCN
0630     LDX ADRESS+1
0640     INX
0650     CPX RAMTOP
0660     BEQ STCN
0670     CLC
0680     ADC #$10
0690 SCN TAY
0700     STA COUNTR
0710     SEC
0720     LDA ADRESS
0730     SBC COUNTR
0740     STA ADRESS
0750     BCS IAD
0760     DEC ADRESS+1
0770 IAD LDA ADRESS
0780     CLC
0790     ADC #$28
0800     STA COUNTR
0810     LDA ADRESS+1
0820     ADC #$00
0830     STA COUNTR+1
0840 NXT1 LDA (COUNTR),Y
0850     STA (ADRESS),Y
0860     INY
0870     BNE NXT1
0880     LDY #$10
0890     LDA ADRESS
0900     CMP #$D8
0910     BEQ STCN
```

```
0920     CLC
0930     ADC #$F0
0940     STA ADRESS
0950     BCC IAD
0960     INC ADRESS+1
0970     BNE IAD
0980 STCN LDX RAMTOP
0990     DEX
1000     STX COUNTR+1
1010     LDX #$D8
1020     STX COUNTR
1030     LDA #$00
1040     LDY #$27
1050 NXT2 STA (COUNTR),Y
1060     DEY
1070     BPL NXT2
1080 ;
1090 ;COMpute LOGical line
1100 ;
1110 COMLOG LDA #$00
1120     STA LOGCOL
1130     LDA ROWCRS
1140     STA HOLD1
1150 LOOP2 LDA HOLD1
1160     JSR LGET2
1170     BCS EXIT
1180     LDA LOGCOL
1190     CLC
1200     ADC #$28
1210     STA LOGCOL
1220     DEC HOLD1
1230     JMP LOOP2
1240 EXIT CLC
1250     LDA LOGCOL
1260     ADC COLCRS
1270     STA LOGCOL
1280     RTS
```

Najpierw wywoływana jest procedura BITROL, która przesuwa o jeden bit w lewo trzy bajty rejestru LOGMAP (LOGical line MAP) zawierających mapę wierszy logicznych ekranu. Gdy włączony jest delikatny przesuw ekranu (wartość $FF w rejestrze FINE), to jego realizację umożliwiają teraz cztery petle opóźniające. Długość ostatniej z nich zależy od tego, czy przesuwany jest cały obraz, czy tylko okno tekstowe.

```
0100 ;BIT ROtate Left
0110 ;
0120 LOGMAP = $02B2
0130 ;
0140     *=  $F732
0150 ;
0160     ROL LOGMAP+2
0170     ROL LOGMAP+1
0180     ROL LOGMAP
0190     RTS
```

Następnie wywoływana jest krótka procedura PUTMSC, która przepisuje adres pamięci obrazu z SAVMSC do pomocniczego rejestru ADRESS.

```
0100 ;PUT Memory SCreen
0110 ;
0120 ADRESS = $64
0130 SAVMSC = $58
0140 ;
0150     *=  $F9A6
0160 ;
0170     LDA SAVMSC
0180     STA ADRESS
0190     LDA SAVMSC+1
0200     STA ADRESS+1
0210     RTS
```

Teraz rozpoczyna się najdłuższa część procedury oznaczona etykietą COMADR. Kolejno obliczane sa rzeczywiste adresy linii obrazu w pamięci i każda linia jest przepisywana (w pętli NXT1) o 40 bajtów niżej (w pamięci - na ekranie wyżej). Na końcu ostatnia linia jest wypełniana zerami (pętla NXT2).

Ostatnia faza rozpoczynająca się od COMLOG służy do obliczenia wiersza logicznego ekranu. Jest ona także wywoływana osobno jako oddzielna procedura. Na początku adres kursora w wierszu logicznym LOGCOL (LOGical line COLumn) jest ustawiany na zero, a pionowa pozycja kursora na ekranie jest przepisywana z ROWCRS do tymczasowego rejestru HOLD1. Następnie z wartością odczytaną z HOLD1 wywoływana jest procedura LOGGET, lecz nie od początku, a od etykiety LGET2.

```
0100 ;LOGical line GET
0110 ;
0120 BITCON = $F723
0130 BITMSK = $6E
0140 ROWCRS = $54
0150 TABMAP = $02A3
0160 ;
0170     *=  $F758
0180 ;
0190     LDA ROWCRS
0200 LGET2 CLC
0210 LGET3 ADC #$78
0220 ;
0230 ;BIT GET
0240 ;
0250    JSR BITCON
0260    CLC
0270    LDA TABMAP,X
0280    AND BITMSK
0290    BEQ EXIT
0300    SEC
0310 EXIT RTS
```

Wywołanie to jest powtarzane tak długo, dopóki po zakończeniu procedury LOGGET bit Carry nie będzie ustawiony, przy czym przed każdym kolejnym wywołaniem zawartość LOGCOL jest zwiększana o 40, a HOLD1 zmniejszana o jeden. Po tym jeszcze do adresu kursora w linii logicznej LOGCOL dodawana jest jego pozycja pozioma COLCRS i procedura się kończy.

Procedura LOGGET według pionowego położenia kursora (lub według zawartości akumulatora - zależnie od punktu początkowego) odczytuje poprzez BITCON wzór bitowy wiersza logicznego i porównuje go z mapą tabulacji TABMAP. Jeżeli ich wartości są różne, to przed opuszczeniem procedury bit Carry jest ustawiany, w przeciwnym przypadku pozostaje skasowany.

```
0100 ;BIT CONversion
0110 ;
0120 MSKTAB = $EEB4
0130 BITMSK = $6E
0140 ;
0150     *=  $F723
0160 ;
0170     PHA
0180     AND #$07
0190     TAX
0200     LDA MSKTAB,X
0210     STA BITMSK
0220     PLA
0230     LSR A
0240     LSR A
0250     LSR A
0260     TAX
0270     RTS
```

Procedura BITCON oddziela najpierw cztery młodsze bity akumulatora i według nich pobiera wartość maski bitowej z tabeli MSKTAB. Wartość ta jest zapisywana do rejestru BITMSK (BIT MaSK). Następnie poprzednia zawartość akumulatora jest trzykrotnie przesuwana w prawo (dzielona przez osiem) i przepisywana do rejestru X.

```
0100 ;MaSK TABle
0110 ;
0120     *=  $EEB4
0130 ;
0140     .BYTE $80,$40,$20,$10
0150     .BYTE $08,$04,$02,$01
```

W celu umieszczenia znaku na ekranie z OUTCHR jest wywoływana procedura OUTPLT. Jeśli znacznik SSFLAG (Start/Stop FLAG) wskazuje zatrzymanie wyprowadzania danych na ekran, to procesor oczekuje na zezwolenie (np. przez naciśnięcie CONTROL-1). Następnie aktualna pozycja kursora na ekranie jest przepisywana z ROWCRS i COLCRS do OLDROW i OLDCOL oraz odczytywana jest zawartość rejestru ATACHR. Uzyskany znak w kodzie ATASCII jest zamieniany przy pomocy tabeli AINCC na znak w kodzie wewnętrznym komputera. Wymaga to odczytania bitów 5 i 6 znaku oraz dodania według nich wartości z tabeli.

```
0100 ;OUTput PLoTted
0110 ;
0120 AINCC = $FB49
0130 ADRESS = $64
0140 ATACHR = $02FB
0150 CHAR =  $02FA
0160 CONVRT = $F5AC
0170 DMASK = $02A0
0180 OLDROW = $5A
0190 ROWCRS = $54
```

```
0200 SHFAMT = $6F
0210 SSFLAG = $02FF
0220 TEMP = $50
0230 ;
0240     *=  $F1CA
0250 ;
0260 OUTPLT LDA SSFLAG
0270     BNE OUTPLT
0280     LDX #$02
0290 NXT LDA ROWCRS,X
0300     STA OLDROW,X
0310     DEX
0320     BPL NXT
0330     LDA ATACHR
0340     TAY
0350     ROL A
0360     ROL A
0370     ROL A
0380     ROL A
0390     AND #$03
0400     TAX
0410     TYA
0420     AND #$9F
0430     ORA AINCC,X
0440 DISPLY STA CHAR
0450     JSR CONVRT
0460     LDA CHAR
0470 JUSTP LSR SHFAMT
0480     BCS MSK
0490     ASL A
0500     JMP JUSTP
0510 MSK AND DMASK
0520     STA TEMP
0530     LDA DMASK
0540     EOR #$FF
0550     AND (ADRESS),Y
0560     ORA TEMP
0570     STA (ADRESS),Y
0580     RTS

0100 ;Atascii to INternal
0110 ;Conversion Constans
0120 ;
0130     *=  $FB49
0140 ;
0150     .BYTE $40,$00,$20,$60
```

Obliczony w ten sposób znak jest zapisywany do CHAR i przez wywołanie procedury CONVRT współrzędne kursora są zamieniane na rzeczywisty adres w pamięci obrazu. Następnie zawartość rejestru SHFAMT (SHiFt AMounT) jest przesuwana w prawo, aż do ustawienia bitu Carry. Po skasowaniu zbędnych bitów przy pomocy maski DMASK (Display MASK) uzyskana wartość jest dodawana do zawartości pamięci obrazu.

Wspomniana już procedura CONVRT odczytuje współrzędne kursora z rejestrów ROWCRS oraz COLCRS i przelicza je na adres kursora w pamięci obrazu. W tym celu najpierw numer wiersza z ROWCRS jest mnożony przez pięć i zapisywany do ADRESS.

```
0100 ;CONVeRT position to address
0110 ;
0120 ADRESS = $64
0130 COLCRS = $55
0140 DINDEX = $57
0150 DMASK = $02A0
0160 MLTTMP = $66
0170 OLDADR = $5E
0180 ROWCRS = $54
0190 SAVADR = $68
0200 SAVMSC = $58
0210 SHFAMT = $6F
0220 TBMSK = $FB04
0230 TDMSK = $EEAD
0240 TLSHC = $EE6D
0250 TRSHC = $EE9D
0260 ;
0270     *=  $F5AC
0280 ;
0290     LDX #$01
0300     STX MLTTMP
0310     DEX
0320     STX ADRESS+1
0330     LDA ROWCRS
0340     ASL A
0350     ROL ADRESS+1
0360     ASL A
0370     ROL ADRESS+1
0380     ADC ROWCRS
0390     STA ADRESS
0400     BCC LSH
0410     INC ADRESS+1
0420 LSH LDY DINDEX
0430     LDX TLSHC,Y
0440 NXT ASL ADRESS
0450     ROL ADRESS+1
0460     DEX
0470     BNE NXT
0480     LDA COLCRS+1
0490     LSR A
0500     LDA COLCRS
0510     LDX TRSHC,Y
0520     BEQ BPS
0530 LOP ROR A
0540     ASL MLTTMP
0550     DEX
0560     BNE LOP
0570 BPS ADC ADRESS
0580     BCC NHI
0590     INC ADRESS+1
0600 NHI CLC
0610     ADC SAVMSC
0620     STA ADRESS
0630     STA OLDADR
0640     LDA ADRESS+1
0650     ADC SAVMSC+1
0660     STA ADRESS+1
0670     STA OLDADR+1
0680     LDX TRSHC,Y
0690     LDA TBMSK,X
```

```
0700     AND COLCRS
0710     ADC MLTTMP
0720     TAY
0730     LDA TDMSK-1,Y
0740     STA DMASK
0750     STA SHFAMT
0760     LDY #$00
0770     RTS
```

Zawartość ADRESS jest z kolei mnożona przez 2 do potęgi pobranej z tabeli TLSHC. Wartość z tabeli jest wybierana według numeru trybu przechowywanego w rejestrze DINDEX. Teraz z numeru kolumny najstarsze bity są przepisywane do pomocniczego rejestru MLTTMP. Liczba wydzielonych bitów jest określana na podstawie tabeli TRSHC.

```
0100 ;Table Left SHift Columns
0110 ;
0120     *=  $EE6D
0130 ;
0140     .BYTE $03,$02,$02,$01
0150     .BYTE $01,$02,$02,$03
0160     .BYTE $03,$03,$03,$03
0170     .BYTE $03,$03,$02,$03

0100 ;Table Right SHift Columns
0110 ;
0120     *=  $EE9D
0130 ;
0140     .BYTE $00,$00,$00,$02
0150     .BYTE $03,$02,$03,$02
0160     .BYTE $03,$01,$01,$01
0170     .BYTE $00,$00,$03,$02
```

Obliczone w ten sposób miejsce w kolumnie jest dodawane do ADRESS i w rezultacie otrzymujemy adres punktu liczony od początku pamięci obrazu. Adres bezwzględny uzyskiwany jest po dodaniu do ADRESS wektora SAVMSC. Wynik ten jest zapisywany jednocześnie w rejestrze OLDADR. Ponieważ jeden bajt pamięci obrazu może odpowiadać kilku punktom na ekranie, to według TRSHC jest odczytywany z tabeli TBMSK numer maski bitowej.

```
0100 ;Table Bit MaSK
0110 ;
0120     *=  $FB04
0130 ;
0140     .BYTE $00,$01,$03,$07
```

Po poprawieniu według zawartości COLCRS i MLTTMP numer ten służy do odczytania z tabeli TDMSK odpowiedniej maski bitowej. Pobrana maska jest przed zakończeniem procedury zapisywana do rejestrów DMASK i SHFAMT.

```
0100 ;Table Display MaSK
0110 ;
0120     *=  $EEAD
0130 ;
0140     .BYTE $FF,$F0,$0F,$C0

0150     .BYTE $30,$0C,$03
```

```
0160 MSKTAB
0170     .BYTE $80,$40,$20,$10
0180     .BYTE $08,$04,$02,$01
```

Po umieszczeniu znaku na ekranie wywoływana jest jeszcze z OUTCH procedura EOLSUB. Ma ona trzy punkty początkowe: EOLSUB, INCRSB i SCRIBT. Różnią się one tylko wprowadzonym znakiem. Przy rozpoczęciu od EOLSUB jest to znak RETURN ($9B), od INCRSB - znak $00, a od SCRIBT znak umieszczony uprzednio w rejestrze INSDAT.

```
0100 BITGET = $F75D
0110 COLCRS = $55
0120 COMLOG = $F88E
0130 DINDEX = $57
0140 INSDAT = $7D
0150 INSLN2 = $F50D
0160 LOGCOL = $63
0170 RETURN = $F665
0180 RMARGN = $53
0190 ROWCRS = $54
0200 RTWSCR = $F661
0210 TMCCN = $EE7D
0220 ;
0230     *=  $F609
0240 ;
0250 END RTS
0260 ;
0270 ;INcrease CuRsor SuBroutine
0280 ;
0290 INCRSB LDA #$00
0300     BEQ SKIP
0310 ;
0320 ;End Of Line SUBroutine
0330 ;
0340 EOLSUB LDA #$9B
0350 SKIP STA INSDAT
0360 ;
0370 ;SCRoll If BoTtom
0380 ;
0390 SCRIBT INC LOGCOL
0400     INC COLCRS
0410     BNE BPS
0420     INC COLCRS+1
0430 BPS LDA COLCRS
0440     LDX DINDEX
0450     CMP TMCCN,X
0460     BEQ GRC
0470     CPX #$00
0480     BNE END
0490     CMP RMARGN
0500     BEQ END
0510     BCC END
0520 GRC CPX #$08
0530     BNE LGC
0540     LDA COLCRS+1
0550     BEQ END
0560 LGC LDA DINDEX
```

```
0570     BNE RETURN
0580     LDA LOGCOL
0590     CMP #$51
0600     BCC NLN
0610     LDA INSDAT
0620     BEQ RETURN
0630     JSR RTWSCR
0640     JMP $F6AB  ;do JMP COMLOG
0650 NLN JSR RETURN
0660     LDA ROWCRS
0670     CLC
0680     ADC #$78
0690     JSR BITGET
0700     BCC EXIT
0710     LDA INSDAT
0720     BEQ EXIT
0730     CLC
0740     JSR INSLN2
0750 EXIT JMP COMLOG
```

Najpierw zwiększana jest pozioma pozycja kursora i jego adres w wierszu logicznym. Gdy kursor nie osiągnął jeszcze końca linii, procedura kończy się rozkazem RTS; w przeciwnym razie, jeśli nie jest to tryb 0, przez skok do RETURN (wewnątrz RTWSCR). W trybie 0, jeśli kursor znajduje się w trzeciej linii wiersza logicznego i w INSDAT zapisany jest znak zero, także następuje skok do RETURN. Każdy inny znak powoduje wywołanie procedury RTWSCR i potem skok do COMLOG.

Jeżeli kursor nie znajduje się jeszcze w ostatniej linii wiersza logicznego, to poprzez wywołanie BITGET sprawdza się, czy jest to początek następnego wiersza logicznego. Jeżeli nie lub gdy znak w INSDAT jest zero, to następuje skok do COMLOG. Gdy kursor znalazł się w nowym wierszu logicznym, przez wywołanie procedury INSLIN (ale od etykiety INSLN2) do aktualnego wiersza dodawana jest następna linia fizyczna.

3.2.4. Odczyt z ekranu

Żądanie odczytu z ekranu powoduje wywołanie przez CIO procedury GETCH. Poprzez serię wywołań innych procedur kolejno jest sprawdzany zakres pozycji kursora na ekranie (RANGE), punkt jest odczytywany z ekranu (GETPLT) i zamieniany na kod ASCII (INATAC). Po przesunięciu kursora na następną pozycję ekranu przez INCRSB procedura kończy się skokiem do KBOPN.

```
0100 ;GET CHaracter
0110 ;
0120 GETPLT = $F18F
0130 INATAC = $F76A
0140 INCRSB = $F60A
0150 KBOPN = $F21E
0160 RANGE = $F6CA
0170 ;
0180     *=  $F180
0190 ;
0200     JSR RANGE
```

```
0210     JSR GETPLT
0220     JSR INATAC
0230     JSR INCRSB
0240     JMP KBOPN
```

Pobierająca znak z ekranu procedura GETPLT najpierw wywołuje CONVRT w celu obliczenia adresu kursora w pamięci. Odczytany bajt jest następnie zamieniany na znak przy pomocy maski bitowej DMASK i rejestru przesunięć SHFAMT. Uzyskany w ten sposób znak jest zapisywany do rejestru CHAR.

```
0100 ;GET PLoTted
0110 ;
0120 ADRESS = $64
0130 CHAR =  $02FA
0140 CONVRT = $F5AC
0150 DMASK = $02A0
0160 SHFAMT = $6F
0170 ;
0180     *=  $F18F
0190 ;
0200     JSR CONVRT
0210     LDA (ADRESS),Y
0220     AND DMASK
0230 NEXT LSR SHFAMT
0240     BCS EXIT
0250     LSR A
0260     BPL NEXT
0270 EXIT STA CHAR
0280     CMP #$00
0290     RTS
```

Następna procedura zamienia znak zapisany w CHAR z kodu wewnętrznego na kod ASCII. Najpierw jednak dokonywane jest sprawdzenie trybu graficznego. Jeżeli jest to tryb bitowy (od $03 do $0B oraz $0E i $0F), to znak bez żadnej zmiany jest przepisywany do rejestru ATACHR i procedura się kończy (przez RTS - powrót z procedury).

```
0100 ;INternal to ATAscii Conversion
0110 ;
0120 ATACHR = $02FB
0130 CHAR =  $02FA
0140 DINDEX = $57
0150 INACC = $FB4D
0160 ;
0170     *=  $F76A
0180 ;
0190     LDA CHAR
0200     LDY DINDEX
0210     CPY #$0E
0220     BCS EXIT
0230     CPY #$0C
0240     BCS CNV
0250     CPY #$03
0260     BCS EXIT
0270 CNV ROL A
0280     ROL A
0290     ROL A
0300     ROL A
```

```
0310     AND #$03
0320     TAX
0330     LDA CHAR
0340     AND #$9F
0350     ORA INACC,X
0360 EXIT STA ATACHR
0370     RTS
```

W trybach tekstowych współczynnik zamiany kodu jest odczytywany z tabeli INACC według bitów 5 i 6 znaku. Po dodaniu tego współczynnika do wartości znaku, wynik umieszczany jest w ATACHR.

```
0100 ;INternal to Atascii
0110 ;Conversion Constans
0120 ;
0130     *=  $FB4D
0140 ;
0150     .BYTE $20,$40,$00,$60
```

Ponieważ procedura GETCH kończy się skokiem do KBOPN, to znak znajdujący się w ATACHR jest tam przepisywany do akumulatora i w ten sposób przekazywany do CIO.

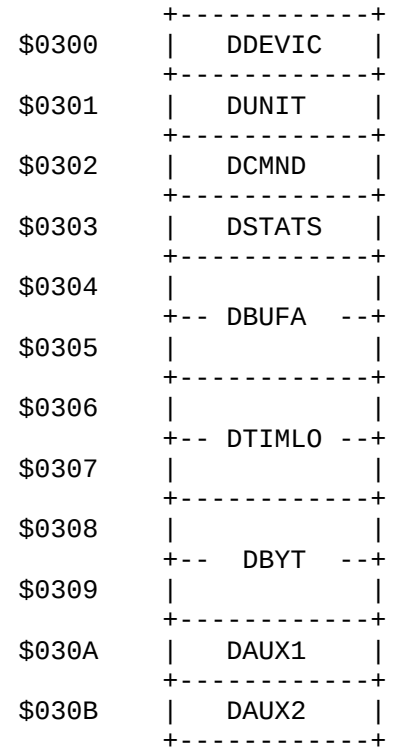
Rozdział 4

TRANSMISJA SZEREGOWA

Większość urządzeń zewnętrznych przyłączonych do systemu komunikuje się z komputerem poprzez złącze szeregowe. Po ustaleniu niezbędnych parametrów sterowniki poszczególnych urządzeń wywołują procedury transmisji przez złącze szeregowe. Ten zespół procedur zwany jest szeregowym wejściem/wyjściem (SIO - Serial Input/Output). Z procedur SIO korzystają zarówno wbudowane sterowniki drukarki, magnetofonu i stacji dysków (DSKINT - zob. rozdział 5), jak i pozostałe sterowniki instalowane podczas inicjowania systemu.

4.1. Blok kontroli urządzeń

Podobnie jak program użytkownika przesyła dane do CIO poprzez bloki IOCB, a CIO do sterowników poszczególnych urządzeń przez blok IOCBZ, tak do przesyłania parametrów transmisji między sterownikami urządzeń i SIO jest wydzielony specjalny obszar pamięci RAM. Nazywa się on blokiem kontroli urządzeń - DCB (Device Control Block) - i zajmuje część trzeciej strony pamięci od adresu $0300 do $030B. Struktura i znaczenie poszczególnych rejestrów DCB również są podobne do IOCB.



Rys.10. Struktura DCB.

DDEVIC - kod identyfikacyjny urządzenia. Stosowane są następujące identyfikatory: $30 - stacja dysków, $40 - drukarka, $50 - interfejs RS-232 i $60 - magnetofon. Dla stacji dysków i interfejsu

RS-232 identyfikator może być dodatkowo zwiększany o numer urządzenia (np. $31 - stacja dysków numer 1, $52 - interfejs numer 2).

DUNIT - numer urządzenia. Dla magnetofonu równy zero, a dla drukarki ustawiany według zawartości ICDNO. W przypadku stacji dysków i RS-232 może być równy zero, jeśli numer urządzenia został już dodany do kodu identyfikacyjnego.

DCMND - kod operacji do wykonania. OS Atari przewiduje pięć różnych operacji:

```
READ   (kod $52 - "R") - odczyt rekordu lub sektora
PUT    (kod $50 - "P") - zapis  rekordu lub sektora bez
weryfikacji
WRITE  (kod $57 - "W") - zapis  rekordu  lub  sektora z
weryfikacją
STATUS (kod $53 - "S") - żądanie statusu urządzenia
FORMAT (kod $21 - "!") - formatowanie  dyskietki (tylko
dla stacji dysków)
```

DSTATS - rodzaj i status operacji. Przed wykonaniem operacji zawiera jej rodzaj: $40 (odczyt) lub $80 (zapis). Po operacji zawiera jej status.

DBUFA - adres bufora danych dla wykonywanej operacji.

DTIMLO - wartość Timeout. Oznacza maksymalny czas wykonywania danej operacji, a więc czas oczekiwania na odpowiedź od urządzenia.

DBYT - długość bufora danych. Dla operacji dyskowych zawiera długość sektora, dla magnetofonu - długość rekordu (razem z trzema bajtami kontrolnymi).

DAUX1 - pierwszy bajt pomocniczy. Przy operacjach dyskowych zawiera starszy bajt numeru sektora, dla magnetofonu jest ignorowany, a dla pozostałych urządzeń zawiera tryb pracy.

DAUX2 - drugi bajt pomocniczy. Przy operacjach dyskowych zawiera młodszy bajt numeru sektora, dla magnetofonu rodzaj przerw między rekordami ($80 - krótkie, $00 - długie), a przy współpracy z drukarką jest ignorowany.

Niektóre źródła podają jeszcze dodatkowe kody operacji SIO ($51 - READ SPIN, $54 - READ ADDRESS, $55 - MOTOR ON i $56 - VERIFY SECTOR), lecz stacje Atari 1050 oraz Top Drive ich nie rozpoznają. Rezultatem tych operacji jest błąd $8B (DEVICE NACK - negatywne potwierdzenie operacji).

4.2. Wstępna procedura SIO

Po ustaleniu parametrów transmisji sterowniki urządzeń wywołują wstępną procedurę SIO - SIOINT. Sprawdza ona, czy chodzi o komunikację z nowym urządzeniem i ewentualnie wywołuje

procedurę, która ją przeprowadza.

Najpierw numer urządzenia jest odkładany na stos, a następnie sprawdzana jest zawartość rejestru PDVMSK (Parallel DeVice MaSK). Jeżeli jest ona równa zero, to znaczy, że transmisja nie dotyczy żadnego z urządzeń dołączonych do szyny równoległej. W takim przypadku wywoływana jest główna procedura SIO.

```
0100 ;SIO INTerface
0110 ;
0120 CRITIC = $42
0130 DSTATS = $0303
0140 DUNIT = $0301
0150 GETLOW = $C9AF
0160 PDIOR = $D805
0170 PDVMSK = $0247
0180 PDVREG = $D1FF
0190 PDVRS = $0248
0200 SIO =   $E971
0210 ;
0220     *=  $C933
0230 ;
0240     LDA #$01
0250     STA CRITIC
0260     LDA DUNIT
0270     PHA
0280     LDA PDVMSK
0290     BEQ FOUND
0300     LDX #$08
0310 NEXT JSR GETLOW
0320     BEQ FOUND
0330     TXA
0340     PHA
0350     JSR PDIOR
0360     PLA
0370     TAX
0380     BCC NEXT
0390     LDA #$00
0400     STA PDVRS
0410     STA PDVREG
0420     BEQ END
0430 FOUND JSR SIO
0440 END PLA
0450     STA DUNIT
0460     LDA #$00
0470     STA CRITIC
0480     STY DSTATS
0490     LDY DSTATS
0500     RTS
```

Jeśli komunikacja będzie prowadzona z nowym urządzeniem, to po wpisaniu do rejestru X wartości $08 rozpoczyna się pętla rozpoznająca urządzenie i wykonująca transmisję. Na początku pętli wywoływana jest procedura GETLOW, która odszukuje urządzenie o najmniejszym numerze. Gdy nie ma takiego urządzenia, również wywoływana jest procedura SIO.

Znalezienie nowego urządzenia żądającego obsługi powoduje zapamiętanie na stosie wartości

rejestru X i wywołanie procedury PDIOR (Parallel Device I/O Routine) z aktualnie aktywnego urządzenia. Procedura ta umieszczona jest w pamięci ROM urządzenia, a pod podanym tu adresem znajduje się tylko jej właściwy adres poprzedzony rozkazem skoku JMP.

Po zakończeniu PDIOR odtwarzana jest wartość rejestru X i pętla jest kontynuowana. Gdy zostaną sprawdzone wszystkie nowe urządzenia, to są one odłączane przez wyzerowanie rejestrów PDVRS i PDVREG.

Procedura SIOINT kończy się w każdym przypadku odtworzeniem ze stosu numeru urządzenia DUNIT, skasowaniem znacznika CRITIC i umieszczeniem w rejestrze Y oraz DSTATS rezultatu operacji.

Procedura GETLOW rozpoczyna się od zmniejszenia rejestru X. Jeśli uzyskana wartość jest ujemna, to po wyzerowaniu rejestrów PDVRS i PDVREG następuje powrót do SIOINT.

```
0100 ;GET LOWest
0110 ;
0120 PDVREG = $D1FF
0130 PDVMSK = $0247
0140 PDVRS = $0248
0150 BITMSK = $CA21
0160 ;
0170     *=  $C9AF
0180 ;
0190 GETLOW DEX
0200     BPL CONT
0210     LDA #$00
0220     STA PDVRS
0230     STA PDVREG
0240     RTS
0250 CONT LDA PDVMSK
0260     AND BITMSK,X
0270     BEQ GETLOW
0280     STA PDVRS
0290     STA PDVREG
0300     RTS
```

W przeciwnym razie z zawartości rejestru PDVMSK jest wydzielany bit określający urządzenie, które wymaga obsługi. Wykonywane jest to przy pomocy maski bitowej pobranej z tabeli BITMSK. Gdy bit ten jest skasowany, następuje powrót do początku GETLOW i sprawdzane jest następne urządzenie.

Wartość różna od zera jest przepisywana do rejestrów PDVRS i PDVREG. Tam ustawiony bit powoduje zaktywizowanie odpowiadającego mu urządzenia (zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego").

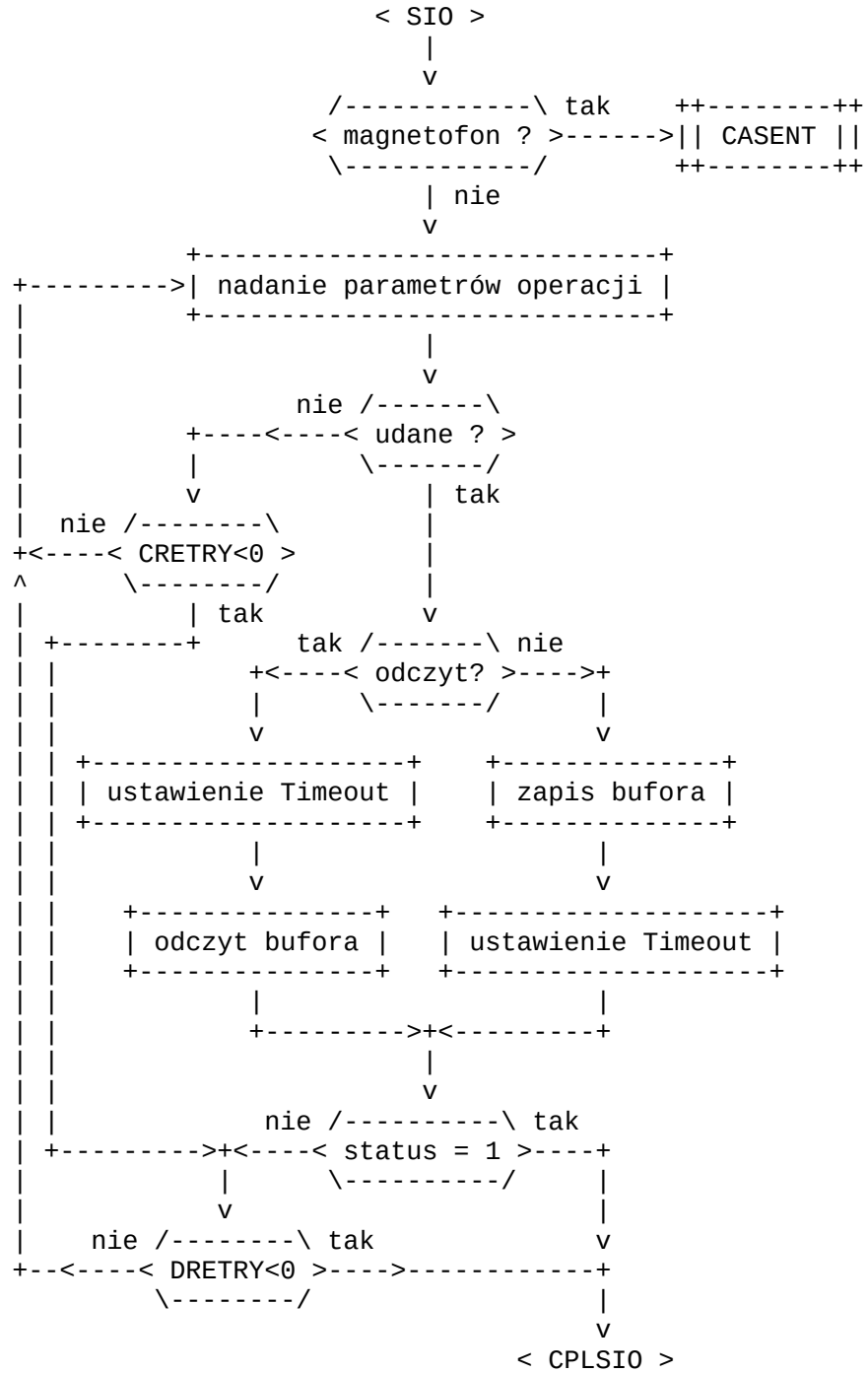
```
0100 ;BIT MaSK table
0110 ;
0120     *=  $CA21
0130 ;
0140     .BYTE $80,$40,$20,$10
```

```
0150     .BYTE $08,$04,$02,$01
```

Każda maska bitowa zawarta w tej tabeli ma ustawiony jeden bit. Ponieważ uszeregowane są one od najstarszego do najmłodszego bitu, to przy odczycie przez GETLOW od końca kolejno sprawdzane są urządzenia o numerze od 0 do 8.

4.3. Główna procedura SIO

Po stwierdzeniu braku transmisji z lub do nowego urządzenia wywoływana jest główna procedura SIO. Przeprowadza ona całą operację transmisji przez złącze szeregowe.



Rys.11. Struktura operacji SIO.

```
0100 ;SIO main routine
0110 ;
0120 AUDF3 = $D204
0130 AUDF4 = $D206
0140 BUFEN = $34
0150 BUFR =  $32
0160 CASENT = $EB9D
0170 CASFLG = $030F
0180 CAUX1 = $023C
0190 CAUX2 = $023D
0200 CCMND = $023B
0210 CDEVIC = $023A
0220 CRETRY = $029C
0230 CRITIC = $42
0240 DAUX1 = $030A
0250 DAUX2 = $030B
0260 DCMND = $0302
0270 DDEVIC = $0300
0280 DRETRY = $02BD
0290 DSTATS = $0303
0300 DUNIT = $0301
0310 ERRFLG = $023F
0320 LODPTR = $EB87
0330 PBCTL = $D303
0340 RECEIV = $EAFD
0350 SENDIN = $ECAF
0360 SETTOT = $EC9A
0370 SNDDIS = $EC84
0380 STACKP = $0318
0390 STATUS = $30
0400 STIMWT = $ECC0
0410 TSTAT = $0319
0420 ;
0430     *=  $E971
0440 ;
0450     TSX
0460     STX STACKP
0470     LDA #$01
0480     STA CRITIC
0490     LDA DDEVIC
0500     CMP #$60
0510     BNE DSK
0520     JMP CASENT
0530 DSK LDA #$00
0540     STA CASFLG
0550     LDA #$01
0560     STA DRETRY
0570 LOOP1 LDA #$0D
0580     STA CRETRY
0590 LOOP2 LDA #$28
0600     STA AUDF3
0610     LDA #$00
0620     STA AUDF4
0630     CLC
0640     LDA DDEVIC
0650     ADC DUNIT
0660     ADC #$FF
0670     STA CDEVIC
0680     LDA DCMND
0690     STA CCMND
0700     LDA DAUX1
0710     STA CAUX1
0720     LDA DAUX2
0730     STA CAUX2
0740     CLC
0750     LDA # <CDEVIC
0760     STA BUFR
0770     ADC #$04
0780     STA BUFEN
0790     LDA # >CDEVIC
0800     STA BUFR+1
0810     STA BUFEN+1
0820     LDA #$34
0830     STA PBCTL
0840     JSR SENDIN
0850     LDA ERRFLG
0860     BNE ERR
0870     TYA
0880     BNE STAT
0890 ERR DEC CRETRY
0900     BPL LOOP2
0910     JMP NEXT
0920 STAT LDA DSTATS
0930     BPL TMOT
0940     LDA #$0D
0950     STA CRETRY
0960     JSR LODPTR
0970     JSR SENDIN
0980     BEQ EXIT
0990 TMOT JSR SETTOT
1000     LDA #$00
1010     STA ERRFLG
1020     JSR STIMWT
1030     BEQ PRC
1040     BIT DSTATS
1050     BVS RCV
1060     LDA ERRFLG
1070     BNE NEXT
1080     BEQ CPLSIO
1090 RCV JSR LODPTR
1100     JSR RECEIV
1110 PRC LDA ERRFLG
1120     BEQ RES
1130     LDA TSTAT
1140     STA STATUS
1150 RES LDA STATUS
1160     CMP #$01
1170     BEQ CPLSIO
1180 NEXT DEC DRETRY
1190     BMI CPLSIO
1200     JMP LOOP1
1210 ;
```

```
1220 ;ComPLete SIO            1260     STA CRITIC
1230 ;                        1270     LDY STATUS
1240 CPLSIO JSR SNDDIS        1280     STY DSTATS
1250     LDA #$00             1290     RTS
```

Przed przystąpieniem do transmisji w rejestrze STACKP (STACK Pointer) zapamiętywany jest wskaźnik stosu procesora, a znacznik CRITIC jest ustawiany na 1 (jest on sprawdzany przez procedurę przerwania SYSVBL - zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego"). Następnie sprawdzany jest kod urządzenia w rejestrze DDEVIC. Jeżeli oznacza on magnetofon, to następuje skok do CASENT, gdzie prowadzona jest komunikacja z magnetofonem.

W przeciwnym razie kasowany jest znacznik magnetofonu CASFLG (CASsette FLaG) i ustalane są liczby prób podjęcia komunikacji z urządzeniem DRETRY (Device RETRY) i CRETRY (Command RETRY). Następnie wpisywane są do rejestrów AUDF3 i AUDF4 wartości określające szybkość transmisji.

Polecenie wykonania operacji jest wysyłane do urządzenia w formie czterobajtowego bloku, zwanego CFB (Command Frame Buffer). Zawartości DDEVIC, DCMND, DAUX1 i DAUX2 są więc przepisywane do odpowiednich rejestrów CFB. Adres pierwszego rejestru CFB (CDEVIC) jest wpisywany jako adres bufora BUFR (BUFfeR), a adres następnego rejestru po CFB jako adres końca bufora BUFEN (BUFfer ENd). Teraz przez rejestr PBCTL sygnalizowane jest nadawanie polecenia i przez wywołanie procedury SENDIN blok CFB jest wysyłany do urządzenia. W przypadku wystąpienia błędu - sygnalizowanego w rejestrze ERRFLG (ERRor FLaG) - czynność ta jest powtarzana, aż do wyzerowania rejestru CRETRY.

Po przyjęciu polecenia przez urządzenie rozpoczyna się właściwa operacja. Kolejność czynności jest przy tym odwrotna dla zapisu i odczytu, a do rozpoznania rodzaju operacji służy rejestr DSTATS (Device STATuS). Ustawienie w nim bitu 7 sygnalizuje zapis, a ustawienie bitu 6 oznacza odczyt.

Przy zapisie najpierw adres bufora danych jest ustawiany przez procedurę LODPTR i informacja jest wysyłana przez wywołanie SENDIN. Następnie ustalany jest (przez procedurę SETTOT) maksymalny czas wykonywania operacji i komputer czeka na odpowiedź urządzenia. Ta czynność jest przeprowadzana przez wywołanie STIMWT.

Podczas odczytu najpierw wywoływane są procedury SETTOT i STIMWT, a dopiero po zgłoszeniu się urządzenia procedura LODPTR ustala adres bufora, a procedura RECEIV dokonuje odczytu przesyłanej informacji.

Jeżeli podczas operacji wystąpił błąd, to jest ona powtarzana od początku, aż do wyzerowania rejestru DRETRY. Pozytywny wynik operacji powoduje przejście do końcowej fazy procedury SIO oznaczonej etykietą CPLSIO. Wywoływana jest tam procedura SNDDIS, która kończy transmisję,

zerowany jest znacznik CRITIC, a rezultat operacji jest przepisywany z rejestru STATUS do DSTATS.

Opis procedur wykorzystywanych przez SIO rozpoczną dwie krótkie procedury pomocnicze LODPTR i SETTOT.

Procedura LODPTR przenosi adres i długość bufora danych z rejestrów DBUFA i DBYT do rejestrów BUFR i BUFEN. Ponieważ BUFEN zawiera adres końca bufora, to przed zapisaniem do tego rejestru długość bufora DBYT musi być dodana do jego adresu DBUFA.

