Stołeczny Ośrodek
Elektronicznej
Techniki Obliczeniowej

INFORMATYKA
mikrokomputerowa

Wojciech Zientara

Mapa pamięci Atari XL/XE: Procedury interpretera BASIC-a

ATARI

WARSZAWA 1989

Spis treści

PRZEDMOWA

Dla umożliwienia porozumiewania się człowieka z komputerem konieczny jest język zrozumiały dla obu stron. Początkowo językiem tym był wewnętrzny kod maszynowy procesora. Ponieważ posługiwanie się nim jest stosunkowo trudne, wymyślono nowe - bardziej skomplikowane - języki, które są zrozumiałe dla przeciętnego użytkownika. W komputerach domowych najczęściej instalowany jest interpreter języka Basic. Nie wynika to z jego szczególnej prostoty i zalet, lecz ze stosunkowo niewielkiej pojemności pamięci zajmowanej przez ten język. Implementacje Basica stosowane w ośmiobitowych komputerach domowych są ponadto znacznie okrojone w stosunku do standardowej wersji Microsoft. Programy pisane w takim języku nie mogą być siłą rzeczy doskonałe. Dzięki znajomości wewnętrznej struktury interpretera można osiągnąć znacznie lepsze wyniki. Możliwe jest również bezpośrednie wywoływanie procedur interpretera, tak jak innych procedur w języku maszynowym, zarówno w programach napisanych w Basicu, jak i w asemblerze.

Niniejsza książka zawiera kompletny opis interpretera Atari Basic, który wbudowany jest we wszystkich modelach Atari XL i XE. Ponieważ interpreter ten (jak wszystkie elementy systemu) ulegał modyfikacjom, to opis dotyczy ostatniej wersji - *Basic Revision C*. Ta wersja interpretera znajduje się we wszystkich komputerach sprowadzonych do Polski przez Pewex, oprócz pierwszej partii (kilkaset sztuk). Posiadaną wersję interpretera można rozpoznać przez sprawdzenie zawartości komórki $A8E2 (43234). Jeżeli znajduje się w niej wartość $60 (96), to mamy do czynienia z wersją *Revision B*, a gdy $EA (234), to z *Revision C*.

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.

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 tabelę 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

Interpreter Atari Basic zajmuje obszar 8 KB pamięci ROM od adresu $A000 (40960) do $BFFF (49151). Jest on umieszczony w przestrzeni adresowej komputera na prawach cartridge'a (zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego"). W przypadku odłączenia go, pojawia się w tym obszarze blok 8 KB pamięci RAM. Ponadto interpreter wykorzystuje do pracy połowę strony zerowej RAM od $80 (128) do $FF (255) oraz obszary stosu kontroli składni $0480-$057D (1152-1405) i bufora wejściowego $0580-$05FF (1408-1535).

Program w języku Basic jest zapisywany począwszy od najniższego wolnego adresu pamięci RAM. Adres ten jest przechowywany w rejestrze MEMLO (MEMory LOw address). Przy braku DOS-u lub innego programu zarządzającego rejestr MEMLO zawiera wartość $0700 (1792), w przeciwnym przypadku wskazuje na pierwszy bajt powyżej obszaru zajętego przez ten program. Struktura programu Basica zapisanego w pamięci RAM jest pokazana na rysunku 1.

Bufor tokenizacji jest 256-bajtowym buforem wejściowym, w którym przechowywany jest wprowadzony wiersz programu podczas procesu tokenizacji. Proces ten zostanie opisany w dalszej części książki.

Tablica nazw zmiennych zawiera zapisane w kodzie ASCII nazwy wszystkich zmiennych użytych w programie. Maksymalna pojemność tablicy wynosi 128 zmiennych. Ponieważ nazwa w Basicu może mieć do 128 znaków, to maksymalna długość tablicy nazw wynosi 16 KB! Poszczególne nazwy nie są od siebie oddzielone, a koniec nazwy jest oznaczony ustawieniem najstarszego bitu w ostatnim znaku nazwy. Dla zmiennych tekstowych ostatnim znakiem nazwy jest zawsze "$", a dla zmiennych tablicowych "(".

Tablica wartości zmiennych zawiera informacje dotyczące aktualnej wartości każdej zmiennej. Do zapisania każdej wartości przeznaczone jest osiem bajtów, przy czym wartości zmiennych różnych typów są zapisywane w różny sposób. We wszystkich przypadkach pierwszy bajt oznacza typ zmiennej, a drugi - jej numer w tablicy nazw. Typy zmiennych są oznaczone w tablicy wartości następująco:

Dla prostej zmiennej liczbowej pozostałe sześć bajtów zawiera wartość zapisaną w kodzie BCD (w formacie liczby zmiennoprzecinkowej). Bajty 3 i 4 zmiennych indeksowanych (tablicowych i tekstowych) zawierają odległość początku danych zmiennej od początku tablicy zmiennych indeksowanych (zob. niżej). Pozostałe bajty określają wielkość zmiennych. Dla zmiennych tablicowych bajty 5 i 6 zawierają pierwszy wymiar zmiennej zwiększony o jeden (liczba kolumn), a bajty 7 i 8 - drugi wymiar, także zwiększony o jeden (liczba wierszy). Zero oznacza w tym przypadku brak wymiaru (zmienna tablicowa jednowymiarowa). Dla zmiennych tekstowych bajty 5 i 6 zawierają aktualną długość zmiennej, a bajty 7 i 8 - zadeklarowany wymiar zmiennej.

Tablica instrukcji zawiera zakodowane w procesie tokenizacji wszystkie wiersze programu oraz wiersze wprowadzone w trybie bezpośrednim. Format zapisu w tej tablicy jest opisany w rozdziale

poświęconym kontroli składni.

Tablica zmiennych indeksowanych służy do przechowania wszystkich wartości zmiennych tablicowych i tekstowych. Dla każdego znaku zmiennej tekstowej jest przewidziany jeden bajt (znak kodu ATASCII), zaś dla każdego elementu zmiennej tablicowej sześć bajtów (liczba zmiennoprzecinkowa).

Stos bieżący służy do przechowywania adresów powrotnych dla instrukcji GOSUB i FOR oraz wartości niezbędnych do prawidłowego przebiegu pętli FOR/NEXT.

Informacja dla instrukcji GOSUB zajmuje cztery bajty. Pierwszy - równy $00 - jest symbolem GOSUB, w dwóch następnych bajtach zapisany jest numer wiersza programu zawierającego instrukcję GOSUB, a w ostatnim bajcie umieszczona jest odległość GOSUB od początku tego wiersza.

W przypadku instrukcji FOR na stos zapisywane jest szesnaście bajtów. Pierwsze dwanaście z nich zajmują wartości końca pętli i kroku licznika, obie w formacie liczb zmiennoprzecinkowych (po 6 bajtów). W trzynastym bajcie znajduje się zwiększony o 128 (ustawiony bit 7) numer zmiennej będącej zmienną sterującą pętli. W bajtach 14 i 15 zapisany jest numer wiersza programu zawierającego instrukcję FOR, a w ostatnim bajcie umieszczona jest odległość FOR od początku tego wiersza.

Wszystkie wymienione wyżej bloki programu są ruchome, a ich aktualne adresy znajdują się w rejestrach RAM na stronie zerowej. Początek bufora tokenizacji wskazywany jest wektorem LOMEM (LOw MEMory address). Tablica nazw zmiennych rozciąga się od adresu zawartego w rejestrze VNTP (Variable Name Table Pointer) do adresu zawartego w VNTD (Variable Name Table enD). Tablica wartości zmiennych jest wskazywana wektorem VVTP (Variable Value Table Pointer), zaś tablica instrukcji wektorem STMTAB (STateMent TABle). Wektor STARP (STring and ARray table Pointer) wskazuje początek tablicy zmiennych indeksowanych. Adres stosu bieżącego zawarty jest w rejestrze RUNSTK (RUNtime STacK). Koniec obszaru zajętego przez program Basica wskazywany jest przez wektor BMEMHI (Basic MEMory HIgh address). Ponadto znajduje się tu wektor STMCUR (STateMent CURrent address), który wskazuje aktualnie wykonywany wiersz programu Basica, a w przypadku oczekiwania na wprowadzenie danych - początek wiersza trybu bezpośredniego.

Rozdział 1

URUCHOMIENIE INTERPRETERA

W czasie inicjowania systemu komputerowego przez procedurę RESET (zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego") inicjowany jest także zainstalowany cartridge. Po zakończeniu inicjowania i ewentualnym odczytaniu DOS-u sterowanie komputerem jest przekazywane do cartridge'a, którego rolę pełni w tym przypadku interpreter Basica. Odpowiednie informacje znajdują się w czterech rejestrach umieszczonych na końcu obszaru zajętego przez cartridge.

Warto przy okazji wspomnieć, że na końcu interpretera pozostało dziewięć pustych bajtów pamięci (równych $00), które można wykorzystać przy jego modyfikowaniu. Te niewykorzystane bajty znajdują się w obszarze $BFF1-$BFF9.

```
0100 ;CARTridge
0110 ;
0120     *=  $BFFA
0130 ;
0140 CARTRUN .WORD $A000
0150 CARTINS .BYTE $00
0160 CARTOPT .BYTE $05
0170 CARTINI .WORD $BFF0
```

Bajt $00 w rejestrze CARTINS (CARTridge INSerted) oznacza, że cartridge jest obecny i należy przekazać mu sterowanie. W celu zainicjowania cartridge'a wywoływana jest procedura, której adres zawiera rejestr CARTINI (CARTridge INItialization address). W przypadku interpretera Basica adres ten wskazuje tylko na instrukcję RTS, a więc interpreter nie wymaga inicjowania i następuje powrót do procedury RESET. Kolejnym sprawdzanym rejestrem jest CARTOPT (CARTridge OPTions). Jeśli zawiera on wartość różną od zera (jak w tym przypadku) to wykonywany jest wstępny odczyt z dyskietki. Po całkowitym zainicjowaniu systemu procedura RESET kończy się skokiem do adresu wskazanego wektorem CARTRUN (CARTridge RUN address). W interpreterze Basica znajduje się tu adres procedury zimnego startu interpretera CDST.

Na początku procedury CDST sprawdzane są znaczniki LOADFLG (LOADing FLaG) i WARMST (WARM STart). Gdy pierwszy z nich jest różny od zera, to znaczy, że start systemu odbywa się w trakcie odczytu z urządzenia zewnętrznego i przeprowadzany jest start zimny. W przeciwnym razie start gorący jest wykonywany, jeśli znacznik WARMST jest różny od zera.

Przy zimnym starcie najpierw wartość dolnej granicy dostępnej pamięci RAM jest przepisywana z rejestru MEMLO do rejestru LOMEM, a znaczniki LOADFLG i MEOLFLG (Modified End Of Line FLaG) są zerowane. Następnie wartość z MEMLO jest umieszczana w rejestrach określających położenie bloków programu od VNTP do BMEMHI (zob. Wprowadzenie). Z kolei poprzez dwukrotne wywołanie procedury INSEL wszystkie bloki powyżej tablicy nazw zmiennych

przesuwane są o jeden bajt, a bloki powyżej tablicy instrukcji jeszcze o trzy bajty. W uzyskane w ten sposób miejsce w tabeli nazw zmiennych wpisywane jest zero (nie ma nazwy żadnej zmiennej), a do tabeli instrukcji ciąg bajtów $00, $80, $03, który oznacza pustą instrukcję w trybie bezpośrednim (dokładnie jest to opisane w rozdziale poświęconym tokenizacji programu). Faza zimnego startu kończy się wpisaniem wartości $0A do rejestru tabulacji PTABW (Position of TABulate Width).

```
0100 ;Start Basic cartridge
0110 ;
0120 CLALL = $BD45
0130 INSEL = $A87A
0140 LOADFLG = $CA
0150 LOMEM = $80
0160 MEOLFLG = $92
0170 MEMLO = $02E7
0180 PRTPRM = $BD62
0190 PSTMAD = $BD9D
0200 PTABW = $C9
0210 RSTBRG = $A8F1
0220 RSTCHN = $BD5B
0230 STARP = $8C
0240 STMCUR = $8A
0250 VNTD =  $84
0260 VNTP =  $82
0270 VVTP =  $86
0280 WARMST = $08
0290 ;
0300     *=  $A000
0310 ;
0320 ;ColD STart
0330 ;
0340 CDST LDA LOADFLG
0350     BNE CONT
0360     LDA WARMST
0370     BNE WMST
0380 CONT LDX #$FF
0390     TXS
0400     CLD
0410 ;
0420 ;eXecute NEW statement
0430 ;
0440 XNEW LDX MEMLO
0450     LDY MEMLO+1
0460     STX LOMEM
0470     STY LOMEM+1
0480     LDA #$00
0490     STA MEOLFLG
0500     STA LOADFLG
0510     INY
0520     TXA
0530     LDX #VNTP
0540 LOOP STA $00,X
0550     INX
0560     STY $00,X
0570     INX
0580     CPX #MEOLFLG
0590     BCC LOOP
0600     LDX #VVTP
0610     LDY #$01
```

```
0620     JSR INSEL
0630     LDX #STARP
0640     LDY #$03
0650     JSR INSEL
0660     LDA #$00
0670     TAY
0680     STA (VNTD),Y
0690     STA (STMCUR),Y
0700     INY
0710     LDA #$80
0720     STA (STMCUR),Y
0730     INY
0740     LDA #$03
0750     STA (STMCUR),Y
0760     LDA #$0A
0770     STA PTABW
0780 WMST JSR RSTBRG
0790 ;
0800 ;WaRM STart
0810 ;
0820 WRMST JSR CLALL
0830 WRMST2 JSR RSTCHN
0840     LDA MEOLFLG
0850     BEQ WAITIN
0860     JSR PSTMAD
0870 ;
0880 ;WAIT for INput
0890 ;
0900 WAITIN JSR PRTPRM
```