```
0100 ;LOaD PoinTeR
0110 ;
0120 BUFEN = $34
0130 BUFR =  $32
0140 DBUFA = $0304
0150 DBYT =  $0308
0160 ;
0170     *=  $EB87
0180 ;
0190     CLC
0200     LDA DBUFA
0210     STA BUFR
0220     ADC DBYT
0230     STA BUFEN
0240     LDA DBUFA+1
0250     STA BUFR+1
0260     ADC DBYT+1
0270     STA BUFEN+1
0280     RTS
```

Procedura SETTOT przygotowuje parametry dla STIMWT. W bitach 0-5 rejestru X umieszczane są bity 2-7 z DTIMLO, a w bitach 6 i 7 rejestru Y bity 0 i 1 z DTIMLO.

```
0100 ;SET TimeOuT
0110 ;
0120 DTIMLO = $0306
0130 ;
0140     *=  $EC9A
0150 ;
0160     LDA DTIMLO
0170     ROR A
0180     ROR A
0190     TAY
0200     AND #$3F
0210     TAX
0220     TYA
0230     ROR A
0240     AND #$C0
0250     TAY
0260     RTS
```

Operacja wysyłania informacji przez złącze szeregowe jest przeprowadzana przez procedurę SENDIN. Jej część od etykiety STIMWT jest także wykorzystywana podczas odczytu z urządzenia.

SENDIN rozpoczyna się od podwójnej pętli opóźniającej. Po jej zakończeniu wywoływana jest procedura SEND, która wysyła informację przez złącze szeregowe. Następnie w rejestrach X i Y ustawiane są parametry czasu oczekiwania na potwierdzenie i wywoływana jest procedura SETT1V, uruchamiająca przerwanie zegara TIMCNT1. Na końcu wywoływana jest procedura WAIT, w czasie której komputer oczekuje na odpowiedź od urządzenia zewnętrznego.

```
0100 ;SEND INitiation
0110 ;
0120 SEND =  $EA88
0130 SETT1V = $EDE2
0140 WAIT =  $EA37
0150 ;
0160     *=  $ECAF
0170 ;
0180     LDX #$01
0190 LOOP1 LDY #$FF
0200 LOOP2 DEY
0210     BNE LOOP2
0220     DEX
0230     BNE LOOP1
0240     JSR SEND
0250     LDY #$02
0260     LDX #$00
0270 ;
0280 ;Set TIMeout and WaiT
0290 STIMWT JSR SETT1V
0300     JSR WAIT
0310     TYA
0320     RTS
```

Zadaniem procedury SETT1V jest przygotowanie systemu do odliczania czasu oczekiwania na odpowiedź. W tym celu wektor przerwania TIMVEC1 jest ustawiany na adres procedury TIM1INT. Następnie wywoływana jest procedura SETVBLV, która ustawia licznik TIMCNT1 według wartości umieszczonych w rejestrach X i Y przed wywołaniem SETT1V. Na zakończenie znacznik TIMFLG (TIMeout FLaG) otrzymuje wartość 1.

```
0100 ;SET Timer 1 Vector
0110 ;
0120 JSETVBV = $E45C
0130 TIM1INT = $EC11
0140 TIMFLG = $0317
0150 TIMVEC1 = $0226
0160 ;
0170     *=  $EDE2
0180 ;
0190     LDA # <TIM1INT
0200     STA TIMVEC1
0210     LDA # >TIM1INT
0220     STA TIMVEC1+1
0230     LDA #$01
0240     SEI
0250     JSR JSETVBV
0260     LDA #$01
0270     STA TIMFLG
0280     CLI
0290     RTS
```

Drugą ważną procedurą wywoływaną przez SENDIN jest WAIT. Oczekuje ona na odpowiedź urządzenia, a następnie bada poprawność przeprowadzonej operacji.

```
0100 ;WAIT for completion
0110 ;
0120 BUFEN = $34
0130 BUFR =  $32
0140 ERRFLG = $023F
0150 NOCKSM = $3C
0160 RECEIV = $EAFD
0170 STATUS = $30
0180 TEMP =  $023E
0190 TSTAT = $0319
0200 ;
0210     *=  $EA37
0220 ;
0230     LDA #$00
0240     STA ERRFLG
0250     CLC
0260     LDA # <TEMP
0270     STA BUFR
0280     ADC #$01
0290     STA BUFEN
0300     LDA # >TEMP
0310     STA BUFR+1
0320     STA BUFEN+1
0330     LDA #$FF
0340     STA NOCKSM
0350     JSR RECEIV
0360     LDY #$FF
0370     LDA STATUS
0380     CMP #$01
0390     BNE ERR
0400     LDA TEMP
0410     CMP #$41
0420     BEQ END
0430     CMP #$43
0440     BEQ END
0450     CMP #$45
0460     BNE NACK
0470     LDA #$90
0480     STA STATUS
0490     BNE ERR
0500 NACK LDA #$8B
0510     STA STATUS
0520 ERR LDA STATUS
0530     CMP #$8A
0540     BEQ TMOT
0550     LDA #$FF
0560     STA ERRFLG
0570     BNE END
0580 TMOT LDY #$00
0590 END LDA STATUS
0600     STA TSTAT
0610     RTS
```

Po wyzerowaniu znacznika błędu ERRFLG wektor bufora BUFR jest ustawiany na adres rejestru tymczasowego TEMP, do którego zostanie odczytana odpowiedź urządzenia. Następnie

wywoływana jest procedura RECEIV, która dokonuje odczytu informacji ze złącza szeregowego.

Jeśli podczas transmisji nie wystąpił błąd (STATUS=$01), to sprawdzana jest odpowiedź urządzenia. Wartości $41 i $43 oznaczają poprawne wykonanie transmisji i przerywają procedurę. Wartość $45 oznacza potwierdzenie negatywne i powoduje wpisanie do rejestru STATUS kodu błędu $8B (DEVICE NACK - negatywne potwierdzenie). Każda inna zawartość rejestru TEMP powoduje wpisanie do STATUS kodu błędu $90 (DEVICE DONE ERROR - błąd wykonania).

Po wykryciu dowolnego błędu sprawdzany jest rejestr STATUS i gdy nie wskazuje przekroczenia czasu ($8A - TIMEOUT), to znacznik ERRFLG otrzymuje wartość $FF. W każdym przypadku procedura kończy się przepisaniem zawartości rejestru STATUS do TSTAT.

4.3.1. Nadawanie na złącze szeregowe

Właściwe przesłanie danych do złącza szeregowego jest wykonywane przez procedurę SEND. Po ustawieniu statusu na 1 wywołuje ona procedurę uruchamiającą nadawanie SNDENBL.

```
0100 ;SEND buffer to serial bus
0110 ;
0120 BUFR =  $32
0130 CHKSNT = $3B
0140 CHKSUM = $31
0150 IRQSTAT = $11
0160 PRBRKK = $EDC7
0170 SEROUT = $D20D
0180 SNDDIS = $EC84
0190 SNDENBL = $EC17
0200 STATUS = $30
0210 XMTDON = $3A
0220 ;
0230     *=  $EA88
0240 ;
0250     LDA #$01
0260     STA STATUS
0270     JSR SNDENBL
0280     LDY #$00
0290     STY CHKSUM
0300     STY CHKSNT
0310     STY XMTDON
0320     LDA (BUFR),Y
0330     STA SEROUT
0340     STA CHKSUM
0350 LOOP LDA IRQSTAT
0360     BNE CONT
0370     JMP PRBRKK
0380 CONT LDA XMTDON
0390     BEQ LOOP
0400     JSR SNDDIS
0410     RTS
```

Następnie zeruje rejestry CHKSUM (CHecK SUM), CHKSNT (CHecK sum SeNT) i XMTDON (eXMiTe DONe). Bajt odczytany z bufora wskazywanego przez BUFR przepisywany jest do rejestrów CHKSUM i SEROUT. Ostatnia czynność powoduje żądanie przerwania i wywołanie

procedury ISRODN.

Teraz komputer czeka na zmianę wartości znacznika XMTDON sygnalizującą zakończenie transmisji. W tym czasie sprawdzany jest rejestr IRQSTAT, a gdy wskaże on naciśnięcie klawisza BREAK, następuje skok do środka procedury BEGNRD w miejsce oznaczone etykietą PRBRKK. Po zakończeniu transmisji wywoływana jest jeszcze procedura SNDDIS.

Zadaniem procedury SNDENBL jest zezwolenie na wysyłanie informacji przez złącze szeregowe. W tym celu wpisuje ona odpowiednie wartości do rejestrów SKCTLS (Serial and Keyboard ConTroL Shadow register) i IRQENS (Interrupt ReQuest ENable Shadow register). Ponadto przy współpracy z magnetofonem ustawiane są częstotliwości generatorów dźwięku numer 1 i 2. Procedura kończy się skokiem do środka procedury RECVEN (zob. rozdział 4.3.2.).

```
0100 ;SeND ENaBLe
0110 ;
0120 AUDF1 = $D200
0130 AUDF2 = $D202
0140 DDEVIC = $0300
0150 ENABLE = $EC56
0160 IRQENS = $10
0170 SKCTLS = $0232
0180 SKSTAT = $D20F
0190 ;
0200     *=  $EC17
0210 ;
0220     LDA #$07
0230     AND SKCTLS
0240     ORA #$20
0250     LDY DDEVIC
0260     CPY #$60
0270     BNE NCR
0280     ORA #$08
0290     LDY #$07
0300     STY AUDF2
0310     LDY #$05
0320     STY AUDF1
0330 NCR STA SKCTLS
0340     STA SKSTAT
0350     LDA #$C7
0360     AND IRQENS
0370     ORA #$10
0380     JMP ENABLE
```

Po nadaniu informacji zadanie zamknięcia komunikacji wypełnia procedura SNDDIS. Ustawia ona rejestry zezwoleń na przerwania IRQEN i IRQENS oraz zeruje rejestry kontroli generatorów dźwięku AUDC1-3.

```
0100 ;SeND DISable
0110 ;
0120 AUDC1 = $D201
0130 IRQEN = $D20E
0140 IRQENS = $10
0150 ;
0160     *=  $EC84
```

```
0170 ;
0180     NOP
0190     LDA #$C7
0200     AND IRQENS
0210     STA IRQENS
0220     STA IRQEN
0230     LDX #$06
0240     LDA #$00
0250 NEXT STA AUDC1,X
0260     DEX
0270     DEX
0280     BPL NEXT
0290     RTS
```

4.3.2. Odczyt ze złącza szeregowego

Właściwy odczyt z szyny szeregowej jest wykonywany przez procedurę RECEIV. Jeżeli odczyt dokonywany jest z magnetofonu, to najpierw zerowany jest rejestr sumy kontrolnej CHKSUM, zaś niezależnie od rodzaju urządzenia zerowane są znaczniki BUFRFL (BUFfeR FulL) i RECVND (RECeiVe eND). Następnie po ustawieniu statusu na 1 wywoływana jest procedura RECVEN, która uruchamia odczyt.

Po zasygnalizowaniu w rejestrze PBCTL gotowości do odczytu komputer oczekuje w pętli na zmianę wartości w rejestrze RECVND, która sygnalizuje zakończenie odczytu. Podczas pętli sprawdzane są jeszcze rejestry IRQSTAT (sygnalizuje naciśnięcie BREAK) i TIMFLG (sygnalizuje błąd Timeout).

Procedura RECEIV może być zakończona trzema sposobami: po poprawnym odczycie rozkazem RTS, po naciśnięciu klawisza BREAK skokiem do PRBRKK lub po przekroczeniu czasu skokiem do ITIMOT. W tym ostatnim przypadku do rejestru STATUS wpisywany jest kod błędu $8A (TIMEOUT ERROR - przekroczenie czasu).

```
0100 ;RECEIVe
0110 ;
0120 BUFRFL = $38
0130 CASFLG = $030F
0140 CHKSUM = $31
0150 IRQSTAT = $11
0160 PBCTL = $D303
0170 PRBRKK = $EDC7
0180 RECVEN = $EC40
0190 RECVND = $39
0200 STATUS = $30
0210 TIMFLG = $0317
0220 ;
0230     *=  $EAFD
0240 ;
0250     LDA #$00
0260     LDY CASFLG
0270     BNE BPS
0280     STA CHKSUM
0290 BPS STA BUFRFL
0300     STA RECVND
```

```
0310     LDA #$01
0320     STA STATUS
0330     JSR RECVEN
0340     LDA #$3C
0350     STA PBCTL
0360 NEXT LDA IRQSTAT
0370     BNE PRC
0380     JMP PRBRKK
0390 PRC LDA TIMFLG
0400     BEQ ITIMOT
0410     LDA RECVND
0420     BEQ NEXT
0430     RTS
0440 ;
0450 ;Indicate TIMeOuT
0460 ;
0470 ITIMOT LDA #$8A
0480     STA STATUS
0490     RTS
```

Procedura RECVEN spełnia taką samą funkcję przy odczycie jak SENDEN przy zapisie. Część od etykiety ENABLE jest nawet wspólna dla obu procedur. Na początku odpowiednie wartości są wpisywane do rejestrów SKCTL, SKCTLS, IRQEN i IRQENS. Następnie ustawiane są parametry generatorów dźwięku w rejestrach kontroli AUDCTL oraz AUDC1-3.

```
0100 ;RECeiVe ENable
0110 ;
0120 AUDC1 = $D201
0130 AUDC2 = $D203
0140 AUDC3 = $D205
0150 AUDCTL = $D208
0160 DDEVIC = $0300
0170 IOSNDEN = $41
0180 IRQEN = $D20E
0190 IRQENS = $10
0200 SKCTL = $D20F
0210 SKCTLS = $0232
0220 SKSTRES = $D20A
0230 ;
0240     *=  $EC40
0250 ;
0260     LDA #$07
0270     AND SKCTLS
0280     ORA #$10
0290     STA SKCTLS
0300     STA SKCTL
0310     STA SKSTRES
0320     LDA #$C7
0330     AND IRQENS
0340     ORA #$20
0350 ENABLE STA IRQENS
0360     STA IRQEN
0370     LDA #$28
0380     STA AUDCTL
0390     LDX #$06
0400     LDA #$A8
0410     LDY IOSNDEN
0420     BNE NEXT
```

```
0430     LDA #$A0
0440 NEXT STA AUDC1,X
0450     DEX
0460     DEX
0470     BPL NEXT
0480     LDA #$A0
0490     STA AUDC3
0500     LDY DDEVIC
0510     CPY #$60
0520     BEQ EXIT
0530     STA AUDC1
0540     STA AUDC2
0550 EXIT RTS
```

4.4. Procedury SIO dla magnetofonu

Rozpoznanie przez procedurę SIO komunikacji z magnetofonem powoduje skok do procedury CASENT. Oddzielenie procedury SIO magnetofonu od procedury obsługującej pozostałe urządzenia jest spowodowane jego specyfiką. Jako jedyne z urządzeń peryferyjnych magnetofon nie posiada układów sterujących i komputer komunikuje się z nim "w ciemno".

Struktura procedury CASENT jest zbliżona do procedury SIO. Tu także jedynie początek i koniec są wspólne dla odczytu i zapisu, natomiast w pozostałej części kolejność faz jest zamieniona. Do rozróżnienia rodzaju operacji wykorzystywana jest zawartość rejestru DSTATS.

Podczas zapisu najpierw ustalane są częstotliwości generatorów dźwięku i wywoływana jest procedura SNDENBL, która uruchamia nadawanie informacji na szynę szeregową. Następnie według zawartości rejestru PALNTS odczytywane są z tabeli STVCTB dane, które przekazane procedurze SETT1V służą do ustalenia czasu operacji.

```
0100 ;System TV Constant TaBle
0110 ;
0120     *=  $EE11
0130 ;
0140     .BYTE $B4,$96,$78,$64
0150     .BYTE $0F,$0D,$0A,$08
0160     .BYTE $83,$9C
```

Po ustawieniu Timeout uruchamiany jest silnik magnetofonu i komputer czeka w pętli WAIT1. W tym czasie na kasecie nagrywany jest sygnał pilotujący. Następnie wywoływana jest procedura LODPTR, która ustawia adresy bufora danych, oraz procedura SEND, która przepisuje zawartość bufora na szynę szeregową.

Zapis kończy się skokiem do ostatniej fazy procedury (wspólnej dla obu operacji). Jeżeli zawartość DAUX2 sygnalizuje długie przerwy między rekordami, to wyłączany jest silnik magnetofonu. Procedura CASENT kończy się skokiem do CPLSIO.

Operacja odczytu rozpoczyna się od ustalenia czasu Timeout przez procedurę SETT1V na podstawie wartości pobranych z tabeli STVCTB. Następnie uruchamiany jest silnik magnetofonu.

Po upływie czasu określonego przez Timeout w celu ustawienia adresu bufora wywoływana jest procedura LODPTR.

W dalszej kolejności wywoływana jest procedura SETTOT, która zwraca parametry Timeout i ponownie SETT1V ustawia procedurę przerwania TIMCNT1. Następnie poprzez BEGNRD rozpoczynany jest odczyt, którego kontynuację przejmuje procedura RECEIV. Po jej zakończeniu komputer przechodzi do ostatniej fazy CASENT (zob. wyżej).

```
0100 ;CASsette ENTer
0110 ;
0120 AUDF3 = $D204
0130 AUDF4 = $D206
0140 BEGNRD = $ED3D
0150 CASFLG = $030F
0160 CPLSIO = $EA2A
0170 DAUX2 = $030B
0180 DSTATS = $0303
0190 LODPTR = $EB87
0200 PACTL = $D302
0210 PALNTS = $62
0220 RECEIV = $EAFD
0230 SEND =  $EA88
0240 SNDENBL = $EC17
0250 SETT1V = $EDE2
0260 SETTOT = $EC9A
0270 STVCTB = $EE11
0280 TIMFLG = $0317
0290 ;
0300     *=  $EB9D
0310 ;
0320     LDA DSTATS
0330     BPL READ
0340     LDA #$CC
0350     STA AUDF3
0360     LDA #$05
0370     STA AUDF4
0380     JSR SNDENBL
0390     LDX PALNTS
0400     LDY STVCTB+4,X
0410     LDA DAUX2
0420     BMI SKIP1
0430     LDY STVCTB,X
0440 SKIP1 LDX #$00
0450     JSR SETT1V
0460     LDA #$34
0470     STA PACTL
0480 WAIT1 LDA TIMFLG
0490     BNE WAIT1
0500     JSR LODPTR
0510     JSR SEND
0520     JMP END
0530 READ LDA #$FF
0540     STA CASFLG
0550     LDX PALNTS
0560     LDY STVCTB+6,X
0570     LDA DAUX2
0580     BMI SKIP2
```

```
0590     LDY STVCTB+2,X
0600 SKIP2 LDX #$00
0610     JSR SETT1V
0620     LDA #$34
0630     STA PACTL
0640 WAIT2 LDA TIMFLG
0650     BNE WAIT2
0660     JSR LODPTR
0670     JSR SETTOT
0680     JSR SETT1V
0690     JSR BEGNRD
0700     JSR RECEIV
0710 END LDA DAUX2
0720     BMI EXIT
0730     LDA #$3C
0740     STA PACTL
0750 EXIT JMP CPLSIO
```

Rozpoczęcie odczytu jest przeprowadzane przez procedurę BEGNRD. Najpierw oczekuje ona w pętli na sygnał (w SKCTL) odczytania informacji z szyny szeregowej. Podczas pętli sprawdzany jest klawisz BREAK (poprzez rejestr IRQSTAT) oraz upływ czasu operacji Timeout (poprzez znacznik TIMFLG). Naciśnięcie BREAK powoduje skok do PRBRKK, gdzie przerywana jest transmisja i ustawiany status operacji (na $80 - BREAK ABORT - przerwanie klawiszem BREAK). Przekroczenie maksymalnego czasu trwania operacji powoduje natomiast skok do ITIMOT.

```
0100 ;BEGiN ReaD
0110 ;
0120 AUDF3 = $D204
0130 AUDF4 = $D206
0140 BUFR =  $32
0150 CBAUD = $02EE
0160 CHKSUM = $31
0170 COMPUT = $ECC8
0180 CPLSIO = $EA2A
0190 INTIM1 = $030C
0200 IRQSTAT = $11
0210 ITIMOT = $EB27
0220 PACTL = $D302
0230 PBCTL = $D303
0240 RTCLOCK = $12
0250 SAVIO = $0316
0260 SNDDIS = $EC84
0270 SKCTL = $D20F
0280 SKCTLS = $0232
0290 STACKP = $0318
0300 STATUS = $30
0310 TEMP3 = $0315
0320 TIMFLG = $0317
0330 VCOUNT = $D40B
0340 ;
0350     *=  $ED3D
0360 ;
0370 BEGNRD LDA IRQSTAT
0380     BNE PRC
0390     JMP PRBRKK
0400 PRC SEI
0410     LDA TIMFLG
0420     BNE CONT
```

```
0430     BEQ TOT
0440 CONT LDA SKCTL
0450     AND #$10
0460     BNE BEGNRD
0470     STA SAVIO
0480     LDX VCOUNT
0490     LDY RTCLOCK+2
0500     STX INTIM1
0510     STY INTIM1+1
0520     LDX #$01
0530     STX TEMP3
0540     LDY #$0A
0550 STT LDA IRQSTAT
0560     BEQ PRBRKK
0570     LDA TIMFLG
0580     BNE CTL
0590 TOT CLI
0600     JMP ITIMOT
0610 CTL LDA SKCTL
0620     AND #$10
0630     CMP SAVIO
0640     BEQ STT
0650     STA SAVIO
0660     DEY
0670     BNE STT
0680     DEC TEMP3
0690     BMI CPT
0700     LDA VCOUNT
0710     LDY RTCLOCK+2
0720     JSR COMPUT
0730     LDY #$09
0740     BNE STT
0750 CPT LDA CBAUD
0760     STA AUDF3
0770     LDA CBAUD+1
0780     STA AUDF4
0790     LDA #$00
0800     STA SKCTL
0810     LDA SKCTLS
0820     STA SKCTL
0830     LDA #$55
0840     STA (BUFR),Y
0850     INY
0860     STA (BUFR),Y
0870     LDA #$AA
0880     STA CHKSUM
0890     CLC
0900     LDA BUFR
0910     ADC #$02
0920     STA BUFR
0930     LDA BUFR+1
0940     ADC #$00
0950     STA BUFR+1
0960     CLI
0970     RTS
0980 ;
0990 ;PRocess BReaK Key
1000 ;
1010 PRBRKK JSR SNDDIS
1020     LDA #$3C
```

```
1030     STA PACTL
1040     LDA #$3C
1050     STA PBCTL
1060     LDA #$80
1070     STA STATUS
1080     LDX STACKP
1090     TXS
1100     DEC IRQSTAT
1110     CLI
1120     JMP CPLSIO
```

Po zasygnalizowaniu w rejestrze SKCTL rozpoczęcia transmisji do rejestru INTIM1 (INterval TIMer 1) przepisywane są stany licznika linii ekranu VCOUNT i zegara RTCLOCK. Teraz komputer oczekuje na następny sygnał w SKCTL. Po jego uzyskaniu aktualne stany VCOUNT i RTCLOCK są zapisywane odpowiednio w akumulatorze i rejestrze Y oraz wywoływana jest procedura COMPUT. Służy ona do obliczenia właściwej prędkości transmisji.

Ustalona prędkość transmisji jest teraz przepisywana do rejestrów częstotliwości generatorów dźwięku. Po wpisaniu wartości początkowych bajtów rekordu (były one wykorzystane do regulacji szybkości) do bufora procedura się kończy.

Procedura COMPUT oblicza rzeczywistą szybkość transmisji z magnetofonu. Umożliwia to dostosowanie się komputera do nierównomiernego przesuwu taśmy w magnetofonie oraz do zmian jej długości. Na początku, po zapisaniu w rejestrze INTIM2 (INterval TIMer 2) przekazanych wartości, dwukrotnie wywoływana jest procedura ADJUST. Jej zadaniem jest poprawienie wartości odczytanych z licznika linii obrazu w zależności od systemu TV, w którym pracuje komputer.

```
0100 ;ADJUST
0110 ;
0120 ADJTAB = $EE1B
0130 PALNTS = $62
0140 ;
0150     *=  $ED2E
0160 ;
0170     CMP #$7C
0180     BMI ADJ
0190     SEC
0200     SBC #$7C
0210     RTS
0220 ADJ CLC
0230     LDX PALNTS
0240     ADC ADJTAB,X
0250     RTS
```

Jeśli przekazana w akumulatorze wartość jest większa od $7C, to jest ona zmniejszana o tą wartość. Gdy jest mniejsza, to dodawana jest wartość pobrana z tabeli ADJTAB według znacznika zastosowanego systemu TV (PALNTS).

```
0100 ;ADJust TABle
0110 ;
0120     *=  $EE1B
0130 ;
```

```
0140     .BYTE $07,$20
```

Następnie różnica między poprawionymi wartościami licznika linii jest zapisywana do przejściowego rejestru TEMP1, a różnica między stanami zegara RTCLOCK do rejestru Y. Odczytana z tabeli STVCTB wartość jest teraz mnożona przez zawartość Y (dodawana Y razy) i do rezultatu dodawana jest zawartość TEMP1. Wynik ten zmniejszony o $16 jest umieszczany w rejestrze X, a jego 3 najmłodsze bity (przed zmniejszeniem) w rejestrze Y.

Zawartość rejestru Y służy dalej jako mnożnik liczby $0B dodawanej do $F5. Od osiągniętego wyniku jest jeszcze odejmowane $07 i gdy rezultat jest ujemny, to w Y znajduje się wartość $FF, a w przeciwnym razie - $00.

```
0100 ;COMPUTe baud rate
0110 ;
0120 ADJUST = $ED2E
0130 CBAUD = $02EE
0140 INTIM1 = $030C
0150 INTIM2 = $0310
0160 PALNTS = $62
0170 POKTAB = $EDF9
0180 STVCTB = $EE11
0190 TEMP1 = $0312
0200 ;
0210     *=  $ECC8
0220 ;
0230     STA INTIM2
0240     STY INTIM2+1
0250     JSR ADJUST
0260     STA INTIM2
0270     LDA INTIM1
0280     JSR ADJUST
0290     STA INTIM1
0300     LDA INTIM2
0310     SEC
0320     SBC INTIM1
0330     STA TEMP1
0340     LDA INTIM2+1
0350     SEC
0360     SBC INTIM1+1
0370     TAY
0380     LDX PALNTS
0390     LDA #$00
0400     SEC
0410     SBC STVCTB+8,X
0420 LOOP1 CLC
0430     ADC STVCTB+8,X
0440     DEY
0450     BPL LOOP1
0460     CLC
0470     ADC TEMP1
0480     TAY
0490     LSR A
0500     LSR A
0510     LSR A
0520     ASL A
0530     SEC
```

```
0540     SBC #$16
0550     TAX
0560     TYA
0570     AND #$07
0580     TAY
0590     LDA #$F5
0600 LOOP2 CLC
0610     ADC #$0B
0620     DEY
0630     BPL LOOP2
0640     LDY #$00
0650     SEC
0660     SBC #$07
0670     BPL SKIP
0680     DEY
0690 SKIP CLC
0700     ADC POKTAB,X
0710     STA CBAUD
0720     TYA
0730     ADC POKTAB+1,X
0740     STA CBAUD+1
0750     RTS
```

Przechowywana w rejestrze X wartość służy teraz do odczytu współczynników z tabeli POKTAB. Są one dodawane do zawartości akumulatora i rejestru Y, po czym otrzymane w rezultacie liczby zostają przepisane do rejestru CBAUD (Cassette BAUD rate). Rejestr ten służy do ustalenia częstotliwości generatorów dźwięku sterujących szybkością transmisji.

```
0100 ;POKey TABle
0110 ;
0120     *=  $EDF9
0130 ;
0140     .BYTE $E8,$03,$43,$04
0150     .BYTE $9E,$04,$F9,$04
0160     .BYTE $54,$05,$AF,$05
0170     .BYTE $0A,$06,$65,$06
0180     .BYTE $C0,$06,$1A,$07
0190     .BYTE $75,$07,$D0,$07
```

Rozdział 5

STEROWNIK DYSKOWY

Komputery Atari do współpracy ze stacją dysków wymagają najpierw wczytania z dyskietki DOS-u (Disk Operating System). Ten powszechny pogląd jest jednak niesłuszny, a dowód na to jest w nim zawarty. Otóż DOS musi być odczytany z dyskietki, a jak to zrobić skoro bez DOS-u nie można.

System operacyjny musi więc zawierać procedurę pozwalającą na komunikację ze stacją dysków. Procedura ta nazywa się DSKINT i jest dostępna dla użytkownika wyłącznie z poziomu języka maszynowego. Wykorzystuje ona do komunikacji ze stacją dysków blok DCB i procedury SIO (zob. rozdział 4).

Przed wywołaniem DSKINT w rejestrze DCMND musi zostać umieszczony kod operacji. Podczas odczytu i zapisu sektora rejestry DAUX1 i DAUX2 muszą zawierać numer sektora, a rejestr DBUFA - adres bufora. Podania adresu bufora wymaga także operacja formatowania dyskietki.

Na początku procedury kod urządzenia jest ustawiany na wartość $31 (stacja dysków numer 1). Oznacza to, że komunikację z inną stacją dysków należy wykonywać przez umieszczenie w akumulatorze jej kodu i wywołanie procedury od adresu DSKINT+2. Następnie ustalana jest wartość Timeout - dla formatowania pobierana jest z rejestru DSKTIM (DiSK TIMeout), a dla pozostałych operacji wynosi $07. Również zawartość rejestru DSTATS zależy od rodzaju operacji: dla zapisu jest ona równa $80, a dla innych operacji - $40.

```
0100 ;DiSK INTerface
0110 ;
0120 BUFADR = $15
0130 DBUFA = $0304
0140 DBYT =  $0308
0150 DCMND = $0302
0160 DDEVIC = $0300
0170 DSCTLN = $02D5
0180 DSKTIM = $0246
0190 DSTATS = $0303
0200 DTIMLO = $0306
0210 DVSTAT = $02EA
0220 JSIOINT = $E459
0230 PUTADR = $C73A
0240 ;
0250     *=  $C6B3
0260 ;
0270     LDA #$31
0280     STA DDEVIC
0290     LDA DSKTIM
0300     LDX DCMND
0310     CPX #'!
```

```
0320     BEQ STM
0330     LDA #$07
0340 STM STA DTIMLO
0350     LDX #$40
0360     LDA DCMND
0370     CMP #'P
0380     BEQ WRT
0390     CMP #'W
0400     BNE READ
0410 WRT LDX #$80
0420 READ CMP #'S
0430     BNE DSL
0440     LDA # >DVSTAT
0450     STA DBUFA
0460     LDA # <DVSTAT
0470     STA DBUFA+1
0480     LDY #$04
0490     LDA #$00
0500     BEQ SPM
0510 DPL LDY DSCTLN
0520     LDA DSCTLN+1
0530 SPM STX DSTATS
0540     STY DBYT
0550     STA DBYT+1
0560     JSR JSIOINT
0570     BPL SUC
0580     RTS
0590 SUC LDA DCMND
0600     CMP #'S
0610     BNE FRMT
0620     JSR PUTADR
0630     LDY #$02
0640     LDA (BUFADR),Y
0650     STA DSKTIM
0660 FRMT LDA DCMND
0670     CMP #'!
0680     BNE EXIT
0690     JSR PUTADR
0700     LDY #$FE
0710 LOOP1 INY
0720     INY
0730 LOOP2 LDA (BUFADR),Y
0740     CMP #$FF
0750     BNE LOOP1
0760     INY
0770     LDA (BUFADR),Y
0780     INY
0790     CMP #$FF
0800     BNE LOOP2
0810     DEY
0820     DEY
0830     STY DBYT
0840     LDA #$00
0850     STA DBYT+1
0860 EXIT LDY DSTATS
0870     RTS
```

Odczyt statusu daje w rezultacie czterobajtowy blok informacji, który umieszczany jest w rejestrze DVSTAT. Dla tej operacji jest więc odpowiednio ustawiany adres bufora. Długość bufora

dla wszystkich operacji jest pobierana z rejestru DSCTLN (Disk SeCTor LeNgth).

Po tych czynnościach przygotowawczych wywoływana jest procedura SIOINT, która przeprowadza żądaną operację. Jeżeli zakończy się ona błędem, to DSKINT jest przerywana rozkazem RTS. W przeciwnym przypadku sprawdzany jest jeszcze kod operacji. Gdy nie jest to ani STATUS, ani FORMAT, to po odczytaniu do rejestru Y zawartości DSTATS procedura także się kończy.

```
0100 ;PUT ADdRess
0110 ;
0120 BUFADR = $15
0130 DBUFA = $0304
0140 ;
0150     *=  $C73A
0160 ;
0170     LDA DBUFA
0180     STA BUFADR
0190     LDA DBUFA+1
0200     STA BUFADR+1
0210     RTS
```

Po operacji formatowania dyskietki w buforze umieszczane są numery wadliwych sektorów, a na końcu wpisywane są dwa bajty $FF. Po wywołaniu procedury PUTADR, która przepisuje adres bufora do rejestru BUFADR (BUFfer ADdRess), poszukiwana jest końcowa sekwencja bajtów w buforze. Gdy zostanie ona znaleziona, to do rejestru DBYT zapisywana jest aktualna długość bufora.

W czasie, gdy powstawał system operacyjny Atari, dyskietki były jeszcze stosunkowo drogie. Stąd zastosowanie takiego sposobu formatowania, który umożliwia korzystanie nawet z uszkodzonych dyskietek. Ponieważ cena dyskietek znacznie się zmniejszyła, to żaden DOS nie wykorzystuje tej możliwości. Przydatność jej okazuje się jednak bardzo duża w przypadku korzystania z twardego dysku. Umożliwia bowiem korzystanie z uszkodzonego dysku, a jego cena jest aktualnie bardzo wysoka (wiosna 1988: twardy dysk 20 MB - około 650,- $).

Po odczycie statusu także wywoływana jest procedura PUTADR. Następnie trzeci bajt odczytanego statusu oznaczający maksymalny czas wykonywania operacji jest przepisywany do rejestru DSKTIM w celu jego uaktualnienia. Po tym DSKINT się kończy.

Rozdział 6

OBSŁUGA NOWYCH URZĄDZEŃ

System operacyjny Atari był projektowany tak, aby umożliwić maksymalną elastyczność. Oczywiste było, że powstaną w przyszłości nowe urządzenia. Z tego powodu tabela procedur obsługi urządzeń zewnętrznych została umieszczona w pamięci RAM i przewidziano możliwość jej rozszerzania do jedenastu wpisów. Projektanci Atari założyli ponadto, że powstaną w przyszłości urządzenia peryferyjne korzystające z szyny równoległej komputera i nazwali je "nowymi urządzeniami" (new device). Do ich obsługi przewidziano liczne procedury systemu operacyjnego oraz wiele rejestrów w obszarze zmiennych systemowych. Jedynym - na razie - takim urządzeniem jest twardy dysk Supra 20 MB.

Dla procedur nowych urządzeń przewidziana jest struktura tzw. listy liniowej. Polega ona na tym, że każdy element listy wskazuje na element następny. Dołączenie do środka listy nowego elementu (np. między elementy n i n+1) wymaga skopiowania wskaźnika z elementu n (wskazującego na n+1) do nowego elementu, a następnie ustawienia wskaźnika elementu n na nowy element.

Trzeba jeszcze powiedzieć, do czego służy ta struktura. Otóż zakłada się, że procedury obsługi nowych urządzeń będą składały się z elementów zorganizowanych w listę liniową. Kolejne elementy mogą zostać umieszczone w dowolnym miejscu pamięci, a ich następstwo jest ustalane właśnie przez listę liniową.

Chciałbym jeszcze przeprosić za ewentualne nieścisłości, które mogą znaleźć się w tym rozdziale. Niestety trudno jest pisać o współpracy komputera z urządzeniami, które są nieznane lub - co gorsza - jeszcze nie istnieją.

6.1. Instalowanie nowego urządzenia

Podczas startu systemu (RESET) wywoływana jest procedura LINKSOM, której zadaniem jest ustawienie elementów listy liniowej. Jej przebieg jest zupełnie odmienny przy starcie zimnym i gorącym. Dlatego najpierw sprawdzana jest zawartość wskaźnika WARMST i na tej podstawie wybierany jest odpowiedni wariant procedury.

```
0100 ;LINK SOMething
0110 ;
0120 CHCKFF = $CB56
0130 CHLINK = $03FB
0140 CKEY =  $03E9
0150 DCBINI = $E7BE
0160 DVSTAT = $02EA
0170 DVTMOT = $02EC
0180 INITLD = $E7DE
0190 LINKCD = $E89E
0200 LINKWM = $E894
```

```
0210 MEMLO = $02E7
0220 MEMTOP = $02E5
0230 REVNUM = $02ED
0240 TEMP1 = $0312
0250 TEMP2 = $0313
0260 WARMST = $08
0270 ZCHAIN = $4A
0280 ;
0290     *=  $E739
0300 ;
0310     LDA WARMST
0320     BEQ CDS
0330     LDA # <CKEY
0340     STA ZCHAIN
0350     LDA # >CKEY
0360     STA ZCHAIN+1
0370 NEXT LDY #$12
0380     CLC
0390     LDA (ZCHAIN),Y
0400     TAX
0410     INY
0420     ADC (ZCHAIN),Y
0430     BEQ EXIT
0440     LDA (ZCHAIN),Y
0450     STA ZCHAIN+1
0460     STX ZCHAIN
0470     JSR CHCKFF
0480     BNE EXIT
0490     JSR LINKWM
0500     BCS EXIT
0510     BCC NEXT
0520 CDS LDA #$00
0530     STA CHLINK
0540     STA CHLINK+1
0550     LDA #$4F
0560     BNE GDV
0570 END LDA #$00
0580     TAY
0590     JSR DCBINI
0600     BPL MEM
0610 EXIT RTS
0620 MEM CLC
0630     LDA MEMLO
0640     ADC DVSTAT
0650     STA TEMP1
0660     LDA MEMLO+1
0670     ADC DVSTAT+1
0680     STA TEMP2
0690     SEC
0700     LDA MEMTOP
0710     SBC TEMP1
0720     LDA MEMTOP+1
0730     SBC TEMP2
0740     BCS ADR
0750 LOOP LDA #$4E
0760 GDV TAY
0770     JSR DCBINI
0780     JMP END
0790 ADR LDA DVTMOT
0800     LDX MEMLO
```

```
0810     STX DVTMOT
0820     LDX MEMLO+1
0830     STX REVNUM
0840     JSR INITLD
0850     BMI LOOP
0860     SEC
0870     JSR LINKCD
0880     BCS LOOP
0890     BCC END
```

Przy zimnym starcie systemu zerowany jest rejestr CHAIN i, po umieszczeniu w akumulatorze i rejestrze Y wartości $4F, wywoływana jest procedura DCBINI. Ma ona stwierdzić obecność urządzenia i jego gotowość do pracy. Następnie przez kolejne wywołania DCBINI odczytywane są elementy procedury urządzenia. Czynność ta jest powtarzana, dopóki kolejne wywołanie DCBINI z wartością $00 w akumulatorze i rejestrze Y nie da wyniku negatywnego.

Sam proces odczytu jest skomplikowany. W wyniku wywołania DCBINI z wartościami $00 w rejestrze DVSTAT znajduje się wielkość elementu (liczba bajtów). Jest ona dodawana do dolnej granicy wolnej pamięci MEMLO i porównywana z górną granicą MEMTOP. Jeśli element nie mieści się w pamięci, to urządzenie jest o tym informowane przez wywołanie DCBINI z wartością $4E, a następnie pętla jest powtarzana.

Jeżeli rozmiar elementu pozwala na umieszczenie go w pamięci, to wywoływana jest procedura INITLD, która przygotowuje i wykonuje odczyt elementu. Błędny odczyt powoduje wywołanie DCBINI z wartością $4E i kolejne powtórzenie pętli, Poprawnie odczytany element jest dołączany do listy liniowej przez procedurę LINK.

Odmienny jest przebieg procedury LINKSOM przy gorącym starcie. Kolejno sprawdzane jest następstwo elementów listy, aż do napotkania wskaźnika zerowego, który oznacza koniec listy. Każdy element jest przy tym sprawdzany przez procedurę CHCKFF. Negatywny wynik powoduje opuszczenie procedury. Jeśli zaś wynik kontroli jest pozytywny, to procedura LINK umieszcza element na końcu listy. Nieprawidłowy przebieg LINK kończy procedurę, a w przeciwnym razie poszukiwany jest następny element.

Procedura DCBINI przepisuje do bloku DCB parametry z tabeli TSIOIN, a w rejestrach DAUX1 i DAUX2 umieszcza wartości przekazane jej odpowiednio w akumulatorze i rejestrze Y. Na końcu wykonywany jest bezpośredni skok do procedury SIOINT, która obsługuje transmisję. Wynikiem wykonanej operacji są cztery bajty umieszczone w rejestrze DVSTAT. Oznaczają one rozmiar elementu oraz adres dla SIO. Ostatni bajt jest niewykorzystany (może zawierać numer wersji).