Dalszy przebieg procedury CDST jest wspólny zarówno dla zimnego, jak i gorącego startu. Najpierw wywoływana jest procedura RSTBRG, która zeruje rejestry DATAD (DATa ADdress), DATALN (DATA LiNe number), ERRCOD (ERRor CODe), STOPLN (STOP LiNe number) i RADFLG (RADian FLaG) oraz umieszcza wartość $80 w rejestrze IRQSTAT (IRQ STATus) i w starszym bajcie rejestru TRAPLN (TRAP LiNe number). Procedura ta kończy się bezpośrednim skokiem do procedury CLALL. Zaraz po zakończeniu procedura ta jest wywoływana ponownie. Jest to spowodowane koniecznością wywoływania procedury CLALL przy wznawianiu pracy interpretera od etykiety WRMST. Natomiast procedura RSTBRG jest wykorzystywana także przez inne procedury interpretera i zakończenie jej skokiem do CLALL oszczędza kilkanaście bajtów pamięci.

Sama procedura CLALL najpierw zeruje rejestry generatorów dźwięku AUDF i AUDC, a więc wyłącza dźwięk. Następnie wywołuje siedem razy procedurę CLCHN umieszczając w rejestrze IODVC (Input/Output DeViCe) wartości od 7 do 1, co powoduje zamknięcie wszystkich kanałów komunikacji z urządzeniami zewnętrznymi poza kanałem 0 używanym przez edytor. Procedura CLCHN jest opisana w rozdziale 6.4. (Obsługa komunikacji).

```
0100 ;ReSeT Basic ReGisters
0110 ;
0120 CLALL = $BD45
0130 DATAD = $B6
0140 IRQSTAT = $11
0150 RADFLG = $FB
0160 TRAPLN = $BC
```

```
0170 ;
0180     *=  $B8F1
0190 ;
0200     LDX #$05
0210     LDY #$00
0220 LOOP STY DATAD,X
0230     DEX
0240     BPL LOOP
0250     STY RADFLG
0260     DEY
0270     STY TRAPLN+1
0280     STY IRQSTAT
0290     JMP CLALL

0100 ;CLose ALL channels
0110 ;
0120 AUDF1 = $D200
0130 CLCHN = $BCF7
0140 IODVC = $C1
0150 ;
0160     *=  $BD45
0170 ;
0180     LDA #$00
0190     LDX #$07
0200 SND STA AUDF1,X
0210     DEX
0220     BNE SND
0230     LDY #$07
0240     STY IODVC
0250 CHN JSR CLCHN
0260     DEC IODVC
0270     BNE CHN
0280     RTS
```

Jeżeli znacznik końca wiersza MEOLFLG jest różny od zera, to teraz jest wywoływana procedura PSTMAD. Jest to możliwe jedynie w przypadku gorącego startu interpretera. Odczytuje ona zawartość rejestru CSTAD (Current STatement ADdress) i umieszcza młodszy bajt w buforze wejściowym, w miejscu wskazanym przez starszy bajt. Na zakończenie zerowany jest znacznik MEOLFLG.

```
0100 ;Put STateMent ADdress
0110 ;
0120 CSTAD = $97
0130 INBUFP = $F3
0140 MEOLFLG = $92
0150 ;
0160     *=  $BD9D
0170 ;
0180     LDY CSTAD+1
0190     LDA CSTAD
0200     STA (INBUFP),Y
0210     LDA #$00
0220     STA MEOLFLG
0230     RTS
```

Procedura startu interpretera kończy się wywołaniem procedury PRTPRM. Powoduje ona umieszczenie na ekranie napisu "READY" oraz znaków końca wiersza (RETURN) przed i za napisem. Dzięki temu napis ten pojawia się zawsze w nowym wierszu ekranu. Procedura PRTPRM

po wywołaniu od etykiety PRTRET służy do umieszczenia na ekranie tylko znaku RETURN. Do służącego jako licznik wyświetlanych znaków rejestru X jest wtedy wpisywana wartość zero zamiast sześć. Wykorzystywana przez PRTPRM procedura PRPCHN jest opisana także w rozdziale "Obsługa komunikacji" (6.4).

```
0100 ;PRinT PRoMpt
0110 ;
0120 CIX =   $F2
0130 PRPCHN = $BA99
0140 ;
0150     *=  $BD62
0160 ;
0170 PRTPRM LDX #$06
0180 LOOP STX CIX
0190     LDA READYP,X
0200     JSR PRPCHN
0210     LDX CIX
0220     DEX
0230     BPL LOOP
0240     RTS
0250 ;
0260 ;READY Prompt
0270 ;
0280 READYP .BYTE $9B,"YDAER",$9B
0290 ;
0300 ;PRinT RETurn character
0310 ;
0320 PRTRET LDX #$00
0330     BEQ LOOP
```

Po wykonaniu wszystkich opisanych wyżej czynności następuje bezpośrednie przejście do procedury kontroli składni programu - SYNTAX, której opis znajduje się w następnym rozdziale. Należy jeszcze zwrócić uwagę na kilka etykiet znajdujących się w procedurze startu. Są one wykorzystywane przez inne procedury interpretera, jeśli wymagane jest ponowne jego zainicjowanie.

Pozostała jeszcze do omówienia wywoływana przez CDST procedura INSEL. Tworzy ona miejsce dla umieszczania wewnątrz obszaru pamięci programu nowych elementów tego programu: wierszy, zmiennych, wartości itd. Przed jej wywołaniem w rejestrze Y musi zostać umieszczona wielkość wstawianego elementu programu, a w rejestrze X adres wektora wskazującego miejsce jego wpisania do programu.

```
0100 ;INSert program ELement
0110 ;
0120 APPMHI = $0E
0130 BMEMHI = $90
0140 CSTAD = $97
0150 INSMER = $B930
0160 LENPEL = $A4
0170 MEMTOP = $02E5
0180 MEOLFLG = $92
0190 MRANGE = $A2
0200 NEWMHI = $9B
0210 OLDMHI = $99
0220 ;
```

```
0230     *=  $A87A
0240 ;
0250     LDA #$00
0260     STY LENPEL
0270     STA LENPEL+1
0280     TYA
0290     SEC
0300     ADC BMEMHI
0310     TAY
0320     LDA BMEMHI+1
0330     ADC LENPEL+1
0340     CMP MEMTOP+1
0350     BCC SET
0360     BNE ERR
0370     CPY MEMTOP
0380     BCC SET
0390     BEQ SET
0400 ERR JMP INSMER
0410 SET SEC
0420     LDA BMEMHI
0430     SBC $00,X
0440     STA MRANGE
0450     LDA BMEMHI+1
0460     SBC $01,X
0470     STA MRANGE+1
0480     CLC
0490     ADC $01,X
0500     STA OLDMHI+1
0510     LDA $00,X
0520     STA OLDMHI
0530     STA CSTAD
0540     ADC LENPEL
0550     STA NEWMHI
0560     LDA $01,X
0570     STA CSTAD+1
0580     ADC LENPEL+1
0590     ADC MRANGE+1
0600     STA NEWMHI+1
0610 LOOP LDA $00,X
0620     ADC LENPEL
0630     STA $00,X
0640     LDA $01,X
0650     ADC LENPEL+1
0660     STA $01,X
0670     INX
0680     INX
0690     CPX #MEOLFLG
0700     BCC LOOP
0710     STA APPMHI+1
0720     LDA BMEMHI
0730     STA APPMHI
0740     LDX MRANGE+1
0750     INX
0760     LDY MRANGE
0770     BNE DCR2
0780     NOP
0790     BEQ NXT2
0800     NOP
0810 DCR1 DEY
0820     DEC OLDMHI+1
0830     DEC NEWMHI+1
```

```
0840 NXT1 LDA (OLDMHI),Y
0850      STA (NEWMHI),Y
0860 DCR2 DEY
0870      BNE NXT1
0880      LDA (OLDMHI),Y
0890      STA (NEWMHI),Y
0900 NXT2 DEX
0910      BNE DCR1
0920      RTS
```

Przekazana do procedury wielkość wstawianego elementu jest najpierw umieszczana w rejestrze LENPEL (LENgth of Program ELement), a następnie dodawana jest do niej aktualna zawartość rejestru BMEMHI. Jeśli uzyskany wynik przekracza wartość MEMTOP (MEMory TOP), to przez skok do procedury INSMER sygnalizowany jest brak wystarczającego obszaru pamięci (INSufficient Memory ERror) - zob. rozdział 3, "Obsługa błędów". W przeciwnym razie wszystkie elementy programu o wyższych adresach są przesuwane w górę, aby uzyskać miejsce dla nowego elementu.

Przed przemieszczeniem obliczana jest wielkość przesuwanego bloku, a uzyskany wynik zapisywany jest w rejestrze MRANGE (Memory RANGE). Aktualny adres końca programu jest przepisywany do rejestrów OLDMHI (OLD Memory HIgh address) oraz CSTAD i, po dodaniu wielkości elementu, umieszczany jest w rejestrze NEWMHI (NEW Memory HIgh address). Teraz wielkość elementu zawarta w LENPEL służy do skorygowania wartości wektorów wskazujących przesuwane bloki programu. Poprawiana jest również zawartość rejestru APPMHI (APPlication Memory HIgh address) wykorzystywanego przez system operacyjny.

Ostatecznie zawartość obszaru pamięci o wielkości określonej przez MRANGE jest przemieszczana od adresu wskazanego wektorem OLDMHI do adresu wskazanego przez NEWMHI. Przemieszczenie to jest wykonywane od wyższych do niższych adresów, aby uniknąć zniszczenia przepisywanej informacji. W opróżnionym w ten sposób miejscu dotychczasowa zawartość pozostaje nadal, aż do wpisania nowych informacji (obszar ten nie jest zerowany).

Procedura INSEL jest często wykorzystywana przez interpreter, posiada także swój odpowiednik o odwrotnym działaniu - DELEL - opisany w następnym rozdziale.

Rozdział 2

ZAPIS PROGRAMU W PAMIĘCI

Gdyby program zapisany był w pamięci komputera w taki sposób, jak to widać na ekranie, to zajmowałby stosunkowo duży obszar tej pamięci. Ponadto wykonywanie tak zapisanego programu byłoby powolne ze względu na konieczność każdorazowego rozpoznawania instrukcji, operacji i zmiennych. Dlatego też interpreter tłumaczy każdy wprowadzony wiersz programu na zestaw specjalnych kodów zwanych tokenami, sprawdzając przy tym jego poprawność składniową, a następnie zakodowany (stokenizowany) wiersz programu umieszcza w tablicy instrukcji.

Podział obszaru zajmowanego przez program był już opisany we wprowadzeniu. Kolejne wiersze właściwego programu są umieszczane w pamięci według kolejności ich numerów, przy czym wiersze wprowadzone w trybie bezpośrednim otrzymują numer 32768 ($8000) i są zapisywane na końcu tablicy instrukcji.