```
0100 ;DCB INItiation
0110 ;
0120 DAUX1 = $030A
0130 DAUX2 = $030B
0140 DDEVIC = $0300
0150 JSIOINT = $E459
0160 TSIOIN = $E7D4
```

```
0170 ;
0180     *=  $E7BE
0190 ;
0200     PHA
0210     LDX #$09
0220 NEXT LDA TSIOIN,X
0230     STA DDEVIC,X
0240     DEX
0250     BPL NEXT
0260     STY DAUX2
0270     PLA
0280     STA DAUX1
0290     JMP JSIOINT
```

Wartości znajdujące się w tabeli TSIOIN oznaczają parametry dla DCB w kolejności: DDEVIC ($4F), DUNIT ($01), DCMND ($40 - GET DATA), DSTAT ($40 - odczyt), DBUFA ($02EA - DVSTAT), DTIMLO ($1E) i DBYT ($04).

```
0100 ;Table SIO INitiation
0110 ;
0120     *=  $E7D4
0130 ;
0140     .BYTE $4F,$01,$40,$40,$EA
0150     .BYTE $02,$1E,$00,$04,$00
```

Procedurą kontrolującą poprawność elementu jest CHCKFF. Oblicza ona sumę 18 bajtów poczynając od bajtu wskazanego przez rejestr ZCHAIN. Suma ta powinna mieć wartość $FF, w akumulatorze jest wtedy $00. Jeśli jest inaczej to akumulator zawiera różnicę między obliczoną wartością a $FF.

```
0100 ;CHeCK for $FF
0110 ;
0120 ZCHAIN = $4A
0130 ;
0140     *=  $CB56
0150 ;
0160     LDY #$11
0170     LDA #$00
0180     CLC
0190 LOOP ADC (ZCHAIN),Y
0200     DEY
0210     BPL LOOP
0220     ADC #$00
0230     EOR #$FF
0240     RTS
```

6.1.1. Odczyt elementu

W celu odczytania elementu struktury wywoływana jest procedura INITLD. Jej zadaniem jest przygotowanie transmisji.

```
0100 ;INITiation LoaDer
0110 ;
0120 DVTMOT = $02EC
0130 GBYTEA = $02CF
0140 GETBYT = $E816
```

```
0150 LOADAD = $02D1
0160 LOADER = $C745
0170 TEMP1 = $312
0180 TEMP2 = $313
0190 TEMP3 = $315
0200 ZLOADA = $02D3
0210 ;
0220     *=  $E7DE
0230 ;
0240     STA TEMP2
0250     LDX #$00
0260     STX TEMP1
0270     DEX
0280     STX TEMP3
0290     LDA DVTMOT
0300     ROR A
0310     BCC LDH
0320     INC DVTMOT
0330     BNE LDH
0340     INC DVTMOT+1
0350 LDH LDA DVTMOT
0360     STA LOADAD
0370     LDA DVTMOT+1
0380     STA LOADAD+1
0390     LDA # <GETBYT
0400     STA GBYTEA
0410     LDA # >GETBYT
0420     STA GBYTEA+1
0430     LDA #$80
0440     STA ZLOADA
0450     JMP LOADER
```

Na początku zawartość akumulatora jest zapisywana do pomocniczego rejestru TEMP2, a rejestry TEMP1 i TEMP3 otrzymują odpowiednio wartości $00 i $FF. Obliczony uprzednio adres wczytywania jest przenoszony z DVTMOT do rejestru LOADAD (LOAD ADdress), a wektor GBYTEA (Get BYTE Address) jest ustawiany na adres procedury GETBYT. Następnie znacznik ZLOADA otrzymuje wartość $80 i wykonywany jest bezpośredni skok do procedury LOADER.

```
0100 ;LOADER routine
0110 ;
0120 GBYTEA = $02CF
0130 HIBYTE = $0288
0140 LCOUNT = $0233
0150 LOADAD = $02D1
0160 NEWVEC = $C8E4
0170 RECLEN = $0245
0180 RUNADR = $02C9
0190 ;
0200     *=  C745
0210 ;
0220     LDX #$05
0230 NEXT LDA #$00
0240     STA RUNADR,X
0250     DEX
0260     BPL NEXT
0270 LOOP1 LDA #$00
0280     STA LCOUNT
0290     JSR GBYTAC
```

```
0300     LDY #$9C
0310     BCS EXIT
0320     STA HIBYTE
0330     JSR GBYTAC
0340     LDY #$9C
0350     BCS EXIT
0360     STA RECLEN
0370     LDA HIBYTE
0380     CMP #$0B
0390     BEQ HENDRT
0400     ROL A
0410     TAX
0420     LDA NEWVEC,X
0430     STA RUNADR
0440     LDA NEWVEC+1,X
0450     STA RUNADR+1
0460 LOOP2 LDA RECLEN
0470     CMP LCOUNT
0480     BEQ LOOP1
0490     JSR GBYTAC
0500     LDY #$9C
0510     BCS EXIT
0520     JSR RUNADC
0530     INC LCOUNT
0540     BNE LOOP2
0550 EXIT RTS
0560 ;
0570 ;Handle END Record Type
0580 ;
0590 HENDRT JSR GBYTAC
0600     LDY #$9C
0610     BCS END
0620     STA RUNADR
0630     JSR GBYTAC
0640     LDY #$9C
0650     BCS END
0660     STA RUNADR+1
0670     LDA RECLEN
0680     CMP #$01
0690     BEQ SUC
0700     BCC ERR
0710     CLC
0720     LDA RUNADR
0730     ADC LOADAD
0740     TAY
0750     LDA RUNADR+1
0760     ADC LOADAD+1
0770 SET STY RUNADR
0780     STA RUNADR+1
0790 SUC LDY #$01
0800 END RTS
0810 ERR LDY #$00
0820     LDA #$00
0830     BEQ SET
0840 GBYTAC JMP (GBYTEA)
0850 RUNADC JMP (RUNADR)
```

Na początku zerowane jest sześć bajtów poczynając od adresu RUNADR (są to rejestry RUNADR, HIUSED i ZHIUSE). Po ustawieniu licznika LCOUNT (Loader COUNTer) na zero

rozpoczyna się pętla odczytu. Najpierw dwukrotnie wywoływana jest procedura GETBYT, a odczytane przez nią wartości są umieszczane kolejno w rejestrach HIBYTE (HIgh BYTE) i RECLEN (RECord LENgth). Zasygnalizowanie błędu po zakończeniu GETBYT zawsze powoduje umieszczenie w rejestrze Y kodu błędu $9C (żadne ze źródeł nie podaje znaczenia tego kodu) i przerwanie procedury LOADER.

Przy pierwszym wywołaniu procedury GETBYT z LOADER rejestr TEMP3 zawiera wartość $FF. Powoduje to wpisanie do TEMP3 wartości $80 i wywołanie procedury GTNXBL i odczyt bloku 128 bajtów. Pierwszy bajt bloku jest umieszczany w akumulatorze i przekazywany do procedury LOADER. Każde następne wywołanie GETBYT powoduje zwrócenie kolejnego bajtu bloku, aż do jego wyczerpania. Wtedy znów wywoływana jest procedura GTNXBL.

Pierwszy bajt bloku (przepisany do HIBYTE) służy jako indeks tabeli wektorów NEWVEC, natomiast drugi (RECLEN) określa długość odczytanego bloku. Jeżeli RECLEN jest równy licznikowi LCOUNT, to pętla jest powtarzana. Jej opuszczenie następuje po odczytaniu wartości HIBYTE równej $0B. Następuje wtedy przejście do etykiety HENDRT.

Przy RECLEN różnym od LCOUNT uruchamiana jest pętla wewnętrzna, która kończy się po zrównaniu tych wartości. Podczas niej kolejne bajty odczytane przez GETBYT są przekazywane do procedury, której adres znajduje się w rejestrze RUNADR. Po jej zakończeniu licznik LCOUNT jest zwiększany i pętla się powtarza.

```
0100 ;GET BYTe
0110 ;
0120 CSCB  = $03FD
0130 GTNXBL = $E833
0140 TEMP3 = $0315
0150 ;
0160     *=  $E816
0170 ;
0180     LDX TEMP3
0190     INX
0200     STX TEMP3
0210     BEQ GPL
0220 NEXT LDX TEMP3
0230     LDA (CSCB-$80),X
0240     CLC
0250     RTS
0260 GPL LDA #$80
0270     STA TEMP3
0280     JSR GTNXBL
0290     BPL NEXT
0300     SEC
0310     RTS
```

Po przerwaniu pętli procedura GETBYT jest wywoływana jeszcze dwa razy. Odczytane bajty umieszczane są w rejestrze RUNADR. Gdy RECLEN ma wartość 1, procedura się kończy, a gdy 0 - to zerowany jest RUNADR. W innym przypadku do zawartości RUNADR dodawana jest jeszcze zawartość LOADAD.

```
0100 ;GeT NeXt BLock
0110 ;
0120 DAUX1 = $030A
0130 DDEVIC = $0300
0140 JSIOINT = $E459
0150 SIOTAB = $E851
0160 TEMP1 = $0312
0170 TEMP2 = $0313
0180 ;
0190     *=  $E833
0200 ;
0210     LDX #$0B
0220 NEXT LDA SIOTAB,X
0230     STA DDEVIC,X
0240     DEX
0250     BPL NEXT
0260     LDX TEMP1
0270     STX DAUX1
0280     INX
0290     STX TEMP1
0300     LDA TEMP2
0310     STA DDEVIC
0320     JMP JSIOINT
```

Procedura GTNXBL wywoływana przez GETBYT przepisuje do DCB parametry transmisji z tabeli SIOTAB. Następnie do rejestru DAUX1 określającego numer rekordu wpisuje wartość pobraną z TEMP1, który z kolei jest zwiększany. Na końcu z rejestru TEMP2 przepisuje kod urządzenia umieszczony tam przez procedurę INITLD i wykonuje skok do SIOINT.

```
0100 ;SIO TABle
0110 ;
0120     *=  $E851
0130 ;
0140     .BYTE $00,$01,$26,$40
0150     .BYTE $FD,$03,$1E,$00
0160     .BYTE $80,$00,$00,$00
```

Wartości znajdujące się w tabeli SIOTAB oznaczają parametry dla DCB w kolejności: DDEVIC ($00), DUNIT ($01), DCMND ($26 - SEND HANDLER), DSTAT ($40 - odczyt), DBUFA ($03FD - CSCB), DTIMLO ($1E), DBYT ($80 - 128 bajtów) oraz DAUX1 i DAUX2 ($00).

6.1.2. Przetwarzanie elementu

Każdy bajt odczytany przez GETBYT jest przekazywany do procedury wskazanej wektorem RUNADR. Wektor ten jest pobierany przez procedurę LOADER z tabeli adresowej NEWVEC według wartości HIBYTE. Warto zwrócić uwagę na fakt, że układ tej tabeli jest bardzo zbliżony do układu tabeli COMTAB, która jest wykorzystywana przez CIO (zob. rozdział 2.1. - str. 15).

```
0100 ;NEW device VECtor
0110 ;
0120 ADD28E = $C86D
0130 ADDGET = $C8B5
0140 ADDWRD = $C892
0150 HENDRT = $C795
0160 PUTCHR = $C7D5
```

```
0170 ;
0180     *=  $C8E4
0190 ;
0200     .WORD PUTCHR
0210     .WORD PUTCHR
0220     .WORD ADDWRD
0230     .WORD ADDWRD
0240     .WORD ADDWRD
0250     .WORD ADDWRD
0260     .WORD ADD28E
0270     .WORD ADD28E
0280     .WORD ADDGET
0290     .WORD ADDGET
0300     .WORD PUTCHR
0310     .WORD HENDRT
```

Teraz zostaną kolejno opisane procedury umieszczone w powyższej tabeli. Trzeba przy tym pamiętać, że w każdej z nich daną wejściową jest bajt odczytany przez GETBYT i umieszczony w akumulatorze.

Procedura PUTCHR jest wywoływana, gdy rejestr HIBYTE ma wartość $00, $01 lub $0A. Jej przebieg jest zależny od stanu licznika LCOUNT. Gdy jest on równy zero, to zawartość akumulatora jest tylko zapisywana do młodszego bajtu rejestrów RELADR i NEWADR, a procedura się kończy. Gdy LCOUNT jest równy 1, to bajt z akumulatora jest umieszczany w starszym bajcie tych rejestrów i następuje przejście do drugiej fazy PUTCHR.

```
0100 ;PUT CHaRacter
0110 ;
0120 HIBYTE = $0288
0130 HIUSED = $02CB
0140 LCOUNT = $0233
0150 LOADAD = $02D1
0160 LTEMP = $36
0170 MEMTOP = $02E5
0180 NEWADR = $028E
0190 RECLEN = $0245
0200 RELADR = $024A
0210 ;
0220     *=  $C7D5
0230 ;
0240     LDY LCOUNT
0250     CPY #$01
0260     BEQ ADD
0270     BCS RELTXT
0280     STA RELADR
0290     STA NEWADR
0300     BCC EXIT
0310 ADD STA RELADR+1
0320     STA NEWADR+1
0330     LDX #$00
0340     LDA HIBYTE
0350     BEQ ADL
0360     CMP #$0A
0370     BEQ ADN
0380     LDX #$02
0390 ADL CLC
```

```
0400     LDA RELADR
0410     ADC LOADAD,X
0420     STA NEWADR
0430     LDA RELADR+1
0440     ADC LOADAD+1,X
0450     STA NEWADR+1
0460 ADN CLC
0470     LDA NEWADR
0480     ADC RECLEN
0490     PHA
0500     LDA #$00
0510     ADC NEWADR+1
0520     TAY
0530     PLA
0540     SEC
0550     SBC #$02
0560     BCS BPS
0570     DEY
0580 BPS PHA
0590     TYA
0600     CMP HIUSED+1,X
0610     PLA
0620     BCC LOD
0630     BNE SAV
0640     CMP HIUSED,X
0650     BCC LOD
0660 SAV STA HIUSED,X
0670     PHA
0680     TYA
0690     STA HIUSED+1,X
0700     PLA
0710 LOD LDX HIBYTE
0720     CPX #$01
0730     BEQ EXIT
0740     CPY MEMTOP+1
0750     BCC EXIT
0760     BNE ERR
0770     CMP MEMTOP
0780     BCC EXIT
0790 ERR PLA
0800     PLA
0810     LDY #$9D
0820 EXIT RTS
0830 ;
0840 ;RELocate TeXT in memory
0850 ;
0860 RELTXT SEC
0870     PHA
0880     LDA LCOUNT
0890     SBC #$02
0900     CLC
0910     ADC NEWADR
0920     STA LTEMP
0930     LDA #$00
0940     ADC NEWADR+1
0950     STA LTEMP+1
0960     PLA
0970     LDY #$00
0980     STA (LTEMP),Y
0990     JMP EXIT
```

Dalsze działanie zależy od wartości HIBYTE. Przy HIBYTE równym zero do RELADR jest dodawane LOADAD, a przy równym jeden ZLOADA i rezultat jest umieszczany w NEWADR. Wartość $0A w HIBYTE powoduje ominięcie tego etapu. Następnie zawartość NEWADR jest zwiększana o długość rekordu RECLEN i zmniejszana o 2. Uzyskany wynik jest porównywany z zawartością rejestru HIUSED (HIgh USED address) i gdy jest większy, to zostaje do niego przepisany. Na końcu wykonywane jest jeszcze porównanie otrzymanego adresu z wartością MEMTOP. Jeżeli adres jest większy, to w rejestrze Y sygnalizowany jest błąd $9D. Wystąpienie błędu każdorazowo powoduje zdjęcie ze stosu dwóch bajtów i powrót z procedury PUTCHR do miejsca wywołania INITLD.

Zupełnie odmienny jest przebieg procedury, gdy licznik LCOUNT ma wartość większą od 1. Do adresu zawartego w NEWADR dodawany jest stan LCOUNT zmniejszony o 2 i uzyskany wynik jest wpisywany do rejestru przejściowego LTEMP (Loader TEMPorary register). Przekazany w akumulatorze znak jest teraz umieszczany pod adresem wskazywanym przez LTEMP i procedura się kończy.

Wartości HIBYTE z zakresu od $02 do $05 powodują wywołanie procedury ADDWRD. Najpierw dodawane są zawartości akumulatora i NEWADR, a wynik umieszczany jest w LTEMP. Następnie do bajtu wskazywanego przez LTEMP dodawana jest zawartość rejestru LOADAD (dla HIBYTE mniejszego od 4) lub ZLOADA (dla HIBYTE większego od 3).

```
0100 ;ADDition for Write/ReaD
0110 ;
0120 HIBYTE = $0288
0130 LOADAD = $02D1
0140 LTEMP = $36
0150 NEWADR = $028E
0160 ;
0170     *=  $C892
0180 ;
0190     LDX #$00
0200     LDY HIBYTE
0210     CPY #$04
0220     BCC BPS
0230     LDX #$02
0240 BPS CLC
0250     ADC NEWADR
0260     STA LTEMP
0270     LDA #$00
0280     ADC NEWADR+1
0290     STA LTEMP+1
0300     LDY #$00
0310     LDA (LTEMP),Y
0320     CLC
0330     ADC LOADAD,X
0340     STA (LTEMP),Y
0350     RTS
```

Przy HIBYTE równym $06 lub $07 wywoływana jest procedura ADD28E. Wykonywane jest przez nią także dodawanie zawartości LOADAD według LTEMP, jak w ADDWRD, lecz teraz operacja dotyczy dwóch bajtów: wskazywanego przez wektor LTEMP i następnego.

```
0100 ;ADDition $028E
0110 ;
0120 LOADAD = $02D1
0130 LTEMP = $36
0140 NEWADR = $028E
0150 ;
0160     *=  $C86D
0170 ;
0180     CLC
0190     ADC NEWADR
0200     STA LTEMP
0210     LDA #$00
0220     ADC NEWADR+1
0230     STA LTEMP+1
0240     LDY #$00
0250     LDA (LTEMP),Y
0260     CLC
0270     ADC LOADAD
0280     STA (LTEMP),Y
0290     INC LTEMP
0300     BNE BPS
0310     INC LTEMP+1
0320 BPS LDA (LTEMP),Y
0330     ADC LOADAD+1
0340     STA (LTEMP),Y
0350     RTS
```

Przebieg procedury ADDGET wywoływanej dla zawartości HIBYTE równej $08 lub $09 jest zależny od parzystości licznika LCOUNT, co jest rozpoznawane według bitu 0.

```
0100 ;ADDition for GET
0110 ;
0120 HIBYTE = $0288
0130 LCOUNT = $0233
0140 LOADAD = $02D1
0150 LTEMP = $36
0160 NEWADR = $028E
0170 ;
0180     *=  $C8B5
0190 ;
0200     PHA
0210     LDA LCOUNT
0220     ROR A
0230     PLA
0240     BCS ADD
0250     CLC
0260     ADC NEWADR
0270     STA LTEMP
0280     LDA #$00
0290     ADC NEWADR+1
0300     STA LTEMP+1
0310     LDY #$00
0320     LDA (LTEMP),Y
0330     STA HIBYTE
```

```
0340 EXIT RTS
0350 ADD CLC
0360     ADC LOADAD
0370     LDA #$00
0380     ADC LOADAD+1
0390     ADC HIBYTE
0400     LDY #$00
0410     STA (LTEMP),Y
0420     BEQ EXIT
```

Przy parzystej wartości LCOUNT (bit 0 skasowany) zawartość akumulatora jest dodawana do NEWADR, a uzyskany wynik jest traktowany jako wektor. Bajt wskazany przez ten wektor jest odczytywany i umieszczany w rejestrze HIBYTE.

Nieparzysta wartość LCOUNT (bit 0 ustawiony) powoduje zsumowanie zawartości akumulatora oraz rejestrów LOADAD i HIBYTE. Uzyskany rezultat jest umieszczany w komórce wskazanej wektorem LTEMP.

6.1.3. Dołączenie elementu do listy

Każdy odczytany element struktury musi zostać umieszczony we właściwym miejscu listy liniowej. Operacja ta jest przeprowadzana przez procedurę LINK. Procedura ta posiada trzy punkty początkowe. Jeden z nich (LINKCD) jest wykorzystywany przy zimnym starcie systemu, drugi (LINKWM) przy starcie gorącym, a trzeci (LINK) jest normalnym adresem wywołania procedury.

```
0100 ;LINK routine
0110 ;
0120 CALVEC = $E900
0130 CHCKFF = $CB56
0140 DVTMOT = $02EC
0150 MEMLO = $02E7
0160 SRCHLS = $E85D
0170 TEMP1 = $0312
0180 TEMP2 = $0313
0190 UNLINK = $E915
0200 ZCHAIN = $4A
0210 ;
0220     *=  $E894
0230 ;
0240 ;LINK WarM start
0250 LINKWM SEC
0260     PHP
0270     BCS INIHND
0280 ;LINK
0290 LINK STA DVTMOT+1
0300     STY DVTMOT
0310 ;
0320 ;LINK ColD start
0330 LINKCD PHP
0340     LDA #$00
0350     TAY
0360     JSR SRCHLS
0370     BCS ERR
0380     LDY #$12
0390     LDA DVTMOT
```

```
0400     STA (ZCHAIN),Y
0410     TAX
0420     INY
0430     LDA DVMOT+1
0440     STA (ZCHAIN),Y
0450     STX ZCHAIN
0460     STA ZCHAIN+1
0470     LDA #$00
0480     STA (ZCHAIN),Y
0490     DEY
0500     STA (ZCHAIN),Y
0510 ;INItialize HaNDler
0520 INIHND JSR CALVEC
0530     BCC SUCC
0540     LDA DVTMOT+1
0550     LDY DVTMOT
0560     JSR UNLINK
0570 ERR PLP
0580     SEC
0590     RTS
0600 SUCC PLP
0610     BCS UPDMLO
0620     LDA #$00
0630     LDY #$10
0640     STA (ZCHAIN),Y
0650     INY
0660     STA (ZCHAIN),Y
0670 UPDMLO CLC
0680     LDY #$10
0690     LDA MEMLO
0700     ADC (ZCHAIN),Y
0710     STA MEMLO
0720     INY
0730     LDA MEMLO+1
0740     ADC (ZCHAIN),Y
0750     STA MEMLO+1
0760     LDY #$0F
0770     LDA #$00
0780     STA (ZCHAIN),Y
0790     JSR CHCKFF
0800     LDY #$0F
0810     STA (ZCHAIN),Y
0820     CLC
0830     RTS
```

Rozpoczęcie procedury od LINKWM powoduje opuszczenie całej pierwszej fazy procedury. Po ustawieniu bitu Carry i zapamiętaniu na stosie rejestru statusu procesora wykonywany jest skok do drugiej części oznaczonej etykietą INIHND.

Po rozpoczęciu od etykiety LINKCD akumulator i rejestr Y są zerowane i wywoływana jest procedura SRCHLS. Wyszukuje ona ostatni element struktury i umieszcza jego adres w rejestrze ZCHAIN. Gdy taki element nie został znaleziony, to po odtworzeniu ze stosu rejestru statusu i ustawieniu bitu Carry procedura LINK jest przerywana. Następnie wskaźnik tego elementu jest ustawiany tak, aby wskazywał na nowy element. Adres nowego elementu jest umieszczany w rejestrze ZCHAIN, a jego wskaźnik jest zerowany.

Teraz element jest poprzez wywołanie procedury CALVEC inicjowany. W przypadku niepowodzenia ponownie pobierany jest jego adres z DVTMOT i wywoływana jest procedura UNLINK, która usuwa element z listy. Dzięki temu lista zawiera tylko prawidłowo zainicjowane struktury.

Potem odtwarzany jest ze stosu rejestr statusu procesora. Jeżeli ma on skasowany bit Carry, to zerowane są bajty 16 i 17 struktury wskazywanej przez wektor ZCHAIN. Oznacza to, że nowy element mieści się poniżej dolnej granicy wolnej pamięci (MEMLO). Następnie zawartości bajtów 16 i 17 są dodawane do wartości MEMLO, aby zabezpieczyć nowy element przed ingerencją programu użytkownika.

Na zakończenie wywoływana jest procedura CHCKFF, która oblicza sumę kontrolną i zwraca wynik odjęcia jej od $FF. Wynik ten jest umieszczany w 15 bajcie elementu. Teraz po wywołaniu CHCKFF bit Carry będzie skasowany, a w akumulatorze znajdzie się zero.

```
0100 ;SeaRCH LiSt
0110 ;
0120 CHCKFF = $CB56
0130 CKEY =  $03E9
0140 TEMP1 = $0312
0150 TEMP2 = $0313
0160 ZCHAIN = $4A
0170 ;
0180     *=  $E85D
0190 ;
0200     STY TEMP1
0210     STA TEMP2
0220     LDA # <CKEY
0230     STA ZCHAIN
0240     LDA # >CKEY
0250     STA ZCHAIN+1
0260 LOOP LDY #$12
0270     LDA (ZCHAIN),Y
0280     TAX
0290     INY
0300     LDA (ZCHAIN),Y
0310     CMP TEMP2
0320     BNE FIND
0330     CPX TEMP1
0340     BNE FIND
0350     CLC
0360     RTS
0370 FIND CMP #$00
0380     BNE FOUND
0390     CPX #$00
0400     BNE FOUND
0410 ERR SEC
0420     RTS
0430 FOUND STX ZCHAIN
0440     STA ZCHAIN+1
0450     JSR CHCKFF
0460     BNE ERR
0470     BEQ LOOP
```

Procedura SRCHLS przeszukuje listę liniową w celu odnalezienia elementu, którego wskaźnik ma parametry umieszczone przed wywołaniem SRCHLS w akumulatorze i rejestrze Y. Przekazane wartości są najpierw zapamiętywane w rejestrach TEMP1 i TEMP2. Następnie sprawdzany jest wskaźnik elementu. Gdy jest to poszukiwany element, procedura kończy się ze skasowanym bitem Carry.

W przeciwnym przypadku sprawdza się, czy jest to ostatni element listy (wskaźnik równy zero). Jeśli tak, to procedura kończy się z ustawionym bitem Carry. Jeżeli nie jest to jeszcze koniec listy, to po przepisaniu adresu do rejestru ZCHAIN wywoływana jest procedura CHCKFF. Pozytywny wynik sprawdzenia sumy kontrolnej powoduje przejście do początku pętli i badanie następnego elementu. Błąd kończy procedurę SRCHLS z ustawionym bitem Carry.

```
0100 ;CALl VECtor
0110 ;
0120 TEMP1 = $0312
0130 TEMP2 = $0313
0140 ZCHAIN = $4A
0150 ;
0160     *=  $E900
0170 ;
0180     CLC
0190     LDA ZCHAIN
0200     ADC #$0C
0210     STA TEMP1
0220     LDA ZCHAIN+1
0230     ADC #$00
0240     STA TEMP2
0250     JMP (TEMP1)
```

Inicjowanie elementu struktury odbywa się poprzez procedurę CALVEC. Odczytuje ona 12 i 13 bajt elementu oraz umieszcza je w rejestrach TEMP1 i TEMP2. Następnie według tego wektora wykonywany jest skok do procedury inicjującej. Wyraźnie widać tu analogię z tabelami adresowymi CIO, w których od dwunastego bajtu znajduje się instrukcja skoku do procedury inicjowania urządzenia.

Gdy zachodzi konieczność usunięcia elementu z listy liniowej (np. po niepoprawnej inicjalizacji), to wywoływana jest procedura UNLINK. Adres elementu do usunięcia jest przekazywany do niej w akumulatorze i rejestrze Y. Z tymi wartościami wywoływana jest najpierw procedura SRCHLS, która wyszukuje element do usunięcia.

Teraz wskaźnik elementu poprzedzającego jest ustawiany na element następny i w ten sposób zbędny element jest już usunięty z listy. Dokładniej metodę wykonania tej operacji można prześledzić na wydruku procedury. Jeżeli usunięcie elementu odbyło się poprawnie, to procedura kończy się ze skasowanym bitem Carry, w innym wypadku Carry jest ustawiany.

```
0100 ;UNLINK routine
0110 ;
0120 CHCKFF = $CB56
0130 COLDST = $0244
```

```
0140 SRCHLS = $E85D
0150 ZCHAIN = $4A
0160 ;
0170 ;
0180     *=  $E915
0190 ;
0200     JSR SRCHLS
0210     BCS EXIT
0220     TAY
0230     LDA ZCHAIN
0240     PHA
0250     LDA ZCHAIN+1
0260     PHA
0270     STX ZCHAIN
0280     STY ZCHAIN+1
0290     LDA COLDST
0300     BNE COLD
0310     LDY #$10
0320     CLC
0330     LDA (ZCHAIN),Y
0340     INY
0350     ADC (ZCHAIN),Y
0360     BNE END
0370     JSR CHCKFF
0380     BNE END
0390 COLD LDY #$12
0400     LDA (ZCHAIN),Y
0410     TAX
0420     INY
0430     LDA (ZCHAIN),Y
0440     TAY
0450     PLA
0460     STA ZCHAIN+1
0470     PLA
0480     STA ZCHAIN
0490     TYA
0500     LDY #$13
0510     STA (ZCHAIN),Y
0520     DEY
0530     TXA
0540     STA (ZCHAIN),Y
0550     CLC
0560     RTS
0570 END PLA
0580     PLA
0590 EXIT SEC
0600     RTS
```

6.2. Komunikacja z nowym urządzeniem

Pierwszą operacją konieczną do współpracy z urządzeniem przez CIO jest otwarcie kanału IOCB dla transmisji. Gdy otwierająca kanał procedura CIOOPN stwierdzi w rejestrze HNDLOD wartość różną od zera lub nie znajdzie wpisu urządzenia w tabeli HATABS, to wywoływana jest procedura SPCHND przygotowująca komunikację z nowym urządzeniem.

```
0100 ;SPeCial HaNDler
0110 ;
0120 DCBINI = $E7BE
```

```
0130 DVSTAT = $02EA
0140 ICAX3 = $034C
0150 ICAX4 = $034D
0160 ICAX5Z = $2E
0170 ICBAZ = $24
0180 ICDNOZ = $21
0190 ICHIDZ = $20
0200 ICPTZ = $26
0210 PUTBYT = $EF26
0220 ;
0230     *=  $EEF9
0240 ;
0250     LDY #$00
0260     LDA (ICBAZ),Y
0270     LDY ICDNOZ
0280     JSR DCBINI
0290     BPL NERR
0300     LDY #$82
0310     RTS
0320 NERR LDA #$7F
0330     STA ICHIDZ
0340     LDA # <[PUTBYT-1]
0350     STA ICPTZ
0360     LDA # >[PUTBYT-1]
0370     STA ICPTZ+1
0380     LDA DVSTAT+2
0390     LDX ICAX5Z
0400     STA ICAX4,X
0410     LDY #$00
0420     LDA (ICBAZ),Y
0430     STA ICAX3,X
0440     LDY #$01
0450     RTS
```

Najpierw wywołuje ona procedurę DCBINI, która sprawdza istnienie urządzenia. Jej negatywny wynik powoduje wpisanie do rejestru Y kodu błędu $82 i przerwanie procedury SPCHND. Po otrzymaniu od urządzenia odpowiedzi, indeks w HATABS (ICHIDZ) jest ustawiany na wartość $7F, która sygnalizuje, że jest to nowe urządzenie. Jako wektor procedury obsługi wpisywany jest adres procedury PUTBYT. Następnie do ICAX4 przepisywany jest adres z DVSTAT+2, a do ICAX3 właściwa nazwa urządzenia z bufora.

Każde następne żądanie komunikacji z nowym urządzeniem musi być poprzedzone wpisaniem do rejestru HNDLOD wartości różnej od zera (rejestr ICHIDZ zawiera $7F). Po stwierdzeniu takich wartości CIOMAIN wywołuje procedurę PRPLNK, która przygotowuje urządzenie do transmisji.

```
0100 ;PRePare LiNK
0110 ;
0120 DFVHND = $E716
0130 ICAX3 = $034C
0140 ICAX4 = $034D
0150 ICAX5Z = $2E
0160 ICCHID = $0340
0170 ICCOMT = $17
0180 ICHIDZ = $20
0190 INIOPN = $E55C
0200 INITLD = $E7DE
0210 LINKCD = $E89E
```

```
0220 NEXDER = $E510
0230 ;
0240     *=  $CA29
0250 ;
0260     LDX ICAX5Z
0270     LDA ICAX4,X
0280     JSR INITLD
0290     BCS EXIT
0300     CLC
0310     JSR LINKCD
0320     BCS EXIT
0330     LDX ICAX5Z
0340     LDA ICAX3,X
0350     JSR FDVHND
0360     BCS EXIT
0370     LDX ICAX5Z
0380     STA ICCHID,X
0390     STA ICHIDZ
0400     LDA #$03
0410     STA ICCOMT
0420     JMP INIOPN
0430 EXIT JMP NEXDER
```

Najpierw do akumulatora przenoszony jest bajt z ICAX4 i wywoływana jest procedura INITLD, która inicjuje odpowiedni element. Następnie procedura LINK umieszcza ten element w liście liniowej. Teraz z rejestru ICAX3 jest pobierany właściwy kod urządzenia i procedura FDVHND (część DEVNUM) odszukuje w tabeli HATABS wektor tabeli adresowej tego urządzenia. Znaleziony indeks w HATABS jest przepisywany do rejestrów ICCHID i ICHIDZ. Wykrycie błędu podczas PRPLNK powoduje skok do procedury CIOMAIN, która sygnalizuje błąd NON EXISTENT DEVICE (urządzenie nie istnieje). W innym razie, po ustaleniu kodu operacji na $03 (OPEN), procedura kończy się skokiem do INIOPN (wewnątrz procedury CIOOPN).

Po otwarciu kanału IOCB dla nowego urządzenia w rejestrze ICPTZ znajduje się adres procedury PUTBYT. Spełnia ona zadania zbliżone do CIO i przy porównaniu obu tych procedur bardzo łatwo wykryć podobieństwa.

Na początku sprawdzana jest poprawność numeru IOCB (identycznie jak w CIOMAIN) oraz wartość znacznika HNDLOD. Pozytywny wynik powoduje przepisanie odpowiedniego IOCB do ZIOCB i wywołanie procedury PRPLNK. W przeciwnym przypadku do rejestru Y wpisywany jest kod błędu i procedura się kończy.

```
0100 ;PUT BYTe routine
0110 ;
0120 HNDLOD = $02E9
0130 ICAX5Z = $2E
0140 ICCHID = $0340
0150 ICHIDZ = $20
0160 ICPTZ = $26
0170 PRPLNK = $CA29
0180 ;
0190     *=  $EF26
0200 ;
0210     PHA
```

```
0220     TXA
0230     PHA
0240     AND #$0F
0250     BNE BIN
0260     CPX #$80
0270     BPL BIN
0280     LDA HNDLOD
0290     BNE SUC
0300     LDY #$82
0310 ERR PLA
0320     PLA
0330     CPY #$00
0340     RTS
0350 BIN LDY #$86
0360     BMI ERR
0370 SUC STX ICAX5Z
0380     LDY #$00
0390 NEXT LDA ICCHID,X
0400     STA ICHIDZ,Y
0410     INX
0420     INY
0430     CPY #$0C
0440     BMI NEXT
0450     JSR PRPLNK
0460     BMI ERR
0470     PLA
0480     TAX
0490     PLA
0500     TAY
0510     LDA ICPTZ+1
0520     PHA
0530     LDA ICPTZ
0540     PHA
0550     TYA
0560     LDY #$92
0570     RTS
```

Po poprawnym zakończeniu PRPLNK wektor z rejestru ICPTZ jest przepisywany na stos. Po wykonaniu rozkazu RTS program jest kontynuowany od umieszczonego na stosie adresu. W CIO analogiczną operację wykonują procedury GOHAND i CIOJMP.

System operacyjny Atari zawiera jeszcze tabelę adresową, która może być wykorzystana przez nowe urządzenie. Ma ona strukturę wymaganą przez CIO i może być wykorzystana po wpisaniu jej adresu do HATABS.

```
0100 ;CALl TABle
0110 ;
0120 NEWINIT = $C90C
0130 NWDVC1 = $C991
0140 NWDVC3 = $C996
0150 NWDVC5 = $C99B
0160 NWDVC7 = $C9A0
0170 NWDVC9 = $C9A5
0180 NWDVCB = $C9AA
0190 ;
0200     *=  $E48F
0210 ;
```

```
0220     .WORD NWDVC1-1
0230     .WORD NWDVC3-1
0240     .WORD NWDVC5-1
0250     .WORD NWDVC7-1
0260     .WORD NWDVC9-1
0270     .WORD NWDVCA-1
0280     JMP NEWINIT
0290     .BYTE $00
```

Zawarte w tej tabeli adresy wskazują na tabelę wywołań procedur. Wszystkie operacje są rozpoczynane przez procedurę CHKNWP, a ich rodzaj jest rozpoznawany według zawartości rejestru Y.

```
0100 ;NEW DeVice Call
0110 ;
0120 CHKNWP = $C9DC
0130 ;
0140     *=  $C991
0150 ;
0160 NWDVC1 LDY #$01
0170     JMP CHKNWP
0180 NWDVC3 LDY #$03
0190     JMP CHKNWP
0200 NWDVC5 LDY #$05
0210     JMP CHKNWP
0220 NWDVC7 LDY #$07
0230     JMP CHKNWP
0240 NWDVC9 LDY #$09
0250     JMP CHKNWP
0260 NWDVCB LDY #$0B
0270     JMP CHKNWP
```

Procedura ta odszukuje poprzez wywołanie GETLOW aktywne urządzenie o najmniejszym numerze i przekazuje mu sterowanie poprzez wywołanie NEWPER. Czynności te powtarzane są w pętli, aż do obsłużenia wszystkich urządzeń. Ponadto znaczną część procedury zajmują operacje przechowania niezbędnych parametrów i ich późniejszego odtworzenia.

```
0100 ;CHecK NeW device Port
0110 ;
0120 CRITIC = $42
0130 GETLOW = $C9AF
0140 NEWPER = $C9CA
0150 PDVREG = $D1FF
0160 PDVRS = $0248
0170 PPTMPA = $024C
0180 PPTMPX = $024D
0190 ;
0200     *=  $C9DC
0210 ;
0220     STA PPTMPA
0230     STX PPTMPX
0240     LDA CRITIC
0250     PHA
0260     LDA #$01
0270     STA CRITIC
0280     LDX #$08
0290 LOOP JSR GETLOW
```

```
0300     BEQ NEXDV
0310     TXA
0320     PHA
0330     TYA
0340     PHA
0350     JSR NEWPER
0360     BCC NEXT
0370     STA PPTMPA
0380     PLA
0390     PLA
0400     JMP END
0410 NEXDV LDY #$82
0420 END LDA #$00
0430     STA PDVRS
0440     STA PDVREG
0450     PLA
0460     STA CRITIC
0470     LDA PPTMPA
0480     STY PPTMPX
0490     LDY PPTMPX
0500     RTS
0510 NEXT PLA
0520     TAY
0530     PLA
0540     TAX
0550     BCC LOOP
```

Działanie procedury NEWPER jest identyczne, jak znajdujących się w podsystemie CIO procedur GOHAND i CIOJMP (zob. str. 24). Jednakże adres wywoływanej procedury jest pobierany z pamięci ROM nowego urządzenia zależnie od operacji. Zawartość rejestru Y określa kolejno: PDVOPV (Parallel DeVice OPen Vector), PDVCLV (Parallel Device CLose Vector), PDVGBV (Parallel Device Get Byte Vector), PDVPBV (Parallel Device Put Byte Vector), PDVSTV (Parallel Device STatus Vector) i PDVSPV (Parallel Device SPecial Vector).