W dalszej części tego rozdziału zostanie przedstawiona lista tokenów i sposób ich zapisu w pamięci komputera oraz proces tokenizacji. Proces ten jest jednakże bardzo skomplikowany ze względu na jednoczesną kontrolę składni instrukcji i jego zrozumienie wymaga bardzo dobrej znajomości języka maszynowego. Z tego powodu kontrola składni będzie przedstawiona tylko w sposób przybliżony, a bardziej zainteresowani tym tematem Czytelnicy muszą samodzielnie dokonać dokładnej analizy.

2.1. Tokeny Atari Basic

Podczas tokenizacji wprowadzony wiersz programu jest zamieniany na ciąg kodów. Rządzą tym oczywiście pewne reguły, które teraz zostaną podane. Jako przykład wykorzystamy krótki wiersz programu następującej postaci:

```
10 DIM NAZWA$(20):NAZWA$="ATARI":PRINT ALFA+5
```

Pierwsze dwa tokeny reprezentują numer wiersza w standardowej postaci - młodszy bajt, starszy bajt - a więc w naszym przypadku $0A, $00. Trzeci bajt określa długość wiersza (tu $28), a czwarty długość instrukcji. Przez długość wiersza należy rozumieć kolejny numer tokena symbolizującego koniec wiersza (znak RETURN - token $16). Podobnie długość instrukcji jest przedstawiona jako numer tokena oznaczającego dwukropek pomiędzy instrukcjami (token $14) lub koniec wiersza. Gdy w jednym wierszu znajduje się kilka instrukcji, to każda z nich rozpoczyna się od tokena określającego jej długość (w przykładzie są to kolejno $10, $1C i $28). Pozostałe tokeny zawierają właściwą treść wiersza programu.

Stałe liczbowe są przedstawione w postaci liczby zmiennoprzecinkowej poprzedzonej kodem określającym stałą liczbową (razem siedem bajtów). Zapis stałych tekstowych także zaczyna się od kodu stałej tekstowej, po którym znajduje się liczba znaków stałej, a następnie stała zapisana

znakami ATASCII. Zmienne są zapisywane przy pomocy ich numerów w tablicy nazw zmiennych. Numery te mają ustawiony najstarszy bit (są zwiększone o $80).

Każda instrukcja programu zaczyna się od tokena oznaczającego komendę języka Basic. Dalej umieszczone są tokeny stałych i zmiennych oraz tokeny operatorów i funkcji. Znaczenie tych tokenów jest przedstawione w poniższych tabelach. Numeracja wszystkich tokenów odpowiada ich położeniu w tablicach nazw instrukcji STNAME i operatorów OPFN oraz w tablicach wektorów instrukcji STVTAB i operatorów OPVTAB.

Tabela 1. Tokeny instrukcji Atari Basic

token HEX	i DEC	instrukcja
$00	0	REM
$01	1	DATA
$02	2	INPUT
$03	3	COLOR
$04	4	LIST
$05	5	ENTER
$06	6	LET
$07	7	IF
$08	8	FOR
$09	9	NEXT
$0A	10	GOTO
$0B	11	GO TO
$0C	12	GOSUB
$0D	13	TRAP
$0E	14	BYE
$0F	15	CONT
$10	16	COM
$11	17	CLOSE
$12	18	CLR
$13	19	DEG
$14	20	DIM
$15	21	END
$16	22	NEW
$17	23	OPEN
$18	24	LOAD
$19	25	SAVE
$1A	26	STATUS
$1B	27	NOTE
$1C	28	POINT
$1D	29	XIO
$1E	30	ON
$1F	31	POKE
$20	32	PRINT
$21	33	RAD
$22	34	READ
$23	35	RESTORE
$24	36	RETURN
$25	37	RUN
$26	38	STOP
$27	39	POP
$28	40	?
$29	41	GET
$2A	42	PUT
$2B	43	GRAPHICS
$2C	44	PLOT

```
$2D    45     POSITION
$2E    46     DOS
$2F    47     DRAWTO
$30    48     SETCOLOR
$31    49     LOCATE
$32    50     SOUND
$33    51     LPRINT
$34    52     CSAVE
$35    53     CLOAD
$36    54     opuszczone LET
$37    55     błąd składni
```

Wyjaśnienia wymagają dwa ostatnie tokeny. Jeśli nie zostanie rozpoznana prawidłowa komenda, to interpreter zakłada, że jest to nazwa zmiennej i wpisuje token instrukcji przypisania z opuszczonym słowem LET. W przypadku napotkania błędu składniowego w tokenizowanym wierszu na jego początku zapisywany jest token błędu składniowego (STNTAX ERROR), a cały wiersz jest przepisywany do tablicy instrukcji bez tokenizacji.

Pierwszą instrukcją naszego przykładu jest DIM, której odpowiada token $14. Taka właśnie liczba jest piątym tokenem wiersza. Kolejna instrukcja jest podstawieniem, w którym opuszczone zostało słowo LET - odpowiada jej token $36 (pozycja 18). Ostatnią instrukcją jest PRINT - token $20 na pozycji 30.

Po każdej z wymienionych instrukcji następuje zmienna. Pierwsza zmienna (NAZWA$) o numerze 0 jest oznaczona tokenem $80 (pozycje 6 i 19), zaś druga zmienna (ALFA) o numerze 1 - tokenem $81 (pozycja 31). Ponadto w przykładzie występują dwie stałe liczbowe: 20 (pozycje 8-14) i 5 (pozycje 33-39) oraz stała tekstowa "ATARI" (pozycje 21-27).

Na tym przykładzie wyraźnie widać oszczędność pamięci uzyskiwaną przez zastosowanie zmiennych zamiast stałych. Każda zmienna zajmuje miejsce w tablicy nazw (zwykle 2-3 bajty), w tablicy wartości zmiennych (8 bajtów) i tyle miejsc,ile razy wystąpi w programie, przy czym raz musi mieć nadaną wartość (12 bajtów). Jeśli stała występuje w programie cztery razy, to zajmie 4*7=28 bajtów. Po zastąpieniu jej przez stałą mamy 2+8+12+4*1=26 bajtów. Warto więc każdą stałą liczbową występującą w programie ponad trzy razy zastępować zmienną. Podobnie jest ze stałymi tekstowymi, lecz opłacalność takiego zabiegu zależy od długości stałej.

Tabela 2. Tokeny operatorów i funkcji Atari Basic

```
token HEX i DEC     operator lub funkcja
---------------------------------------
      $0E    14     stała liczbowa
      $0F    15     stała tekstowa
      $10    16     niewykorzystany
      $11    17     niewykorzystany
      $12    18     , -przecinek
      $13    19     $ -znak dolara
      $14    20     : -koniec instrukcji
      $15    21     ; -średnik
      $16    22     EOL -znak RETURN
      $17    23     GOTO -w ON
      $18    24     GOSUB -w ON
```

```
$19    25     TO -w FOR
$1A    26     STEP -w FOR
$1B    27     THEN -w IF
$1C    28     # -znak "numer"
$1D    29     <=  -+
$1E    30     <>   |
$1F    31     >=   | porównanie
$20    32     <    |   liczb
$21    33     >    |
$22    34     =   -+
$23    35     ^ -potęgowanie
$24    36     * -mnożenie
$25    37     + -dodawanie
$26    38     - -odejmowanie
$27    39     / -dzielenie
$28    40     NOT
$29    41     OR
$2A    42     AND
$2B    43     ( -wyr. arytmet.
$2C    44     ) -kończący
$2D    45     = -LET liczbowe
$2E    46     = -LET tekstowe
$2F    47     <=  -+
$30    48     <>   |
$31    49     >=   | porównanie
$32    50     <    |  tekstów
$33    51     >    |
$34    52     =   -+
$35    53     + -znak liczby
$36    54     - -znak liczby
$37    55     ( -zm. tekstowa
$38    56     ( -zm. tablicowa
$39    57     ( -wymiar zm. tabl.
$3A    58     ( -funkcja
$3B    59     ( -wymiar zm. tekst.
$3C    60     , -w zmiennej tablic.
$3D    61     STR$
$3E    62     CHR$
$3F    63     USR
$40    64     ASC
$41    65     VAL
$42    66     LEN
$43    67     ADR
$44    68     ATN
$45    69     COS
$46    70     PEEK
$47    71     SIN
$48    72     RND
$49    73     FRE
$4A    74     EXP
$4B    75     LOG
$4C    76     CLOG
$4D    77     SQR
$4E    78     SGN
$4F    79     ABS
$50    80     INT
$51    81     PADDLE
$52    82     STICK
$53    83     PTRIG
$54    84     STRIG
```

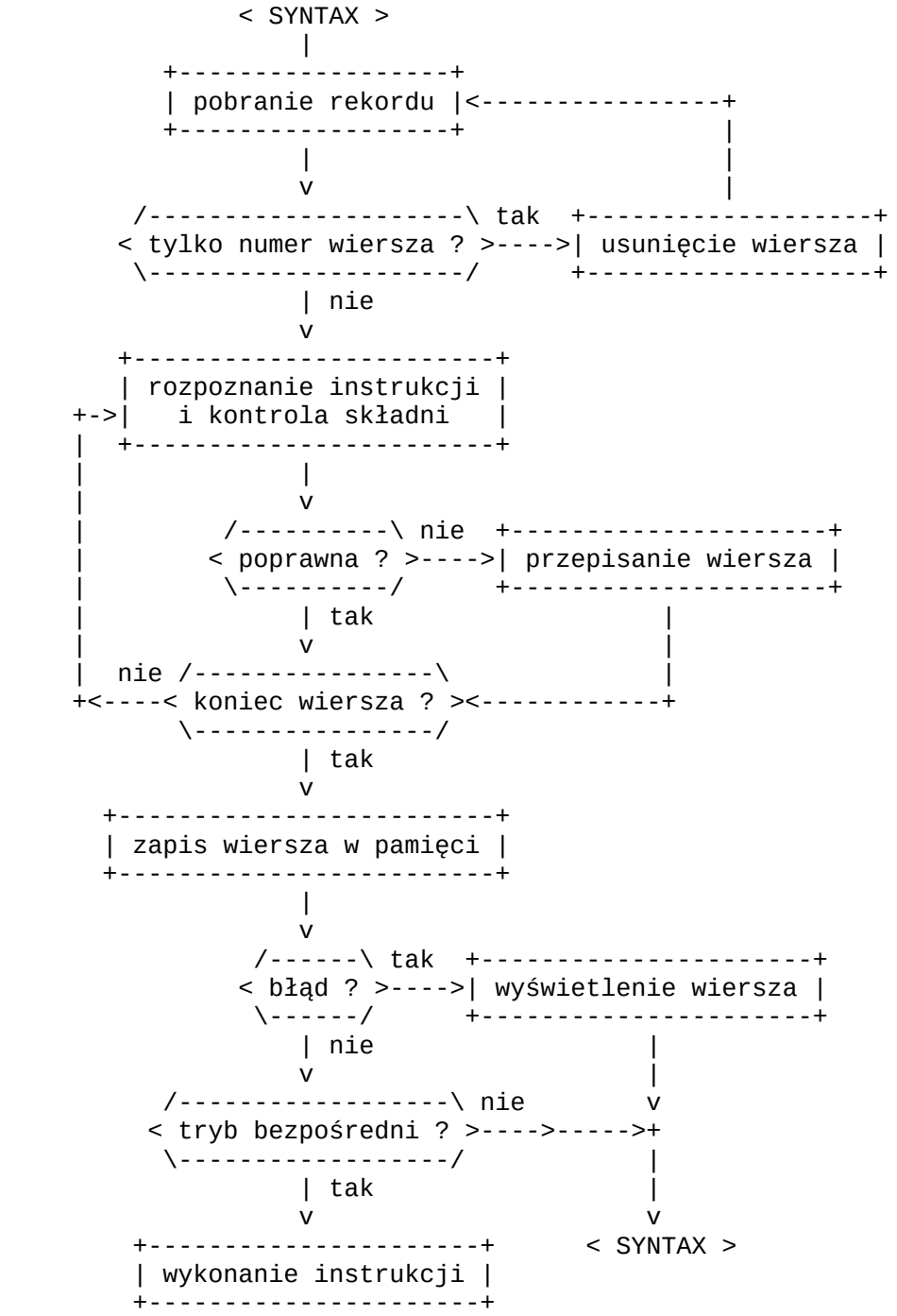
W tabeli tokenów operatorów zwraca uwagę duża liczba różnych tokenów dla nawiasów oraz oddzielenie identycznych operatorów wyrażeń tekstowych i liczbowych. Pomimo identycznego wyglądu wykonywane przez nie funkcje różnią się znacznie i zróżnicowanie tokenów powoduje następnie wybór różnych procedur wykonawczych przy realizacji programu.