```
0100 ;NEW device Port ERror
0110 ;
0120 PDVOPV = $D80D
0130 PPTMPA = $024C
0140 PPTMPX = $024D
0150 ;
0160     *=  $C9CA
0170 ;
0180     LDA PDVOPV,Y
0190     PHA
0200     DEY
0210     LDA PDVOPV,Y
0220     PHA
0230     LDA PPTMPA
0240     LDX PPTMPX
0250     LDY #$92
0260     RTS
```

Rozdział 7

TWORZENIE OBRAZU

Tworzeniem obrazu na monitorze zajmuje się w komputerach Atari specjalizowany układ scalony - ANTIC (AlphaNumeric Television Interface Controller). Jest to specjalnie zaprojektowany dla Atari mikroprocesor, który posiada własny zestaw rozkazów i własny program. Według tego programu ANTIC pobiera z pamięci komputera dane umieszczone tam przez jednostkę centralną (procesor 6502) i po odpowiedniej interpretacji przesyła je do monitora.

Obraz telewizyjny generowany przez ANTIC jest jednak monochromatyczny. Kolory są dodawane przez inny układ specjalny - GTIA (Graphics Television Interface Adaptor), który także został zaprojektowany specjalnie dla Atari. Poza kolorem GTIA tworzy jeszcze ruchome obiekty ekranowe zwane popularnie duszkami, a według nomenklatury Atari - grafiką graczy i pocisków (P/MG - Player/Missile Graphics). Te funkcje GTIA są opisane w rozdziale 8.

Spośród układów graficznych stosowanych w komputerach domowych ANTIC wyróżnia się dwoma cechami. Przede wszystkim może bezpośrednio korzystać z pamięci komputera poprzez tzw. DMA (Direct Memory Access). Odciąża więc w ten sposób procesor centralny, co pozwala znacznie przyspieszyć jego pracę. Ponadto ANTIC posiada kompletną 16-bitową szynę adresową, dzięki czemu ma dostęp do całych 64 KB pamięci. Umożliwia to umieszczenie danych dla obrazu oraz programu ANTIC-a w dowolnym miejscu pamięci.

Przed przejściem do dalszego opisu trzeba wiedzieć, jak wygląda obraz telewizyjny tworzony przez ANTIC. Obraz ten jest wyświetlany na ekranie przez wiązkę elektronów poruszającą się poziomo od lewej do prawej krawędzi ekranu (patrząc od strony użytkownika) i tworzącą jedną linię ekranu. Wiązka jest wygaszana na czas przejścia do początku nowej linii - ten okres nazywamy synchronizacją poziomą. Po utworzeniu całego obrazu wiązka jest wygaszana na czas przejścia z prawego, dolnego rogu ekranu do lewego, górnego. Nazywa się to synchronizacją pionową. Cała operacja powtarza się co 1/50 sekundy, a więc 50 razy na sekundę.

Pozioma wielkość linii ekranu jest mierzona w specjalnych jednostkach zwanych cyklami koloru. Jest to czas wykonywania przez procesor 6502 połowy cyklu maszynowego. Pełna linia ekranu ma 228 cykli koloru, co pozwala na przedstawienie 456 pojedynczych punktów, a pełny obraz zawiera 310 linii (w systemie PAL). Jednak tak utworzony obraz nie mieściłby się w całości na ekranie telewizora. Aby uniknąć utraty części informacji, obraz tworzony przez ANTIC ma nieco ograniczone wymiary.

Maksymalna wielkość obrazu generowanego przez ANTIC może wynosić 216 linii, ponieważ na więcej nie wystarcza czasu. Utworzenie większej liczby linii spowoduje "zerwanie" synchronizacji

czyli pionowe przesuwanie się obrazu. Aby górna krawędź obrazu nie wychodziła poza ekran, w górnej części ANTIC tworzy 24 linie puste. Pozostają więc 192 linie i taka właśnie jest maksymalna rozdzielczość pionowa obrazu. Można ją nieco zwiększyć przez zmniejszenie liczby pustych linii i zwiększenie liczby linii obrazu, lecz suma wszystkich linii nigdy nie może przekraczać 216.

Także w poziomie nie jest wykorzystywana cała długość linii. W tym przypadku możliwe są jedynie trzy rozwiązania. Wyświetlana linia obrazu może posiadać 128, 160 lub 192 cykle koloru. Normalnie używany jest obraz o szerokości 160 cykli koloru, co daje 320 punktów ekranu. Zmiana szerokości linii jest dokonywana przez zmianę zawartości odpowiedniego rejestru ANTIC-a i programista nie może jej ustalić dowolnie.

Pozostaje jeszcze wyjaśnienie różnicy między linią ekranu i linią obrazu oraz między punktem ekranu i pixelem (punktem obrazu). Wszystko to, co zostało napisane wyżej dotyczy linii i punktów ekranu, ponieważ stanowią one podstawowe elementy ekranu (niezależne od komputera). Natomiast najmniejszy element obrazu (PICture ELement = pixel), na którego wygląd może wpływać programista zależy od wybranego trybu graficznego. Ogólnie można powiedzieć, że linia obrazu składa się z jednej lub kilku linii ekranu, zaś pixel z jednego lub kilku punktów ekranu.

ANTIC posiada kilkanaście rejestrów, które sterują jego pracą. Poprzez zmianę ich zawartości można wpływać na wygląd obrazu. Najważniejszym rejestrem jest DMACTL (DMA ConTroL - $D400) kontrolujący dostęp ANTIC-a do pamięci komputera. Bity 6 i 7 tego rejestru są niewykorzystane, a znaczenie pozostałych jest następujące:

Bit 5 włącza (gdy jest ustawiony) bezpośredni dostęp ANTIC-a do pamięci w celu czytania programu. Skasowanie tego bitu uniemożliwia więc wyświetlanie obrazu pomimo tego, że znajduje się on w pamięci.

Bity 2-4 są wykorzystywane do sterowania grafiką graczy i pocisków. Ich działanie jest szczegółowo opisane w rozdziale 8.

Bity 0 i 1 regulują dostęp ANTIC-a do pamięci w celu odczytu danych obrazu. Gdy oba są równe zero, to ANTIC nie może odczytywać danych i nie wyświetla obrazu. Inne kombinacje ustalają szerokość wyświetlanego obrazu: 01 - obraz wąski (128 cykli koloru), 10 - obraz normalny (160 cykli koloru) i 11 - obraz szeroki (192 cykle koloru).

Podczas wykonywania programu często zachodzi konieczność przeprowadzenia jakiejś operacji (np. zmiany koloru) między liniami ekranu, a nie w trakcie tworzenia linii. Pomocny jest w tym rejestr WSYNC (Wait for SYNChronization - $D40A). Wpisanie dowolnej wartości do tego rejestru powoduje zatrzymanie pracy CPU (6502) do zakończenia tworzenia na ekranie aktualnej linii.

Także licznik linii ekranu VCOUNT (Vertical COUNT - $D40B) jest zwykle wykorzystywany

podczas przerwań DLI. Zawiera on numer aktualnie tworzonej linii ekranu podzielony przez dwa. Podczas tworzenia obrazu VCOUNT zlicza więc od 0 do 155.

Stosunkowo najrzadziej wykorzystywane są dwa następne rejestry - LPENH (Light PEN Horizontal - $D40C) i LPENV (LPEN Vertical). Wskazują one aktualną pozycję pióra świetlnego na ekranie. Rejestr LPENH zawiera numer cyklu koloru (od 0 do 227), a LPENV numer linii podzielony przez dwa (jak VCOUNT).

7.1. Program ANTIC-a

ANTIC tworzy obraz według programu zawartego w pamięci RAM komputera, oznaczonego skrótem DL (Display List). Adres tego programu jest wpisywany do wektora DLPTRS (Display List PoinTeR Shadow register), a następnie podczas przerwania synchronizacji pionowej VBLK (w tym czasie obraz nie jest generowany) przepisywany do DLPTR.

Program ANTIC-a jest układany przez system operacyjny i zapisywany w pamięci podczas procedury otwarcia ekranu (SCOPN - zob. str. 49). Dla każdego trybu program ANTIC-a jest inny, ale jest tylko jeden. Użytkownik może ingerować w jego treść, aby mieszać między sobą różne tryby oraz dla uzyskania innych specjalnych efektów.

7.1.1. Rozkazy ANTIC-a

Aby zmieniać program ANTIC-a trzeba przede wszystkim znać kody jego rozkazów i ich znaczenie. Wszystkie rozkazy, którymi dysponuje ANTIC, można podzielić na trzy zasadnicze grupy: rozkazy tworzenia pustych linii, rozkazy skoków i rozkazy tworzenia linii trybu. Rodzaj rozkazu jest rozpoznawany według czterech młodszych bitów. Cztery starsze bity służą do modyfikacji podstawowego rozkazu.

We wszystkich rozkazach ustawienie najstarszego bitu powoduje wywołanie przez ANTIC (po wykonaniu rozkazu) przerwania niemaskowalnego. Przerwanie to nazywane jest przerwaniem programu ANTIC-a (DLI - Display List Interrupt). Nie należy przez to rozumieć, że przerywany jest program ANTIC-a, lecz, że wywołuje on przerwanie. Po otrzymaniu żądania przerwania DLI procesor wykonuje procedurę przerwania, której adres znajduje się w rejestrze DLIV (DLI Vector - $0200).

Rozkazy tworzenia pustych linii mają cztery młodsze bity skasowane, są więc postaci $x0. Pozostałe bity (4-6, bo bit 7 sygnalizuje przerwanie DLI) zawierają liczbę tworzonych pustych linii zmniejszoną o jeden. Pełna lista rozkazów tworzących puste linie jest następująca:

	bez DLI	z DLI
1 pusta linia	$00	$80
2 puste linie	$10	$90
3 puste linie	$20	$A0
4 puste linie	$30	$B0
5 pustych linii	$40	$C0
6 pustych linii	$50	$D0

```
7 pustych linii       $60              $E0
8 pustych linii       $70              $F0
```

Rozkazy skoków mają najmłodszy bit równy jeden, są więc postaci $x1. Bity 4 i 5 są niewykorzystane, zaś bit 6 określa rodzaj skoku. Gdy bit 6 jest skasowany, to tworzona jest jedna pusta linia, a dwa następne bajty programu przepisywane są do licznika programu w kolejności LSB/MSB. Powoduje to wykonywanie dalszej części programu od wskazanego adresu, czyli po prostu skok bezwzględny (JMP). Rozkaz ten jest podobny do rozkazu JMP procesora 6502.

Gdy bit 6 jest ustawiony, to ANTIC wykonuje te same czynności, lecz dalsza realizacja programu jest zatrzymywana, aż do synchronizacji pionowej. Jest to więc skok z oczekiwaniem na synchronizację VBLK (JVB) i musi on występować na końcu programu ANTIC-a (i tylko tam).

Kody rozkazów skoku przedstawia poniższa tabela:

```
                        bez DLI          z DLI
JMP                       $01             $81
JVB                       $41             $C1
```

Rozkazy tworzenia linii trybu zawierają w czterech młodszych bitach numer trybu (od $02 do $0F), który określa sposób interpretacji danych z pamięci obrazu. Chodzi tu oczywiście o numery trybów ANTIC-a, które są inne niż numery stosowane przez OS. Wszystkie numery trybów w tym rozdziale oznaczają tryby ANTIC-a, o ile nie zostało specjalnie zaznaczone, że chodzi o tryb OS. Pozostałe cztery bity oznaczają modyfikacje rozkazu. Znaczenie bitu 7 (DLI) zostało już opisane, teraz kolej na pozostałe modyfikacje.

Bit 4 (HSC - Horizontal SCrolling) oznacza, gdy jest ustawiony, włączenie poziomego przesuwu tworzonej linii obrazu (dalsze informacje na ten temat znajdują się w rozdziale 7.4. - str. 180).

Bit 5 (VSC - Vertical SCrolling) oznacza, gdy jest ustawiony, włączenie pionowego przesuwu tworzonej linii obrazu.

Bit 6 (LMS - Load Memory Scan) powoduje, gdy jest ustawiony przepisanie dwóch następnych bajtów programu do licznika pamięci obrazu. Licznik obrazu wskazuje ANTIC-owi miejsce w pamięci, z którego ma on pobrać dane obrazu. Ta modyfikacja rozkazu jest bardzo ważna i wymaga szerszego omówienia. ANTIC po rozpoznaniu rozkazu tworzenia linii pobiera z pamięci obrazu odpowiednią liczbę bajtów (zależną od trybu) i po zinterpretowaniu wyświetla na ekranie. Pobieranie danych dla następnej linii rozpoczyna się od następnego bajtu po ostatnio odczytanym. Trzeba więc na początku programu odpowiednio ustawić adres tych danych i to z dwóch powodów. Po pierwsze dlatego, aby ANTIC wiedział skąd ma pobierać dane. Po drugie dlatego, że program ANTIC-a jest wykonywany w całości 50 razy na sekundę i po pobraniu danych dla jednego obrazu, dane dla następnego byłyby pobierane z innego miejsca pamięci.

Licznik obrazu ma specyficzną konstrukcję, która wpływa na sposób jego działania. Posiada on

szesnaście bitów ustalanych przez wpisanie adresu odczytanego z programu ANTIC-a. Podczas pobierania danych z pamięci zmieniane jest jednak tylko 12 młodszych bitów. Ogranicza to zakres działania licznika do bloku 4 KB. Jeżeli dane obrazu zajmują więcej niż 4 KB, to licznik musi być ponownie ustawiony. W przeciwnym razie po osiągnięciu końca bloku 4 KB następne dane będą pobierane z początku tego samego bloku. Trzeba przy tym uważać, aby dane żadnej linii obrazu nie przekraczały granicy bloku, gdyż nie można zmienić stanu licznika w środku tworzonej linii. Sytuacja taka występuje w trybach E i F. Normalnie rozkaz LMS występuje w programie ANTIC-a razem z rozkazem tworzenia pierwszej linii obrazu i razem z rozkazem tworzenia pierwszej linii okna tekstowego (jeśli ono jest).

Poniższe tabele zawierają kody wszystkich rozkazów ANTIC-a tworzących linie trybu.

Bez przerwania DLI

modyfikacje	-	HSC	-	HSC	-	HSC	-	HSC
	-	-	VSC	VSC	-	-	VSC	VSC
	-	-	-	-	LMS	LMS	LMS	LMS
tryb								
2	$02	$12	$22	$32	$42	$52	$62	$72
3	$03	$13	$23	$33	$43	$53	$63	$73
4	$04	$14	$24	$34	$44	$54	$64	$74
5	$05	$15	$25	$35	$45	$55	$65	$75
6	$06	$16	$26	$36	$46	$56	$66	$76
7	$07	$17	$27	$37	$47	$57	$67	$77
8	$08	$18	$28	$38	$48	$58	$68	$78
9	$09	$19	$29	$39	$49	$59	$69	$79
A	$0A	$1A	$2A	$3A	$4A	$5A	$6A	$7A
B	$0B	$1B	$2B	$3B	$4B	$5B	$6B	$7B
C	$0C	$1C	$2C	$3C	$4C	$5C	$6C	$7C
D	$0D	$1D	$2D	$3D	$4D	$5D	$6D	$7D
E	$0E	$1E	$2E	$3E	$4E	$5E	$6E	$7E
F	$0F	$1F	$2F	$3F	$4F	$5F	$6F	$7F

Z przerwaniem DLI

modyfikacje	-	HSC	-	HSC	-	HSC	-	HSC
	-	-	VSC	VSC	-	-	VSC	VSC
	-	-	-	-	LMS	LMS	LMS	LMS
tryb								
2	$82	$92	$A2	$B2	$C2	$D2	$E2	$F2
3	$83	$93	$A3	$B3	$C3	$D3	$E3	$F3
4	$84	$94	$A4	$B4	$C4	$D4	$E4	$F4
5	$85	$95	$A5	$B5	$C5	$D5	$E5	$F5
6	$86	$96	$A6	$B6	$C6	$D6	$E6	$F6
7	$87	$97	$A7	$B7	$C7	$D7	$E7	$F7
8	$88	$98	$A8	$B8	$C8	$D8	$E8	$F8
9	$89	$99	$A9	$B9	$C9	$D9	$E9	$F9
A	$8A	$9A	$AA	$BA	$CA	$DA	$EA	$FA
B	$8B	$9B	$AB	$BB	$CB	$DB	$EB	$FB
C	$8C	$9C	$AC	$BC	$CC	$DC	$EC	$FC
D	$8D	$9D	$AD	$BD	$CD	$DD	$ED	$FD
E	$8E	$9E	$AE	$BE	$CE	$DE	$EE	$FE
F	$8F	$9F	$AF	$BF	$CF	$DF	$EF	$FF

7.1.2. Struktura programu ANTIC-a

Program ANTIC-a musi spełniać kilka warunków, aby jego wykonanie dało na ekranie pożądany efekt. Przede wszystkim musi koniecznie zawierać rozkaz LMS na początku oraz rozkaz JVB na końcu. Aby górna krawędź obrazu nie wychodziła poza ekran, przed rozkazami tworzenia linii trybu powinny znaleźć się rozkazy tworzenia pustych linii (może ich być mniej niż 24). Typowy program ANTIC-a tworzony przez system operacyjny jest przedstawiony na następnej stronie.

```
START   $70     -+
        $70      ¦  3*8=24 puste linie
        $70     -+
        $4x     linia trybu + LMS
        <SCRM   -+  adres pamięci obrazu
        >SCRM   -+  w kolejności LSB/MSB
        $0x     -+
        $0x      ¦
        ...      :  linie trybu
        $0x      ¦
        $0x     -+
        $42     linia trybu 2 + LMS
        <TXTM   -+  adres pamięci okna
        >TXTM   -+  tekstowego (LSB/MSB)
        $02     -+
        $02      ¦  3 linie trybu 2
        $02     -+
        $41     JVB
        <START  -+  adres początku
        >START  -+  programu ANTIC-a
```

Oczywiście fragment programu dotyczący okna tekstowego występuje tylko wtedy, gdy to okno istnieje. Najkrótsze programy ANTIC-a są ustalane dla trybów 5 (tryb 13 OS) i 7 (2) bez okna tekstowego, mają one po 20 bajtów. Tryby E (15) i F (8) bez okna mają najdłuższe programy - po 202 bajty. Jak widać, czym większa rozdzielczość, tym dłuższy program ANTIC-a.

7.1.3. Kolory

Informacja o obrazie utworzona przez ANTIC przed wysłaniem do monitora jest jeszcze uzupełniana informacją o kolorze. Ta czynność jest wykonywana przez GTIA. Do realizacji tego zadania GTIA wykorzystuje dziewięć rejestrów, w których przechowywane są wartości kolorów. Znajdują się one w obszarze od $D012 do $D01A, a ich rejestry-cienie od $02C0 do $02C8.

Wszystkie rejestry koloru mają jednakową strukturę. Cztery starsze bity zawierają numer barwy, a cztery młodsze stopień jasności. Ponieważ bit 0 nie jest odczytywany, to mamy do dyspozycji 16 barw w 8 jasnościach. Daje to razem 128 kolorów, a we wszystkich informacjach o Atari jest napisane, że można wybierać z palety 256 kolorów. Skąd ta różnica? Obie podane wyżej informacje są prawdziwe. Rzeczywiście do wyboru jest 128 kolorów. Natomiast szczególna interpretacja danych obrazu w trybie 9 (tryb systemu operacyjnego) pozwala na uzyskanie szesnastu zamiast ośmiu stopni jasności i teraz można wybierać z 256 kolorów. Tak więc we wszystkich trybach mamy dostępne 128 kolorów, a tylko w trybie 9 (OS) - 256. Nie jest to całkowicie prawdziwe, ale pozostawmy tą kwestię na później. Szczegóły są wyjaśnione przy opisie trybów GTIA.

Teraz wypada przedstawić kolory, które odpowiadają różnym wartościom umieszczonym w rejestrach koloru. Ze stopniami jasności sprawa jest prosta: wartość najmniejsza ($00) oznacza kolor najciemniejszy, a największa ($0F) - najjaśniejszy. Znacznie trudniejsze jest przedstawienie barw. Przede wszystkim dlatego, że barwa zależy także od jasności, np. barwa $00 z jasnością $00 to kolor czarny, zaś z jasnością $0E - biały. Poza tym kolory w dużym stopniu zależą od konkretnego egzemplarza monitora, nie mówiąc już o systemie telewizyjnym (wszyskie źródła podają barwy w systemie NTSC, a nie PAL). Po przedstawieniu tych zastrzeżeń, czas na wykaz barw (x oznacza jasność).

```
kod          kolor
$0x       szary
$1x       złoty
$2x       pomarańczowy
$3x       czerwony
$4x       różowy
$5x       purpurowy
$6x       fioletowy
$7x       niebieski zimny
$8x       niebieski
$9x       błękitny
$Ax       turkusowy
$Bx       niebiesko-zielony
$Cx       zielony
$Dx       żółto-zielony
$Ex       pomarańczowo-zielony
$Fx       żółty
```

Na zakończenie jeszcze jedna ważna informacja. Podczas tworzenia obrazu kolor jest określany zawsze według rejestru sprzętowego. Ten z kolei jest zapisywany wartością z rejestru-cienia podczas każdego przerwania VBLK, czyli 50 razy na sekundę. Po zmianie koloru w rejestrze-cieniu kolor na ekranie zmieni się więc najpóźniej po 1/50 sekundy. Jeśli zmiana zostanie dokonana w rejestrze sprzętowym, to kolor zmieni się natychmiast, lecz nie da się tego zobaczyć. Najpóźniej bowiem po 1/50 sekundy kolor zostanie odtworzony według starej wartości z rejestru-cienia. Gdy jednak kolor w rejestrze sprzętowym będzie zmieniany podczas każdego tworzenia obrazu (np. przez procedurę przerwania DLI), to zmiana będzie trwała, ale tylko w tej części obrazu, która jest tworzona po przerwaniu. Podczas przerwania VBLK stara wartość zostanie odtworzona i część obrazu do ponownego wywołania procedury DLI będzie miała stary kolor. Jest to najczęstsze zastosowanie przerwań DLI.

7.2. Tryby bitowe

W trybach bitowych dostępny jest każdy pixel obrazu. Poszczególne tryby różnią się między sobą liczbą dostępnych kolorów oraz rozdzielczością, czyli wielkością pixela. Od tych parametrów zależy też wielkość obszaru pamięci zajmowanego przez dane obrazu. Ponieważ pixele w niektórych trybach zajmują kilka linii ekranu (w pionie), to dla zaoszczędzenia czasu ANTIC umieszcza dane pobrane dla jednej linii w specjalnym rejestrze. Przy tworzeniu następnej linii ekranu należącej do tej samej linii obrazu dane są pobierane nie z pamięci, lecz z tego specjalnego

rejestru. Pozostawia to więcej czasu jednostce centralnej (6502) na wykonywanie właściwego programu. Trzeba bowiem pamiętać, że ANTIC zatrzymuje pracę 6502 na czas pobierania danych z pamięci.

ANTIC dysponuje ośmioma trybami bitowymi. Wszystkie tryby są dostępne dla systemu operacyjnego. Ponadto OS poprzez zmianę wartości w rejestrze GTIACTL może uzyskać trzy dodatkowe tryby bitowe. Poniżej opisane są wszystkie tryby bitowe. Podane wartości liczbowe odnoszą się, o ile nie określono inaczej, do normalnej szerokości obrazu, równej 160 cykli koloru.

Tryb 8

W tym trybie pixel ma wysokość ośmiu linii ekranu i szerokość czterech cykli koloru. W jednej linii znajduje się więc 40 pixeli. Maksymalna liczba linii obrazu wynosi 192/8=24. ANTIC dysponuje przy tym czteroma kolorami. Do wyboru koloru pixela potrzebne są dwa bity. Razem dane całej linii zajmują 40*2=80 bitów, czyli 80/8=10 bajtów. Każda para bitów umieszczona w pamięci obrazu służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr COLBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2
```

Tryb ten odpowiada trybowi $03 systemu operacyjnego.

Tryb 9

W tym trybie pixel ma wysokość czterech linii ekranu i szerokość dwóch cykli koloru. W jednej linii znajduje się więc 80 pixeli. Maksymalna liczba linii obrazu wynosi 192/4=48. ANTIC dysponuje przy tym dwoma kolorami. Do wyboru koloru pixela potrzebny jest jeden bit. Razem dane całej linii zajmują 80*1=80 bitów, czyli 80/8=10 bajtów. Każdy skasowany bit w pamięci obrazu powoduje wybór koloru z rejestru COLBAK, a każdy bit ustawiony z rejestru COLPF0. Tryb ten odpowiada trybowi $04 systemu operacyjnego.

Tryb A

W tym trybie pixel ma wysokość czterech linii ekranu i szerokość dwóch cykli koloru. W jednej linii znajduje się więc 80 pixeli. Maksymalna liczba linii obrazu wynosi 192/4=48. ANTIC dysponuje przy tym czteroma kolorami. Do wyboru koloru pixela potrzebne są dwa bity. Razem dane całej linii zajmują 80*2=160 bitów, czyli 160/8=20 bajtów. Każda para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr COLBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2
```

Tryb ten odpowiada trybowi $05 systemu operacyjnego.

Tryb B

W tym trybie pixel ma wysokość dwóch linii ekranu i szerokość jednego cyklu koloru. W jednej linii znajduje się więc 160 pixeli. Maksymalna liczba linii obrazu wynosi 192/2=96. ANTIC dysponuje przy tym dwoma kolorami. Do wyboru koloru pixela potrzebny jest jeden bit. Razem dane całej linii zajmują 160*1=160 bitów, czyli 160/8=20 bajtów. Każdy skasowany bit w pamięci obrazu powoduje wybór koloru z rejestru COLBAK, a każdy bit ustawiony z rejestru COLPF0. Tryb ten odpowiada trybowi $06 systemu operacyjnego.

Tryb C

W tym trybie pixel ma wysokość jednej linii ekranu i szerokość jednego cyklu koloru. Pixel jest więc prostokątny. W jednej linii znajduje się 160 pixeli. Maksymalna liczba linii obrazu wynosi 192/1=192. ANTIC dysponuje przy tym dwoma kolorami. Do wyboru koloru pixela potrzebny jest jeden bit. Razem dane całej linii zajmują 160/1=160 bitów, czyli 160/8=20 bajtów. Każdy skasowany bit w pamięci obrazu powoduje wybór koloru z rejestru COLBAK, a każdy bit ustawiony z rejestru COLPF0. Tryb ten odpowiada trybowi $0E systemu operacyjnego.

Tryb D

W tym trybie pixel ma wysokość dwóch linii ekranu i szerokość jednego cyklu koloru. W jednej linii znajduje się więc 160 pixeli. Maksymalna liczba linii obrazu wynosi 192/2=96. ANTIC dysponuje przy tym dwoma kolorami. Do wyboru koloru pixela potrzebne są dwa bity. Razem dane całej linii zajmują 160*2=320 bitów, czyli 320/8=40 bajtów. Każda para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr COLBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2
```

Tryb ten odpowiada trybowi $07 systemu operacyjnego.

Tryb E

W tym trybie pixel ma wysokość jednej linii ekranu i szerokość jednego cyklu koloru. Pixel jest więc prostokątny. W jednej linii znajduje się 160 pixeli. Maksymalna liczba linii obrazu wynosi 192/1=192. ANTIC dysponuje przy tym czterema kolorami. Do wyboru koloru pixela potrzebne są dwa bity. Razem dane całej linii zajmują 160*2=320 bitów, czyli 320/8=40 bajtów. Pełna pamięć obrazu w tym trybie (192 linie) przekracza granicę 4 KB. Każda para bitów wybiera kolor według schematu:

```
para bitów 00 - rejestr COLBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2
```

Tryb ten odpowiada trybowi $0F systemu operacyjnego.

Tryb F

W tym trybie pixel ma wysokość jednej linii ekranu i szerokość pół cyklu koloru. W jednej linii znajduje się więc 320 pixeli. Maksymalna liczba linii obrazu wynosi 192/1=192. Ponieważ nie można dokonać zmiany koloru w obrębie jednego cyklu koloru, to pixele mogą się różnić tylko jasnością. Do wyboru jasności pixela potrzebny jest jeden bit. Razem dane całej linii zajmują 320*1=320 bitów, czyli 320/8=40 bajtów. Pełna pamięć obrazu w tym trybie (192 linie) przekracza granicę 4 KB. Każdy skasowany bit w pamięci obrazu powoduje wybór koloru z rejestru COLPF2, a każdy bit ustawiony wybiera jasność z rejestru COLPF1, a kolor z COLPF2. Tryb ten odpowiada trybowi $08 systemu operacyjnego.

Trzy dodatkowe tryby bitowe można uzyskać przez zmianę dwóch najstarszych (6 i 7) bitów rejestru GTIACTL (GTIA ConTroL). Normalnie mają one wartość 00 i obraz jest tworzony przez ANTIC tak, jak to opisano przy poszczególnych trybach bitowych i znakowych. System operacyjny dla uzyskania dodatkowych trybów wybiera tryb ANTIC-a F ($08 OS) i zmienia bity w GTIACTL. Pomimo, iż OS nie przewiduje zmiany sposobu pracy GTIA w innych trybach ANTIC-a, to jest to możliwe i pozwala niekiedy na uzyskanie ciekawych efektów graficznych (także w trybach znakowych).

Ponieważ ANTIC w dodatkowych trybach OS pracuje w trybie F, to możliwe jest uzyskanie na ekranie 192 linii po 320 punktów ekranu w każdej. Dane każdej linii składają się więc z 40 bajtów. Sposób interpretacji tych danych jest jednak inny i zależy od stanu bitów 6 i 7 w GTIACTL. Zawsze jednak GTIA interpretuje każde cztery bity (pół bajtu) danych pamięci obrazu jako dane dla jednego pixela. Można więc uzyskać w jednej linii 80 pixeli, z których każdy ma wysokość jednej linii ekranu i szerokość dwóch cykli koloru.

Tryb $09 (OS)

Tryb ten uzyskiwany jest przy wartości najstarszych bitów GTIACTL równej 01. Wartość bitów każdego pixela jest interpretowana jako stopień jasności. Kolor jest przy tym pobierany z rejestru COLBAK. Pozwala to na uzyskanie 16 stopni jasności jednego koloru.

Tryb $0A (OS)

Tryb ten uzyskiwany jest przy wartości najstarszych bitów GTIACTL równej 10. Wartość bitów każdego pixela jest interpretowana jako numer rejestru koloru. Ponieważ GTIA posiada dziewięć rejestrów, to można uzyskać 9 różnych barw o różnych jasnościach.

Tryb $0B (OS)

Tryb ten uzyskiwany jest przy wartości najstarszych bitów GTIACTL równej 11. Wartość bitów każdego pixela jest interpretowana jako numer barwy. Jasność jest przy tym pobierana z rejestru COLBAK. Pozwala to na uzyskanie 16 różnych barw o jednakowym stopniu jasności.

Zbiorcze zestawienie trybów bitowych zawiera poniższa tabela. Liczba linii i rozmiar pamięci podane są w niej dwukrotnie - pierwsza wartość dotyczy obrazu z oknem tekstowym, a druga bez. Rozmiar pamięci określa liczbę bajtów zajmowaną przez pamięć obrazu i program ANTIC-a.

tryb ANTIC-a	tryb OS	liczba linii	liczba pixeli	liczba kolorów	bajty w linii	rozmiar pamięci
8	$03	20/24	40	4	10	434/432
9	$04	40/48	80	2	10	694/696
A	$05	40/48	80	4	20	1174/1176
B	$06	80/96	160	2	20	2174/2184
C	$0E	160/192	160	2	20	4270/4296
D	$07	80/96	160	4	40	4190/4200
E	$0F	160/192	160	4	40	8112/8138
F	$08	160/192	320	1,5	40	8112/8138
F	$09	-/192	80	1(16 j.)	40	-/8138
F	$0A	-/192	80	9	40	-/8138
F	$0B	-/192	80	16(1 j.)	40	-/8138

7.3. Tryby znakowe

W trybach znakowych obraz jest tworzony w sposób odmienny niż w trybach bitowych. W trybach bitowych dane z pamięci obrazu są bezpośrednio wyświetlane na monitorze. W trybach znakowych bajty danych są interpretowane jako numery znaków w zestawie. ANTIC pobiera więc z pamięci obrazu numer znaku i według tego numeru wybiera z zestawu znak do wyświetlenia. Dane linii są przy tym umieszczane w specjalnym rejestrze, podobnie jak w trybach bitowych.

Adres zestawu znaków (a właściwie jego starszy bajt) jest umieszczony w rejestrze CHBASE (CHaracter BASE - $D409). Każdy znak zajmuje w tym zestawie 8 bajtów. Przy tworzeniu linii ekranu obliczany jest adres konkretnego bajtu pamięci, który należy umieścić na ekranie. Odbywa się to według wzoru:

```
CHBASE*$0100 + numer_znaku*$08 + numer_linii_ekranu
```

Dla zaoszczędzenia czasu CPU numer znaku jest pobierany z rejestru ANTIC-a, do którego zostały przepisane dane linii. Numer znaku nie jest wykorzystywany w całości do wyboru znaku. Jego bity 6 i 7 lub tylko 7 służą do wyboru koloru znaku. Tak więc w trybach używających sześciu bitów można przedstawić 64 znaki, a w gdy używane jest siedem bitów, to zestaw zawiera 128 znaków. Ponieważ z rejestru CHBASE odczytuje się także tylko część bitów (1-7 lub 2-7), to zestaw znaków powinien rozpoczynać się odpowiednio od granicy 512 bajtów lub 1 KB. Sposób tworzenia adresu bajtu danych dla różnych trybów jest przedstawiony na rysunkach 12 i 13, które znajdują się na następnej stronie.

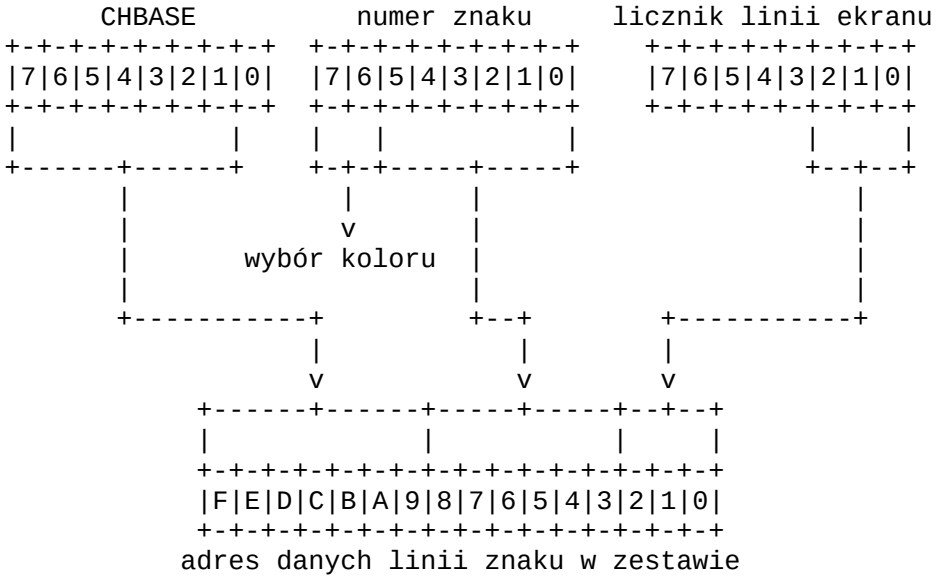
Ponadto w trybach 2 i 3 wyglądem znaków na ekranie steruje rejestr CHRCTL (CHaRacter ConTroL - $D401). Bity 3-7 tego rejestru są niewykorzystane, a pozostałe mają następujące znaczenie:

Bit 0 działa tylko na znaki, które mają ustawiony bit 7.

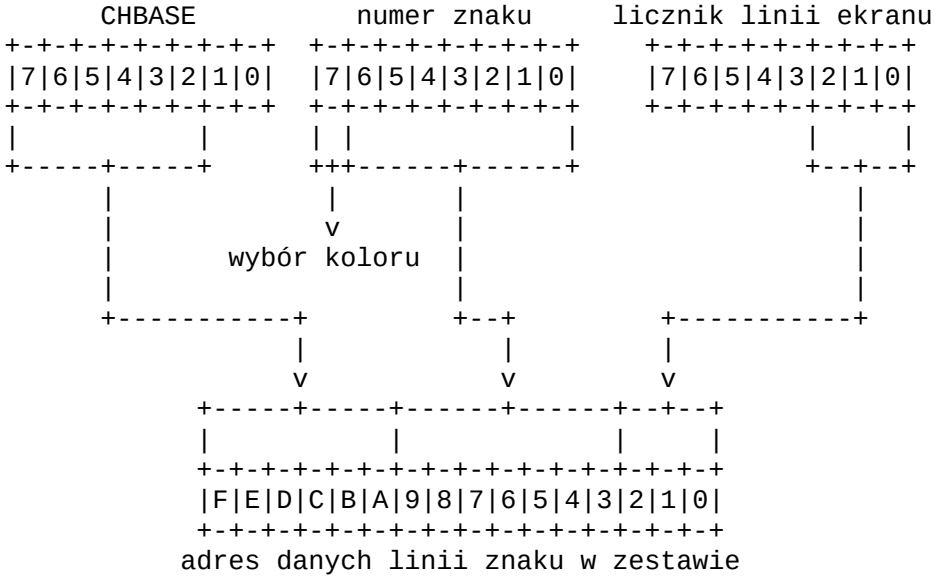
Gdy jest równy jeden, znaki te nie są wyświetlane na ekranie. Przez okresowe przełączanie tego bitu można więc uzyskać miganie znaku.

Bit 1 także działa tylko na znaki, które mają ustawiony bit 7. Gdy jest równy jeden, znaki te mają zamienione kolory punktów i tła. Powoduje to wyświetlenie znaku w negatywie.

Bit 2 odwraca kolejność pobierania danych znaku z pamięci. Gdy na początku wyświetlania linii obrazu jest ustawiony, to wszystkie znaki w tej linii są odwrócone "do góry nogami".



Rys.12. Tworzenie adresu znaku z wykorzystaniem 6 bitów



Rys.13. Tworzenie adresu znaku z wykorzystaniem 7 bitów

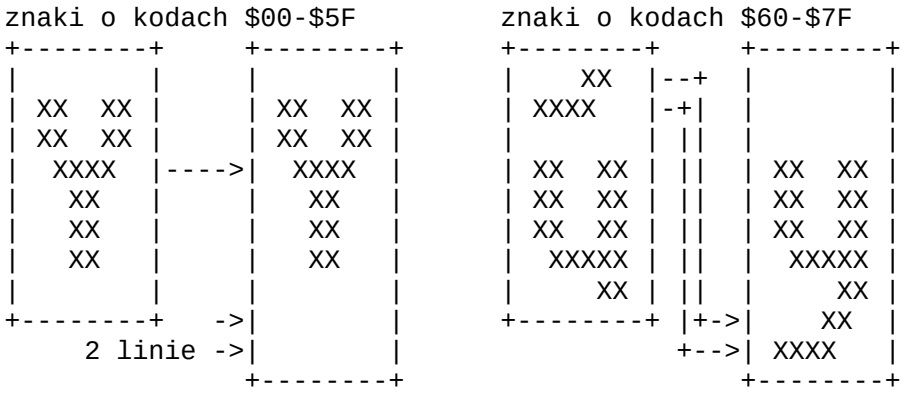
ANTIC dysponuje sześcioma trybami znakowymi. Tryb 3 jest niedostępny dla systemu operacyjnego. Pozostałe tryby mają swoje odpowiedniki w trybach OS. Poniżej opisane są wszystkie tryby znakowe. Podane wartości liczbowe odnoszą się, o ile nie określono inaczej, do normalnej szerokości obrazu, równej 160 cykli koloru.