Wygląd stokenizowanego przykładowego wiersza programu jest przedstawiony na następnej stronie.

pozycja	token	znaczenie
1	$0A	; numer wiersza
2	$00	; $000A = 10
3	$28	długość wiersza (40)
4	$10	dł. instrukcji (16)
5	$14	instrukcja DIM
6	$80	zmienna numer 0
7	$3B	(-nawias wymiaru
8	$0E	stała liczbowa
9	$40	|
10	$20	|
11	$00	| liczba 20
12	$00	| w kodzie BCD
13	$00	|
14	$00	|
15	$2C	) -nawias kończący
16	$14	: -koniec instrukcji
17	$1C	dł. instrukcji (28)
18	$36	opuszczone LET
19	$80	zmienna numer 0
20	$2E	= -LET tekstowe
21	$0F	stała tekstowa
22	$05	długość stałej
23	$41	A |
24	$54	T |
25	$41	A | kody ATASCII
26	$52	R |
27	$49	I |
28	$14	: -koniec instrukcji
29	$28	dł. instrukcji (40)
30	$20	instrukcja PRINT
31	$81	zmienna numer 1
32	$25	+ -dodawanie
33	$0E	stała liczbowa
34	$40	|
35	$05	|
36	$00	| liczba 5
37	$00	| w kodzie BCD
38	$00	|
39	$00	|
40	$16	EOL - koniec wiersza

2.2. Proces tokenizacji

Tokenizacja wiersza programu jest nieodłącznie związana z procesem kontroli jego składni. Czynności te są wykonywane przez procedurę SYNTAX, która jest największym elementem interpretera.



Rys.2. Struktura procedury SYNTAX

```
0100 ;check SYNTAX           0660     STA COX
0110 ;                       0670     STA SXFLG
0120 AUXBR = $AF             0680     STA TMPIX
```

```
0130 BHLD1 = $B0
0140 BHLD2 = $B1
0150 BUFIX = $9F
0160 CDST =  $A000
0170 CIX =   $F2
0180 COX =   $94
0190 CSTAD = $97
0200 DELEL = $A8F7
0210 FNDCST = $A9A2
0220 FR0 =   $D4
0230 GETREC = $BDED
0240 GIRQST = $A9F2
0250 GLNLEN = $A9DC
0260 INBSS = $DBA1
0270 INBUFP = $F3
0280 INIX =  $A8
0290 INSEL = $A87A
0300 LOADFLG = $CA
0310 LOMEM = $80
0320 LSTPLN = $B58E
0330 MAXLN = $AD
0340 NXTSTM = $A9D0
0350 OUTIX = $A7
0360 PLINEN = $A19A
0370 PRCSTM = $A95E
0380 PROMPT = $C2
0390 PRTPLN = $B5AA
0400 RECNAM = $A454
0410 SAVTKN = $A2C4
0420 STBV =  $DA51
0430 STMCUR = $8A
0440 STMSX = $A1BE
0450 STMTAB = $88
0460 STNAME = $A49F
0470 SXFLG = $A6
0480 TMPIX = $B3
0490 VNTD =  $84
0500 ;
0510     *=  $A060
0520 ;
0530 SYNTAX LDA LOADFLG
0540     BNE CDST
0550     LDX #$FF
0560     TXS
0570     JSR STBV
0580     LDA #$5D
0590     STA PROMPT
0600     JSR GETREC
0610     JSR GIRQST
0620     BEQ SYNTAX
0630     LDA #$00
0640     STA CIX
0650     STA BUFIX

0690     STA BHLD1
0700     STA BHLD2
0710     LDA VNTD
0720     STA MAXLN
0730     LDA VNTD+1
0740     STA MAXLN+1
0750     JSR INBSS
0760     JSR PLINEN
0770     JSR SAVTKN
0780     LDA FR0+1
0790     BPL DCD
0800     STA SXFLG
0810 DCD JSR INBSS
0820     LDY CIX
0830     STY INIX
0840     LDA (INBUFP),Y
0850     CMP #$9B
0860     BNE DCDSTM
0870     BIT SXFLG
0880     BMI SYNTAX
0890     JMP DELPLN
0900 ;
0910 DeCoDe STateMent
0920 ;
0930 DCDSTM LDA COX
0940     STA OUTIX
0950     JSR SAVTKN
0960     JSR INBSS
0970     LDA # >STNAME
0980     LDY # <STNAME
0990     LDX #$02
1000     JSR RECNAM
1010     STX CIX
1020     LDA AUXBR
1030     JSR SAVTKN
1040     JSR INBSS
1050     JSR STMSX
1060     BCC SAVL
1070     LDY BUFIX
1080     LDA (INBUFP),Y
1090     CMP #$9B
1100     BNE MCH
1110     INY
1120     STA (INBUFP),Y
1130     DEY
1140     LDA #$20
1150 MCH ORA #$80
1160     STA (INBUFP),Y
1170     LDA #$40
1180     ORA SXFLG
1190     STA SXFLG
1200     LDY INIX
1210     STY CIX

1220     LDX #$03
1230     STX OUTIX
1240     INX
1250     STX COX
1260     LDA #$37
1270 LP1 JSR SAVTKN
1280 ;

1670     JSR NXTSTM
1680     LDX #STMCUR
1690     JSR DELEL+1
1700 STORE LDY COX
1710 LP2 DEY
1720     LDA (LOMEM),Y
1730     STA (STMCUR),Y
```

```
1290 ;MOVe ToKeNs
1300 ;
1310 MOVTKN LDY CIX
1320     LDA (INBUFP),Y
1330     INC CIX
1340     CMP #$9B
1350     BNE LP1
1360     JSR SAVTKN
1370 SAVL LDA COX
1380     LDY OUTIX
1390     STA (LOMEM),Y
1400     LDY CIX
1410     DEY
1420     LDA (INBUFP),Y
1430     CMP #$9B
1440     BNE DCDSTM
1450     LDY #$02
1460     LDA COX
1470     STA (LOMEM),Y
1480     JSR FNDCST
1490     LDA #$00
1500     BCS LEN
1510     JSR GLNLEN
1520 LEN SEC
1530     SBC COX
1540     BEQ STORE
1550     BCS SML
1560     EOR #$FF
1570     TAY
1580     INY
1590     LDX #STMCUR
1600     JSR INSEL
1610     LDA CSTAD
1620     STA STMCUR
1630     LDA CSTAD+1
1640     STA STMCUR+1
1650     BNE STORE
1660 SML TAY
1740     TYA
1750     BNE LP2
1760     BIT SXFLG
1770     BVC EXIT
1780     LDA BHLD2
1790     ASL A
1800     ASL A
1810     ASL A
1820     LDX #STMTAB
1830     JSR DELEL
1840     SEC
1850     LDA VNTD
1860     SBC MAXLN
1870     TAY
1880     LDA VNTD+1
1890     SBC MAXLN+1
1900     LDX #VNTD
1910     JSR DELEL+3
1920     BIT SXFLG
1930     BPL LIST
1940     JSR PRTPLN
1950     JMP SYNTAX
1960 LIST JSR LSTPLN
1970 LOOP JMP SYNTAX
1980 EXIT BPL LOOP
1990     JMP PRCSTM
2000 ;
2010 ;DELete Program LiNe
2020 ;
2030 DELPLN JSR FNDCST
2040     BCS LOOP
2050     JSR GLNLEN
2060     TAY
2070     JSR NXTSTM
2080     LDX #STMCUR
2090     JSR DELEL+1
2100     JMP SYNTAX
```

SYNTAX rozpoczyna się od wywołania procedury STBV (zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego"), która wpisuje adres bufora LBUFF (Line BUFFer) do rejestru INBUFF (INput BUFfer Pointer). Następnie przez wywołanie procedury GETREC (jest ona częścią PUTRET - zob. rozdział 6.6.10) odczytywany jest z edytora wiersz programu. Przed przystąpieniem do jego analizy wywoływana jest jeszcze procedura GIRQST. Służy ona do sprawdzenia, czy został naciśnięty klawisz BREAK. Jeżeli tak, to następuje skok do początku procedury SYNTAX, a w przeciwnym wypadku zerowane są niezbędne rejestry (CIX, COX, BUFIX, TMPIX, SXFLG, BHLD1 i BHLD2), a wektor VNTD jest przepisywany do MAXLN. Dopiero po tym wstępie rozpoczyna się kontrola składni wprowadzonego wiersza i jego tokenizacja.

```
0100 ;Get IRQ STatus
0110 ;
0120 IRQSTAT = $11
0130 ;
0140     *=  $A9F2
0150 ;
0160     LDY IRQSTAT
```

```
0170     BNE END
0180     DEC IRQSTAT
0190     TYA
0200 END RTS
```

Podczas tych operacji często są wywoływane procedury INBSS i SAVTKN. Pierwsza z nich znajduje się w pakiecie procedur zmiennoprzecinkowych i ustawia licznik bufora CIX (Current Input indeX) na następnym znaku innym niż spacja. Druga zapisuje przekazany jej w akumulatorze token na odpowiedniej pozycji bufora tokenizacji. Pozycja ta jest wskazywana przez licznik COX (Current Output indeX), a przekroczenie wartości równej $FF powoduje sygnalizację nadmiernej długości wiersza (błąd Line Too LonG ERror). Warto zauważyć, że dla zaoszczędzenia bajtu pamięci SAVTKN korzysta z rozkazu RTS umieszczonego w poprzedzającej ją procedurze.

```
0100 ;SAVe ToKeN
0110 ;
0120 COX =   $94
0130 LOMEM = $80
0140 LTLGER = $B918
0150 ;
0160     *=  $A2C4
0170 ;
0180     LDY COX
0190     STA (LOMEM),Y
0200     INC COX
0210     BNE $A2C3   ;RTS
0220     JMP LTLGER
```

Jako pierwszy rozpoznawany jest numer wprowadzonego wiersza. Wykonuje to procedura PLINEN. Wywołuje ona najpierw procedurę AFP (pakiet FP), która zamienia ciąg znaków ASCII na liczbę. Jeśli jest to niemożliwe (znaki ASCII nie są cyframi), to licznik CIX jest zerowany, a wiersz otrzymuje numer $8000, co oznacza tryb bezpośredni. Po rozpoznaniu liczby wywoływana jest krótka procedura GETINT zamieniająca otrzymany wynik na dwubajtową liczbę całkowitą. Gdy wartość tej liczby przekracza $8000, to także jest redukowana do tej wartości. Otrzymany w rezultacie numer wiersza jest umieszczany w rejestrze CLNN (Current LiNe Number) oraz przez dwukrotne wywołanie SAVTKN przepisywany do bufora tokenizacji.

```
0100 ;Program LINE Number
0110 ;
0120 AFP =   $D800
0130 CIX =   $F2
0140 CLNN =  $A0
0150 FR0 =   $D4
0160 GETINT = $AD41
0170 SAVTKN = $A2C4
0180 ;
0190     *=  $A19A
0200 ;
0210     JSR AFP
0220     BCC LIN
0230 DIR LDA #$00
0240     STA CIX
0250     LDY #$80
0260     BMI STR
0270 LIN JSR GETINT
0280     LDY FR0+1
```

```
0290     BMI DIR
0300     LDA FR0
0310 STR STY CLNN+1
0320     STA CLNN
0330     JSR SAVTKN
0340     LDA CLNN+1
0350     STA FR0+1
0360     JMP SAVTKN
```

Wspomniana wcześniej procedura GETINT została wprowadzona, aby umożliwić właściwą sygnalizację błędu. Wywołuje ona procedurę FPI z pakietu FP i, w przypadku niepoprawnego wyniku, sygnalizuje błędną wartość przez skok do BVALER (Bad VALue ERror).

```
0100 ;GET INTeger value
0110 ;
0120 BVALER = $B92E
0130 FPI =   $D9D2
0140 ;
0150     *=  $AD41
0160 ;
0170     JSR FPI
0180     BCS ERR
0190     RTS
0200 ERR JSR BVALER
```

Kolejne wywołanie procedury SAVTKN ma na celu zwiększenie licznika COX, aby pozostawić miejsce na token określający długość wprowadzonego wiersza. Jeśli wiersz ten jest w trybie bezpośrednim, to znacznik SXFLG (SyntaX FLaG) otrzymuje wartość $80. Teraz sprawdzany jest kod następnego znaku. Jeśli jest to koniec wiersza ($9B) i wiersz jest w trybie bezpośrednim, to następuje powrót do początku procedury SYNTAX. Gdy wiersz zawiera tylko poprawny numer, następuje skok do etykiety DELPLN, gdzie wykonywane jest usunięcie wiersza o podanym numerze z tablicy instrukcji (zob. 2.2.1. Usuwanie wiersza).

W przypadku nierozpoznania końca wiersza aktualna zawartość licznika COX jest przepisywana do rejestru OUTIX (OUTput IndeX) i przez wywołanie SAVTKN rezerwowane jest miejsce na token długości instrukcji. Kolejnym etapem jest przeszukiwanie tablicy nazw instrukcji przez procedurę RECNAM. Tablica nazw STNAME - oprócz samych nazw - zawiera wektory wskazujące położenie w tablicy składni danych określających wymaganą składnię instrukcji.

```
0100 ;STatement NAME table    0570     .CBYTE "LET"
0110 ;                        0580     .WORD SIF-1
0120 SBYE =  $A6B9            0590     .CBYTE "IF"
0130 SCOLOR = $A6B8           0600     .WORD SFOR-1
0140 SCLOSE = $A71B           0610     .CBYTE "FOR"
0150 SDATA = $A7C6            0620     .WORD SNEXT-1
0160 SDIM =  $A75A            0630     .CBYTE "NEXT"
0170 SENTER = $A71E           0640     .WORD SCOLOR-1
0180 SFOR =  $A6CD            0650     .CBYTE "GOTO"
0190 SGET =  $A6E3            0660     .WORD SCOLOR-1
0200 SIF =   $A78F            0670     .CBYTE "GO TO"
0210 SINPUT = $A6EF           0680     .WORD SCOLOR-1
0220 SLET =  $A6BB            0690     .CBYTE "GOSUB"
0230 SLIST = $A72D            0700     .WORD SCOLOR-1
0240 SLOCAT = $A6DD           0710     .CBYTE "TRAP"
```

```
0250 SLPRNT = $A6FB
0260 SNEXT = $A6E5
0270 SNOTE = $A744
0280 SON =   $A75D
0290 SOPEN = $A714
0300 SPOKE = $A757
0310 SPRINT = $A6F7
0320 SPUT =  $A6B5
0330 SREAD = $A6F0
0340 SREM =  $A7C3
0350 SRSTR = $A6EA
0360 SRUN =  $A721
0370 SSETC = $A755
0380 SSOUND = $A753
0390 SSTAT = $A73B
0400 SXIO =  $A712
0410 ;
0420     *=  $A49F
0430 ;
0440     .WORD SREM-1
0450     .CBYTE "REM"
0460     .WORD SDATA-1
0470     .CBYTE "DATA"
0480     .WORD SINPUT-1
0490     .CBYTE "INPUT"
0500     .WORD SCOLOR-1
0510     .CBYTE "COLOR"
0520     .WORD SLIST-1
0530     .CBYTE "LIST"
0540     .WORD SENTER-1
0550     .CBYTE "ENTER"
0560     .WORD SLET-1

1040     .WORD SON-1
1050     .CBYTE "ON"
1060     .WORD SPOKE-1
1070     .CBYTE "POKE"
1080     .WORD SPRINT-1
1090     .CBYTE "PRINT"
1100     .WORD SBYE-1
1110     .CBYTE "RAD"
1120     .WORD SREAD-1
1130     .CBYTE "READ"
1140     .WORD SRSTR-1
1150     .CBYTE "RESTORE"
1160     .WORD SBYE-1
1170     .CBYTE "RETURN"
1180     .WORD SRUN-1
1190     .CBYTE "RUN"
1200     .WORD SBYE-1
1210     .CBYTE "STOP"
1220     .WORD SBYE-1
1230     .CBYTE "POP"
1240     .WORD SPRINT-1
1250     .CBYTE "?"
1260     .WORD SGET-1
1270     .CBYTE "GET"
1280     .WORD SPUT-1
1290     .CBYTE "PUT"
0720     .WORD SBYE-1
0730     .CBYTE "BYE"
0740     .WORD SBYE-1
0750     .CBYTE "CONT"
0760     .WORD SDIM-1
0770     .CBYTE "COM"
0780     .WORD SCLOSE-1
0790     .CBYTE "CLOSE"
0800     .WORD SBYE-1
0810     .CBYTE "CLR"
0820     .WORD SBYE-1
0830     .CBYTE "DEG"
0840     .WORD SDIM-1
0850     .CBYTE "DIM"
0860     .WORD SBYE-1
0870     .CBYTE "END"
0880     .WORD SBYE-1
0890     .CBYTE "NEW"
0900     .WORD SOPEN-1
0910     .CBYTE "OPEN"
0920     .WORD SENTER-1
0930     .CBYTE "LOAD"
0940     .WORD SENTER-1
0950     .CBYTE "SAVE"
0960     .WORD SSTAT-1
0970     .CBYTE "STATUS"
0980     .WORD SNOTE-1
0990     .CBYTE "NOTE"
1000     .WORD SNOTE-1
1010     .CBYTE "POINT"
1020     .WORD SXIO-1
1030     .CBYTE "XIO"

1300     .WORD SCOLOR-1
1310     .CBYTE "GRAPHICS"
1320     .WORD SPOKE-1
1330     .CBYTE "PLOT"
1340     .WORD SPOKE-1
1350     .CBYTE "POSITION"
1360     .WORD SBYE-1
1370     .CBYTE "DOS"
1380     .WORD SPOKE-1
1390     .CBYTE "DRAWTO"
1400     .WORD SSETC-1
1410     .CBYTE "SETCOLOR"
1420     .WORD SLOCAT-1
1430     .CBYTE "LOCATE"
1440     .WORD SSOUND-1
1450     .CBYTE "SOUND"
1460     .WORD SLPRNT-1
1470     .CBYTE "LPRINT"
1480     .WORD SBYE-1
1490     .CBYTE "CSAVE"
1500     .WORD SBYE-1
1510     .CBYTE "CLOAD"
1520     .WORD SLET-1
1530     .CBYTE $00,$00
1540     .WORD $2A00
1550 ERMSG .CBYTE "ERROR-  "
```

Przed wywołaniem procedury RECNAM w akumulatorze i rejestrze Y umieszczany jest adres przeszukiwanej tablicy. Rejestr X zawiera natomiast indeks określający, od którego znaku należy porównywać nazwę. Zwykle jest to zero, lecz dla tablicy nazw instrukcji indeks jest równy 2, gdyż dwa pierwsze bajty są wektorem do tablicy składni. Przed rozpoczęciem przeszukiwania indeks z rejestru X jest przepisywany do rejestru STPTR (STack PoinTeR).

Procedura RECNAM działa w pętli. Najpierw adres tablicy jest zapisywany do rejestru POKADR (POKe ADdRess), a kolejny numer sprawdzanej nazwy (licząc od zera) do rejestru AUXBR (AUXiliary Basic Register). Z tablicy odczytywany jest bajt i jeśli jest równy zero (koniec tablicy), to procedura się kończy, a ustawiony bit Carry sygnalizuje niepowodzenie. W przeciwnym wypadku kolejne znaki są odczytywane z bufora wejściowego i porównywane z zawartością tablicy. Rezultat porównania jest zapisywany na stosie. Przerwanie porównywania znaków następuje po osiągnięciu końca nazwy w tablicy, a w przypadku instrukcji także po odczytaniu z bufora kropki. Koniec nazwy jest zawsze oznaczony w tablicy ustawieniem najstarszego bitu w ostatnim znaku nazwy.

Teraz wynik porównania jest odczytywany ze stosu. Gdy jest on negatywny, pętla się powtarza. Odnalezienie nazwy w tablicy jest sygnalizowane przez skasowanie bitu Carry. W takim przypadku po zakończeniu procedury RECNAM rejestr AUXBR zawiera numer kolejny rozpoznanej nazwy, a w rejestrze X znajduje się indeks następnego znaku w buforze wejściowym i po powrocie do miejsca wywołania licznik CIX jest uaktualniany według tej wartości. Przy przeszukiwaniu tablicy STNAME trzeba zwrócić uwagę na fakt, że jeśli nie została rozpoznana poprawna nazwa instrukcji, to rejestr AUXBR zawiera wartość $36, czyli token instrukcji przypisania z opuszczonym słowem LET. Każda nazwa, inna niż umieszczona w tablicy STNAME, jest więc traktowana jako nazwa zmiennej.

```
0100 ;RECognize NAMe
0110 ;
0120 AUXBR = $AF
0130 CIX =   $F2
0140 LBUFF = $0580
0150 POKADR = $95
0160 STPTR = $AA
0170 ;
0180     *=  $A454
0190 ;
0200 RECNAM STX STPTR
0210     LDX #$FF
0220     STX AUXBR
0230 SAV STA POKADR+1
0240     STY POKADR
0250     INC AUXBR
0260     LDX CIX
0270     LDY STRPTR
0280     LDA (POKADR),Y
0290     BEQ END
0300     LDA #$00
0310     PHP
0320 RDC LDA LBUFF,X
0330     AND #$7F
```

```
0340     CMP #
0350     BEQ CST
0360 CHK EOR (POKADR),Y
0370     ASL A
0380     BEQ NXT
0390     PLA
0400     PHP
0410 NXT INY
0420     INX
0430     BCC RDC
0440     PLP
0450     BEQ $A452   ;CLC,RTS
0460 ;
0470 ;NeXT NAMe
0480 ;
0490 NXTNAM CLC
0500     TYA
0510     ADC POKADR
0520     TAY
0530     LDA POKADR+1
0540     ADC #$00
0550     BNE SAV
0560 END SEC
0570     RTS
0580 CST LDA #$02
0590     CMP STPTR
0600     BNE CHK
0610 CHR LDA (POKADR),Y
0620     BMI LST
0630     INY
0640     BNE CHR
0650 LST SEC
0660     BCS NXT
```

Rozkaz AND #$7F w procedurze RECNAM (wiersz 330) kasuje najstarszy bit znaku pobranego z bufora wejściowego. Dzięki temu - wbrew rozpowszechnionym opiniom - interpreter Atari Basic rozpoznaje wszystkie instrukcje, funkcje i operatory zapisane w negatywie (inverse video)! Sygnalizację błędu powoduje jedynie wpisanie w negatywie stałych liczbowych oraz nazw zmiennych. Przykład ten świadczy o tym, jak często negatywne opinie o Atari są powtarzane bez sprawdzenia stanu faktycznego.

Odczytany token instrukcji jest teraz zapisywany do bufora tokenizacji i wywoływana jest procedura STMSX, która sprawdza poprawność składniową tej instrukcji. Procedura STMSX jest przedstawiona w rozdziale 2.2.2.

Jeśli został stwierdzony błąd składni, to miejsce jego wystąpienia jest zaznaczane ustawieniem najstarszego bitu znaku, a jako piąty token wiersza wpisywana jest wartość $37 (SYNTAX ERROR). Następnie cała zawartość bufora wejściowego jest przepisywana do bufora tokenizacji. Dodatkowo w rejestrze SXFLG ustawiany jest bit 6, co umożliwia późniejsze rozpoznanie błędnego wiersza.

Po stokenizowaniu całej instrukcji aktualny stan licznika COX jest zapisywany w miejscu wskazanym przez zawartość rejestru OUTIX, czyli na początku instrukcji. W ten sposób

otrzymujemy token określający długość instrukcji. Jeśli ostatnim odczytanym znakiem nie był znak końca wiersza (EOL), to procedura się powtarza od rozpoznania nazwy następnej instrukcji w wierszu. W przeciwnym razie stan licznika COX jest wpisywany jako trzeci token, a więc jako długość wiersza.