Tryb 2

W tym trybie znak składa się z ośmiu linii po osiem pixeli, z których każdy ma szerokość pół cyklu koloru. W jednej linii znajduje się więc 40 znaków (czyli 40 bajtów). Maksymalna liczba linii obrazu wynosi 192/8=24. Ponieważ nie można dokonać zmiany koloru w obrębie jednego cyklu koloru, to pixele mogą się różnić tylko jasnością. Każdy bit znaku określa jasność punktu ekranu. Bit skasowany powoduje wybór koloru punktu z rejestru COLPF2, a bit ustawiony wybiera jasność z rejestru COLPF1, a kolor z COLPF2. Do określenia adresu znaku wykorzystywane jest siedem bitów, więc zestaw zawiera 128 znaków. Najstarszy bit (7) steruje wyglądem znaku poprzez rejestr CHRCTL. Tryb ten odpowiada trybowi $00 systemu operacyjnego i jest używany przez edytor.

Tryb 3

W tym trybie znak składa się z dziesięciu linii po osiem pixeli, z których każdy ma szerokość pół cyklu koloru. W jednej linii znajduje się więc 40 znaków (czyli 40 bajtów). Kolory są wybierane analogicznie, jak w trybie 2 i bit 7 także steruje wyglądem znaku poprzez rejestr CHRCTL. W trybie 3 szczególny jest sposób tworzenia znaków. Ponieważ dane znaku składają się z ośmiu bajtów, to normalnie dwie ostatnie linie ekranu są puste. Inaczej jest przy tworzeniu znaków z ostatniej ćwiartki zestawu (o kodach $60-$7F). ANTIC pozostawia w tym przypadku puste dwie pierwsze linie, zaś dwa pierwsze bajty danych znaku umieszcza w dwóch ostatnich liniach (zob. rys. 14). Takie tworzenie znaków wymaga zaprojektowania odrębnego zestawu znaków, w którym będą przedefiniowane znaki zawarte w ostatniej ćwiartce. Dzięki temu tryb ten jest szczególnie przydatny do tworzenia tekstu (wykorzystuje go np. "SpeedScript"). Nie ma on jednak odpowiednika w systemie operacyjnym.



Rys.14. Tworzenie znaków w trybie 3 ANTIC-a

Tryb 4

W tym trybie znak składa się z ośmiu linii po cztery pixele, z których każdy ma szerokość jednego cyklu koloru. W jednej linii znajduje się więc 40 znaków (40 bajtów). Maksymalna liczba linii obrazu wynosi 192/8=24. Do określenia adresu znaku wykorzystywane jest siedem bitów, więc zestaw zawiera 128 znaków. ANTIC dysponuje przy tym pięcioma kolorami. Kolor pixela jest wybierany przez dwa bity danych znaku oraz bit 7 numeru znaku. Każda para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr CLOBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2 (gdy bit 7 = 0)
para bitów 11 - rejestr COLPF3 (gdy bit 7 = 1)
```

Tryb ten odpowiada trybowi $0C systemu operacyjnego. Ponieważ w polu o szerokości czterech pixeli trudno jest zdefiniować literę, to tryb 4 jest wykorzystywany głównie do tworzenia wielokolorowej grafiki (np. w grach), która zajmuje przy tym stosunkowo niewielki obszar pamięci.

Tryb 5

W tym trybie znak składa się z szesnastu linii po cztery pixele, w których każdy ma szerokość jednego cyklu koloru i zajmuje dwie linie ekranu (znak jest dwukrotnie wyższy niż w trybie 4). W jednej linii znajduje się więc 40 znaków (40 bajtów). Maksymalna liczba linii obrazu wynosi 192/16=12. Do określenia adresu znaku wykorzystywane jest siedem bitów, więc zestaw zawiera 128 znaków. ANTIC dysponuje przy tym pięcioma kolorami. Kolor pixela jest wybierany przez dwa bity danych znaku oraz bit 7 numeru znaku. Każda para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr CLOBAK
para bitów 01 - rejestr COLPF0
para bitów 10 - rejestr COLPF1
para bitów 11 - rejestr COLPF2 (gdy bit 7 = 0)
para bitów 11 - rejestr COLPF3 (gdy bit 7 = 1)
```

Tryb ten odpowiada trybowi $0D systemu operacyjnego.

Tryb 6

W tym trybie znak składa się z ośmiu linii po osiem pixeli, z których każdy ma szerokość jednego cyklu koloru. W jednej linii znajduje się więc 20 znaków (20 bajtów). Maksymalna liczba linii obrazu wynosi 192/8=24. Do określenia adresu znaku wykorzystywane jest sześć bitów, więc zestaw zawiera 64 znaki. ANTIC dysponuje przy tym pięcioma kolorami. Kolor tła jest zawsze ustalany przez rejestr COLBAK. Kolor wszystkich pixeli znaku jest wybierany przez dwa najstarsze bity numeru znaku. Ta para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr COLPF0
para bitów 01 - rejestr COLPF1
para bitów 10 - rejestr COLPF2
para bitów 11 - rejestr COLPF3
```

Tryb ten odpowiada trybowi $01 systemu operacyjnego.

Tryb 7

W tym trybie znak składa się z szesnastu linii po osiem pixeli, z których każdy ma szerokość jednego cyklu koloru i zajmuje dwie linie ekranu. W jednej linii znajduje się więc 20 znaków (20 bajtów). Maksymalna liczba linii obrazu wynosi 192/16=12. Do określenia adresu znaku wykorzystywane jest sześć bitów, więc zestaw zawiera 64 znaki. ANTIC dysponuje przy tym pięcioma kolorami. Kolor tła jest zawsze ustalany przez rejestr COLBAK. Kolor wszystkich pixeli znaku jest wybierany przez dwa najstarsze bity numeru znaku. Ta para bitów służy do wyboru koloru według schematu:

```
para bitów 00 - rejestr COLPF0
para bitów 01 - rejestr COLPF1
para bitów 10 - rejestr COLPF2
para bitów 11 - rejestr COLPF3
```

Tryb ten odpowiada trybowi $02 systemu operacyjnego.

Warto na koniec wskazać różnice pomiędzy poszczególnymi trybami znakowymi. Tryby 6 i 7 mają znaki o szerokości dwukrotnie większej, a tryby 5 i 7 o wysokości dwukrotnie większej niż pozostałe. W trybach 4-7 dostępne jest 5 kolorów, lecz w różny sposób. W trybach 4 i 5 kolor określają bity znaku, a w trybach 6 i 7 bity numeru znaku. W pierwszym przypadku można więc uzyskać znak wielokolorowy, zaś w drugim cały znak musi mieć jednakowy kolor.

Zbiorcze zestawienie trybów znakowych zawiera poniższa tabela. Liczba linii i rozmiar pamięci podane są w niej dwukrotnie - pierwsza wartość dotyczy obrazu z oknem tekstowym, a druga bez. Rozmiar pamięci określa liczbę bajtów zajmowaną przez pamięć obrazu i program ANITC-a.

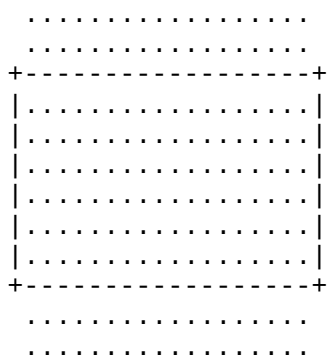
tryb ANTIC-a	tryb OS	liczba linii	liczba znaków	liczba kolorów	bajty w linii	rozmiar pamięci
2	$00	-/24	40	1,5	40	-/992
3	-	-/20	40	1,5	40	-/828
4	$0C	20/24	40	5	40	1154/1152
5	$0D	10/12	40	5	40	664/660
6	$01	20/24	20	5	20	674/672
7	$02	10/12	20	5	20	424/420

7.4. Przesuwanie obrazu

Często informacja, która powinna być wyświetlona na ekranie, po prostu się na nim nie mieści. Konieczne jest więc przemieszczenie tej informacji. Większość komputerów wymaga przepisania całej zawartości pamięci obrazu na nowe miejsce. Ponieważ w Atari adres pamięci obrazu jest zmienny, to wystarczy tylko zmienić ten adres w rozkazie LMS i już cała zawartość ekranu jest przesunięta.

Wykonanie takiego przesunięcia w pionie jest wyjątkowo proste. Dane obrazu muszą obejmować

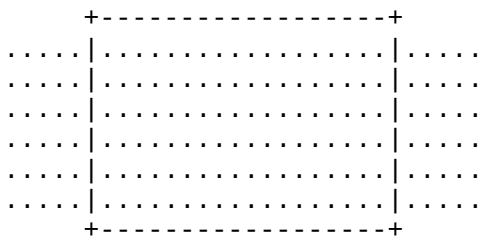
więcej wierszy, a adres LMS wskazuje początek odpowiedniego wiersza (rys. 15). Przesuw obrazu uzyskuje się przez zmianę adresu LMS, czyli przez zmianę tylko dwóch bajtów. Da to zadawalający efekt nawet w tak wolnym języku, jak Atari Basic.



Rys.15. Pamięć obrazu przy przesuwaniu pionowym

Gdy ten sposób zostanie zastosowany do przesuwania obrazu w poziomie, to uzyskany efekt będzie niezbyt interesujący. Po zmniejszeniu adresu w rozkazie LMS wszystkie znaki z prawej krawędzi obrazu znajdą się na lewej, a przy zwiększeniu odwrotnie.

Aby tego uniknąć należy rozkaz LMS umieścić w każdej linii obrazu, a dane obrazu muszą zawierać więcej informacji dla każdej linii (rys. 16). Oczywiście szybka zmiana kilkunastu lub kilkudziesięciu adresów wymaga już użycia języka maszynowego. Ponadto musi to zostać wykonane w czasie, gdy obraz nie jest tworzony, a więc podczas przerwania VBLK. Mimo to i tak jest to dużo prostsze i szybsze niż przepisywanie całej zawartości pamięci obrazu.



Rys.16. Pamięć obrazu przy przesuwaniu poziomym

Opisana wyżej metoda ma jedną zasadniczą wadę. Obraz nie może być przesuwany poziomo po jednym punkcie ekranu, a pionowo można to uzyskać tylko w trybach, w których linia obrazu składa się z jednej linii ekranu. Przesuwanie ekranu odbywa się więc skokowo, co nie wygląda zbyt ciekawie.

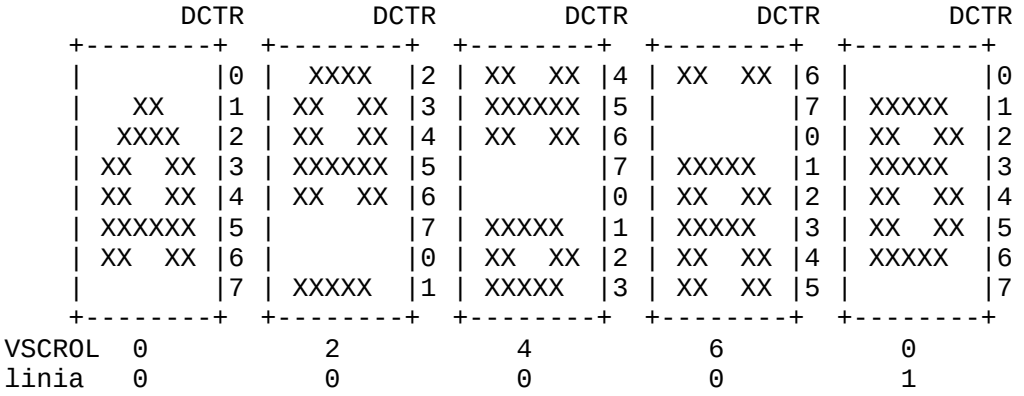
Konstrukcja ANTIC-a umożliwia jednak również przesuwanie obrazu po jednym punkcie lub po jednej linii ekranu. Taki przesuw zwany jest przesuwem delikatnym. Uzyskanie tego efektu jest możliwe dzięki rejestrom ANTIC-a HSCROL (Horizontal SCROLling - $D404) i VSCROL (Vertical SCROLling - $D405).

Rejestry te wpływają tylko na linie obrazu, w których ustawiony jest odpowiednio bit 4 (HSC) lub bit 5 (VSC). Ponadto linia nie może być jednocześnie przesuwana poziomo i pionowo.

7.4.1. Przesuw pionowy

Zanim przystąpimy do opisu delikatnego przesuwu obrazu w pionie, musimy poznać jeszcze jeden wewnętrzny licznik ANTIC-a. Licznik ten nazywa się DCTR (Delta CounTeR) i zlicza linie ekranu w każdej linii obrazu. Ponieważ zliczanie rozpoczyna się od zera, to najwyższy stan DCTR jest o jeden mniejszy od liczby linii ekranu w linii obrazu dla danego trybu.

ANTIC przesuwa w górę te linie obrazu, które mają ustawiony bit 5, o liczbę zawartą w rejestrze VSCROL. Jest to wykonywane w następujący sposób. Po napotkaniu pierwszej linii obrazu, która ma ustawiony bit 5, licznik DCTR rozpoczyna zliczanie od wartości umieszczonej w VSCROL zamiast od zera. Powoduje to wyświetlenie tylko dolnej części tej linii. Następne linie z ustawionym bitem 5 są tworzone normalnie, a więc DCTR liczy od zera do wartości maksymalnej dla danego trybu. Dopiero po napotkaniu pierwszej linii ze skasowanym bitem 5 zmienia się sposób liczenia. DCTR zlicza teraz linie ekranu od zera do wartości zawartej w VSCROL, dzięki czemu na ekranie pojawia się tylko górna część tej linii.



Rys.17. Delikatny przesuw obrazu w pionie

Przy pomocy rejestru VSCROL można przesuwać obraz tylko w obrębie jednej linii. Aby zwiększyć zakres przesuwania obrazu trzeba połączyć przesuwanie delikatne ze zgrubnym (na rys. 17 jest to zasygnalizowane przez numer linii).

Oczywiście zarówno zmiana stanu rejestru VSCROL, jak i zmiana adresu pamięci obrazu musi być wykonana podczas przerwania VBLK. Odpowiedni fragment procedury, która wykonuje tą operację, jest przedstawiony poniżej (wartości oznaczone $xxxx zależą od położenia w pamięci procedury, prorgamu ANTIC-a i danych obrazu). Zostały przy tym poczynione następujące założenia:

1. Obraz jest przesuwany tylko w pionie, a jego pamięć jest zorganizowana w sposób pokazany na rys. 15.

2. Przesuwanie obrazu odbywa się tylko w górę, a po osiągnięciu końca pamięci obrazu (oznaczonego przez FINISH) przesuwanie się kończy.

3. Etykieta MSC oznacza miejsce programu ANTIC-a, w którym znajduje się adres pamięci obrazu.

4. Obraz jest tworzony w takim trybie ANTIC-a, w którym znak ma wysokość ośmiu linii ekranu i szerokość 8 punktów ekranu (tryby 2 lub 4).

5. We wszystkich rozkazach tworzenia linii trybu, oprócz ostatniego bit 5 jest ustawiony.

```
0100 ;Vertical Fine Scrolling
0110 ;
0120 JEXITVB = $E462
0130 VSCROL = $D405
0140 FINISH = $xxxx
0150 MSC   = $xxxx
0160 ;
0170     *=  $xxxx
0180 ;
0190     LDA MSC+1
0200     CMP # >FINISH
0210     BCC CONT
0220     LDA MSC
0230     CMP # <FINISH
0240     BCS EXIT
0250 CONT INC VSCROL
0260     LDA VSCROL
0270     CMP #$08
0280     BNE EXIT
0290     LDA #$00
0300     STA VSCROL
0310     CLC
0320     LDA MSC
0330     ADC #$28
0340     STA MSC
0350     BCC EXIT
0360     INC MSC+1
0370 EXIT JMP JEXITVB
```

Powyższa procedura ma niewielkie zastosowanie praktyczne, gdyż po jej uruchomieniu obraz szybko przesunie się w górę i tak już pozostanie. Aby nadać jej znaczenie praktyczne trzeba napisać drugą procedurę, która będzie przesuwać obraz do góry (zob. niżej) oraz uzależnić wykonanie jednej z tych procedur od jakiegoś parametru. Może to być położenie joysticka lub naciśnięcie klawisza.

Dla przesuwania obrazu w dół trzeba jeszcze przyjąć dodatkowe założenie, że dane obrazu rozpoczynają się od adresu START i po jego osiągnięciu przesuwanie zostanie przerwane.

```
0100 ;Vertical Fine Scrolling 2
0110 ;
0120 JEXITVB = $E462
```

```
0130 VSCROL = $D405
0140 START = $xxxx
0150 MSC   = $xxxx
0160 ;
0170     *=  $xxxx
0180 ;
0190     LDA # >START
0200     CMP MSC+1
0210     BCC CONT
0220     LDA # <START
0230     CMP MSC
0240     BCS EXIT
0250 CONT DEC VSCROL
0260     LDA VSCROL
0270     BPL EXIT
0280     LDA #$08
0290     STA VSCROL
0300     SEC
0310     LDA MSC
0320     SBC #$28
0330     STA MSC
0340     BCS EXIT
0350     DEC MSC+1
0360 EXIT JMP JEXITVB
```

7.4.2. Przesuw poziomy

Ten sam problem występuje przy przesuwaniu poziomym, gdyż każdy bajt danych tworzy kilka pixeli lub cały znak. Przesunięcie o cały bajt przesuwa więc obraz o kilka cykli koloru. Rolę licznika dla przesuwu poziomego pełni w ANTIC-u rejestr HSCROL - identycznie jak VSCROL dla przesuwu pionowego. Między tymi rejestrami jest jednak pewna różnica. O ile VSCROL pozwala przesunąć obraz o jedną linię, to HSCROL przesuwa o jeden cykl koloru, a nie o jeden punkt. Przesunięcie bowiem obrazu o jeden punkt (a więc jeden bit w danych) spowodowałoby zmianę kolorów obrazu, gdyż kolory są zwykle ustalane przez pary bitów.

Rejestr VSCROL ma również ograniczenie podobne do tego, jakie wprowadza licznik DCTR. ANTIC uwzględnia jedynie cztery młodsze bity HSCROL, co pozwala przesunąć obraz jedynie o 15 cykli koloru. Większe przesunięcie można uzyskać przez połączenie delikatnego i zgrubnego przesuwu.

Tworzenie obrazu przesuniętego jest analogiczne, jak przy przesuwie pionowym. Po napotkaniu rozkazu tworzenia linii obrazu, który ma ustawiony bit 4, ANTIC opuszcza liczbę cykli koloru odpowiadającą zawartości HSCROL, a brakujące cykle koloru dodaje na końcu linii. Te brakujące cykle należą już do następnego znaku, więc przy przesuwie poziomym dane linii obrazu muszą być dłuższe conajmniej o jeden bajt. Ponadto pamięć obrazu musi być zorganizowana w sposób pokazany na rys. 16. Dlatego każdy rozkaz tworzenia linii musi mieć ustawiony bit 6 (LMS) w celu ponownego ustawienia licznika pamięci obrazu.

Procedury obsługujące przesuw poziomy są zbliżone do procedur przesuwu pionowego, więc zamieszczona zostanie tylko jedna i już bez komentarza (założenia pozostają bez zmian).

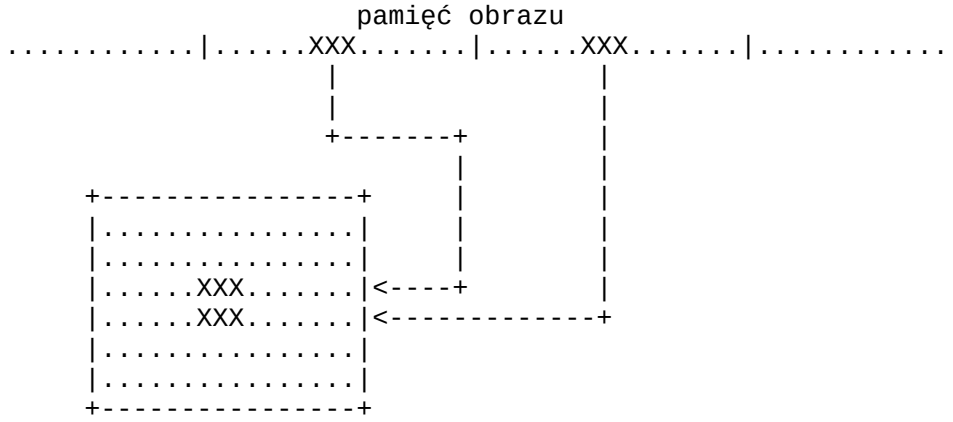
```
0100 ;Horizontal Fine Scrolling
0110 ;
0120 JEXITVB = $E462
0130 HSCROL = $D404
0140 FINISH = $xxxx
0150 MSC   = $xxxx
0160 ;
0170     *=  $xxxx
0180 ;
0190     LDA MSC+1
0200     CMP # >FINISH
0210     BCC CONT
0220     LDA MSC
0230     CMP # <FINISH
0240     BCS EXIT
0250 CONT INC HSCROL
0260     LDA HSCROL
0270     CMP #$04
0280     BNE EXIT
0290     LDA #$00
0300     STA HSCROL
0310     LDX #$46
0320 LOOP INC MSC,X
0330     BNE NEXT
0340     INC MSC+1,X
0350 NEXT DEX
0360     DEX
0370     DEX
0380     BPL LOOP
0390 EXIT JMP JEXITVB
```

Rozdział 8

GRAFIKA GRACZY I POCISKÓW

Jedną z najważniejszych czynności wykonywanych przez komputery domowe jest animacja. Stosowana jest ona nie tylko w grach komputerowych, lecz także w wielu programach edukacyjnych i użytkowych. Uzyskanie szybkiej i płynnej animacji jest jednak bardzo trudne ze względu na sposób przechowywania danych obrazu.

We wszystkich komputerach pamięć obrazu jest zorganizowana liniowo. Oznacza to, że dane dla kolejnych linii obrazu są umieszczone w pamięci jedne po drugich. Utworzenie na ekranie niewielkiego obiektu polega na wpisaniu informacji o tym obiekcie w różne miejsca pamięci, oddalone od siebie (rys. 18). Przesunięcie obiektu wymaga usunięcia jego danych z pamięci obrazu, wpisania w to miejsce danych tła i umieszczenie obiektu w nowym miejscu. Ponieważ poszczególne fragmenty danych są od siebie oddalone, to jest to pracochłonne i stosunkowo powolne. Dodatkowym utrudnieniem jest to, że w trybach bitowych każdy bajt zawiera dane dla kilku punktów obrazu i trzeba zmieniać tylko określone bity tego bajtu. Taki sposób animacji wymaga więc od procesora dużej liczby obliczeń, a na dodatek nie zapewnia płynności ruchu.

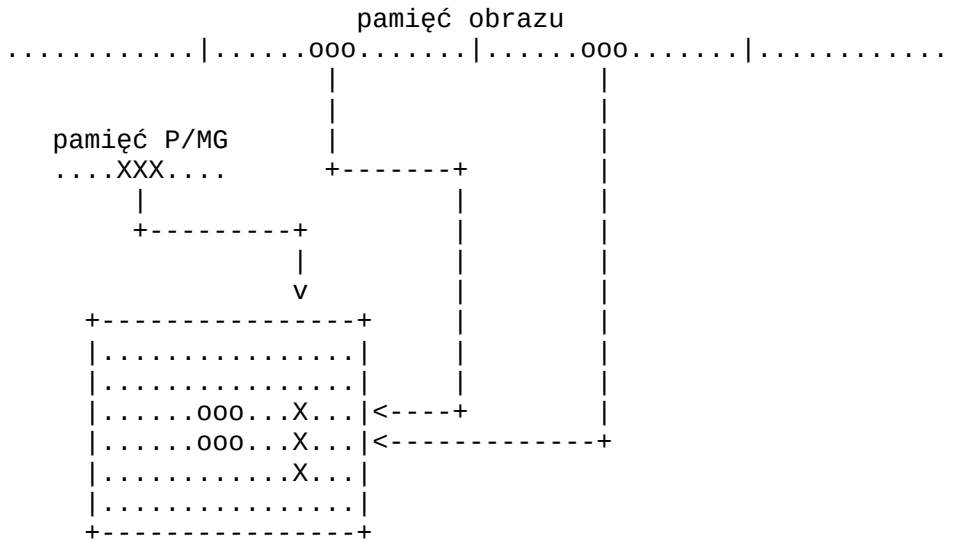


Rys.18. Rozmieszczenie danych obrazu w pamięci

Stosowane są różne sposoby rozwiązywania tego problemu. Najczęściej stosuje się specjalny układ scalony, który umożliwia utworzenie tzw. ruchomych obiektów ekranowych. W komputerach Atari XL/XE układem tym jest GTIA, a technika uzyskiwania obiektów nazywa się grafiką graczy i pocisków (Player/Missile Graphics - P/MG). Umożliwia ona jednoczesne przedstawienie na ekranie czterech dużych obiektów zwanych graczami oraz czterech małych zwanych pociskami. Większa liczba obiektów jest dostępna poprzez zmiany parametrów P/MG przez procedury przerwania DLI.

Zasada działania P/MG polega na umieszczeniu danych obiektu obok siebie w pamięci, dzięki czemu ich przemieszczanie staje się bardzo proste i, co ważniejsze, szybkie (rys. 19). Podczas tworzenia obrazu GTIA do danych aktualnej linii ekranu dodaje dane P/MG i w ten sposób obiekt

pojawia się na ekranie. Oczywiście jest to znacznie bardziej skomplikowane i wymaga od komputera wykonania wielu operacji, lecz są one wykonywane przez układy komputera i nie angażują programisty.



Rys.19. Umieszczenie danych P/MG na obrazie

Podczas tworzenia obrazu ANTIC pobiera z pamięci, oprócz danych dla linii, także dane grafiki P/M i umieszcza je w rejestrze grafiki GRAFP lub GRAFM znajdującym się w GTIA. Gdy GTIA przenosi na ekran dane linii, to dodaje do nich dane z rejestrów grafiki. W ten sposób na ekranie uzyskujemy sumę danych obrazu i danych P/MG.

Grafika graczy i pocisków nie jest wykorzystywana przez system operacyjny. Pomimo, iż działa ona sprzętowo, to jednak wszystkie czynności przygotowawcze oraz poruszanie obiektów muszą być zaprogramowane. Do tego celu konieczna jest znajomość działania układów ANTIC i GTIA, a przede wszystkim ich rejestrów służących do sterowania grafiką P/M.

8.1. Tworzenie P/MG

Grafika graczy i pocisków, tak jak obraz, musi posiadać obszar pamięci, w którym będą przechowywane dane do umieszczenia na ekranie. Wybór tego obszaru nie jest dowolny, lecz zależy od sposobu odczytu danych przez ANTIC. Ponieważ dostęp ANTIC-a do pamięci jest kontrolowany poprzez rejestr DMACTL, to trzeba zacząć od niego.

Wcześniej było już opisane znaczenie bitów 0-2 i 5-7 rejestru DMACTL (str. 166). Pozostałe trzy bity sterują dostępem do danych P/MG.

Bit 3 kontroluje dostęp ANTIC-a do obszaru pamięci dla pocisków. Gdy jest ustawiony, to dane te są przepisywane z pamięci do rejestru GRAFM (GRAphics For Missile).

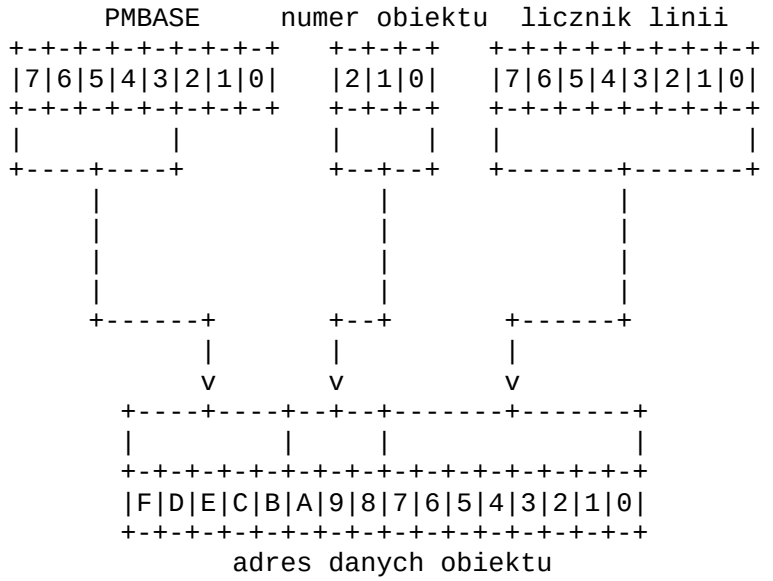
Bit 4 kontroluje dostęp ANTIC-a do obszaru pamięci dla graczy. Gdy jest ustawiony, to dane te

są przepisywane z pamięci do rejestrów GRAFP0-3 (GRAphics For Player 0-3).

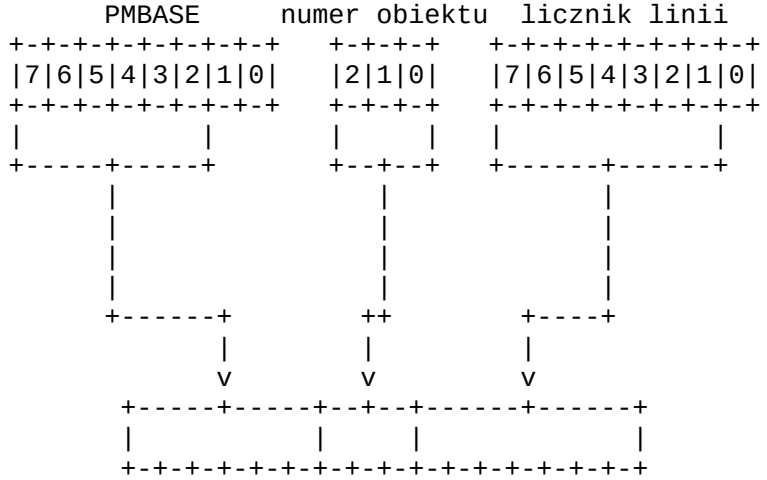
Bit 5 ustala rozdzielczość grafiki P/MG. Gdy jest skasowany, to dane P/MG są pobierane z pamięci podczas tworzenia co drugiej linii ekranu. Każdy pixel obiektu ma więc wysokość dwóch linii ekranu. Po ustawieniu bitu 5 dane są odczytywane przy tworzeniu każdej linii, więc pixel ma wysokość jednej linii ekranu. Nazywa się to odpowiednio rozdzielczością dwuliniową lub jednoliniową.

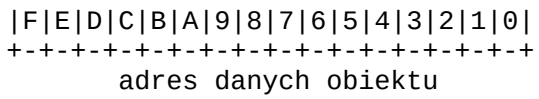
8.1.1. Pamięć P/MG

Bazowy adres pamięci graczy i pocisków jest umieszczany w rejestrze PMBASE (Player/Missile BASE - $D407). Całkowity adres danych obiektu dla konkretnej linii ekranu jest tworzony z trzech elementów: zawartości PMBASE, numeru gracza lub pocisku i licznika linii ekranu. Ponieważ częstość pobierania danych z pamięci zależy od wybranej rozdzielczości, to różny jest sposób zestawienia adresu. Różnice te wyjaśniają rysunki 20 i 21.



Rys.20. Adres obiektu przy rozdzielczości jednoliniowej





Rys.21. Adres obiektu przy rozdzielczości dwuliniowej

Numer obiektu jest wybierany przez ANTIC na drodze sprzętowej według następującego schematu:

```
000 - kombinacja niewykorzystana
001 - kombinacja niewykorzystana
010 - kombinacja niewykorzystana
011 - odczyt danych dla pocisków
100 - odczyt danych dla gracza 0
101 - odczyt danych dla gracza 1
110 - odczyt danych dla gracza 2
111 - odczyt danych dla gracza 3
```

Wynika z tego, że dane poszczególnych obiektów muszą być umieszczone w pamięci następująco:

```
            rozdz. dwuliniowa          rozdz. jednoliniowa
pociski  PMBASE+$0180-PMBASE+$01FF  PMBASE+$0300-PMBASE+$03FF
gracz 0  PMBASE+$0200-PMBASE+$027F  PMBASE+$0400-PMBASE+$04FF
gracz 1  PMBASE+$0280-PMBASE+$02FF  PMBASE+$0500-PMBASE+$05FF
gracz 2  PMBASE+$0300-PMBASE+$037F  PMBASE+$0600-PMBASE+$06FF
gracz 3  PMBASE+$0380-PMBASE+$03FF  PMBASE+$0700-PMBASE+$07FF
```

Pozostawia to wolny obszar pamięci o wielkości 384 lub 768 bajtów. Zwykle pamięć P/MG jest zabezpieczana przed ingerencją systemu operacyjnego (przez zmianę RAMTOP), więc ten obszar można przeznaczyć na umieszczenie procedur w języku maszynowym. Jeżeli zostanie ustawiona rozdzielczość jednoliniowa i nie będą wykorzystywane pociski, to pozostaje wolny obszar 1 KB, w którym można umieścić dodatkowy zestaw znaków.

Widoczne jest także ograniczenie dla wartości PMBASE. Ponieważ nie wszystkie bity tego rejestru są wykorzystywane do tworzenia adresu obiektu, to PMBASE musi wskazywać początek bloku 1 KB przy rozdzielczości dwuliniowej i 2 KB przy jednoliniowej.

Obiekty mogą też być tworzone na ekranie bez pośrednictwa ANTIC-a i bez korzystania z opisanego wyżej obszaru pamięci. Trzeba w tym celu wpisać bajt wzoru bezpośrednio do rejestru grafiki GRAFM lub GRAFP. Każdy rejestr GRAFP odpowiada bajtowi danych gracza, zaś w rejestrze GRAFM każdemu pociskowi odpowiadają dwa bity. Umieszczone w ten sposób dane będą wyświetlane na całym ekranie, otrzymamy więc pionowy pas. Zmienić zawartość rejestrów grafiki, tak aby uzyskać widoczny efekt na ekranie, można tylko podczas tworzenia obrazu, czyli przy pomocy procedury przerwania DLI.

Niezależnie od sposobu zapisywania danych w rejestrach grafiki (bezpośrednio lub przez ANTIC) należy jeszcze zasygnalizować układowi GTIA, że dane z tych rejestrów mają być przeniesione na ekran. Do tego celu służą dwa najmłodsze bity rejestru PMCTL (Player/Missile

ConTroL - $D01D). Bit 0 steruje pociskami, a bit 1 graczami. Ustawienie tych bitów powoduje dodawanie danych gracza lub pocisku do danych tworzonej linii.

8.1.2. Wielkość i kolor obiektów

Kolejny wniosek dotyczy efektu uzyskiwanego na ekranie. ANTIC przenosi po jednym bajcie danych obiektu dla każdej tworzonej linii ekranu. Oznacza to, że można uzyskać obiekt o wysokości całego ekranu i szerokości jednego bajtu (gracz) lub dwóch bitów (pocisk). Szerokość wyrażona w bitach nic jeszcze nie mówi o rzeczywistej szerokości obiektu. Każdy bit reprezentuje jeden pixel obrazu. Pixel ten jest niezależny od aktualnego trybu obrazu.

Wysokość pixela jest wyznaczana przez wybraną rozdzielczość i jest jednakowa dla wszystkich obiektów. Szerokość pixela jest ustalana przez zawartość rejestru SIZEM lub SIZEP. W rejestrach SIZEP0-3 (SIZE Player 0-3) szerokość określają dwa najmłodsze bity (pozostałe są niewykorzystane). W rejestrze SIZEM (SIZE Missiles) każda para bitów określa szerokość pixeli odpowiedniego pocisku: bity 0 i 1 - pocisk 0, bity 2 i 3 - pocisk 1, bity 4 i 5 - pocisk 2 oraz bity 6 i 7 - pocisk 3. Szerokość pixeli jest wyznaczana według schematu:

```
para bitów 00 - pixel o szerokości 1 cyklu koloru
para bitów 01 - pixel o szerokości 2 cykli koloru
para bitów 10 - pixel o szerokości 1 cyklu koloru
para bitów 11 - pixel o szerokości 4 cykli koloru
```

Maksymalna szerokość gracza wynosi więc 6*4=32 cykle koloru. Jest to 1/5 normalnej szerokości obrazu. Przy użyciu czterech graczy i czterech pocisków można całkowicie pokryć obraz.

Wygląd obiektu na ekranie jest określany przez jego dane. Każdy bajt danych określa jedną linię pixeli. Natomiast wygląd poszczególnych pixeli określają bity tego bajtu. Skasowany bit oznacza przezroczysty pixel, czyli pixel w takim kolorze jaki znajduje się w tym miejscu na ekranie. Bit ustawiony powoduje nadanie odpowiadającemu mu pixelowi koloru z rejestru COLPM0-3 (COLor of Player/Missile 0-3). Kolory te są identyczne parami, to znaczy gracz 0 i pocisk 0 mają kolor z rejestru COLPM0, gracz 1 i pocisk 1 z rejestru COLPM1 itd.

Gdy obiekt znajduje się na tle obrazu, nie ma żadnych kłopotów. Problem pojawia się dopiero wtedy, gdy obraz zawiera jakąś treść. Chodzi o to, czy obiekt będzie zasłaniał zawartość obrazu, czy odwrotnie. Nazywamy to priorytetem - kolor o wyższym priorytecie będzie zasłaniał kolory o niższym priorytecie. Do określenia kolejności kolorów służy rejestr GTIACTL (GTIA ConTroL - $D01B). Jego bity 0-3 ustalają 2następujące priorytety kolorów:

bit 0	bit 1	bit 2	bit 3
COLPM0	COLPM0	COLPF0	COLPF0
COLPM1	COLPM1	COLPF1	COLPF1
COLPM2	COLPF0	COLPF2	COLPM0
COLPM3	COLPF1	COLPF3	COLPM1
COLPF0	COLPF2	COLPM0	COLPM2
COLPF1	COLPF3	COLPM1	COLPM3

```
COLPF2      COLPM2      COLPM2      COLPF2
COLPF3      COLPM3      COLPM3      COLPF3
COLBAK      COLBAK      COLBAK      COLBAK
```

Ustawienie jednego z tych bitów ustala określone przez niego priorytety. Ustawienie więcej niż jednego bitu powoduje niejednoznaczne określenie priorytetów. W takim przypadku, gdy obszary o jednakowym priorytecie będą się pokrywały, to uzyskają one kolor czarny. Przy ustalaniu priorytetów trzeba pamiętać, że w niektórych trybach ANTIC-a (2, 3 i F) z rejestru COLPF1 jest pobierana tylko jasność pixela. W tych trybach wszystkie pixele obrazu mają priorytet taki jak COLPF2, niezależnie od aktualnego priorytetu COLPF1.

Przy tworzeniu P/MG wykorzystywane są jeszcze dwa bity rejestru GTIACTL (4 i 5). Ustawienie bitu 4 powoduje nadanie wszystkim pociskom koloru określonego rejestrem COLPF3. Można w ten sposób uzyskać piątego gracza. Każda część takiego gracza jest jednak niezależna - może być poruszana oddzielnie i mieć różną szerokość.

Bit 5 wywołuje po ustawieniu zmianę priorytetów par graczy 0-1 i 2-3. Każdy gracz z takiej pary ma kolor określony odpowiednim rejestrem. Gdy pixele graczy pokrywają się, to kolor wynikowy jest otrzymywany poprzez operację OR wykonaną na bitach kolorów (np. dla graczy 0 i 1 - COLMP0 ORA COLMP1). Także w tym przypadku pozostałe parametry graczy są ustalane odrębnie.

8.1.3. Umieszczenie obiektu na ekranie

Teraz nadeszła już pora przeniesienia obiektu na ekran. Podczas tworzenia kolejnych linii ekranu GTIA odczytuje zawartość rejestrów grafiki GRAFP0-3 i GRAFM i dodaje ją do danych linii. W ten sposób pionowa pozycja obiektu na ekranie odpowiada dokładnie położeniu danych tego obiektu w 128- lub 256-bajtowym obszarze pamięci P/MG.

Wybranie rozdzielczości dwuliniowej uniemożliwia jednak umieszczenie obiektu na ekranie z dokładnością jednej linii ekranu. Aby usunąć tą niedogodność w GTIA został przewidziany rejestr VDELAY (Vertical DELAY - $D01C). Jeżeli zawarty w nim bit jest skasowany, to odpowiadający mu obiekt jest umieszczany na ekranie tak, jak to zostało wcześniej opisane. Natomiast ustawienie bitu powoduje opóźnienie o jedną linię przesyłania danych z rejestrów grafiki na ekran. Dzięki temu cały obiekt jest przesunięty o jedną linię ekranu w dół. Bity rejestru VDELAY odpowiadają kolejno pociskom od 0-3 i graczom 0-3:
bit 0 - pocisk 0 bit 1 - pocisk 1 bit 2 - pocisk 2 bit 3 - pocisk 3 bit 4 - gracz 0 bit 5 - gracz 1 bit 6 - gracz 2 bit 7 - gracz 3 GTIA musi jeszcze wiedzieć, w którym miejscu linii ma umieścić dane obiektu. Do tego celu służy osiem rejestrów poziomej pozycji obiektów - HPOSP0-3 (Horizontal POSition of Player 0-3) i HPOSM0-3 (Horizontal POSition of Missile 0-3). Każdy z nich zawiera numer cyklu koloru, w którym rozpoczyna się przenoszenie na ekran danych odpowiedniego obiektu.

Aby obiekt był widoczny na ekranie, jego pozycja nie może być dowolna, lecz musi mieścić się w pewnych granicach. Dla pionowej pozycji granice te zależą od przyjętej rozdzielczości. Przy rozdzielczości jednoliniowej w obrębie obrazu mieszczą się pozycje od 32 do 224. Dla rozdzielczości dwuliniowej są to pozycje od 16 do 112.

Znacznie większe jest zróżnicowanie pozycji poziomych - zależą one bowiem od szerokości obiektów. Wartości te zawiera poniższa tabela.

obiekt całkowicie:		na obrazie		poza obrazem	
granica od strony:		lewej	prawej	lewej	prawej
szerokość:					
pojedyncza	- gracz	48	200	40	208
	- pocisk	48	206	46	208
podwójna	- gracz	48	192	32	208
	- pocisk	48	204	44	208
poczwórna	- gracz	48	176	16	208
	- pocisk	48	200	40	208

Na zakończenie tego opisu krótka procedura umieszczająca na ekranie gracza 0.