Teraz stokenizowany wiersz musi być umieszczony w odpowiednim miejscu tablicy instrukcji. Odszukanie tego miejsca jest zadaniem procedury FNDCST. Przeszukuje ona całą tablicę instrukcji, aż do napotkania wiersza o takim samym lub wyższym numerze. Sygnalizuje przy tym rezultat poszukiwania przez ustawienie bitu Carry, gdy numery są różne, lub jego skasowanie, gdy są równe. Odpowiednio do tego w akumulatorze jest umieszczana wartość zero lub długość wiersza odczytana przez ponowne wywołanie procedury GLNLEN.

```
0100 ;FiND Current STatement
0110 ;
0120 CLNN =  $A0
0130 GLNLEN = $A9DC
0140 NXTSTM = $A9D0
0150 SAVCUR = $BE
0160 STMCUR = $8A
0170 STMTAB = $88
0180 ;
0190     *=  $A9A2
0200 ;
0210     LDA STMCUR
0220     STA SAVCUR
0230     LDA STMCUR+1
0240     STA SAVCUR+1
0250     LDA STMTAB+1
0260     LDY STMTAB
0270     STA STMCUR+1
0280     STY STMCUR
0290 LOOP LDY #$01
0300     LDA (STMCUR),Y
0310     CMP CLNN+1
0320     BCC NXT
0330     BNE END
0340     DEY
0350     LDA (STMCUR),Y
0360     CMP CLNN
0370     BCC NXT
0380     BNE END
0390     CLC
0400 END RTS
0410 NXT JSR GLNLEN
0420     JSR NXTSTM
0430     JMP LOOP
```

Przy przeszukiwaniu zawartości tablicy instrukcji są wykorzystywane dwie krótkie procedury GLNLEN i NXTSTM. Pierwsza z nich odczytuje token określający długość aktualnie sprawdzanego wiersza. Druga dodaje tą długość do adresu wiersza dając w wyniku adres następnego wiersza programu.

```
0100 ;Get LiNe LENgth
0110 ;
0120 STMCUR = $8A
```

```
0130 ;
0140     *=  $A9DC
0150 ;
0160     LDY #$02
0170     LDA (STMCUR),Y
0180     RTS

0100 ;NeXT STateMent
0110 ;
0120 STMCUR = $8A
0130 ;
0140     *=  $A9D0
0150 ;
0160     CLC
0170     ADC STMCUR
0180     STA STMCUR
0190     LDA STMCUR+1
0200     ADC #$00
0210     STA STMCUR+1
0220     RTS
```

Znajdująca się w akumulatorze długość wiersza jest porównywana z długością nowego wiersza określoną przez stan licznika COX. Zależnie od różnicy długości starego i nowego wiersza programu wykonywana jest odpowiednia sekwencja instrukcji. Jeśli nowy wiersz jest dłuższy, to miejsce dla niego tworzone jest przez wywołanie procedury INSEL, a gdy krótszy, to nadmiar miejsca kasuje procedura DELEL. Przy równych długościach starego i nowego wiersza ten fragment jest pomijany. Po przygotowaniu miejsca stokenizowany wiersz programu jest przepisywany do tablicy instrukcji z bufora tokenizacji.

Pozostaje jeszcze zakończyć procedurę SYNTAX. W celu wybrania odpowiedniego wariantu zakończenia testowany jest rejestr SXFLG. Gdy ma on ustawiony bit 6, to oznacza, że wpisany wiersz jest niepoprawny. W takim przypadku przez dwukrotne wywołanie procedury DELEL usuwane są z tablic nazw i wartości zmiennych informacje wprowadzone tam podczas tokenizacji tego wiersza. Następnie błędny wiersz jest wyświetlany na ekranie przez procedurę LSTPLN (wiersz w trybie programowym) lub PRTPLN (wiersz w trybie bezpośrednim), po czym wykonywany jest skok do początku procedury SYNTAX.

Powyższa informacja przeczy obiegowej opinii, że interpreter Atari Basic gromadzi w tablicach nazw i wartości zmiennych śmieci powstałe na skutek błędów przy wpisywaniu programu. Opinia ta jest prawdziwa TYLKO w przypadku poprzedniej wersji interpretera (Revision B).

Jeżeli wiersz jest poprawny, to w przypadku trybu programowego także następuje powrót do początku SYNTAX, zaś w trybie bezpośrednim wykonywany jest skok do procedury PRCSTM, która powoduje wykonanie wprowadzonego wiersza (zob. rozdział 2.2.3).

2.2.1. Usuwanie wiersza

Usuwanie wiersza z programu jest wykonywane przez część procedury SYNTAX oznaczoną etykietą DELPLN. Jej działanie jest bardzo proste. Najpierw przez wywołanie procedury FNDCST

odszukiwany jest w pamięci wiersz do usunięcia. Jeśli go nie ma, to procedura jest przerywana skokiem do początku SYNTAX. Po odnalezieniu wiersza odczytywana jest jego długość (przez GLNLEN), a następnie obliczany jest adres początkowy kolejnego wiersza (przez NXTSTM). Teraz wywoływana jest procedura DELEL, która przesuwa znajdującą się wyżej zawartość pamięci i w ten sposób kasuje wskazany element programu. Procedura DELPLN kończy się skokiem do początku procedury SYNTAX.

Wielokrotnie już wymieniana procedura DELEL jest odwrotnością procedury INSEL. Przemieszcza ona podobnie zawartość obszaru pamięci, lecz w przeciwnym kierunku. Stosowane są trzy sposoby wywołania procedury DELEL: od DELEL, DELEL+1 i DELEL+3. W pierwszym i drugim przypadku następuje przemieszczenie na odległość nie przekraczającą jednej strony pamięci, gdyż starszy bajt wielkości przemieszczenia ma wtedy wartość zero. Młodszy bajt jest przekazywany do procedury w akumulatorze (przy wywołaniu od DELEL) lub w rejestrze Y (przy wywołaniu od DELEL+1). Przemieszczenie obszaru pamięci na odległość przekraczającą jedną stronę (256 bajtów) następuje po wywołaniu od DELEL+3. Akumulator musi wtedy zawierać starszy bajt wielkości przemieszczenia, a rejestr Y młodszy bajt.

```
0100 ;DELete program ELement
0110 ;
0120 APPMHI = $0E
0130 BMEMHI = $90
0140 LENPEL = $A4
0150 MEOLFLG = $92
0160 MRANGE = $A2
0170 NEWMHI = $9B
0180 OLDMHI = $99
0190 ;
0200     *=  $A8F7
0210 ;
0220     TAY
0230     LDA #$00
0240     STY LENPEL
0250     STA LENPEL+1
0260     SEC
0270     LDA BMEMHI
0280     SBC $00,X
0290     EOR #$FF
0300     TAY
0310     INY
0320     STY MRANGE
0330     LDA BMEMHI+1
0340     SBC $01,X
0350     STA MRANGE+1
0360     LDA $00,X
0370     SBC MRANGE
0380     STA OLDMHI
0390     LDA $01,X
0400     SBC #$00
0410     STA OLDMHI+1
0420     STX NEWMHI
0430 LOOP SEC
0440     LDA $00,X
0450     SBC LENPEL
0460     STA $00,X
```

```
0470     LDA $01,X
0480     SBC LENPEL+1
0490     STA $01,X
0500     INX
0510     INX
0520     CPX #MEOLFLG
0530     BCC LOOP
0540     STA APPMHI+1
0550     LDA BMEMHI
0560     STA APPMHI
0570     LDX NEWMHI
0580     LDA $00,X
0590     SBC MRANGE
0600     STA NEWMHI
0610     LDA $01,X
0620     SBC #$00
0630     STA NEWMHI+1
0640 ;
0650 ;MOVe MEMory
0660 ;
0670 MOVMEM LDX MRANGE+1
0680     INX
0690     LDY MRANGE
0700     BNE MOVE
0710     DEX
0720     BNE MOVE
0730     RTS
0740 NEXT INC OLDMHI+1
0750     INC NEWMHI+1
0760 MOVE LDA (OLDMHI),Y
0770     STA (NEWMHI),Y
0780     INY
0790     BNE MOVE
0800     DEX
0810     BNE NEXT
0820     RTS
```

Szczegółowy opis działania procedury DELEL jest zbędny, gdyż stanowi ona odwrotność INSEL i zasada działania obu procedur jest jednakowa. Trzeba jednak zwrócić uwagę na etykietę MOVMEM. Wywołanie procedury od tego miejsca umożliwia przemieszczenie dowolnego obszaru pamięci w stronę niższych adresów, a jeśli stary i nowy obszar nie pokrywają się, to także w stronę wyższych adresów. Zamiast wymyślać do swoich programów nowe procedury przemieszczeń warto więc wykorzystać już istniejący fragment interpretera.

2.2.2. Kontrola składni

Po rozpoznaniu instrukcji wywoływana jest procedura STMSX, która dokonuje kontroli składni tej instrukcji i jednocześnie przeprowadza jej tokenizację. Przed przystąpieniem do tej czynności wektor umieszczony w tabeli STNAME przed nazwą instrukcji jest przepisywany do rejestrów PCNTC (Program CouNTer Current register) i BPRCNT (Basic PRogram CouNTer). Przepisywane są także stany liczników: COX do BOUTIX (Basic OUTput IndeX) oraz CIX do BINIX (Basic INput IndeX).

```
0100 ;STateMent SyntaX        0550     LDX #$FF
0110 ;                        0560 ADPC CLC
0120 BINIX = $0480            0570     ADC PCNTC
```

```
0130 BOUTIX = $0481
0140 BPRCNT = $0482
0150 BUFIX = $9F
0160 CIX =   $F2
0170 COX =   $94
0180 GTCCHR = $A293
0190 INCPRC = $A28C
0200 LTLGER = $B918
0210 PCNTC = $9D
0220 POKADR = $95
0230 STIX =  $A9
0240 VALUE = $A29B
0250 ;
0260     *=  $A1BE
0270 ;
0280 STMSX LDY #$01
0290     LDA (POKADR),Y
0300     STA PCNTC+1
0310     STA BPRCNT+1
0320     DEY
0330     LDA (POKADR),Y
0340     STA PCNTC
0350     STA BPRCNT
0360     STY STIX
0370     LDA COX
0380     STA BOUTIX
0390     LDA CIX
0400     STA BINIX
0410 CHAR JSR GTCCHR
0420     BMI NEG
0430     CMP #$01
0440     BCC PHADR
0450     BNE CVL
0460     JSR PHADR
0470     JMP RSM
0480 CVL CMP #$05
0490     BCC GPC
0500     JSR VALUE
0510     JMP RSM
0520 NEG SEC
0530     SBC #$C1
0540     BCS ADPC

1000     PLA
1010     STA PCNTC
1020     JMP CHAR
1030 ERR JMP LTLGER
1040 GPC LDX STIX
1050     BEQ EXIT
1060     LDA BPRCNT,X
1070     STA PCNTC
1080     LDA BPRCNT+1,X
1090     STA PCNTC+1
1100     DEX
1110     DEX
1120     DEX
1130     DEX
1140     STX STIX
1150 RSM BCS NXT
1160     JMP CHAR
1170 NXT JSR GTCCHR
0580     PHA
0590     TXA
0600     ADC PCNTC+1
0610     PHA
0620     JMP SAVCPM
0630 ;
0640 ;PusH ADdRess
0650 ;
0660 PHADR JSR STCCHR
0670     PHA
0680     JSR GTCCHR
0690     PHA
0700     BCC SAVCPM
0710     PLA
0720     TAY
0730     PLA
0740     TAX
0750     TYA
0760     PHA
0770     TXA
0780     PHA
0790 EXIT RTS
0800 ;
0810 ;SAVe Current ParaMeters
0820 ;
0830 SAVCPM LDX STIX
0840     INX
0850     INX
0860     INX
0870     INX
0880     BEQ ERR
0890     STX STIX
0900     LDA CIX
0910     STA BINIX,X
0920     LDA COX
0930     STA BOUTIX,X
0940     LDA PCNTC
0950     STA BPRCNT,X
0960     LDA PCNTC+1
0970     STA BPRCNT+1,X
0980     PLA
0990     STA PCNTC+1

1190     CMP #$02
1200     BCS NVL
1210     JSR INCPRC
1220     JSR INCPRC
1230     BNE NXT
1240 NVL CMP #$03
1250     BEQ GPC
1260     BCS NXT
1270     LDA CIX
1280     CMP BUFIX
1290     BCC GTIX
1300     STA BUFIX
1310 GTIX LDX STIX
1320     LDA BINIX,X
1330     STA CIX
1340     LDA BOUTIX,X
1350     STA COX
1360     JMP CHAR
```

```
1180     BMI NXT
```

Podczas kontroli składni z tablicy składni SXTAB są odczytywane kolejne liczby, które oznaczają elementy składniowe instrukcji. Odczyt ten jest wykonywany przez procedurę GTCCHR. Najpierw zwiększa ona stan licznika PCNTC przez wywołanie procedury INCPRC, a następnie odczytuje wskazany przezeń kod.

```
0100 ;GeT Current CHaRacter
0110 ;
0120 INCPRC = $A28C
0130 PCNTC = $9D
0140 ;
0150     *=  $A293
0160 ;
0170     JSR INCPRC
0180     LDX #$00
0190     LDA (PCNTC,X)
0200     RTS
```