```
0100 ;Player 0 routine
0110 ;
0120 COLPM0S = $02C0
0130 DMACTLS = $022F
0140 GTICTLS = $026F
0150 HPOSP0 = $D000
0160 PMBASE = $D407
0170 PMCTL = $D01D
0180 PMSTRT = $9000
0190 SIZEP0 = $D008
0200 ;
0210     *=  $9000
0220 ;
0230     LDX #$00
0240 LOOP LDA SHAPE,X
0250     STA PMSTRT+$0400+VPOS,X
0260     INX
0270     CPX #$19
0280     BNE LOOP
0290     LDA # >PMSTRT
0300     STA PMBASE
0310     LDY #$02
0320     STY PMCTL
0330     DEY
0340     STY GTICTLS
0350     DEY
0360     STY SIZEP0
0370     LDA HPOS
0380     STA HPOSP0
0390     LDA #$0E
0400     STA COLPM0S
0410     LDA #$3A
0420     STA DMACTLS
0430     RTS
0440 ;
0450 VPOS .BYTE $78
0460 HPOS .BYTE $7A
```

```
0470 SHAPE .BYTE $00,$00,$0C,$18
0480     .BYTE $38,$7C,$54,$7C
0490     .BYTE $28,$38,$38,$3C
0500     .BYTE $7E,$FB,$B9,$38
0510     .BYTE $38,$3C,$1C,$1C
0520     .BYTE $0E,$06,$03,$00,$00
```

8.2. Przemieszczanie obiektów

Po tym, co zostało napisane w poprzednim rozdziale, łatwo już zrozumieć sposób przemieszczania obiektu po ekranie. Trzeba jednak wspomnieć o kilku ograniczeniach i dodatkowych możliwościach.

Rejestry pozycji poziomej obiektów, jak prawie wszystkie rejestry GTIA, są jednokierunkowe - w tym przypadku można jedynie do nich zapisywać. Próba odczytu da nam w efekcie wartość zupełnie bez sensu. Jeśli zatem zachodzi potrzeba zmiany pozycji poziomej obiektu o pewną wartość, to trzeba aktualną zawartość rejestru HPOS... przechowywać w dodatkowym rejestrze RAM. Jest to uwidocznione w zamieszczonych przykładach.

Istnienie rejestrów pozycji poziomej umożliwia uzyskanie praktycznie dowolnej liczby obiektów na ekranie. Jest tylko jedno ograniczenie - w jednej linii ekranu nigdy nie można umieścić więcej niż czterech graczy i cztery pociski. Zwielokrotnienie obiektu wykonuje się przez zapisanie w jego polu pamięci danych dla kilku obiektów. Wszystkie te obiekty będą przy tym położone jeden nad drugim i będą się poruszały razem. Do ich usamodzielnienia należy wykorzystać najczęściej wymienianą w tej książce procedurę - przerwanie DLI. Podczas procedury, która je obsługuje można wpisać do rejestru pozycji nową wartość i następny obiekt znajdzie się w zupełnie innym miejscu. Trzeba jednak pamiętać o jednym - rejestry HPOS... nie są odnawiane podczas przerwania VBLK. Należy więc użyć tej procedury dwukrotnie, a w przypadku większej liczby obiektów - tyle razy, ile obiektów znajduje się w jednym polu danych P/MG.

Więcej kłopotu sprawia przemieszczanie w pionie. Tu konieczne jest przepisywanie danych wewnątrz obszaru pamięci dla danego obiektu. Ponieważ obszar taki nigdy nie przekracza jednej strony pamięci, to napisanie odpowiedniej procedury jest bardzo proste. Dla przesunięcia zawartości całego pola pamięci o jeden bajt w górę przy rozdzielczości jednoliniowej wystarczy taka procedura:

```
0100 ;Move Up
0110 ;
0120     LDX #$01
0130 LOOP LDA PLAYER,X
0140     STA PLAYER-1,X
0150     INX
0160     BNE LOOP
```

Podobna procedura przesuwa pole pamięci o jeden bajt w dół.

```
0100 ;Move Down
0110 ;
```

```
0120     LDX #$FF
0130 LOOP LDA PLAYER-1,X
0140     STA PLAYER,X
0150     DEX
0160     BNE LOOP
```

Dla rozdzielczości dwuliniowej należy tylko zamienić rozkazy BNE LOOP na BPL LOOP i w drugiej procedurze rozkaz LDX #$FF na LDX #$7F.

Gdy przemieszczenie ma dotyczyć tylko części pola (np. przy wykorzystywaniu go dla kilku obiektów), to trzeba wprowadzić ograniczenia pętli LOOP. Zwiększa to długość procedury, lecz skraca czas jej wykonywania. Przykład takiej procedury jest pokazany dalej.

Przy wykorzystywaniu rozdzielczości dwuliniowej przesunięcie obiektu o jeden bajt powoduje przemieszczenie obiektu o dwie linie ekranu. Czasami może to wywoływać wrażenie niezbyt płynnego ruchu. Można to ominąć poprzez naprzemienne kasowanie i ustawianie odpowiedniego bitu w rejestrze VDELAY oraz przesuwanie obiektu. Uzyskuje się wtedy jeszcze większą płynność ruchu.

Teraz dalszy ciąg procedury pokazanej w poprzednim rozdziale (należy to po prostu dopisać do niej usuwając rozkaz RTS). Przesuwa ona utworzony obiekt w zależności od położenia joysticka (w celu praktycznego wykorzystania trzeba ją dołączyć do programu i nieco zmienić zakończenie, aby nie działała w pętli bez końca).

```
0530 ;Player 0 Movement
0540 ;
0550 JOY0  = $0278
0560 ;
0570 MOVE LDA JOY0
0580     EOR #$FF
0590     AND #$0F
0600     BEQ MOVE
0610     CLC
0620     ROR A
0630     BCC E1
0640     LDX #$01
0650 LP1 LDA PMSTRT+$0400,X
0660     STA PMSTRT+$03FF,X
0670     INX
0680     BNE LP1
0690     BEQ MOVE
0700 E1  ROR A
0710     BCC E2
0720     LDX #$FF
0730 LP2 LDA PMSTRT+$03FF,X
0740     STA PMSTRT+$0400,X
0750     DEX
0760     BNE LP2
0770     BEQ MOVE
0780 E2  ROR A
0790     BCC E3
0800     DEC HPOS
0810     JMP HP
```

```
0820 E3  INC HPOS
0830 HP  LDA HPOS
0840     STA HPOSP0
0850     JMP MOVE
```

8.3. Zderzenia między obiektami

Dodatkowym zyskiem z zastosowania grafiki graczy i pocisków jest możliwość sprzętowego wykrywania kolizji. Są to przypadki zetknięcia się poszczególnych obiektów między sobą oraz z elementami obrazu (zwanego tu polem gry), które mają kolor inny niż kolor tła. Ponieważ jest to działanie sprzętowe (wbudowane w strukturę układu GTIA), to użytkownik musi jedynie sprawdzić w odpowiednim rejestrze, czy takie zetknięcie wystąpiło i odpowiednio zareagować.

Można w ten sposób wykrywać wszelkie spotkania obiektów, bez przeprowadzania żmudnych obliczeń sprawdzających, czy dwa pixele należące do różnych obiektów zajmują to samo miejsce na ekranie. W tym celu należy jedynie zbadać zawartość jednego z rejestrów kolizji.

Tych rejestrów kolizji jest w układzie GTIA aż szesnaście. Można je podzielić na cztery grupy: rejestry kolizji pocisków z polem gry (KOLMPF), graczy z polem gry (KOLPPF), pocisków z graczami (KOLMP) i graczy z graczami (KOLPP). Niemożliwe jest w ten sposób jedynie wykrycie kolizji pocisku z innym pociskiem. We wszystkich wymienionych rejestrach do wykrywania kolizji służą bity 0-3, a bity 4-7 są niewykorzystane. Normalnie wszystkie bity są skasowane. Ustawienie bitu oznacza zaistnienie kolizji obiektu, do którego należy ten rejestr, z odpowiednim obiektem lub kolorem pola gry.

Znaczenie poszczególnych bitów w rejestrach wskazujących kolizje obiektów z polem gry (KOLM0-3PF - KOLlision Missile 0-3 to Play Field i KOLP0-3PF - KOLlision Player 0-3 to Play Field) jest następujące:

```
bit 0 - kolizja z obszarem o kolorze z COLPF0
bit 1 - kolizja z obszarem o kolorze z COLPF1
bit 2 - kolizja z obszarem o kolorze z COLPF2
bit 3 - kolizja z obszarem o kolorze z COLPF3
```

Podobnie w rejestrach wskazujących kolizje obiektów z innymi obiektami (KOLM0-3P - KOLlision Missile 0-3 to Player i KOLP0-3P - KOLlision Player 0-3 to Player) poszczególne bity oznaczają:

```
bit 0 - kolizja z graczem 0
bit 1 - kolizja z graczem 1
bit 2 - kolizja z graczem 2
bit 3 - kolizja z graczem 3
```

W przypadku rejestrów kolizji graczy z graczami (KOLPP) bit odpowiadający graczowi, do którego należy ten rejestr jest zawsze skasowany. Na przykład w rejestrze KOLP0P zawsze skasowany jest bit 0, a w KOLP1P bit 1.

Wykrycie kolizji i ustawienie odpowiedniego bitu następuje zawsze dopiero po wyświetleniu linii na ekranie. Aby uniknąć pomyłek najlepiej wszystkie kolizje sprawdzać dopiero po utworzeniu całego obrazu, czyli podczas przerwania VBLK. Jeżeli procedura wywoływana przez zderzenie obiektów jest bardzo długa, to można zastosować przepisywanie zawartości rejestru kolizji do określonego miejsca w pamięci RAM. Jeżeli nie jest istotne natychmiastowe wykrycie kolizji, to sprawdzenie można wykonać w dowolnej chwili, gdyż rejestry kolizji nie są kasowane po przerwaniu zetknięcia.

No tak! Rejestry kolizji można tylko odczytywać, a bity wskazujące kolizje nie są kasowane. Z tego wynika, że są to rejestry jednorazowego użytku. Po wykryciu kolizji trzeba wyłączyć komputer i uruchomić ponownie, aby wykryć następne spotkanie tych samych obiektów. To jakie to ułatwienie? Na szczęście nie jest tak źle. Wszystkie rejestry kolizji są kasowane przez wpisanie dowolnej wartości do rejestru HITCLR (HIT CLeaR - $D01E). Rejestr ten nie służy ponadto do niczego, więc jest obojętne, co zostanie w nim zapisane.

Procedura wykrywania kolizji między obiektami powinna kolejno sprawdzać wszystkie konieczne rejestry kolizji i wywoływać odpowiednie części programu lub odczytane wartości przepisywać do innych komórek. Następnie musi zapisać dowolną wartość do HITCLR, aby umożliwić wskazanie następnych kolizji i już... może się powtarzać od początku. A oto przykład:

```
0100 ;Player 0 Collisions Detection
0110 ;
0120 HITCLR = $D01E
0130 KOLP0PF = $D004
0140 KOLP0P = $D00D
0150 ;
0160     *=  $0600
0170 ;
0180     LDA #$00
0190     LDX #$07
0200 LOOP STA PF0,X
0210     DEX
0220     BPL LOOP
0230     LDY #$FF
0240     LDX #$00
0250     LDA KOLP0PF
0260     AND #$0F
0270     CLC
0280 LP1 ROR A
0290     BCC NX1
0300     STY PF0,X
0310 NX1 INX
0320     CPX #$04
0330     BNE LP1
0340     LDA KOLP0P
0350     AND #$0F
0360     CLC
0370 LP2 ROR A
0380     BCC NX2
0390     STY PF0,X
0400 NX2 INX
0410     CPX #$08
```

```
0420     BNE LP2
0430     STY HITCLR
0440     RTS
0450 ;
0460 PF0 .BYTE $00,$00,$00,$00
0470 PL0 .BYTE $00,$00,$00,$00
```

Rozdział 9

DŹWIĘK

Wszystkie komputery domowe są wyposażone w lepsze lub gorsze układy służące do tworzenia dźwięku. Komputery Atari XL/XE nie są tu wyjątkiem i posiadają cztery niezależne generatory dźwięku. Stanowią one integralną część specjalizowanego układu scalonego POKEY (POtentiometr and KEYboard chip). Do sterowania tymi generatorami POKEY posiada dziewięć rejestrów (tylko do zapisu). Oprócz tego generatory służą jako sprzętowe liczniki i mogą wywoływać przerwania IRQ.

9.1. Generowanie dźwięku

Ponieważ POKEY jest wykorzystywany przede wszystkim do obsługi klawiatury i złącza szeregowego, to podczas każdej operacji wejścia/wyjścia zmieniane są stany rejestrów sterujących generatorami. Aby uzyskać prawidłowe efekty dźwiękowe należy go więc ponownie zainicjować po takiej operacji. Wykonuje się to przez wpisanie wartości zero do rejestru AUDCTL ($D208) i wartości 3 do rejestru SKCTL ($D20F).

Każdy z czterech generatorów dźwięku posiada dwa odrębne rejestry. Jeden z nich (AUDF1-4 - AUDio Frequency 1-4) służy do ustalenia częstotliwości dźwięku, a drugi (AUDC1-4 - AUDio Control 1-4) steruje głośnością i zniekształceniami. Ponadto wszystkie generatory są sterowane poprzez wspólny rejestr AUDCTL (AUDio ConTroL).

Rejestr częstotliwości AUDF jest wykorzystywany jako dzielnik impulsów. POKEY zlicza impulsy sygnału wejściowego doprowadzonego do generatora i przesyła impuls do wyjścia, gdy stan licznika jest zgodny z zawartością rejestru AUDF zwiększoną o jeden. Przy wartości zero w AUDF przepuszczany jest każdy impuls wejściowy, a przy wartości, np. 3 - co czwarty. Dzielnik ma więc zakres od 1 do 256. Ponieważ sygnał wyjściowy składa się z pojedynczych impulsów, to na końcu jest jeszcze dzielony przez dwa przy pomocy przerzutnika. Otrzymuje się wtedy sygnał o wypełnieniu 1:1 (stosunek czasu trwania impulsu do czasu trwania przerwy między impulsami). Ta ostatnia operacja jest konieczna dla uzyskania właściwej barwy dźwięku.

Nieco bardziej skomplikowane jest działanie rejestrów sterujących generatorami - AUDC. Poszczególne bity tych rejestrów mają różne funkcje. Bity 0-3 określają natężenie dźwięku. Ponieważ w czterech bitach można zapisać wartości od 0 do 15, to taka właśnie jest skala uzyskiwanej siły głosu. Bit 4 wyłącza dzielnik częstotliwości. Powoduje to pojawienie się na wyjściu stałego napięcia o wartości określonej przez bity 0-3. Efektem akustycznym jest stuk w głośniku. Szerszy opis wykorzystania tego bitu znajduje się w dalszej części. Pozostałe bity AUDC kontrolują zniekształcenia dźwięku.

Teraz trzeba przyjrzeć się nieco bliżej konstrukcji układu POKEY. Zawiera on trzy rejestry przesuwające pracujące z częstotliwością zegara systemu (1,79 MHz w systemie NTSC i 1,77 MHz w systemie PAL). Mają one długość 4, 5 i 17 bitów, przy czym rejestr 17-bitowy można przełączyć na 9-bitowy. Impulsy z generatora, po przejściu przez dzielnik częstotliwości, lecz przed podziałem przez dwa, są doprowadzane na wejście wybranego rejestru. Pojawiają się na wyjściu z rejestru w tej samej kolejności (rejestr przesuwający działa jak kolejka w sklepie). Nie byłoby w tym nic nadzwyczajnego, gdyby nie dodatkowa operacja. Bity pobierane są nie tylko z wyjścia, lecz także ze środka rejestru przesuwającego. Bity te poddawane są operacji logicznej NOR i ponownie przesyłane do wejścia rejestru. Powoduje to uzyskanie zakłóceń pseudolosowych, których powtarzalność zależy od długości rejestru.

Po tym wyjaśnieniu łatwo będzie zrozumieć działanie trzech najstarszych bitów (5-7) rejestru AUDC. Służą one do wyboru rejestru przesuwającego, który będzie użyty do zniekształcenia dźwięku. Kolejne kombinacje tych bitów mają następujące znaczenie:

```
bity 000 - rejestr 5-bitowy i 17-bitowy
bity 001 - rejestr 5-bitowy
bity 010 - rejestr 5-bitowy i 4-bitowy
bity 011 - rejestr 5-bitowy
bity 100 - rejestr 17-bitowy
bity 101 - bez rejestru przesuwającego
bity 110 - rejestr 4-bitowy
bity 111 - bez rejestru przesuwającego
```

Pomimo użycia trzech bitów kombinacji jest tylko sześć, gdyż pary 001 i 011 oraz 101 i 111 są identyczne. Poniższa tabela (zaczerpnięta z "De Re Atari") przedstawia rodzaje dźwięku uzyskiwane przy różnych częstotliwościach i różnych wartościach bitów 5-7 w AUDC.

```
kombinacja                  częstotliwości
bitów 5-7      niskie           średnie          wysokie
   000     licznik Geigera   pęd powietrza      strumień
   0x1     karabin maszyn.   silnik elektr.   transformator
   010        ognisko           samochód      wizg silnika
   100     walący się dom   zakłócenia radiowe   wodospad
   1x1              c z y s t e   d ź w i ę k i
   110         samolot          kosiarka          golarka
```

Pozostało jeszcze do opisania znaczenie bitu 4. Ponieważ steruje on wyłącznie siłą głosu, to można przede wszystkim generować przy jego pomocy rytm. Częstotliwość tego rytmu jest dowolna - od ułamków Hertza aż do kilkuset kHz. Jednak zastosowanie tego bitu jest znacznie szersze. Po jego ustawieniu głośnik otrzymuje stałe napięcie określone przez bity 0-3. Można więc generować dowolny dźwięk przez zmianę tylko i wyłącznie głośności. Wymaga to oczywiście precyzyjnego odliczania czasu, lecz nie jest to żaden problem.

Na przykład instrumenty dęte blaszane emitują falę dźwiękową o przebiegu trójkątnym. Normalnie dźwięk wytwarzany przez komputer ma przebieg prostokątny, różnica barwy jest więc bardzo wyraźna. Wygenerowanie przebiegu trójkątnego wymaga jedynie ustawienia bitu 4 i cyklicznego wpisywania do bitów 0-3 wartości rosnących liniowo od 0 do 15, a następnie

malejących od 15 do 0 (razem 16-31 i 31-16). Jest to bardzo proste, lecz Atari Basic jest zbyt wolny i trzeba to wykonać w języku maszynowym (oczywiście procedurę maszynową można także dołączyć do programu w Basicu).

Ostatni rejestr dźwięku POKEY-a - AUDCTL - steruje jednocześnie wszystkimi generatorami. Jego użycie znacznie zwiększa możliwości dźwiękowe Atari. Poszczególne bity tego rejestru mają następujące znaczenie:

Bit 0 służy do wyboru zegara bazowego, czyli źródła impulsów dla wszystkich generatorów. Normalnie zegar bazowy ma częstotliwość 64 kHz, a po ustawieniu bitu 0 jest przełączany na 15 kHz. Jeśli pozostałe parametry pozostaną niezmienione, to spowoduje to czterokrotne zmniejszenie częstotliwości wszystkich dźwięków.

Ustawienie bitu 1 włącza filtr górnoprzepustowy w generatorze 2. Filtr ten jest sterowany generatorem 4 i powoduje, że generator 2 emituje tylko dźwięki o częstotliwościach wyższych od częstotliwości generatora 4.

Bit 2 włącza analogiczny filtr górnoprzepustowy dla generatora 1 sterowany generatorem 3.

Bit 3, gdy jest ustawiony, łączy dzielniki generatorów 3 i 4 w jeden dzielnik 16-bitowy, który umożliwia bardziej precyzyjną regulację częstotliwości. Poszerza to również zakres uzyskiwanych częstotliwości ze 125 Hz - 32 kHz do 0,5 Hz - 32 kHz, a po zmianie zegara bazowego przy pomocy bitu 5 na 16 Hz - 1 MHz.

Bit 4 łączy dzielniki 1 i 2 w jeden dzielnik 16-bitowy analogicznie jak bit 3.

Bit 5 przełącza (gdy jest ustawiony) zegar bazowy dla generatora 3 na częstotliwość 1.77 MHz (wszystkie źródła podają tu częstotliwość 1,79 MHZ - jest to częstotliwość kwarcowego zegara systemu, ale w NTSC! - komputery w wersji PAL mają zegar 1,77MHz). Przełączenie to powoduje zwiększenie częstotliwości ponad 30 razy. Tak wysokie dźwięki nie nadają się do słuchania - wykorzystuje się tą możliwość prawie zawsze jednocześnie z ustawieniem bitu 3.

Bit 6 przełącza zegar bazowy dla generatora 1 na częstotliwość 1,77MHz, podobnie jak bit 5 (wszystkie podane tam uwagi odnoszą się także do tego bitu).

Bit 7 przełącza 17-bitowy rejestr przesuwający na rejestr 9-bitowy. Pozwala to na uzyskanie dwóch dodatkowych sposobów zniekształcenia dźwięku (dla bitów 5-7 w AUDC równych 000 lub 100).

Ubocznym efektem działania generatorów dźwięku jest liczba pseudolosowa znajdująca się w rejestrze RANDOM ($D20A). Umieszczane jest tam osiem najstarszych bitów z 17-bitowego

rejestru przesuwającego. Przez odczyt tego rejestru można otrzymać przypadkową wartość z zakresu od 0 do 255. Możliwość ta jest często wykorzystywana, szczególnie przez programy napisane w języku maszynowym.

9.2. Liczniki POKEY-a

Wspomniane wcześniej liczniki POKEY-a są jedną z najrzadziej wykorzystywanych możliwości Atari, warto więc opisać bliżej ich działanie, choć odbiega to od głównego tematu rozdziału. Opis ten dotyczy liczników 1, 2 i 4 - licznik 3 nie jest używany w systemie operacyjnym (w taki sposób).

Po ustawieniu i uruchomieniu liczniki zliczają wstecz, to znaczy od wartości ustawionej do zera. W chwili wyzerowania licznika POKEY generuje sygnał żądania przerwania maskowalnego IRQ. Każdemu z liczników odpowiada jeden bit (0, 1 i 2) w rejestrze zezwoleń przerwań IRQEN oraz w rejestrze sygnalizacji przerwań IRQST. Po rozpoznaniu źródła przerwania CPU wywołuje procedurę przerwania, której wektor jest umieszczony odpowiednio w rejestrze VTIMR1 ($0210), VTIMR2 ($0212) lub VTIMR4 ($0214). A oto kolejność postępowania, konieczna dla wykorzystania tych przerwań.

Najpierw trzeba ustawić w rejestrze AUDCTL wymaganą częstotliwość zegara bazowego. Następnie w rejestrze AUDC odpowiadającym wykorzystywanemu licznikowi należy umieścić zero. Początkowy stan licznika wpisuje się do rejestru AUDF. Oczywiście w pamięci musi znajdować się procedura obsługi przerwania, a odpowiedni wektor VTIMR (Vector TIMeR) musi zawierać jej adres. Także właściwe bity w IRQEN i IRQENS muszą zezwalać na przerwanie.

Teraz wpisanie dowolnej wartości do rejestru STIMER (Start TIMERs - $D209) powoduje umieszczenie zawartości rejestrów AUDF w licznikach i ich uruchomienie. Ponieważ liczniki POKEY-a zliczają z częstotliwością znacznie większą niż pozostałe zegary systemu, to w ten sposób można uzyskać odliczanie czasu krótszego od 1/50 sekundy. Dla czasów dłuższych wykorzystywane są przerwania synchronizacji pionowej VBLK oraz liczników TIMCNT1-5.

Rozdział 10

OBSŁUGA MANIPULATORÓW

W każdym komputerze ważną rolę pełnią manipulatory. Szczególnie ważne jest ich zastosowanie w komputerach domowych, gdzie służą do gier. Zasadniczo można wyróżnić dwa rodzaje manipulatorów stosowanych w sprzęcie komputerowym: manipulatory analogowe i manipulatory cyfrowe.

Sygnałem z manipulatora analogowego jest napięcie. W najprostszych rozwiązaniach uzyskuje się to przez włączenie między bieguny zasilania komputera potencjometru. Rozwiązaniem bardziej skomplikowanym może być przetwornik zamieniający np. szybkość obrotową na napięcie. Do manipulatorów analogowych zalicza się potencjometry (paddle), myszy i tabliczki graficzne.

Całkowicie odmienne jest działanie manipulatora cyfrowego. Zawiera on kilka styków, które mogą mieć tylko dwa stany - zwarty i rozwarty. Najczęściej spotykanym rozwiązaniem jest takie, w którym styk ma napięcie 5 V i jest zwierany do masy. Ten zespół styków jest dołączony do jednego z układów komputera, skąd położenie poszczególnych styków jest odczytywane jako stany bitów rejestru. Manipulatorem cyfrowym jest przede wszystkim joystick. Ponadto w manipulatorach analogowych stosuje się jeden lub dwa przyciski, których działanie jest oparte na zasadzie zwierania styków, a więc cyfrowe.

Komputery Atari 400/800 miały po cztery gniazda manipulatorów. Do każdego z tych gniazd można było dołączyć joystick lub dwa potencjometry. W sumie daje to ogromną liczbę czterech joysticków lub ośmiu potencjometrów. Dało tu o sobie znać pochodzenie komputerów Atari od gier telewizyjnych. W nowych modelach XL i XE liczba gniazd została zredukowana do przyzwoitej liczby dwóch, lecz ze względu na zachowanie zgodności oprogramowania pozostały rejestry w pamięci RAM.

Ponieważ manipulatory dwóch wymienionych wyżej typów mają zupełnie odmienne działanie, to są one obsługiwane przez różne układy komputera. Są to trzy układy, a nie dwa, jak można by przypuszczać, gdyż przyciski joysticków są obsługiwane oddzielnie.

10.1. Potencjometry

Do dwóch gniazd manipulatorów można dołączyć w Atari cztery potencjometry (parami). Sygnał analogowy (napięcie) z gniazda jest przesyłany do układu POKEY i tam poddawany konwersji na postać cyfrową. Wykonuje tą operację specjalny przetwornik analogowo-cyfrowy (A/DC - Analog/Digital Converter).

Przetworniki te sprawiają konstruktorom komputerów dużo kłopotów i to nie ze względów

technicznych. Osiągnięcie nawet bardzo dużej dokładności nie stanowi żadnego problemu. Wymaga za to czasu. Doprowadzany sygnał jest kolejno porównywany z napięciami wzorcowymi o rosnących wartościach. Czym więcej napięć wzorcowych, tym dokładniej określone jest napięcie sygnału, ale także tym dłużej trwa pomiar. Największą trudnością jest znalezienie właściwego kompromisu pomiędzy dokładnością i szybkością przetwornika.

Rozwiązanie zastosowane w Atari jest następujące: POKEY mierzy jednocześnie osiem sygnałów wejściowych (modele XL i XE wykorzystują tylko cztery z nich). Ponieważ wykonuje to co 1/50 sekundy (z częstotliwością synchronizacji obrazu - 50 Hz), to w tym czasie możliwe jest porównanie każdego sygnału z 228 napięciami wzorcowymi. Zamiana sygnału analogowego na cyfrowy daje więc w rezultacie wartości z zakresu od 0 do 228.

Obliczone wartości mogą być odczytane z rejestrów POT0-7 (POTentiometr 0-7 - $D200-$D207). Niestety w języku maszynowym nie jest to proste (Atari Basic i większość języków wyższego poziomu sama wykonuje opisane niżej czynności). Rejestry te bowiem są jednocześnie licznikami sprzętowymi układu POKEY. Ich stan jest stopniowo zmniejszany, a po wyzerowaniu któregokolwiek wywoływane jest przerwanie IRQ (TIMER 1, 2 i 4), natomiast licznik jest ponownie ustawiany na 228 lub według zawartości rejestrów-cieni. Dzięki temu mogą one być wykorzystane do zliczania okresów czasu krótszych od częstotliwości obrazu (50 Hz, czyli 1/50 sekundy) - jest to opisane w poprzednim rozdziale.

Prawidłowy odczyt stanu potencjometru umożliwiają dwa dodatkowe rejestry. Rejestr POTGO (POTentiometrs GOes - $D20B) uruchamia odczyt i konwersję sygnału wejściowego po wpisaniu do niego dowolnej wartości. Zakończenie zamiany sygnału analogowego na cyfrowy jest sygnalizowane skasowaniem bitu w rejestrze POTSTAT (POTentiometrs STATus - $D208). Każdemu bitowi tego rejestru (0-7) odpowiada potencjometr o tym samym numerze. Ustawienie bitu w POTSTAT oznacza więc, że wartość odpowiadającego mu potencjometru jest jeszcze obliczana i odczyt z POT... da nieprawidłowy wynik.

Jeżeli znacznie ważniejsza od dokładności jest szybkość przetwarzania sygnału, to można ją zmienić poprzez rejestr SKCTL (Serial and Keyboard ConTroL - $D20F). Gdy bit 2 tego rejestru jest skasowany (stan normalny), to przetwarzanie sygnału trwa 20 milisekund, co odpowiada utworzeniu na ekranie 228 linii. Po ustawieniu tego bitu czas przetwarzania skraca się do 128 mikrosekund (dwie linie ekranu), jednak kosztem znacznego zmniejszenia dokładności pomiaru. Przy zmianie zawartości tego rejestru trzeba pamiętać, aby nie zmienić innych, wcześniej ustawionych bitów (normalnie wszystkie są skasowane).

Stosowane do gier potencjometry (tzw. paddle) są ponadto wyposażone w przyciski - każdy paddle ma jeden przycisk. Sygnały z tych przycisków są dołączone do układu obsługującego joysticki (zob. rozdział następny). Odpowiadają one przesunięciom joysticka w lewo i w prawo:

```
przycisk potencjometru 0 = joystick 0 w lewo
```

```
przycisk potencjometru 1 = joystick 0 w prawo
przycisk potencjometru 2 = joystick 1 w lewo
przycisk potencjometru 3 = joystick 1 w prawo
przycisk potencjometru 4 = joystick 2 w lewo
przycisk potencjometru 5 = joystick 2 w prawo
przycisk potencjometru 6 = joystick 3 w lewo
przycisk potencjometru 7 = joystick 3 w prawo
```

Potencjometry 4-7 oraz joysticki 2 i 3 występują tylko w komputerach Atari 400/800.

10.2 Joysticki

Wykorzystanie joysticków jest znacznie prostsze ponieważ uzyskiwany jest z nich sygnał cyfrowy. W Atari obsługą joysticków zajmuje się układ scalony PIA (typowy układ 6520 lub 6820), a właściwie jego port A, gdyż PIA składa się z dwóch identycznych, niezależnych od siebie części. W modelach 400/800 port B obsługiwał dwa dalsze gniazda joysticków.

Joystick ma cztery zasadnicze położenia (oprócz neutralnego): w lewo, w prawo, naprzód i wstecz. Pozostałe położenia są ich kombinacją. Do cyfrowego przedstawienia tych położeń wystarczą cztery bity, dla dwóch joysticków potrzebny jest więc jeden rejestr 8-bitowy. Takim rejestrem jest właśnie PORTA (PORT A - $D300). Jego poszczególne bity są przypisane następującym położeniom joysticka:

```
bit 0 - joystick 0 naprzód
bit 1 - joystick 0 wstecz
bit 2 - joystick 0 w lewo
bit 3 - joystick 0 w prawo
bit 4 - joystick 1 naprzód
bit 5 - joystick 1 wstecz
bit 6 - joystick 1 w lewo
bit 7 - joystick 1 w prawo
```

Jak wynika z wstępnego opisu, normalnie wszystkie bity są ustawione. Wykonanie ruchu joystickiem powoduje skasowanie odpowiedniego bitu (lub bitów). Języki wyższego poziomu odczytują położenie joysticka z rejestrów-cieni - $0278 (joystick 0) lub $0279 (joystick 1). Odczyt położenia bezpośrednio z PORTA stosowany przeważnie w języku maszynowym wymaga natomiast wykonania kilku operacji. Przedstawia to poniższa procedura (analogicznie przeprowadzane jest przepisanie zawartości PORTA do rejestrów-cieni podczas przerwania VBLK).

```
0100 ;Read Joystick's Movements
0110 ;
0120 PORTA = $D300
0130 ;
0140     *=  $0600
0150 ;
0160 ;Joystick 0
0170 ;
0180 JOY0 LDA PORTA
0190     AND #$0F
0200     RTS
0210 ;
0220 ;Joystick 1
```

```
0230 ;
0240 JOY1 LDA PORTA
0250      LSR A
0260      LSR A
0270      LSR A
0280      LSR A
0290      RTS
```

Dla uproszczenia późniejszego rozpoznania położenia joysticka zwykle po rozkazie LDA PORTA stosuje się jeszcze rozkaz EOR #$FF, który zamienia wartości wszystkich bitów na przeciwne. Teraz sprawdzenie konkretnego bitu jest wykonywane przez sekwencję rozkazów AND, BNE. Na przykład ruch w lewo wykrywany jest rozkazami AND #$04, BNE LEFT. Jeżeli bit 2 był ustawiony (po EOR #$FF oznacza to ruch w lewo), wykonywany jest skok do miejsca oznaczonego etykietą LEFT. Podobna metoda została zastosowana w procedurze zamieszczonej na stronie 194. Oczywiście można także sprawdzać po kilka bitów jednocześnie (np. AND #$06 wykrywa ruch w lewo i wstecz).

Podobnie jak potencjometry, także każdy joystick wyposażony jest w przycisk (najczęściej jest ich kilka, lecz połączone są razem). Do wykrywania ich stanu służą rejestry układu GTIA (w PIA zabrakło miejsca) - TRIG0 i TRIG1 (TRIGger 0-1). W rejestrach tych bity 1-7 są niewykorzystywane, a stan przycisku jest sygnalizowany jedynie przez bit 0. Normalnie bit ten jest ustawiony, a jego skasowanie oznacza, że przycisk został wciśnięty.

Wszystkie wymienione wyżej rejestry mają swoje kopie w pamięci RAM. Zawartości ich są uaktualniane podczas przerwania synchronizacji pionowej VBLK. Spojrzenie na mapę pamięci ujawnia, że rejestrów-cieni jest dwa razy więcej! Jest to pozostałość po modelach 400/800 utrzymana w celu zachowania zgodności oprogramowania. Przy przepisywaniu do rejestrów-cieni wartości dotyczące joysticka 0 są umieszczane w rejestrach joysticków 0 i 2, a joysticka 1 w rejestrach 1 i 3. Dotyczy to zarówno położenia joysticka, jak i stanu jego przycisku.

10.3. Wyjście z gniazd joysticków

Układ PIA, do którego dołączone są gniazda joysticków, jest programowalnym układem wejścia/wyjścia. Można więc, przeprogramować go tak, aby zamiast odczytywać sygnał z tego gniazda, zapisywał go tam. Do tego celu trzeba wykorzystać rejestr PACTL (Port A ConTroL - $D302).

Układ PIA ma wiele funkcji, które są sterowane między innymi przez PACTL. Trzeba więc uważać, aby nie wywołać zamieszania przy ingerowaniu w jego zawartość. Jedynym interesującym nas bitem tego rejestru jest bowiem bit 2. Gdy jest on ustawiony, rejestr PORTA działa jako rejestr przesyłania danych. Oznacza to, że sygnał doprowadzony z jednej strony rejestru jest przekazywany na drugą. Specjalnie używam enigmatycznych określeń stron: "jedna" i "druga", gdyż kierunek przesyłania jest zmienny. Skasowanie bitu 2 zamienia rejestr PORTA w rejestr porządkowania danych. Teraz wpisanie jakiejś wartości do tego rejestru ustawia kierunek przesyłania danych i to

oddzielnie dla każdego bitu! Jeżeli bit wpisanej wartości jest skasowany, to ten bit rejestru PORTA będzie działał jako wejście (odczyt sygnału z gniazda joysticka). Ustawiony bit wpisanej wartości ustawia bit rejestru PORTA jako wyjście (zapis sygnału na gniazdo joysticka). Oto przykładowe wartości:

```
$00 (bin 00000000) - wszystkie bity jako wejście
$FF (bin 11111111) - wszystkie bity jako wyjście
$F0 (bin 11110000) - gniazdo  joysticka  0  jako wejście, a
                     joysticka 1 jako wyjście
$AA (bin 10101010) - bity  parzyste  (0,  2,  4  i  6) jako
                     wejście, bity nieparzyste (1, 3, 5 i 7) jako wyjście
```

Po odpowiednim ustawieniu portu należy odtworzyć jego właściwą funkcję przez ustawienie bitu 2 w PACTL. Wszystkie te operacje można przeprowadzić nawet z poziomu Basica, a w języku maszynowym na przykład tak:

```
0100 DIR = $xx  ;kierunek transmisji
0110 PACTL = $D302
0120 PORTA = $D300
0130 ;
0140     LDY #DIR
0150     LDA PACTL
0160     AND #$FB
0170     STA PACTL
0180     STY PORTA
0190     ORA #$04
0200     STA PACTL
0210     RTS
```

Sygnały wejściowe i wyjściowe w gniazdach joysticków odpowiadają standardowi TTL. Układ PIA jest dodatkowo separowany od gniazd rezystorami 2,2 kOhma, co zabezpiecza go przed przeciążeniem prądowym. Bardzo łatwe jest więc zaprojektowanie układu sterującego dowolnym urządzeniem, a nawet kilkoma urządzeniami. Można też zastosować komputer jako inteligentny sterownik, który będzie reagował na impulsy z urządzenia wysyłaniem do niego odpowiednich poleceń. Stworzono tu ogromne pole do popisu dla inwencji użytkownika. Warto jednak pamiętać, że komputer jest urządzeniem delikatnym i drogim. Przy stosowaniu napięć wyższych od 5 V należy stosować galwaniczne oddzielenie obwodów za pomocą transoptorów lub przekaźników. Niektórzy zalecają nawet oba sposoby jednocześnie i stopniowanie napięcia: 5 V, 12 V i dopiero 220 V.

Zamiast podsumowania przedstawiam zaczerpnięty z miesięcznika "Atari User" program, który jest przykładem wykorzystania portu A (a jednocześnie przykładem własnej procedury obsługi przerwania).

```
0100 ;Joystick Driver
0110 ;for device "J:"
0120 ;
0130 ICAX1Z = $2A
0140 ICAX2Z = $2B
0150 BOOT? = $09
0160 PACTL = $D302
```

```
0170 PORTA = $D300
0180 HATABS = $031A
0190 ;
0200     *=  $0600
0210 ;
0220 ;construct HATABS entry
0230 ;
0240     PLA
0250 JINIT LDX #$00
0260 NEXTENT LDA HATABS,X
0270     BEQ TABENT
0280     INX
0290     INX
0300     INX
0310     CPX #$22
0320     BCS NOROOM
0330     JMP NEXTENT
0340 NOROOM BRK
0350 ;
0360 ;now to put the entry in
0370 ;
0380 TABENT LDA #'J
0390     STA HATABS,X
0400     INX
0410     LDA # <JDRIVER
0420     STA HATABS,X
0430     INX
0440     LDA # >JDRIVER
0450     STA HATABS,X
0460     RTS
0470 ;
0480 ;vector table
0490 ;
0500 JDRIVER .WORD OPEN-1
0510     .WORD CLOSE-1
0520     .WORD GET-1
0530     .WORD PUT-1
0540     .WORD STATUS-1
0550     .WORD SPEC-1
0560     JMP INIT
0570 ;
0580 ;open routines
0590 ;
0600 OPEN LDA ICAX1Z
0610     CMP #$0C
0620     BEQ IOOP
0630     CMP #$08
0640     BEQ OPOP
0650     CMP #$04
0660     BEQ IPOP
0670     LDY #$92
0680     RTS
0690 ;
0700 ;open for INPUT/OUTPUT
0710 ;
0720 IOOP LDX ICAX2Z
0730 PATCH LDA #$38
0740     STA PACTL
0750     STX PORTA
0760     LDA #$3C
```

```
0770     STA PACTL
0780 OK  LDY #$01
0790 INIT RTS
0800 ;
0810 ;open for OUTPUT only
0820 ;
0830 OPOP LDX #$FF
0840     JMP PATCH
0850 ;
0860 ;open for INPUT only
0870 ;
0880 IPOP LDX #$00
0890     JMP PATCH
0900 ;
0910 ;normal is INPUT, so...
0920 ;
0930 CLOSE JMP IPOP
0940 ;
0950 ;put a byte to port
0960 ;
0970 PUT STA PORTA
0980     JMP OK
0990 ;
1000 ;get a byte from port
1010 ;
1020 GET LDA PORTA
1030     JMP OK
1040 ;
1050 ;function not implemented
1060 ;
1070 STATUS
1080 SPEC RTS
```

Program ten instaluje w tabeli HATABS urządzenie o nazwie "J:". Może być ono otwarte do odczytu, zapisu lub odczytu i zapisu. W tym ostatnim przypadku można dowolnie ustalić, które styki będą wejściem, a które wyjściem. Po zainstalowaniu urządzenie można wykorzystywać z poziomu Basica. Dostępne są wtedy następujące instrukcje:

```
OPEN #n,4,0,"J:" - otwarcie do odczytu
OPEN #n,8,0,"J:" - otwarcie do zapisu
OPEN #n,12,x,"J:" - otwarcie do zapisu i odczytu (x określa
                    bity wejściowe i wyjściowe)
GET #n,A - odczyt bajtu
PUT #n,A - zapis bajtu
CLOSE #n - zamknięcie;  ustawia  PORTA do odczytu (normalny
           stan portu)
```

Rozdział 11

ZARZĄDZANIE PAMIĘCIĄ

Druga połówka układu PIA - port B - wykorzystywana była w starych modelach do obsługi gniazd joysticków 2 i 3. Usunięcie tych joysticków nie zostało podyktowane nadmiarem manipulatorów.

Podczas kolejnych modernizacji systemu komputerowego przede wszystkim przybywało pamięci:

```
Atari 400 - 16 KB RAM i 10 KB ROM, razem 26 KB
Atari 800 - 48 KB RAM i 10 KB ROM, razem 64 KB
Atari 600XL - 16 KB RAM i 24 KB ROM, razem 40 KB
Atari 800XL - 64 KB RAM i 24 KB ROM, razem 88 KB
Atari 65XE - 64 KB RAM i 24 KB ROM, razem 88 KB
Atari 130XE - 128 KB RAM i 24 KB ROM, razem 152 KB
```

Początkowo planowano jeszcze model 260XE z pamięcią 256 KB RAM i 24 KB ROM.

Niestety przestrzeń adresowa (możliwy do zaadresowania obszar pamięci) mikroprocesorów 6502 i ANTIC wynosi tylko 64 KB i w żaden sposób nie można jej zwiększyć. Jedynym sposobem umożliwienia dostępu do zwiększonej przestrzeni jest podzielenie jej na części (tzw. banki) i dołączanie ich w miarę potrzeby. Funkcję takiego przełącznika banków pełni specjalny układ logiczny MMU (Memory Management Unit).

Do sterowania MMU potrzebny jest jednak rejestr w już istniejącej strukturze komputera. Do tego celu wykorzystano właśnie PORTB (Port B - $D302) układu PIA, co wymusiło rezygnację z dwóch gniazd joysticków.

Rejestr PBCTL (Port B ConTroL - $D303) sterujący między innymi pracą PORTB jest teraz w tym zakresie nieużyteczny. Przy wykorzystywaniu innych jego funkcji należy jednak uważać, aby nie zmieniać wartości bitu 2. Spowoduje to bowiem nieuchronne zawieszenie systemu operacyjnego.

Ponadto niektóre operacje przełączania pamięci są wykonywane sprzętowo na podstawie sygnałów doprowadzanych do odpowiednich styków zewnętrznych złączy komputera. Umożliwia to dołączenie cartridge'ów i nowych urządzeń.

W rodzinie komputerów Atari jest jeszcze jeden (niezbyt udany) model - 1200XL. W tym modelu bity 2 i 3 portu B zostały użyte do sterowania diodami elektroluminescencyjnymi. Gdy jeden z tych bitów jest skasowany, to dioda LED świeci się, w przeciwnym razie jest zgaszona. Dioda LED 1

sterowana bitem 2 oznacza zablokowanie klawiatury. Dioda LED 2 (bit 3) oznacza korzystanie z międzynarodowego zestawu znaków.

11.1. Atari 65XE/800XL

Modele 800XL i 65XE mają po 64 KB pamięci RAM. Obszar pamięci ROM musi się więc pokrywać z częścią RAM (ma takie same adresy). Cały obszar ROM można podzielić funkcjonalnie na trzy podstawowe zespoły: system operacyjny, interpreter Atari Basic i program testujący (selftest).

Dla przełączania trzech obszarów wystarczą trzy bity, więc tylko taka część PORTB jest wykorzystywana. Normalna konfiguracja systemu komputerowego jest następująca: program testujący wyłączony, interpreter Atari Basic wyłączony, system operacyjny włączony. Natomiast normalną zawartością rejestru PORTB jest $FF - wszystkie bity ustawione. Od razu widać, że sterowanie obszarem ROM zawierającym OS musi być odwrotne niż pozostałymi częściami ROM.

Czas już na szczegóły. W rejestrze PORTB wykorzystywane są tylko bity 0, 1 i 7. Ich przyporządkowanie jest następujące:

Bit 0 steruje obszarem ROM zawierającym system operacyjny (adresy $C000-$CFFF oraz $D800-$FFFF). Jego skasowanie powoduje odłączenie OS. Jeśli w tym obszarze RAM nie ma innego systemu operacyjnego, to komputer zawiesi się podczas najbliższego przerwania synchronizacji pionowej. Aby tego uniknąć, trzeba zablokować przerwania NMI i IRQ w odpowiednich rejestrach. Z tej możliwości korzysta wiele programów. Używają one własnego systemu operacyjnego lub modyfikują istniejący system po uprzednim przepisaniu go do pamięci RAM. Ponadto obszar pamietu procedur zmiennoprzecinkowych ($D800-$DFFF) może zostać odłączony na drodze sprzętowej i w to miejsce podstawiany jest obszar pamięci przyłączony do szyny równoległej. Trzeba w tym celu podać sygnał niski (zewrzeć do masy) styk MPD szyny równoległej (800XL - styk 43 w Parallel Bus, 130XE - styk 4 w Enhanced Cartridge Interface, w 65XE jest to niemożliwe).

Bit 1 steruje obszarem ROM zawierającym interpreter Atari Basic ($A000-$BFFF). Jego skasowanie powoduje dołączenie interpretera. W przeciwnym przypadku w tym obszarze adresowym znajduje się blok 8 KB RAM. Podanie wysokiego sygnału (5 V) na styk En5 w szczelinie cartridge'a (styk 14) powoduje odłączenie zarówno interpretera, jak i pamięci RAM oraz umieszczenie w tym obszarze adresowym pamięci cartridge'a. Analogicznie sygnał wysoki podany na styk En4 (styk A) odłącza RAM w obszarze $8000-$9FFF.

Bit 7 steruje obszarem ROM zawierającym program testujący. Fizycznie zajmuje on obszar od $D000 do $D7FF. Po skasowaniu bitu 7,oprócz uaktywnienia tego obszaru,przełączane są jeszcze linie adresowe. W ten sposób program testujący jest przenoszony do obszaru adresowego od $5000 do $57FF.

11.2. Atari 130XE

W modelu 130XE został podwojony obszar pamięci RAM. Dodatkowe 64 KB RAM są podzielone na cztery banki (0-3) po 16 KB każdy. Można je dołączyć w miejsce bloku 16 KB standardowej pamięci RAM ($4000-$7FFF). Ponieważ fizycznie nie zwiększa to dostępnego obszaru pamięci zastosowano jeszcze możliwość indywidualnego przełączania banków pamięci dla obu procesorów (CPU i ANTIC). Do realizacji tych funkcji wykorzystywane są cztery dalsze bity PORTB (bity 0, 1 i 7 mają znaczenie takie jak w 800XL i 65XE).

Bity 2 i 3 służą do wyboru banku dodatkowej pamięci, który będzie umieszczony w obszarze $4000-$7FFF. Wszystkie dodatkowe banki mają te same adresy, choć niektóre źródła podają to inaczej. Wartość tej pary bitów odpowiada numerowi przyłączonego banku:

```
para bitów 00 - bank 0
para bitów 01 - bank 1
para bitów 10 - bank 2
para bitów 11 - bank 3
```

Jednak sama zmiana numeru banku nic jeszcze nie zmienia w konfiguracji systemu. Aby umożliwić dostęp do dodatkowej pamięci trzeba wykorzystać dwa następne bity w PORTB.

Bity 4 i 5 służą do ustalenia dostępu do dodatkowej pamięci. Bit 4 steruje dostępem CPU, a bit 5 dostępem ANTIC-a. Jeżeli bit jest ustawiony, to procesor, któremu bit ten odpowiada, korzysta z pamięci podstawowej. Po skasowaniu bitu część przestrzeni adresowej procesora jest ustawiana na dodatkowy bank RAM o numerze określonym przez bity 2 i 3. Gdy bity 4 i 5 są ustawione, to niezależnie od wartości bitów 2 i 3 oba procesory korzystają z pamięci podstawowej. W tym trybie pracy 130XE jest w pełni zgodny z 65XE i 800XL.

bit 4	bit 5	pamięć CPU	pamięć ANTIC-a
0	0	dodatkowa	dodatkowa
0	1	dodatkowa	główna
1	0	główna	dodatkowa
1	1	główna	główna

Korzystanie z dodatkowej pamięci 130XE jest możliwe z poziomu dowolnego języka programowania. Najczęściej jednak jest ona wykorzystywana jako ramdysk. Prawie wszystkie dyskowe systemy operacyjne zawierają programy automatycznie zakładające ramdysk podczas inicjowania systemu. Ponadto wiele programów istnieje w dwóch wersjach: pełnej dla 130XE oraz nieco okrojonej dla pozostałych modeli.

Przy wykorzystaniu dodatkowych banków pamięci dla ANTIC-a bardzo łatwo można uzyskać wiele interesujących efektów. Na przykład szybkie przełączanie banków ułatwia tworzenie animacji. Wykorzystanie wszystkich możliwości zależy tylko od inwencji programisty.

11.3. Rozszerzenia pamięci

Jak mówi jedno z popularnych praw Murphy'ego, każdy komputer ma zbyt mało pamięci. Wychodząc naprzeciw zapotrzebowaniu użytkowników wiele firm opracowało rozszerzenia pamięci RAM dla komputerów XL/XE do 256 i 512 KB, a nawet do 1 MB. Zarządzanie takim dużym obszarem pamięci jest rozwiązywane na wiele różnych sposobów.

Najprostszym sposobem jest inne wykorzystanie bitów 2-6 rejestru PORTB. Cała pamięć RAM komputera jest w takim przypadku dzielona na banki po 16 KB. Przy pojemności 256 KB daje to 16 banków, a przy 512 KB - 32 banki. Do wyboru banku potrzeba więc odpowiednio cztery lub pięć bitów. Liczba ta nie przekracza ilości niewykorzystanych bitów rejestru PORTB.

Wybrany bank pamięci jest ustawiany w obszarze $4000-$7FFF. Ponieważ normalnie wszystkie bity w PORTB są ustawione, to po zainicjowaniu systemu jest to zawsze bank o najwyższym numerze. Trzy z pozostałych banków tworzą resztę podstawowej pamięci RAM komputera. Pozostaje do zagospodarowania jeszcze 192 (12 banków) lub 448 KB (28 banków), a właściwie o 16 KB więcej, gdyż jeden bank podstawowy zajmuje obszar ROM i można go bez obaw wykorzystać. W tym przypadku nie ma możliwości oddzielnego adresowania banków przez ANTIC i CPU.

Dwie najpopularniejsze na Zachodzie wersje rozszerzeń pamięci działają na nieco innej zasadzie. Są one bowiem dostosowane do wszystkich wersji Atari, co uniemożliwia użycie rejestru PORTB. W rozszerzeniu firmy Axlon do wyboru banków pamięci służy rejestr umieszczony pod adresem $CFFF, a firma Mosaic umieściła ten rejestr w $FFC0. Zastosowanie oddzielnego rejestru oddaje do dyspozycji układu zarządzania pamięcią aż osiem bitów.

W obu przypadkach rejestry znajdują się w przestrzeni zajmowanej normalnie przez pamięć ROM. Rozwiązanie to zmusza do użycia dodatkowych układów logicznych, które rozwiązują ten problem. Ponieważ w Polsce takie rozwiązania nie są spotykane, to ich opis pominiemy.

DODATKI

Dodatek A

Adresy procedur OS

```
$C0CD - PLARTI - powrót z przerwania
$C272 - SETVBLV - ustawianie wektorów przerwania VBLK
$C6B3 - DSKINT - główna procedura dyskowa
$C73A - PUTADR - przepisanie adresu bufora dyskowego
$C745 - LOADER - odczyt rekordu z nowego urządzenia
$C795 - HENDRT - zakończenie odczytu z nowego urządzenia
$C7D5 - PUTCHR - odczyt znaku z nowego urządzenia
$C86D - ADD28E - obliczanie adresu wczytywania
$C892 - ADDWRD - obliczanie adresu do zapisu i odczytu
$C8B5 - ADDGET - obliczanie adresu do odczytu
$C8E4 - NEWVEC - tabela wektorów nowych urządzeń
$C90C - NEWINIT - inicjowanie nowych urządzeń
$C933 - SIOINT - wstępna procedura złącza szeregowego
$C991 - NWDVC - tabela procedur I/O nowych urządzeń
$C9AF - GETLOW - odszukanie nowego urządzenia do operacji I/O
$C9CA - NEWPER - wywołanie procedury I/O nowego urządzenia
$C9DC - CHKNWP - wybór i uaktywnienie nowego urządzenia
$CA21 - BITMASK - maski bitowe dla NEWIOREQ i GETLOW
$CA29 - PRPLNK - przygotowanie operacji I/O nowego urządzenia
$CB56 - CHCKFF - obliczenie sumy kontrolnej listy liniowej
$CC00 - CHARSET2 - zestaw znaków międzynarodowych
$D805 - PDIOR - operacja I/O nowego urządzenia
$D80D - PDVOPV - wektor operacji OPEN nowego urządzenia
$D80F - PDVCLV - wektor operacji CLOSE nowego urządzenia
$D811 - PDVGBV - wektor operacji GET BYTE nowego urządzenia
$D813 - PDVPBV - wektor operacji PUT BYTE nowego urządzenia
$D815 - PDVSTV - wektor operacji STATUS nowego urządzenia
$D817 - PDVSPV - wektor operacji SPECIAL nowego urządzenia
$E000 - CHARSET1 - standardowy zestaw znaków
$E400 - EDTVEC - wektory obsługi edytora
$E410 - SCRVEC - wektory obsługi ekranu
$E420 - KBDVEC - wektory obsługi klawiatury
$E430 - PRTVEC - wektory obsługi drukarki
$E440 - CASVEC - wektory obsługi magnetofonu
$E450 - JMPTAB - tabela skoków
$E453 - JDSKINT - skok do DSKINT
$E456 - JCIOMAIN - skok do CIOMAIN
$E459 - JSIOINT - skok do SIOINT
$E45C - JSETVBV - skok do SETVBLV
$E462 - JEXITVB - skok do EXITVBL
$E468 - JSNDENB - skok do SNDENBL
$E47A - JCASRDBL - skok do CASRDBL
$E47D - JCASOPIN - skok do CASOPIN
$E48F - CALTAB - tabela adresowa nowych urządzeń
$E49B - NEWINITC - skok do NEWINIT
$E4DC - CIONOPN - kanał I/O nie otwarty
$E4DF - CIOMAIN - procedura obsługi urządzeń
$E510 - NEXDER - błąd: urządzenie nie istnieje
$E53F - CIOOPN - procedura otwarcia kanału IOCB
$E55C - INIOPN - przygotowanie otwarcia IOCB
```

$E57C - CIOCLS - procedura zamknięcia kanału IOCB
$E597 - CIOSTSP - odczyt statusu i operacje specjalne CIO
$E5B2 - CIOREAD - odczyt z kanału IOCB
$E61E - CIOWRT - zapis do kanału IOCB
$E670 - CIORET - powrót z procedury CIO
$E672 - CPLCIO - zakończenie operacji CIO
$E695 - CMPENT - ustalenie adresu procedury operacji I/O
$E6BB - DECBUFL - zmniejszenie długości bufora danych
$E6C8 - DECBUFP - zmniejszenie adresu bufora danych
$E6D1 - INCBUFP - zwiększenie adresu bufora danych
$E6D8 - SUBBUFL - zmniejszenie długości bufora danych
$E6EA - GOHAND - przejście do procedury operacji I/O
$E6F4 - CIOJMP - skok do procedury według adresu ze stosu
$E6FF - DEVNUM - ustalenie numeru urządzenia
$E712 - DVSRCH - poszukiwanie urządzenia w HATABS
$E716 - FDVHND - poszukiwanie urządzenia w HATABS
$E72D - COMTAB - tabela indeksów adresów operacji I/O
$E739 - LINKSOM - procedura dołączania urządzeń
$E7BE - DCBINI - inicjowanie SIO dla nowego urządzenia
$E7D4 - TSIOIN - tabela wartości do inicjowania SIO
$E7DE - INITLD - inicjowanie odczytu z nowego urządzenia
$E816 - GETBYT - odczyt bajtu z nowego urządzenia
$E833 - GTNXBL - odczyt bloku z nowego urządzenia
$E851 - SIOTAB - tabela parametrów DCB dla GTNXBL
$E85D - SRCHLS - przeszukiwanie listy liniowej
$E894 - LINKWM - dołączanie elementu przy gorącym starcie
$E89E - LINKCD - dołączanie elementu przy zimnym starcie
$E900 - CALVEC - tabela wektorów nowych urządzeń
$E915 - UNLINK - usunięcie elementu z listy liniowej
$E971 - SIO - obsługa złącza szeregowego
$EA2A - CLPSIO - zakończenie operacji SIO
$EA37 - WAIT - oczekiwanie na potwierdzenie
$EA88 - SEND - nadawanie na szynę szeregową
$EAFD - RECEIV - procedura odczytu SIO
$EB27 - ITIMOT - sygnalizacja błędu Timeout
$EB87 - LODPTR - przepisanie adresu i długości bufora
$EB9D - CASENT - procedura SIO dla magnetofonu
$EC11 - TIM1INT - przerwanie licznika 1
$EC17 - SNDENBL - zezwolenie na zapis danych
$EC40 - RECVEN - zezwolenie na odczyt danych
$EC56 - ENABLE - zezwolenie na transmisję
$EC84 - SNDDIS - zabronienie zapisu danych
$EC9A - SETTOT - ustalenie Timeout
$ECAF - SENDIN - rozpoczęcie operacji zapisu
$ECC0 - STIMWT - ustawienie Timeout i oczekiwanie
$ECC8 - COMPUT - obliczenie rzeczywistej szybkości odczytu
$ED2E - ADJUST - poprawienie licznika w/g systemu TV
$ED3D - BEGNRD - rozpoczęcie odczytu z magnetofonu
$EDC7 - PRBRKK - obsługa klawisza BREAK
$EDE2 - SETT1V - ustawienie przerwania TIMCNT1
$EDF9 - POKTAB - tabela współczynników szybkości transmisji
$EE11 - STVCTB - tabela współczynników systemu TV
$EE1B - ADJTAB - tabela poprawek licznika
$EE1D - TSMAL - tabela adresów pamięci obrazu
$EE2D - TDLEC - tabela wielkości programów ANTIC-a
$EE4D - TAGRM - tabela trybów ANTIC-a
$EE5D - TDLVL - tabela wielkości trybów ANTIC-a
$EE6D - TLSHC - tabela przesunięć do obliczania adresu
$EE7D - TMCCN - tabela wielkości kolumn obrazu
$EE8D - TMRCN - tabela wielkości wierszy obrazu

```
$EE9D - TRSHC - tabela przesunięć do obliczania adresu
$EEAD - TDMSK - tabela masek bitowych dla obrazu
$EEB4 - MSKTAB - tabela masek bitowych
$EEBC - NEWDEV - wpisanie nowego urządzenia do HATABS
$EEF9 - SPCHND - specjalna procedura obsługi I/O
$EF26 - PUTBYT - procedura obsługi nowego urządzenia
$EF6E - POWERON - inicjowanie edytora
$EF8E - SCOPN - otwarcie kanału dla ekranu
$EF94 - EDOPN - otwarcie kanału dla edytora
$F180 - GETCH - odczyt znaku z ekranu
$F18F - GETPLT - odczyt punktu z ekranu
$F1A4 - OUTCH - zapis znaku na ekranie
$F1B4 - TSTRET - porównanie znaku z RETURN ($9B)
$F1CA - OUTPLT - zapis punktu na ekranie
$F1E9 - DISPLY - przekształcenie znaku na ekranie
$F20B - RETURM - powrót z procedur monitora
$F21E - KBOPN - otwarcie kanału dla klawiatury
$F22D - EDSP - procedura specjalna dla edytora
$F22E - SCRFIN - zamknięcie kanału dla ekranu
$F24A - EGETCH - odczyt znaku z edytora
$F2AD - JSRIND - skok pośredni do procedury
$F2B0 - EOUTCH - zapis znaku do edytora
$F2BE - PRCCHR - obsługa znaku w edytorze
$F2F8 - IGNORE - odczyt następnego znaku z klawiatury
$F2FD - KBGBYT - odczyt znaku z klawiatury
$F302 - KGETCH - pobranie znaku z klawiatury
$F3E0 - ESCAPE - procedura znaku "Escape"
$F3E6 - CRSUP - kursor o jeden wiersz w górę
$F3F3 - CRSDWN - kursor o jeden wiersz w dół
$F400 - CRSLFT - kursor o jeden znak w lewo
$F40A - CRSRMR - kursor do prawego marginesu
$F411 - CRSRGT - kursor o jeden znak w prawo
$F41B - CRSLMR - kursor do lewego marginesu
$F420 - CLRSCR - czyszczenie ekranu
$F440 - CRSHOM - kursor do lewego, górnego rogu ekranu
$F450 - CRSBS - kasowanie znaku w lewo od kursora
$F47A - CRSTAB - kursor na następną pozycję tabulacji
$F495 - CRSSTB - ustawianie pozycji tabulacji
$F49A - CRSCTB - kasowanie pozycji tabulacji
$F49F - INSCHR - wstawienie znaku pod kursorem
$F4D5 - DELCHR - usunięcie znaku spod kursora
$F50C - INSLIN - wstawienie linii pod kursorem
$F50D - INSLN2 - wstawienie linii pod kursorem
$F520 - DELLIN - usunięcie linii spod kursora
$F527 - DELROW - usunięcie linii fizycznej
$F556 - BELL - brzęczyk edytora
$F55F - BTMLIN - kursor do lewego, dolnego rogu
$F565 - DBDEC - dwukrotne zmniejszenie dwubajtowego wektora
$F569 - STDFS - ustawienie parametrów przesuwu obrazu
$F570 - STDDSP - ustawienie parametrów obrazu
$F578 - SGDEC - jednokrotne zmniejszenie dwubajtowego wektora
$F57A - DCUSAC - zmniejszenie wektora według akumulatora
$F5A0 - SSDLE - ustawienie adresu DL przy przesuwie obrazu
$F5AC - CONVRT - zamiana pozycji kursora na adres
$F60A - INCRSB - zwiększenie pozycji kursora
$F60E - EOLSUB - koniec wiersza logicznego
$F612 - SCRIBT - sprawdzenie końca wiersza logicznego
$F661 - RTWSCR - przesunięcie ekranu z dodaniem wiersza
$F665 - RETURN - umieszczenie znaku RETURN
$F6AE - SUBEND - aktualizacja wartości ROWAC lub COLAC
```

$F6BC - ERANGE - sprawdzenie zakresu edytora
$F6CA - RANGE - sprawdzenie zakresu ekranu
$F718 - OFFCRS - odtworzenie znaku spod kursora
$F723 - BITCON - zamiana znaku na maskę bitową
$F732 - BITROL - przesunięcie LOGMAP o jeden bit w lewo
$F73C - BITPUT - ustawienie bitu w TABMAP
$F73E - BITPT2 - ustawienie bitu w TABMAP
$F74A - BITCLR - skasowanie bitu w TABMAP
$F758 - LOGGET - obliczenie wiersza logicznego
$F75A - LGET2 - obliczenie wiersza logicznego
$F75B - LGET3 - obliczenie wiersza logicznego
$F75D - BITGET - obliczenie maski bitowej
$F76A - INATAC - zamiana kodu Internal na ATASCII
$F78E - LININS - dodanie nowego wiersza na ekranie
$F7C2 - EXTEND - dodanie do LOGMAP nowego wiersza
$F7E2 - CLRLIN - skasowanie zawartości linii na ekranie
$F7F7 - DOSCR - wykonanie przesunięcia obrazu
$F82A - COMADR - obliczenie adresu linii obrazu
$F88E - COMLOG - obliczenie wiersza logicznego
$F8B1 - DOBUFC - obliczenie chwilowej długości wiersza
$F90C - STRBEG - zapis początku bufora wiersza
$F918 - DELTIE - skasowanie pustej linii ekranu
$F923 - DELEML - skasowanie pustej linii ekranu
$F93C - TSTCNT - sprawdzenie znaków kontrolnych
$F94C - PHACRS - zapamiętanie pozycji kursora
$F957 - PLACRS - odtworzenie pozycji kursora
$F962 - SWAP - przełączanie między ekranem a oknem
$F983 - KEYCLK - dźwięk klawiatury
$F997 - SCLED - kursor na lewej krawędzi obrazu
$F9A6 - PUTMSC - przepisanie adresu pamięci obrazu
$F9AF - DRAW - procedura specjalna dla ekranu
$FB04 - TBMSK - tabela numerów masek bitowych obrazu
$FB08 - COLTAB - tabela standardowych kolorów obrazu
$FB0D - CNTRLS - tabela wektorów znaków kontrolnych
$FB49 - AINCC - tabela zamiany z ATASCII na Internal
$FB4D - INACC - tabela zamiany z Internal na ATASCII
$FB51 - KEYDEF - tabela definicji klawiszy
$FC11 - FKDEF - tabela definicji klawiszy funkcyjnych
$FCC4 - FSDL - przerwanie NMI programu ANTIC-a
$FCDB - CASINIT - inicjowanie magnetofonu
$FCE5 - CASSP - procedura specjalna dla magnetofonu
$FCE6 - CASOPN - otwarcie kanału dla magnetofonu
$FCF7 - CASOPIN - początek odczytu z magnetofonu
$FD7A - CASRDBT - odczyt bajtu z magnetofonu
$FD8D - CASRDBL - odczyt bloku z magnetofonu
$FDB4 - CASWRT - zapis na magnetofon
$FDCC - CASST - odczyt statusu magnetofonu
$FDCF - CASCLS - zamknięcie kanału dla magnetofonu
$FDFC - BPWT - dźwięk i oczekiwanie na naciśnięcie klawisza
$FE36 - PHVRT - przejście do odczytu klawisza
$FE3F - SYSBUF - ustawienie DCB dla magnetofonu
$FE7C - WSIOSB - przygotowanie do zapisu na magnetofon
$FE8D - TABCAS - tabela wartości dla magnetofonu
$FE99 - PRINIT - inicjowanie drukarki
$FE9F - PRSTAD - adres bufora statusu drukarki
$FEA1 - PRCHAR - adres bufora danych drukarki
$FEA3 - PRSTAT - odczyt statusu drukarki
$FEC1 - PRRDSP - procedura specjalna i odczyt z drukarki
$FEC2 - PROPN - otwarcie kanału dla drukarki
$FECB - PRWRT - zapis na drukarkę

```
$FEED - FPBUF - wypełnienie bufora drukarki
$FF14 - SETDCB - ustawienie DCB dla drukarki
$FF44 - PRPUT - ustawienie Timeout dla drukarki
$FF4B - PRMODE - ustalenie trybu pracy drukarki
```

Dodatek B

Rejestry OS w pamięci RAM

```
  $08 - WARMST - znacznik gorącego startu
  $0E - APPMHI - najwyższy adres RAM zajęty przez program
  $10 - IRQENS - rejestr-cień IRQEN
  $11 - IRQSTAT - rejestr-cień IRQST
  $12 - RTCLOK - zegar czasu rzeczywistego
  $15 - BUFADR - adres bufora dla operacji dyskowych
  $17 - ICCOMT - rejestr przejściowy kodu operacji I/O
  $20 - ZIOCB - zerostronicowy blok kontroli I/O
  $20 - ICHIDZ - indeks wpisu urządzenia w HATABS
  $21 - ICDNOZ - numer urządzenia
  $22 - ICCOMZ - kod operacji wejścia/wyjścia
  $23 - ICSTZ - status operacji wejścia/wyjścia
  $24 - ICBAZ - adres bufora danych dla operacji I/O
  $26 - ICPTZ - adres procedury obsługi dla operacji
  $28 - ICBLZ - długość bufora danych dla operacji I/O
  $2A - ICAX1Z - rejestr pomocniczy ZIOCB
  $2B - ICAX2Z - rejestr pomocniczy ZIOCB
  $2C - ICAX3Z - rejestr pomocniczy ZIOCB
  $2D - ICAX4Z - rejestr pomocniczy ZIOCB
  $2E - ICAX5Z - rejestr pomocniczy ZIOCB
  $2F - ICAX6Z - rejestr pomocniczy ZIOCB
  $30 - STATUS - status aktualnej operacji SIO
  $31 - CHKSUM - suma kontrolna dla operacji SIO
  $32 - BUFR - adres bufora danych dla SIO
  $34 - BUFEN - adres końca bufora danych dla SIO
  $36 - LTEMP - pomocniczy wektor odczytu listy liniowej
  $38 - BUFRFL - znacznik zapełnienia bufora SIO
  $39 - RECVND - znacznik końca odczytu
  $3A - XMTDON - znacznik końca transmisji
  $3B - CHKSNT - znacznik nadania sumy kontrolnej
  $3C - NOCKSM - znacznik braku sumy kontrolnej
  $3D - BPTR - licznik bufora magnetofonu
  $3E - GAPTYP - znacznik długości przerwy między blokami
  $3F - FEOF - znacznik końca zbioru
  $40 - FREQ - licznik dźwięku przy otwarciu magnetofonu
  $41 - IOSNDEN - znacznik dźwięku przy transmisji
  $42 - CRITIC - znacznik krytycznych czasowo operacji I/O
  $4A - ZCHAIN - rejestr następstwa listy liniowej
  $4C - DSTAT - status klawiatury i ekranu
  $50 - TEMP - pomocniczy rejestr przejściowy
  $51 - HOLD1 - pomocniczy rejestr przejściowy
  $52 - LMARGIN - lewy margines obrazu
  $53 - RMARGIN - prawy margines obrazu
  $54 - ROWCRS - pionowa pozycja kursora
  $55 - COLCRS - pozioma pozycja kursora
  $57 - DINDEX - numer trybu graficznego OS
  $58 - SAVMSC - adres pamięci obrazu
  $5A - OLDROW - poprzednia pionowa pozycja kursora
  $5B - OLDCOL - poprzednia pozioma pozycja kursora
```

```
  $5D - OLDCHR - poprzedni znak na ekranie
  $5E - OLDADR - poprzedni adres znaku na ekranie
  $60 - FKDEFP - wektor tabeli definicji klawiszy F1-F4
  $62 - PALNTS - wskaźnik systemu TV
  $63 - LOGCOL - adres kursora w wierszu logicznym
  $64 - ADRESS - rejestr adresowy dla procedur edytora
  $66 - MLTTMP - pomocniczy rejestr przejściowy
  $68 - SAVADR - pomocniczy rejestr adresowy
  $6A - RAMTOP - liczba stron pamięci RAM
  $6B - BUFCNT - licznik bufora edytora
  $6C - BUFSTR - adres bufora dla edytora
  $6E - BITMSK - maska bitowa do wyświetlenia znaku
  $6F - SHFAMT - liczba przesunięć punktu
  $70 - ROWAC - aktualny wiersz przy rysowaniu
  $72 - COLAC - aktualna kolumna przy rysowaniu
  $74 - ENDPT - znacznik końca rysowanej linii
  $76 - DELTAR - przyrost pionowej pozycji kursora
  $77 - DELTAC - przyrost poziomej pozycji kursora
  $79 - KEYDEFP - wektor tabeli definicji klawiszy
  $7B - SWPFLG - znacznik kursora w trybach z oknem
  $7C - HOLDCH - przechowywanie wartości znaku
  $7D - INSDAT - rejestr pomocniczy edytora
  $7E - COUNTR - licznik pamięci obrazu dla DOSCR i DRAW
$0100 - STACK - stos mikroprocesora 6502
$0200 - DLIV - wektor przerwania programu ANTIC-a
$0210 - VTIMR1 - wektor przerwania licznika 1 POKEY-a
$0212 - VTIMR2 - wektor przerwania licznika 2 POKEY-a
$0214 - VTIMR4 - wektor przerwania licznika 4 POKEY-a
$0218 - TIMCNT1 - pierwszy licznik systemu
$0226 - TIMVEC1 - wektor przerwania licznika TIMCNT1
$022A - TIMFLG3 - znacznik wyzerowania licznika TIMCNT3
$022F - DMACTLS - rejestr-cień DMACTL
$0230 - DLPTRS - rejestr-cień DLPTR
$0232 - SKCTLS - rejestr-cień SKCTL
$0233 - LCOUNT - licznik odczytu dla nowego urządzenia
$0234 - LPENHS - rejestr-cień LPENH
$0235 - LPENVS - rejestr-cień LPENV
$023A - CDEVIC - kod urządzenia dla SIO
$023B - CCMND - kod operacji dla SIO
$023C - CAUX1 - pierwszy bajt pomocniczy dla SIO
$023D - CAUX2 - drugi bajt pomocniczy dla SIO
$023E - TEMP - tymczasowy rejestr odpowiedzi urządzenia
$023F - ERRFLG - znacznik błędu operacji SIO
$0244 - COLDST - znacznik zimnego startu systemu
$0245 - RECLEN - długość rekordu z nowego urządzenia
$0246 - DSKTIM - wartość Timeout dla stacji dysków
$0247 - PDVMSK - maska obecności nowych urządzeń
$0248 - PDVRS - rejestr-cień PDVREG
$024A - RELADR - adres procedury przemieszczalnej
$024C - PPTMPA - rejestr przechowania zawartości akumulatora
$024D - PPTMPX - rejestr przechowania zawartości rejestru X
$026B - CHSPTR - wektor nieużywanego zestawu znaków
$026C - VSFLAG - znacznik przesuwu pionowego obrazu
$026E - FINE - znacznik delikatnego przesuwu obrazu
$026F - GTICTLS - rejestr-cień GTIACTL
$0270 - PADDL0 - rejestr-cień POT0
$0278 - JSTICK0 - położenie joysticka 0
$0279 - JSTICK1 - położenie joysticka 1
$027A - JSTICK2 - położenie joysticka 0
$027B - JSTICK3 - położenie joysticka 1
```