Procedura INCPRC jest bardzo prosta, gdyż jej zadaniem jest jedynie zwiększenie stanu dwubajtowego licznika PCNTC. Opis działania jest tu całkowicie zbędny.

```
0100 ;INCrease PRogram Counter
0110 ;
0120 PCNTC = $9D
0130 ;
0140     *=  $A28C
0150 ;
0160     INC PCNTC
0170     BNE END
0180     INC PCNTC+1
0190 END RTS
```

Kontrola składni instrukcji jest bardzo złożonym i skomplikowanym procesem. Postępowanie interpretera zależy od kodu odczytanego z tabeli składni. Dla ułatwienia analizy tej tablicy podam znaczenie poszczególnych kodów. Kod $00 oznacza przepisanie dwóch kolejnych bajtów na stos, a następnie do licznika PCNTC, a więc skok w inne miejsce tablicy (w wydruku oznaczony jako jsr). Kod $01 oznacza wykonanie skoku do procedury o adresie zawartym w dwóch następnych bajtach (jmp). Możliwe odmiany składni są rozdzielone kodem $02 (or). Kod $03 sygnalizuje koniec fragmentu składniowego (end). Kod $0D powoduje opuszczenie następnego znaku z bufora (skip). Kod $0E wskazuje, że w tym miejscu wymagana jest wartość - stała lub zmienna (value). Kod $0F nakazuje przepisanie następnego kodu jako tokena do wprowadzanego wiersza (token). Kody większe od $80 powodują skoki o wartość wyrażenia kod-$C1 (adresy skoków są oznaczone przez Qnn). Pozostałe kody są kodami tokenów. A oto cała tablica składni:

```
0100 ;SyntaX TABle
0110 ;
0120 MOVLIN = $A2DB
0130 NUMCNS = $A3F5
0140 NUMVAR = $A320
0150 SCDSTM = $A2CF
0160 STRCNS = $A41C
0170 STRVAR = $A324
0180 ;
```

```
0190     *=  $A605
0200 ;
0210 Q01 .BYTE $CD,$C4,$02,$C2    ; Q03 Q02 or Q02
0220     .BYTE $03                ; end
0230 Q02 .BYTE $2B,$BA,$2C,$DB    ; ( Q01 ) Q05
0240     .BYTE $02,$CD,$D8,$03    ; or Q04 Q05 end
0250 Q03 .BYTE $25,$0F,$35,$02    ; + token + or
0260     .BYTE $26,$0F,$36,$02    ; - token - or
0270     .BYTE $28,$03            ; NOT end
0280 Q04 .BYTE $FE,$02,$E8,$02    ; Q08 or L1 or
0290     .BYTE $01                ; jmp
0300     .WORD NUMCNS-1
0310     .BYTE $02,$00            ; or jsr
0320     .WORD L3-1
0330     .BYTE $03                ; end
0340 Q05 .BYTE $C4,$9C,$02,$03    ; Q06 Q01 or end
0350 Q06 .BYTE $23,$02,$25,$02    ; ^ or + or
0360     .BYTE $26,$02,$24,$02    ; - or * or
0370     .BYTE $27,$02,$1D,$02    ; / or <= or
0380     .BYTE $1F,$02,$1E,$02    ; >= or <> or
0390     .BYTE $20,$02,$21,$02    ; < or > or
0400     .BYTE $22,$02,$2A,$02    ; = or AND or
0410     .BYTE $29,$03            ; OR end
0420 L1  .BYTE $01                ; jmp
0430     .WORD NUMVAR-1
0440     .BYTE $C2,$03            ; Q07 end
0450 Q07 .BYTE $0D,$2B,$0F,$38    ; skip ( token ixv
0460     .BYTE $0E,$C4,$2C,$02    ; value L2 ) or
0470     .BYTE $03                ; end
0480 L2  .BYTE $12,$0F,$3C,$0E    ; , token , value
0490     .BYTE $02,$03            ; or end
0500 Q08 .BYTE $44,$D2,$02,$00    ; ATN Q10 or jsr
0510     .WORD L7-1
0520     .BYTE $D3,$02,$C2,$03    ; Q11 or Q09 end
0530 Q09 .BYTE $3F,$2B,$0F,$3A    ; USR ( token (
0540     .BYTE $00                ; jsr
0550     .WORD L9-1
0560     .BYTE $2C,$03            ; ) end
0570 Q10 .BYTE $2B,$0F,$3A,$0E    ; ( token ( value
0580     .BYTE $2C,$03            ; ) end
0590 Q11 .BYTE $2B,$0F,$3A,$C7    ; ( token ( L4
0600     .BYTE $2C,$03            ; ) end
0610 L3  .BYTE $C4,$E3,$C2,$03    ; L4 Q16 L4 end
0620 L4  .BYTE $C8,$02,$CB,$02    ; Q12 or Q13 or
0630     .BYTE $01                ; jmp
0640     .WORD STRCNS-1
0650     .BYTE $03                ; end
0660 Q12 .BYTE $00                ; jsr
0670     .WORD L8-1
0680     .BYTE $A5,$03            ; Q10 end
0690 Q13 .BYTE $01                ; jmp
0700     .WORD STRVAR-1
0710     .BYTE $C2,$03            ; Q14 end
0720 Q14 .BYTE $2B,$0F,$37,$0E    ; ( token ( value
0730     .BYTE $C4,$2C,$02,$03    ; Q15 ) or end
0740 Q15 .BYTE $12,$0F,$3C,$0E    ; , token , value
0750     .BYTE $02,$03            ; or end
0760 Q16 .BYTE $1D,$0F,$2F,$02    ; <= token <= or
0770     .BYTE $1E,$0F,$30,$02    ; <> token <> or
0780     .BYTE $1F,$0F,$31,$02    ; >= token >= or
0790     .BYTE $20,$0F,$32,$02    ; < token < or
```

```
0800     .BYTE $21,$0F,$33,$02    ; > token > or
0810     .BYTE $22,$0F,$34,$03    ; = token = end
0820 SPUT .BYTE $1C,$0E,$12       ; # value ,
0830 SCOLOR .BYTE $0E             ; value
0840 SBYE .BYTE $FA,$03           ; Q18 end
0850 SLET .BYTE $00               ; jsr
0860     .WORD L1-1
0870     .BYTE $22,$0F,$2D,$0E    ; = token = value
0880     .BYTE $F1,$02,$86,$22    ; Q18 or Q13 =
0890     .BYTE $0F,$2E,$00        ; token = jsr
0900     .WORD L4-1
0910     .BYTE $E8,$03            ; Q18 end
0920 SFOR .BYTE $01               ; jmp
0930     .WORD NUMVAR-1
0940     .BYTE $22,$0F,$2D,$0E    ; = token = value
0950     .BYTE $19,$0E,$C3,$DC    ; TO value Q17 Q18
0960     .BYTE $03                ; end
0970 Q17 .BYTE $1A,$0E,$02,$03    ; STEP value or end
0980 SLOCAT .BYTE $0E,$12,$0E     ; value , value
0990     .BYTE $12,$C4,$03        ; , SNEXT end
1000 SGET .BYTE $DD,$12           ; Q19 ,
1010 SNEXT .BYTE $01              ; jmp
1020     .WORD NUMVAR-1
1030     .BYTE $CB,$03            ; Q18 end
1040 SRSTR .BYTE $0E,$C8,$02,$C6  ; value Q18 or Q18
1050     .BYTE $03                ; end
1060 SINPUT .BYTE $F7             ; Q23
1070 SREAD .BYTE $DB,$C2,$03      ; Q21 Q18 end
1080 Q18 .BYTE $14,$02,$16,$03    ; : or EOL end
1090 SPRINT .BYTE $C9,$BB,$02,$EC ; Q19 Q18 or Q23
1100 SLPRNT .BYTE $00             ; jsr
1110     .WORD L5-1
1120     .BYTE $B5,$03            ; Q18 end
1130 Q19 .BYTE $1C,$0E,$03        ; # value end
1140 Q20 .BYTE $01                ; jmp
1150     .WORD NUMVAR-1
1160     .BYTE $02,$01            ; or jmp
1170     .WORD STRVAR-1
1180     .BYTE $03                ; end
1190 Q21 .BYTE $B8,$C2,$03        ; Q20 Q22 end
1200 Q22 .BYTE $12,$BC,$02,$03    ; , Q21 or end
1210 SXIO .BYTE $0E,$12           ; value ,
1220 SOPEN .BYTE $AC,$12,$F9,$12  ; Q19 , Q27 ,
1230     .BYTE $F3,$9A,$03        ; Q26 Q18 end
1240 SCLOSE .BYTE $A5,$97,$03     ; Q19 Q18 end
1250 SENTER .BYTE $ED,$94,$03     ; Q26 Q18 end
1260 SRUN .BYTE $EA,$91,$02,$8F   ; Q26 Q18 or Q18
1270     .BYTE $03                ; end
1280 Q23 .BYTE $9A,$12,$02,$97    ; Q19 , or Q19
1290     .BYTE $15,$02,$03        ; ; or end
1300 SLIST .BYTE $DE,$85,$02      ; Q26 Q18 or
1310     .BYTE $DB,$12,$C4,$02    ; Q26 , Q24 or
1320     .BYTE $C2,$03            ; Q24 end
1330 Q24 .BYTE $00                ; jsr
1340     .WORD L6-1
1350     .BYTE $F4,$03            ; Q31 end
1360 SSTAT .BYTE $C3,$F1,$03      ; Q25 Q31 end
1370 Q25 .BYTE $82,$12,$00        ; Q19 , jsr
1380     .WORD L1-1
1390     .BYTE $03                ; end
1400 SNOTE .BYTE $BA,$12,$00      ; Q25 , jsr
```

```
1410     .WORD L1-1
1420     .BYTE $E4,$03            ; Q31 end
1430 Q26 .BYTE $00                ; jsr
1440     .WORD L4-1
1450     .BYTE $03                ; end
1460 Q27 .BYTE $0E,$12,$0E,$03    ; value , value end
1470 SSOUND .BYTE $0E,$12         ; value ,
1480 SSETC .BYTE $0E,$12          ; value ,
1490 SPOKE .BYTE $B8,$D5,$03      ; Q27 Q31 end
1500 SDIM .BYTE $ED,$D2,$03       ; Q33 Q31 end
1510 SON .BYTE $0E,$C4,$C7        ; value Q28 Q29
1520     .BYTE $CD,$03            ; Q31 end
1530 Q28 .BYTE $17,$02,$18,$03    ; GOTO or GOSUB end
1540 Q29 .BYTE $0E,$C2,$03        ; value Q30 end
1550 Q30 .BYTE $12,$BC,$02,$03    ; , Q29 or end
1560 Q31 .BYTE $14,$02,$16,$03    ; : or EOL end
1570 Q32 .BYTE $01                ; jmp
1580     .WORD NUMVAR-1
1590     .BYTE $0D,$2B,$0F,$39    ; skip ( token dixv
1600     .BYTE $0E,$00            ; value jsr
1610     .WORD L2-1
1620     .BYTE $2C,$02,$01        ; ) or jmp
1630     .WORD STRVAR-1
1640     .BYTE $2B,$0F,$3B,$0E    ; ( token ( value
1650     .BYTE $2C,$03            ; ) end
1660 Q33 .BYTE $AA,$C3,$02,$03    ; Q32 Q34 or end
1670 Q34 .BYTE $12,$BB,$02,$03    ; , Q33 or end
1680 SIF .BYTE $0E,$1B,$C3,$9B    ; value THEN Q35 Q31
1690     .BYTE $03                ; end
1700 Q35 .BYTE $01                ; jmp
1710     .WORD NUMCNS-1
1720     .BYTE $02,$01            ; or jmp
1730     .WORD SCDSTM-1
1740 L5  .BYTE $C9,$02,$D4,$C3    ; Q37 or Q39 Q36
1750     .BYTE $02,$03            ; or end
1760 Q36 .BYTE $C3,$02,$03        ; Q37 or end
1770 Q37 .BYTE $C3,$C8,$03        ; Q38 Q44 end
1780 Q38 .BYTE $0E,$02,$00        ; value or jsr
1790     .WORD L4-1
1800     .BYTE $03                ; end
1810 Q44 .BYTE $C4,$B3,$02,$03    ; Q39 Q36 or end
1820 Q39 .BYTE $C6,$C2,$03        ; Q41 Q40 end
1830 Q40 .BYTE $BD,$02,$03        ; Q39 or end
1840 Q41 .BYTE $12,$02,$15,$03    ; , or ; end
1850 L6  .BYTE $0E                ; value
1860     .BYTE $C3,$02,$03        ; Q42 or end
1870 Q42 .BYTE $12,$0E,$02,$03    ; , value or end
1880 SREM .BYTE $01               ; jmp
1890     .WORD MOVLIN-1
1900 SDATA .BYTE $01              ; jmp
1910     .WORD MOVLIN-1
1920 L7  .BYTE $40,$02,$41,$02    ; ASC or VAL or
1930     .BYTE $43,$02,$42,$03    ; ADR or LEN end
1940 L8  .BYTE $3D,$02,$3E,$03    ; STR$ or CHR$ end
1950 L9  .BYTE $0E,$C2,$03        ; value Q43 end
1960 Q43 .BYTE $12,$0F,$3C,$BA    ; , token , L9
1970     .BYTE $02,$03            ; or end
```