```
$027C - PTRIG0 - przycisk potencjometru 0
$027D - PTRIG1 - przycisk potencjometru 1
$0284 - TRIG0S - przycisk joysticka 0, rejestr-cień TRIG0
$0285 - TRIG1S - przycisk joysticka 1, rejestr-cień TRIG1
$0286 - TRIG2S - rejestr-cień TRIG0
$0287 - TRIG3S - rejestr-cień TRIG1
$0288 - HIBYTE - indeks operacji nowego urządzenia
$0289 - WMODE - znacznik sposobu dostępu do magnetofonu
$028A - BLIM - długość bufora magnetofonu
$028E - NEWADR - adres procedury nowego urządzenia
$0290 - TXTROW - wiersz kursora w oknie tekstowym
$0291 - TXTCOL - kolumna kursora w oknie tekstowym
$0293 - TINDEX - tryb graficzny OS w oknie tekstowym
$0294 - TXTMSC - adres pamięci okna tekstowego
$029C - CRETRY - liczba powtórzeń rozkazu operacji
$029D - HOLD3 - pomocniczy rejestr przejściowy
$029E - SUBTMP - pomocniczy rejestr przejściowy
$02A0 - DMASK - maska punktów obrazu
$02A2 - ESCFLG - znacznik klawisza ESC
$02A3 - TABMAP - mapa pozycji tabulacji
$02B2 - LOGMAP - mapa linii logicznych
$02B6 - INVFLG - znacznik klawisza inverse video
$02B7 - FILFLG - znacznik wypełniania obrazu
$02B8 - TMPROW - tymczasowy rejestr pozycji kursora
$02B9 - TMPCOL - tymczasowy rejestr pozycji kursora
$02BB - SCRFLG - znacznik przesuwu obrazu
$02BC - HOLD4 - pomocniczy rejestr przejściowy
$02BD - DRETRY - liczba powtórzeń wywołań urządzenia
$02BE - SHFLOK - znacznik klawiszy SHIFT i CONTROL
$02BF - BOTSCR - liczba wierszy tekstu
$02C0 - COLPM0S - rejestr-cień COLPM0
$02C1 - COLPM1S - rejestr-cień COLPM1
$02C2 - COLPM2S - rejestr-cień COLPM2
$02C3 - COLPM3S - rejestr-cień COLPM3
$02C4 - COLPF0S - rejestr-cień COLPF0
$02C5 - COLPF1S - rejestr-cień COLPF1
$02C6 - COLPF2S - rejestr-cień COLPF2
$02C7 - COLPF3S - rejestr-cień COLPF3
$02C8 - COLBAKS - rejestr-cień COLBAK
$02C9 - RUNADR - adres procedury nowego urządzenia
$02CB - HIUSED - adres końcowy procedury nowego urządzenia
$02CF - GBYTEA - adres procedury nowego urządzenia
$02D1 - LOADAD - adres wczytywania z nowego urządzenia
$02D3 - ZLOADA - pomocniczy rejestr adresu wczytywania
$02D5 - DSCTLN - długość sektora dyskowego
$02DB - NOCLIK - znacznik dźwięku klawiatury
$02DE - PBPNT - licznik bufora drukarki
$02DF - PBUFSZ - długość bufora drukarki
$02E5 - MEMTOP - adres górnej granicy wolnej pamięci RAM
$02E7 - MEMLO - adres dolnej granicy wolnej pamięci RAM
$02E9 - HNDLOD - znacznik relokowalnej procedury obsługi I/O
$02EA - DVSTAT - dodatkowy rejestr statusu urządzenia
$02EC - DVTMOT - dodatkowy rejestr Timeout urządzenia
$02ED - REVNUM - numer wersji nowego urządzenia
$02EE - CBAUD - prędkość transmisji z magnetofonu
$02F0 - CRSINH - znacznik widoczności kursora
$02F3 - CHACT - rejestr-cień CHRCTL
$02F4 - CHBAS - rejestr-cień CHBASE
$02F5 - NEWROW - nowa pozycja pionowa kursora
$02F6 - NEWCOL - nowa pozycja pozioma kursora
```

$02F8 - ROWINC - zmiana pionowej pozycji kursora
$02F9 - COLINC - zmiana poziomej pozycji kursora
$02FA - CHAR - kod wewnętrzny znaku
$02FB - ATACHR - kod ATASCII znaku
$02FC - KBCODES - rejestr-cień KBCODE
$02FD - FILDAT - numer koloru dla wypełniania
$02FE - DSPFLG - znacznik wyświetlania znaków kontrolnych
$02FF - SSFLAG - znacznik start/stop dla przesuwu obrazu
$0300 - DDEVIC - kod identyfikacyjny urządzenia
$0301 - DUNIT - numer identyfikacyjny urządzenia
$0302 - DCMND - bajt rozkazu dla urządzenia
$0303 - DSTATS - status urządzenia
$0304 - DBUFA - adres bufora danych
$0306 - DTIMLO - wartość Timeout dla urządzenia
$0308 - DBYT - długość bufora danych
$030A - DAUX1 - rejestr pomocniczy dla operacji I/O
$030B - DAUX2 - rejestr pomocniczy dla operacji I/O
$030C - INTIM1 - rejestr czasu przy odczycie z magnetofonu
$030F - CASFLG - znacznik operacji z magnetofonem
$0310 - INTIM2 - rejestr czasu przy odczycie z magnetofonu
$0312 - TEMP1 - pomocniczy rejestr przejściowy
$0313 - TEMP2 - pomocniczy rejestr przejściowy
$0314 - PTIMOT - wartość Timeout dla drukarki
$0315 - TEMP3 - pomocniczy rejestr przejściowy
$0316 - SAVIO - rejestr przejściowy dla operacji SIO
$0317 - TIMFLG - znacznik upłynięcia czasu Timeout
$0318 - STACKP - rejestr wskaźnika stosu dla SIO
$0319 - TSTAT - przejściowy rejestr statusu SIO
$031A - HATABS - tabela wektorów procedur obsługi
$0340 - IOCB0 - blok kontroli I/O numer 0
$0350 - IOCB1 - blok kontroli I/O numer 1
$0360 - IOCB2 - blok kontroli I/O numer 2
$0370 - IOCB3 - blok kontroli I/O numer 3
$0380 - IOCB4 - blok kontroli I/O numer 4
$0390 - IOCB5 - blok kontroli I/O numer 5
$03A0 - IOCB6 - blok kontroli I/O numer 6
$03B0 - IOCB7 - blok kontroli I/O numer 7
$0340 - ICCHID - indeks wpisu urządzenia w HATABS
$0341 - ICDNO - numer urządzenia
$0342 - ICCMD - kod rozkazu operacji I/O
$0343 - ICSTAT - status operacji I/O
$0344 - ICBUFA - adres bufora danych dla operacji I/O
$0346 - ICPUTB - adres procedury przesyłania danych
$0348 - ICBUFL - długość bufora danych dla operacji I/O
$034A - ICAX1 - rejestr pomocniczy dla operacji I/O
$034B - ICAX2 - rejestr pomocniczy dla operacji I/O
$034C - ICAX3 - rejestr pomocniczy dla operacji I/O
$034D - ICAX4 - rejestr pomocniczy dla operacji I/O
$034E - ICAX5 - rejestr pomocniczy dla operacji I/O
$034F - ICAX6 - rejestr pomocniczy dla operacji I/O
$03C0 - PRNBUF - bufor drukarki
$03E8 - SUPERF - znacznik stosowany przy odczycie klawiatury
$03E9 - CKEY - znacznik klawisza START przy zimnym starcie
$03EC - DERRF - znacznik błędu przy otwieraniu edytora
$03FB - CHLINK - rejestr elementów listy liniowej
$03FD - CSCB - bajty kontroli szybkości magnetofonu
$03FF - CRCB - bajt długości rekordu magnetofonowego
$0400 - CASBUF - bufor magnetofonu
$047F - CASBEN - koniec bufora magnetofonu
$D000 - HPOSP0 - pozioma pozycja gracza 0 (Z)

```
$D000 - KOLM0PF - kolizja pocisku 0 z polem gry (O)
$D001 - HPOSP1 - pozioma pozycja gracza 1 (Z)
$D001 - KOLM1PF - kolizja pocisku 1 z polem gry (O)
$D002 - HPOSP2 - pozioma pozycja gracza 2 (Z)
$D002 - KOLM2PF - kolizja pocisku 2 z polem gry (O)
$D003 - HPOSP3 - pozioma pozycja gracza 3 (Z)
$D003 - KOLM3PF - kolizja pocisku 3 z polem gry (O)
$D004 - HPOSM0 - pozioma pozycja pocisku 0 (Z)
$D004 - KOLP0PF - kolizja gracza 0 z polem gry (O)
$D005 - HPOSM1 - pozioma pozycja pocisku 1 (Z)
$D005 - KOLP1PF - kolizja gracza 1 z polem gry (O)
$D006 - HPOSM2 - pozioma pozycja pocisku 2 (Z)
$D006 - KOLP2PF - kolizja gracza 2 z polem gry (O)
$D007 - HPOSM3 - pozioma pozycja pocisku 3 (Z)
$D007 - KOLP3PF - kolizja gracza 3 z polem gry (O)
$D008 - SIZEP0 - poziomy rozmiar gracza 0 (Z)
$D008 - KOLM0P - kolizja pocisku 0 z graczem (O)
$D009 - SIZEP1 - poziomy rozmiar gracza 1 (Z)
$D009 - KOLM1P - kolizja pocisku 1 z graczem (O)
$D00A - SIZEP2 - poziomy rozmiar gracza 2 (Z)
$D00A - KOLM2P - kolizja pocisku 2 z graczem (O)
$D00B - SIZEP3 - poziomy rozmiar gracza 3 (Z)
$D00B - KOLM3P - kolizja pocisku 3 z graczem (O)
$D00C - SIZEM - poziomy rozmiar pocisków (Z)
$D00C - KOLP0P - kolizja gracza 0 z innym graczem (O)
$D00D - GRAFP0 - rejestr grafiki gracza 0 (Z)
$D00D - KOLP1P - kolizja gracza 1 z innym graczem (O)
$D00E - GRAFP1 - rejestr grafiki gracza 1 (Z)
$D00E - KOLP2P - kolizja gracza 2 z innym graczem (O)
$D00F - GRAFP2 - rejestr grafiki gracza 2 (Z)
$D00F - KOLP3P - kolizja gracza 3 z innym graczem (O)
$D010 - GRAFP3 - rejestr grafiki gracza 3 (Z)
$D010 - TRIG0 - stan przycisku joysticka 0 (O)
$D011 - GRAFM - rejestr grafiki pocisków (Z)
$D011 - TRIG1 - stan przycisku joysticka 1 (O)
$D012 - COLPM0 - rejestr koloru gracza i pocisku 0 (Z)
$D013 - COLPM1 - rejestr koloru gracza i pocisku 1 (Z)
$D013 - TRIG3 - znacznik dołączenia cartridge'a (O)
$D014 - COLPM2 - rejestr koloru gracza i pocisku 2 (Z)
$D014 - PAL - znacznik systemu TV (O)
$D015 - COLPM3 - rejestr koloru gracza i pocisku 3 (Z)
$D016 - COLPF0 - rejestr koloru pola gry 0 (Z)
$D017 - COLPF1 - rejestr koloru pola gry 1 (Z)
$D018 - COLPF2 - rejestr koloru pola gry 2 (Z)
$D019 - COLPF3 - rejestr koloru pola gry 3 (Z)
$D01A - COLBAK - rejestr koloru tła (Z)
$D01B - GTIACTL - rejestr kontroli układu GTIA
$D01C - VDELAY - licznik opóźnienia pionowego P/MG
$D01D - PMCNTL - rejestr kontroli graczy i pocisków
$D01E - HITCLR - rejestr kasowania rejestrów kolizji
$D01F - CONSOL - rejestr stanu klawiszy konsoli
$D1FF - PDVREG - rejestr wyboru nowego urządzenia
$D200 - AUDF1 - częstotliwość pracy generatora 1 (Z)
$D200 - POT0 - rejestr położenia potencjometru 0 (O)
$D201 - AUDC1 - rejestr kontroli dźwięku generatora 1 (Z)
$D201 - POT1 - rejestr położenia potencjometru 1 (O)
$D202 - AUDF2 - częstotliwość pracy generatora 2 (Z)
$D202 - POT2 - rejestr położenia potencjometru 2 (O)
$D203 - AUDC2 - rejestr kontroli dźwięku generatora 2 (Z)
$D203 - POT3 - rejestr położenia potencjometru 3 (O)
```

```
$D204 - AUDF3 - częstotliwość pracy generatora 3 (Z)
$D205 - AUDC3 - rejestr kontroli dźwięku generatora 3 (Z)
$D206 - AUDF4 - częstotliwość pracy generatora 4 (Z)
$D207 - AUDC4 - rejestr kontroli dźwięku generatora 4 (Z)
$D208 - AUDCTL - rejestr kontroli generatorów dźwięku (Z)
$D208 - POTST - status odczytu potencjometrów (O)
$D209 - STIMER - rejestr zerowania liczników (Z)
$D209 - KBCODE - kod ostatnio naciśniętego klawisza (O)
$D20A - SKSTRES - reset statusu złącza szeregowego (Z)
$D20A - RANDOM - rejestr liczby losowej (O)
$D20B - POTG0 - znacznik przetwornika analogowo-cyfrowego (Z)
$D20D - SEROUT - szeregowy rejestr wyjściowy (Z)
$D20D - SERIN - szeregowy rejestr wejściowy (O)
$D20E - IRQEN - zezwolenia przerwań IRQ (Z)
$D20E - IRQST - status przerwań IRQ (O)
$D20F - SKCTL - rejestr kontroli złącza szeregowego (Z)
$D20F - SKSTAT - rejestr statusu złącza szeregowego (O)
$D300 - PORTA - port A układu PIA
$D301 - PORTB - port B układu PIA
$D302 - PACTL - rejestr kontroli portu A
$D303 - PBCTL - rejestr kontroli portu B
$D400 - DMACTL - rejestr kontroli dostępu do pamięci
$D401 - CHRCTL - rejestr kontroli wyświetlania znaków
$D402 - DLPTR - adres programu ANTIC-a
$D404 - HSCROL - znacznik poziomego przesuwu obrazu
$D405 - VSCROL - znacznik pionowego przesuwu obrazu
$D407 - PMBASE - adres pamięci graczy i pocisków
$D409 - CHBASE - adres zestawu znaków
$D40A - WSYNC - znacznik oczekiwania na synchronizację poziomą
$D40B - VCOUNT - licznik linii obrazu
$D40C - LPENH - poziome położenie pióra świetlengo
$D40D - LPENV - pionowe położenie pióra świetlnego
$D40E - NMIEN - rejestr zezwoleń na przerwania NMI
$D40F - NMIST - rejestr statusu przerwań NMI
```

Dodatek C

Zmienne systemowe

```
AUDC
bity 0-3 - regulacja siły głosu (do 0 do 15)
bity 4 - kontrola siły głosu bez modulacji
bity 5-7 - sterowanie zniekształceniami dźwięku
     000 = rejestr 5-bitowy i 17-bitowy
     001 = rejestr 5-bitowy
     010 = rejestr 5-bitowy i 4-bitowy
     011 = rejestr 5-bitowy
     100 = rejestr 17-bitowy
     101 = bez rejestru przesuwającego
     110 = rejestr 4-bitowy
     111 = bez rejestru przesuwającego

AUDCTL
bit 0 - wybór zegara bazowego (0 = 64 kHz, 1 = 15 kHz)
bit 1 - filtr  w generatorze 2 sterowany przez generator 4 (1 =
włączony)
bit 2 - filtr  w generatorze 1 sterowany przez generator 3 (1 =
włączony)
```

```
bit 3 - łączenie  dzielników  3  i 4  w dzielnik 16-bitowy (1 =
włączone)
bit 4 - łączenie  dzielników  1  i 2  w dzielnik 16-bitowy (1 =
włączone)
bit 5 - wybór zegara bazowego dla generatora 3 (0 = w/g bitu 0,
1 = 1,77 MHz)
bit 6 - wybór zegara bazowego dla generatora 1 (0 = w/g bitu 0,
1 = 1,77 MHz)
bit 7 - przełączanie  rejestru przesuwającego (0 = 17-bitowy, 1
= 9-bitowy)

BOTSCR
    $00 - brak tekstu (cały obraz graficzny)
    $04 - okno tekstowe (4 wiersze tekstu)
    $18 - tryb 0 (24 wiersze tekstu)

CHRCTL/CHACT
bit 0 - tłumienie znaków (1 = znaki w inverse niewidoczne)
bit 1 - odwracanie znaków (0 = inverse nie działa)
bit 2 - odbicie znaków (1 = lustrzane odbicie znaków w pionie)
bitu 3-7 - niewykorzystane

CONSOL
bit 0 - klawisz START (0 = wciśnięty)
bit 1 - klawisz SELECT (0 = wciśnięty)
bit 2 - klawisz OPTION (0 = wciśnięty)

CRCB
    $FA - długość rekordu mniejsza od 128
    $FC - pełny rekord (128 bajtów)
    $FE - rekord końca zbioru (128 zer)

CRITIC
    0 - brak ograniczeń czasowych
    nie 0 - krytyczna czasowo operacja I/O

CRSINH
    0 - kursor widoczny
    nie 0 - kursor niewidoczny

DMACTL/DMACTLS
bity 0-1 - szerokość obrazu na ekranie:
           00 = brak obrazu
           01 = obraz wąski (192 punkty)
           10 = obraz normalny (320 punktów)
           11 = obraz szeroki (384 punkty)
bit 2 - DMA dla pocisków (1 = włączony)
bit 3 - DMA dla graczy (1 = włączony)
bit 4 - rozdzielczość P/MG (0 = dwu-, 1 = jednoliniowa)
bit 5 - DMA dla programu ANTIC-a (1 = włączony)
bity 6-7 - niewykorzystane

DSPFLG
    0 - wykonywanie znaków kontrolnych
    nie 0 - wyświetlanie znaków kontrolnych

ESCFLG
    $00 - wykonanie następnego znaku
    $80 - wyświetlenie następnego znaku
```

```
FEOF
    0 - brak rekordu końcowego zbioru
    nie 0 - napotkany rekord końcowy zbioru

FILFLG
    $00 - rysowanie linii (DRAW)
    $01 - rysowanie linii z wypełnieniem (FILL)

FINE
    $00 - przesuw zgrubny (o jeden wiersz)
    $FF - przesuw delikatny (o jedną linię ekranu)

FREQ
    $01 - jeden dźwięk (odczyt z magnetofonu)
    $02 - dwa dźwięki (zapis na magnetofon)

GAPTYP
    $00 - długa przerwa między rekordami
    $80 - krótka przerwa między rekordami

GTIACTL/GTICTLS
bity 0-3 - priorytet graczy i pocisków
      0001 - P0, P1, P2, P3, PF0, PF1, PF2, PF3, BAK
      0010 - P0, P1, PF0, PF1, PF2, PF3, P2, P3, BAK
      0100 - PF0, PF1, PF2, PF3, P0, P1, P2, P3, BAK
      1000 - PF0, PF1, P0, P1, P2, P3, PF2, PF3, BAK
bit 4 - łączenie pocisków w 5 gracza (1 = włączone)
bit 5 - gracze wielokolorowi (1 = włączone)
bity 6-7 - dodatkowe tryby graficzne GTIA:
           00 - GRAPHICS 8
           01 - GRAPHICS 9
           10 - GRAPHICS 10
           11 - GRAPHICS 11

INVFLG
    $00 - normalne wyświetlanie znaków
    $FF - wyświetlanie znaków w negatywie

IRQEN/IRQENS
zezwolenia przerwań IRQ - 0 = zabronione
bit 0 - przerwanie TIMERa 1
bit 1 - przerwanie TIMERa 2
bit 2 - przerwanie TIMERa 4
bit 3 - przerwanie końca transmisji
bit 4 - przerwanie zapisu przez złącze szeregowe
bit 5 - przerwanie odczytu przez złącze szeregowe
bit 6 - przerwanie klawiatury
bit 7 - przerwanie klawisza BREAK

IRQST/IRQSTAT
status przerwań IRQ - 0 = wystąpiło
bit 0 - przerwanie TIMERa 1
bit 1 - przerwanie TIMERa 2
bit 2 - przerwanie TIMERa 4
bit 3 - przerwanie końca transmisji
bit 4 - przerwanie zapisu przez złącze szeregowe
bit 5 - przerwanie odczytu przez złącze szeregowe
bit 6 - przerwanie klawiatury
bit 7 - przerwanie klawisza BREAK
```

```
KOLPxPF/KOLMxPF
bit 0 - kolizja z polem gry 0 (1 = wystąpiła)
bit 1 - kolizja z polem gry 1 (1 = wystąpiła)
bit 2 - kolizja z polem gry 2 (1 = wystąpiła)
bit 3 - kolizja z polem gry 3 (1 = wystąpiła)
bity 4-7 - niewykorzystane

KOLPxP/KOLMxP
bit 0 - kolizja z graczem 0 (1 = wystąpiła)
bit 1 - kolizja z graczem 1 (1 = wystąpiła)
bit 2 - kolizja z graczem 2 (1 = wystąpiła)
bit 3 - kolizja z graczem 3 (1 = wystąpiła)
bity 4-7 - niewykorzystane

NMIEN
zezwolenia przerwań NMI - 0 = zabronione
bity 0-5 - niewykorzystane
bit 6 - przerwanie synchronizacji (VBLKI)
bit 7 - przerwanie programu ANTIC-a (DLI)

NMIST
status przerwań NMI - 1 = wystąpiło
bity 0-4 - niewykorzystane
bit 5 - przerwanie klawisza RESET
bit 6 - przerwanie synchronizacji (VBLKI)
bit 7 - przerwanie programu ANTIC-a (DLI)

NOCLK
    0 - dźwięk klawiatury dozwolony
    nie 0 - dźwięk klawiatury zabroniony

PACTL
bit 0 - zezwolenie na żądanie przerwania IRQ przez urządzenie A
(1 = dozwolone)
bit 1 - niewykorzystywany (zawsze 0)
bit 2 - sterowanie pracą PORTA (0 = rejestr porządkowania
danych, 1 = rejestr przesyłania danych)
bit 3 - sterowanie silnikiem magnetofonu (0 = włączony, 1 =
wyłączony)
bit 4 - niewykorzystany (zawsze 1)
bit 5 - niewykorzystany (zawsze 1)
bit 6 - niewykorzystany (zawsze 0)
bit 7 - rejestr statusu przerwania urządzenia A (1 = wystąpiło)

PBCTL
bit 0 - zezwolenie na żądanie przerwania IRQ przez urządzenie B
(1 = dozwolone)
bit 1 - niewykorzystywany (zawsze 0)
bit 2 - sterowanie pracą PORTB (0 = rejestr porządkowania
danych, 1 = rejestr przesyłania danych)
bit 3 - identyfikacja poleceń dla urządzeń zewnętrznych
bit 4 - niewykorzystany (zawsze 1)
bit 5 - niewykorzystany (zawsze 1)
bit 6 - niewykorzystany (zawsze 0)
bit 7 - rejestr statusu przerwania urządzenia B (1 = wystąpiło)

PMCTL
bit 0 - przenoszenie danych pocisków z rejestru GRAFM na ekran
(1 = włączone)
bit 1 - przenoszenie danych graczy z rejestrów GRAFP na ekran
```

```
(1 = włączone)
bit 2 - odczyt przycisków joysticków (1 = zablokowany)
bity 3-7 - niewykorzystane

PORTA
bit 0 - joystick 0 naprzód (0 = tak)
bit 1 - joystick 0 wstecz (0 = tak)
bit 2 - joystick 0 w lewo, przycisk potencjometru 0 (0 = tak)
bit 3 - joystick 0 w prawo, przycisk potencjometru 1 (0 = tak)
bit 4 - joystick 1 naprzód (0 = tak)
bit 5 - joystick 1 wstecz (0 = tak)
bit 6 - joystick 1 w lewo, przycisk potencjometru 2 (0 = tak)
bit 7 - joystick 1 w prawo, przycisk potencjometru 3 (0 = tak)

PORTB
bit 0 - system operacyjny (0 = odłączony)
bit 1 - interpreter Atari Basic (1 = odłączony)
bity 2-3 - wybór banku pamięci (tylko 130XE)
bit 4 - sterowanie  dostępem CPU do pamięci (0 = dodatkowa, 1 =
główna; tylko 130XE)
bit 5 - sterowanie  dostępem ANTIC-a do pamięci (0 = dodatkowa,
1 = główna; tylko 130XE)
bit 6 - niewykorzystany
bit 7 - program testujący (1 = odłączony)

SHFLOK
    $00 - małe litery (wciśnięty CAPS)
    $40 - duże litery (stan normalny)
    $80 - znaki z CONTROL

SIZEM/SIZEPx
    $00 - normalna wielkość obiektu (pixel = 1 cykl koloru)
    $01 - podwójna wielkość obiektu (pixel = 2 cykle koloru)
    $10 - normalna wielkość obiektu (pixel = 1 cykl koloru)
    $11 - poczwórna wielkość obiektu (pixel = 4 cykle koloru)

SKCTL/SKCTLS
bit 0 - razem z bitem 1 resetuje POKEY
bit 1 - obsługa klawiatury
bit 2 - częstość przetwarzania A/C (0 = 20 ms, 1 = 128 µs)
bit 3 - przesyłanie dwutonowe (1 = włączone)
bity 4-6 - sterowanie szybkością transmisji
   000 - zewnętrzne
   001 - odczyt: w/g generatora 4, zapis: zewnętrzne
   010 - w/g generatora 4
   011 - kombinacja zabroniona
   100 - odczyt: zewnętrzne, zapis: w/g generatora 4
   101 - kombinacja zabroniona
   110 - odczyt: w/g generatora 4, zapis: w/g 2
   111 - j.w., zablokowane wejście i wyjście taktujące
bit 7 - nadanie sygnału SPACE (1 = włączone)

SKSTAT
bit 0 - niewykorzystany (= 1)
bit 1 - przesyłanie danych (0 = trwa)
bit 2 - dowolny klawisz (0 = naciśnięty)
bit 3 - klawisz SHIFT (0 = naciśnięty)
bit 4 - kopia rejestru wejścia szeregowego
bit 5 - bufor klawiatury (0 = przepełniony)
bit 6 - bufor wejścia szeregowego (0 = przepełniony)
```

```
bit 7 - Framing Error (0 = wystąpił)

SSFLAG
    $00 - dane są wyprowadzane na ekran
    $FF - zatrzymanie wyprowadzania danych

SWPFLG
    $00 - kursor w oknie tekstowym
    $FF - kursor na obrazie graficznym

VDELAY
bit 0 - obniżenie pocisku 0 o jedną linię ekranu (1 = tak)
bit 1 - obniżenie pocisku 1 o jedną linię ekranu (1 = tak)
bit 2 - obniżenie pocisku 2 o jedną linię ekranu (1 = tak)
bit 3 - obniżenie pocisku 3 o jedną linię ekranu (1 = tak)
bit 4 - obniżenie gracza 0 o jedną linię ekranu (1 = tak)
bit 5 - obniżenie gracza 1 o jedną linię ekranu (1 = tak)
bit 6 - obniżenie gracza 2 o jedną linię ekranu (1 = tak)
bit 7 - obniżenie gracza 3 o jedną linię ekranu (1 = tak)

WMODE
    $00 - odczyt z magnetofonu
    $80 - zapis na magnetofon
```

Dodatek D

Słownik terminów informatycznych

ANTIC
    AlphaNumeric Television Interface Controller - drugi, dodatkowy mikroprocesor, którego zasadniczym zadaniem jest tworzenie obrazu. Układ specjalnie zaprojektowany dla komputerów Atari.

ASCII
    American Standard Code of Information Interchange - amerykański, standardowy kod wymiany informacji, kod przypisujący liczbom od 0 do 127 znaczenie liter, liczb i znaków kontrolnych, powszechnie używany w komputerach. Każda firma stosuje jednak nieco zmodyfikowany kod, np. w Atari jest używany ATASCII (ATari ASCII).

BCD
    Binary Coded Decimal - liczba dziesiętna kodowana dwójkowo, kod zapisu liczb dziesiętnych, w którym każdej cyfrze odpowiadają cztery bity. W ten sposób w jednym bajcie można zapisać dwie cyfry dziesiętne. Maksymalna wartość półbajtu w kodzie BCD wynosi 9. Procesor 6502 po rozkazie SED pracuje w trybie dziesiętnym, czyli na liczbach w kodzie BCD.

CIO
    Central Input/Output - zespół procedur obsługujących komunikację komputera z urządzeniami zewnętrznymi.

DCB
    Device Control Block - blok kontroli urządzeń, obszar pamięci RAM od $0300 do $030B wykorzystywany przez procedury SIO do operacji wejścia/wyjścia.

DL
    Display List - program ANTIC-a, program w kodzie maszynowym ANTIC-a wskazujący mu sposób generowania obrazu. Program ANTIC-a może wywoływać przerwania niemaskowalne zwane DLI (Display List Interrupt).

DMA
    Direct Memory Access - bezpośredni dostęp do pamięci, sposób dostępu do pamięci komputera z pominięciem pośrednictwa procesora. Oszczędza to czas pracy procesora i zwiększa szybkość pracy systemu. W Atari DMA jest wykorzystywany przez ANTIC (procesor obrazowy).

EOF
    End Of File - koniec zbioru, znak oznaczający ten koniec lub kod błędu informujący, że wszystkie dane zostały już ze zbioru odczytane.

EOL
    End Of Line - koniec linii, znak końca wiersza logicznego w edytorze, a w innych urządzeniach znak końca bloku przesyłanej informacji. Kod EOL jest taki jak znaku RETURN - $9B.

FP
    Floating Point - liczby, operacje i procedury na liczbach rzeczywistych czyli zmiennoprzecinkowych.

GTIA
    Graphics Television Interface Adaptor - specjalizowany układ scalony służący do tworzenia kolorów i obsługi grafiki graczy i pocisków (Player/Missile Graphics).

I/O
    Input/Output - wejście/wyjście, ogólna nazwa operacji służących do komunikacji komputera z urządzeniami zewnętrznymi.

IOCB
    Input/Output Control Block - blok kontroli I/O, 16-bajtowy obszar pamięci RAM wykorzystywany przez procedury CIO do operacji wejścia/wyjścia. Istnieje IOCB strony zerowej i osiem IOCB w obszarze od $0340 do $03BF.

IRQ
    Interrupt Request - żądanie przerwania, nazwą tą określa się wszystkie przerwania maskowalne, to znaczy takie, które mogą nie zostać przyjęte do realizacji przez procesor.

LSB
    Least Significant Byte - mniej znaczący bajt, bajt adresu lub danej zawierający dwie mniej znaczące cyfry (szesnastkowe). LSB=ADR-MSB*$0100. Przy adresowaniu wskazuje numer komórki na stronie.

MSB
    Most Significant Byte - bardziej znaczący bajt, bajt adresu lub danej zawierający dwie bardziej znaczące cyfry. MSB=INT(ADR/$0100). Przy adresowaniu wskazuje numer strony oamięci.

NMI
Non Maskable Interrupt - przerwanie niemaskowalne czyli takie, które musi zostać wykonane przez procesor (nie może być zignorowane).

nowe urządzenie
New Device - nazwa stosowana w systemie operacyjnym Atari dla określenia urządzeń zewnętrznych przyłączanych do szyny równoległej, które nie istniały jeszcze w czasie projektowania systemu.

OS
Operating System - system operacyjny, zestaw procedur zapisanych przeważnie w pamięci ROM, które sterują pracą komputera i jego współpracą z urządzeniami zewnętrznymi.

PIA
Peripheral Interface Adaptor - standardowy układ I/O typu 6520. Jego zadaniem jest w Atari obsługa portów joysticków i zarządzanie pamięcią komputera.

pixel
Picture element - najmniejszy element obrazu dostępny dla programisty. Wielkość pixela zależy od wybranego trybu graficznego. Najmniejszy pixel ma wysokość jednej linii ekranu i szerokość pół cyklu koloru, a największy wysokość ośmiu linii ekranu i szerokość czterech cykli koloru.

POKEY
Potentiometr & Keyboard - specjalizowany układ scalony, którego zadaniem jest obsługa klawiatury, potencjometrów oraz komunikacji komputera z urządzeniami zewnętrznymi poprzez złącze szeregowe.

port
Rejestr służący do komunikacji komputera z urządzeniami peryferyjnymi, zwykle jest to rejestr specjalnego układu scalonego znajdujący się w przestrzeni adresowej procesora. Zwany jest także bramą.

rejestr
Zespół komórek (w Atari 1-6) pamięci RAM służących do przechowywania różnych wartości niezbędnych do pracy OS lub programu. Podstawowymi rodzajami rejestrów są wektory i znaczniki (wskaźniki).

rejestr-cień
Specjalizowane układy scalone komputerów Atari mają własne rejestry, które znajdują się w obszarze adresowym procesora. Większość z nich posiada kopie w normalnych rejestrach RAM, tzw. rejestry-cienie (shadow register). Zawartość rejestrów-cieni jest przepisywana do rejestrów sprzętowych lub odwrotnie podczas przerwania VBLK.

rekord
Część informacji (programu lub danych) zapisywana lub odczytywana jednorazowo z urządzenia zewnętrznego. Długość rekordu jest zwykle równa długości bufora, który służy do jego przechowania, a nigdy nie może być większa.

SIO
    Serial Input/Output - zespół procedur obsługujących komunikację z urządzeniami zewnętrznymi poprzez złącze szeregowe.

strona
    Część pamięci komputera o wielkości 256 bajtów. Starszy bajt adresu wskazuje zawsze numer strony, a młodszy - komórkę na stronie.

Timeout
    Maksymalny czas wykonywania operacji przez urządzenie zewnętrzne i jednocześnie czas oczekiwania komputera na odpowiedź od urządzenia po wysłaniu rozkazu wykonania operacji. Timeout jest różny dla różnych urządzeń i różnych operacji.

VBLK
    Vertical Blank - synchronizacja pionowa, okres wygaszania strumienia elektronów tworzących obraz i przesunięcia go z prawego, dolnego rogu ekranu do lewego, górnego. Podczas VBLK wywoływane jest przerwanie VBLKI (VBLK Interrupt).

wektor
    Rejestr zawierający adres procedury lub danych (wskazujący adres). Wektor dwubajtowy zawiera pełny adres, a wektor jednobajtowy tylko numer strony.

wskaźnik
    To samo co znacznik.

znacznik
    Rejestr, którego zawartość sygnalizuje stan jakiegoś elementu systemu lub wariant operacji. Znacznik może być traktowany jako całość, ale często poszczególne bity znacznika mają odrębne znaczenie.

Dodatek E

Tabela przeliczeń DEC-BIN-HEX

Poniższa tabela może służyć do szybkiej zamiany liczb zapisanych w systemach: szesnastkowym (HEX), dziesiętnym (DEC) i dwójkowym (BIN). Dodatkowo dla ułatwienia orientacji w stosowanym przez procesor 6502 systemie adresowania podana jest wartość dziesiętna pomnożona przez 256 (strona).

HEX	DEC	strona	BIN	HEX	DEC	strona	BIN
00	0	0	00000000	80	128	32768	10000000
01	1	256	00000001	81	129	33024	10000001
02	2	512	00000010	82	130	33280	10000010
03	3	768	00000011	83	131	33536	10000011
04	4	1024	00000100	84	132	33792	10000100
05	5	1280	00000101	85	133	34048	10000101
06	6	1536	00000110	86	134	34304	10000110
07	7	1792	00000111	87	135	34560	10000111
08	8	2048	00001000	88	136	34816	10001000
09	9	2304	00001001	89	137	35072	10001001
0A	10	2560	00001010	8A	138	35328	10001010

0B	11	2816	00001011	8B	139	35584	10001011
0C	12	3072	00001100	8C	140	35840	10001100
0D	13	3328	00001101	8D	141	36096	10001101
0E	14	3584	00001110	8E	142	36352	10001110
0F	15	3840	00001111	8F	143	36608	10001111
10	16	4096	00010000	90	144	36864	10010000
11	17	4352	00010001	91	145	37120	10010001
12	18	4608	00010010	92	146	37376	10010010
13	19	4864	00010011	93	147	37632	10010011
14	20	5120	00010100	94	148	37888	10010100
15	21	5376	00010101	95	149	38144	10010101
16	22	5632	00010110	96	150	38400	10010110
17	23	5888	00010111	97	151	38656	10010111
18	24	6144	00011000	98	152	38912	10011000
19	25	6400	00011001	99	153	39168	10011001
1A	26	6656	00011010	9A	154	39424	10011010
1B	27	6912	00011011	9B	155	39680	10011011
1C	28	7168	00011100	9C	156	39936	10011100
1D	29	7424	00011101	9D	157	40192	10011101
1E	30	7680	00011110	9E	158	40448	10011110
1F	31	7936	00011111	9F	159	40704	10011111
20	32	8192	00100000	A0	160	40960	10100000
21	33	8448	00100001	A1	161	41216	10100001
22	34	8704	00100010	A2	162	41472	10100010
23	35	8960	00100011	A3	163	41728	10100011
24	36	9216	00100100	A4	164	41984	10100100
25	37	9472	00100101	A5	165	42240	10100101
26	38	9728	00100110	A6	166	42496	10100110
27	39	9984	00100111	A7	167	42752	10100111
28	40	10240	00101000	A8	168	43008	10101000
29	41	10496	00101001	A9	169	43264	10101001
2A	42	10752	00101010	AA	170	43520	10101010
2B	43	11008	00101011	AB	171	43776	10101011
2C	44	11264	00101100	AC	172	44032	10101100
2D	45	11520	00101101	AD	173	44289	10101101
2E	46	11776	00101110	AE	174	44544	10101110
2F	47	12032	00101111	AF	175	44800	10101111
30	48	12288	00110000	B0	176	45056	10110000
31	49	12544	00110001	B1	177	45312	10110001
32	50	12800	00110010	B2	178	45568	10110010
33	51	13056	00110011	B3	179	45824	10110011
34	52	13312	00110100	B4	180	46080	10110100
35	53	13568	00110101	B5	181	46336	10110101
36	54	13824	00110110	B6	182	46592	10110110
37	55	14080	00110111	B7	183	46848	10110111
38	56	14336	00111000	B8	184	47104	10111000
39	57	14592	00111001	B9	185	47360	10111001
3A	58	14848	00111010	BA	186	47616	10111010
3B	59	15104	00111011	BB	187	47872	10111011
3C	60	15360	00111100	BC	188	48128	10111100
3D	61	15616	00111101	BD	189	48384	10111101
3E	62	15872	00111110	BE	190	48640	10111110
3F	63	16128	00111111	BF	191	48896	10111111
40	64	16384	01000000	C0	192	49152	11000000
41	65	16640	01000001	C1	193	49408	11000001
42	66	16896	01000010	C2	194	49664	11000010
43	67	17152	01000011	C3	195	49920	11000011
44	68	17408	01000100	C4	196	50176	11000100
45	69	17664	01000101	C5	197	50432	11000101
46	70	17920	01000110	C6	198	50688	11000110

47	71	18176	01000111	C7	199	50944	11000111
48	72	18432	01001000	C8	200	51200	11001000
49	73	18688	01001001	C9	201	51456	11001001
4A	74	18944	01001010	CA	202	51712	11001010
4B	75	19200	01001011	CB	203	51968	11001011
4C	76	19456	01001100	CC	204	52224	11001100
4D	77	19712	01001101	CD	205	52480	11001101
4E	78	19968	01001110	CE	206	52736	11001110
4F	79	20224	01001111	CF	207	52992	11001111
50	80	20480	01010000	D0	208	53248	11010000
51	81	20736	01010001	D1	209	53504	11010001
52	82	20992	01010010	D2	210	53760	11010010
53	83	21248	01010011	D3	211	54016	11010011
54	84	21504	01010100	D4	212	54272	11010100
55	85	21760	01010101	D5	213	54528	11010101
56	86	22016	01010110	D6	214	54784	11010110
57	87	22272	01010111	D7	215	55040	11010111
58	88	22528	01011000	D8	216	55296	11011000
59	89	22784	01011001	D9	217	55552	11011001
5A	90	23040	01011010	DA	218	55808	11011010
5B	91	23296	01011011	DB	219	56064	11011011
5C	92	23552	01011100	DC	220	56320	11011100
5D	93	23808	01011101	DD	221	56576	11011101
5E	94	24064	01011110	DE	222	56832	11011110
5F	95	24320	01011111	DF	223	57088	11011111
60	96	24576	01100000	E0	224	57344	11100000
61	97	24832	01100001	E1	225	57600	11100001
62	98	25088	01100010	E2	226	57856	11100010
63	99	25344	01100011	E3	227	58112	11100011
64	100	25600	01100100	E4	228	58368	11100100
65	101	25856	01100101	E5	229	58624	11100101
66	102	26112	01100110	E6	230	58880	11100110
67	103	26368	01100111	E7	231	59136	11100111
68	104	26624	01101000	E8	232	59392	11101000
69	105	26880	01101001	E9	233	59648	11101001
6A	106	27136	01101010	EA	234	59904	11101010
6B	107	27392	01101011	EB	235	60160	11101011
6C	108	27648	01101100	EC	236	60416	11101100
6D	109	27904	01101101	ED	237	60672	11101101
6E	110	28160	01101110	EE	238	60928	11101110
6F	111	28416	01101111	EF	239	61184	11101111
70	112	28672	01110000	F0	240	61440	11110000
71	113	28928	01110001	F1	241	61696	11110001
72	114	29184	01110010	F2	242	61952	11110010
73	115	29440	01110011	F3	243	62208	11110011
74	116	29696	01110100	F4	244	62464	11110100
75	117	29952	01110101	F5	245	62720	11110101
76	118	30208	01110110	F6	246	62976	11110110
77	119	30464	01110111	F7	247	63232	11110111
78	120	30720	01111000	F8	248	63488	11111000
79	121	30976	01111001	F9	249	63744	11111001
7A	122	31232	01111010	FA	250	64000	11111010
7B	123	31488	01111011	FB	251	64256	11111011
7C	124	31744	01111100	FC	252	64512	11111100
7D	125	32000	01111101	FD	253	64768	11111101
7E	126	32256	01111110	FE	254	65024	11111110
7F	127	32512	01111111	FF	255	65280	11111111

Dodatek F

Tabela różnic asemblerów

Asemblery dostępne na Atari XL/XE różnią się nieco między sobą stosowanymi słowami kluczowymi. Poniższa tabela zawiera różnice występujące w kilku najpopularniejszych asemblerach: Macroassembler 65 (MAC/65), Atari Assembler/Editor (ASM/EDIT), Atari Macroassembler (AMAC) i Synapse Assembler (SYNASM).

MAC/65	ASM/EDIT	AMAC	SYNASM
*=	*=	ORG	.OR
*	*	*O	*
=	=	= lub EQU	.EQ
.BYTE	.BYTE	DB	.AT
.SBYTE	.BYTE	DC	.HS
.DBYTE	.BYTE	brak	.AS
.WORD	.WORD	DW	.DA
=+	*=*+	DS	.BS

Dodatek G

Bibliografia

1. Chadwick Ian: Mapping the Atari, COMPUTE! Books, Greensboro, USA, 1985.
2. Eichler Lutz, Grohmann Bernd: Atari 600XL/800XL Intern, Data Becker, Dusseldorf, RFN, 1984.
3. Hofacker Winfried: Hacker book. Atari computer tips + tricks, ELCOMP Publishing, Pomona, USA, 1983.
4. Praca zbiorowa: De Re Atari. A guide to effective programing 400/800 Home Computers, Byte Publishing, USA, 1981.
5. Ruszczyc Jan: Asembler 6502, SOETO, Warszawa, 1987.
6. Zientara Wojciech: PEEK-POKE 2 (opracowanie), B.U.K. "GLAD", Warszawa, 1987.
7. Zientara Wojciech: Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego, SOETO, Warszawa, 1988.
8. Zientara Wojciech: Mapa pamięci Atari XL/XE. Dyskowe systemy operacyjne, SOETO, Warszawa, 1988.
9. Zientara Wojciech: Mapa pamięci Atari XL/XE. Procedury interpretera Basica, SOETO, Warszawa, 1988.