W trakcie kontroli składni wywoływane są dodatkowe procedury pomagające w tokenizacji elementów programu. Najprostszą z nich jest procedura MOVLIN, wywoływana po rozpoznaniu

tokenu instrukcji REM lub DATA. Po ustawieniu wskaźnika stosu na $FF wykonuje ona skok do etykiety MOVTKN, co powoduje przepisanie bez zmian pozostałej części wiersza.

```
0100 ;MOVe LINe
0110 ;
0120 MOVTKN = $A0FB
0130 ;
0140     *=  $A2DB
0150 ;
0160     LDX #$FF
0170     TXS
0180     JMP MOVTKN
```

Równie prosta procedura jest wywoływana po rozpoznaniu instrukcji IF nie zakończonej numerem wiersza. Procedura SCDSTM zapisuje w stokenizowanym wierszu długość instrukcji IF, a następnie przechodzi do rozpoznawania następnych instrukcji znajdujących się w tym samym wierszu.

```
0100 ;Start of ConDitional STateMent
0110 ;
0120 COX =   $94
0130 DCDSTM = $A0B1
0140 LOMEM = $80
0150 OUTIX = $A7
0160 ;
0170     *=  $A2CF
0180 ;
0190     LDX #$FF
0200     TXS
0210     LDA COX
0220     LDY OUTIX
0230     STA (LOMEM),Y
0240     JMP DCDSTM
```

Jeśli spodziewana jest stała liczbowa, to wywoływana jest procedura NUMCNS. Najpierw zamienia ona przy pomocy procedury AFP ciąg znaków ASCII na liczbę zmiennoprzecinkową. W przypadku niepowodzenia ustawiany jest bit Carry i procedura się kończy. Poprawny wynik zamiany powoduje wpisanie tokena $0E, który oznacza stałą liczbową, oraz przepisanie sześciu bajtów liczby. W tym przypadku przed opuszczeniem procedury bit Carry jest kasowany.

```
0100 ;NUMber CoNStant
0110 ;
0120 AFP =   $D800
0130 CIX =   $F2
0140 COX =   $94
0150 FR0 =   $D4
0160 INBSS = $DBA1
0170 LOMEM = $80
0180 SAVTKN = $A2C4
0190 TIX =   $AC
0200 ;
0210     *=  $A3F5
0220 ;
0230     JSR INBSS
0240     LDA CIX
0250     STA TIX
0260     JSR AFP
```

```
0270     BCC SUCC
0280     LDA TIX
0290     STA CIX
0300     RTS
0310 SUCC LDA #$0E
0320     JSR SAVTKN
0330     INY
0340     LDX #$00
0350 LOOP LDA FR0,X
0360     STA (LOMEM),Y
0370     INY
0380     INX
0390     CPX #$06
0400     BCC LOOP
0410     STY COX
0420     CLC
0430     RTS
```

Zbliżone jest postępowanie przy tokenizacji stałej tekstowej. Wykonuje to procedura STRCNS. Najpierw sprawdzany jest pierwszy znak stałej, i jeśli nie jest to cudzysłów ("), procedura jest przerywana z ustawionym bitem Carry. W przeciwnym wypadku zapisywany jest token stałej tekstowej $0F, po czym kolejne znaki z bufora wejściowego są przepisywane do bufora tokenizacji, aż do napotkania drugiego cudzysłowu lub końca wiersza. Po zapisaniu długości stałej procedura kończy się ze skasowanym bitem Carry.

```
0100 ;STRing CoNStant
0110 ;
0120 CIX =   $F2
0130 COX =   $94
0140 INBSS = $DBA1
0150 INBUFP = $F3
0160 LOMEM = $80
0170 SAVTKN = $A2C4
0180 TOX =   $AB
0190 ;
0200     *=  $A41C
0210 ;
0220     JSR INBSS
0230     LDY CIX
0240     LDA (INBUFP),Y
0250     CMP #'"
0260     BNE $A3F3   ;SEC,RTS
0270     LDA #$0F
0280     JSR SAVTKN
0290     LDA COX
0300     STA TOX
0310     JSR SAVTKN
0320 LOOP INC CIX
0330     LDY CIX
0340     LDA (INBUFP),Y
0350     CMP #$9B
0360     BEQ END
0370     CMP #'"
0380     BEQ NXT
0390     JSR SAVTKN
0400     JMP LOOP
0410 NXT INC CIX
0420 END CLC
0430     LDA COX
```

```
0440     SBC TOX
0450     LDY TOX
0460     STA (LOMEM),Y
0470     CLC
0480     RTS
```

Jeżeli interpreter oczekuje nazwy zmiennej, to wywoływana jest procedura odpowiednia do typu zmiennej. Dla zmiennej liczbowej jest to procedura NUMVAR, zaś dla tekstowej STRVAR. Przebieg obu procedur jest wspólny poza początkową fazą, w której zapisywany jest w rejestrze VART (VARiable Type) typ zmiennej ($00 - liczbowa, $80 - tekstowa).

```
0100 ADBT =  $DBAF
0110 AUXBR = $AF
0120 BHLD1 = $B0
0130 BHLD2 = $B1
0140 CIX =   $F2
0150 CMPLTR = $A3E8
0160 CSTAD = $97
0170 EXPSX = $A2E1
0180 INBSS = $DBA1
0190 INBUFP = $F3
0200 INSEL = $A87A
0210 LBUFF = $0580
0220 NXTNAM = $A482
0230 RECNAM = $A454
0240 SAVTKN = $A2C4
0250 STMNUM = $B2
0260 STMTAB = $88
0270 TIX =   $AC
0280 TMVRER = $B92C
0290 VARN =  $D3
0300 VART =  $D2
0310 VNTD =  $84
0320 VNTP =  $82
0330 ;
0340     *=  $A320
0350 ;
0360 ;NUMber VARiable
0370 ;
0380 NUMVAR LDA #$00
0390     BEQ EXE
0400 ;
0410 ;STRing VARiable
0420 ;
0430 STRVAR LDA #$80
0440 EXE STA VART
0450     JSR INBSS
0460     LDA CIX
0470     STA TIX
0480     JSR CMPLTR
0490     BCS BAD
0500     JSR EXPSX
0510     LDA BHLD1
0520     BEQ CCR
0530     LDY STMNUM
0540     LDA (INBUFP),Y
0550     CMP #'0
0560     BCC BAD
0570 CCR INC CIX
0580     JSR CMPLTR
```

```
0590     BCC CCR
0600     JSR ADBT
0610     BCC CCR
0620     LDA (INBUFP),Y
0630     CMP #'$
0640     BEQ STR
0650     BIT VART
0660     BPL DIM
0670 BAD SEC
0680     RTS
0690 STR BIT VART
0700     BPL BAD
0710     INY
0720     BNE FND
0730 DIM LDA (INBUFP),Y
0740     CMP #'(
0750     BNE FND
0760     INY
0770     LDA #$40
0780     ORA VART
0790     STA VART
0800 FND LDA TIX
0810     STA CIX
0820     STY TIX
0830     LDA VNTP+1
0840     LDY VNTP
0850     LDX #$00
0860     JSR RECNAM
0870 NXT BCS NEW
0880     CPX TIX
0890     BEQ SAV
0900     JSR NXTNAM
0910     JMP NXT
0920 NEW SEC
0930     LDA TIX
0940     SBC CIX
0950     STA CIX
0960     TAY
0970     LDX #VNTD
0980     JSR INSEL
0990     LDA AUXBR
1000     STA VARN
1010     LDY CIX
1020     DEY
1030     LDX TIX
1040     DEX
1050 PUT LDA LBUFF,X
1060     STA (CSTAD),Y
1070     DEX
1080     DEY
1090     BPL PUT
1100     LDY CIX
1110     DEY
1120     LDA (CSTAD),Y
1130     ORA #$80
1140     STA (CSTAD),Y
1150     LDY #$08
1160     LDX #STMTAB
1170     JSR INSEL
1180     INC BHLD2
1190     LDY #$02
```

```
1200     LDA #$00
1210 RST STA VART,Y
1220     INY
1230     CPY #$08
1240     BCC RST
1250     DEY
1260 GET LDA VART,Y
1270     STA (CSTAD),Y
1280     DEY
1290     BPL GET
1300 SAV BIT VART
1310     BVC BPS
1320     DEC TIX
1330 BPS LDA TIX
1340     STA CIX
1350     LDA AUXBR
1360     BMI ERR
1370     ORA #$80
1380     CLC
1390     JMP SAVTKN
1400 ERR JMP TMVRER
```

Rozpoznawanie zmiennej rozpoczyna się od sprawdzenia poprawności jej zapisu. Najpierw pierwszy znak nazwy jest kontrolowany przy pomocy procedury CMPLTR. Jeśli nie jest on literą, to sygnalizowany jest błąd. Następnie wywoływana jest procedura EXPSX, która przeszukuje tablicę nazw funkcji OPFN. Gdy badana nazwa nie występuje w tej tablicy, to sprawdzane są jej kolejne znaki, aż do napotkania znaku innego niż litera lub cyfra. Jeżeli znakiem tym jest dolar ($), to rejestr VART musi zawierać wartość $80. Wystąpienie błędu podczas wyżej opisanych czynności jest sygnalizowane ustawieniem bitu Carry i przerwaniem procedury.

Teraz sprawdzany jest pierwszy znak następujący po nazwie. Jeśli jest to nawias, ustawiany jest bit 6 w rejestrze VART (wymiar lub indeks zmiennej tablicowej). Ponieważ zmienna może się już znajdować w tablicy nazw zmiennych, to teraz procedura RECNAM przeszukuje tą tablicę. W rezultacie tego przeszukania w rejestrze AUXBR znajduje się numer zmiennej (gdy zmienna nie została znaleziona, jest to liczba o jeden większa od numeru ostatniej zmiennej w tablicy). Nowa zmienna jest następnie wpisywana do tablicy nazw i tablicy wartości. W tym celu jest wykorzystywana opisana już procedura INSEL. Jeżeli zmienna jest już w tablicy, to ten fragment jest pomijany.

Na samym końcu procedury kontrolowany jest jeszcze numer zmiennej. Gdy jest on większy od $7F (więcej niż 128 zmiennych), to wykonywany jest skok do TMVRER w celu zasygnalizowania błędu (Too Many VaRiables ERror). Poprawny numer zmiennej jest zwiększany o $80 i przez wywołanie SAVTKN zapisywany w buforze tokenizacji.

Do sprawdzenia znaku pobranego z bufora wejściowego służy procedura CMPLTR. Odczytuje ona znak i sprawdza, czy jest on literą z zakresu A-Z. Przy wywołaniu tej procedury od etykiety CHKLTR sprawdzeniu jest poddawany znak zawarty w akumulatorze.

```
0100 ;CoMPare for LeTteR
0110 ;
```

```
0120 CIX =   $F2
0130 INBUFP = $F3
0140 ;
0150     *=  $A3E8
0160 ;
0170 CMPLTR LDY CIX
0180     LDA (INBUFP),Y
0190 ;
0200 ;CHecK for LeTteR
0210 ;
0220 CHKLTR CMP #'A
0230     BCC ERR
0240     CMP #'[
0250     RTS
0260 ERR SEC
0270     RTS
```

Kolejną procedurą wykorzystywaną podczas kontroli składni jest VALUE. Wywoływana jest ona po napotkaniu kodu większego od $04 i mniejszego od $80. Jeżeli jest to kod $0F, to następny znak z tablicy składni jest przepisywany do bufora tokenizacji i procedura się kończy. Gdy kodem tym jest $0E, to znajdujący się na stosie adres jest zastępowany adresem początkowym tablicy SXTAB. W pozostałych przypadkach wykonywany jest skok do procedury EXPSX, przy czym kod $0D powoduje pominięcie jednego znaku z bufora wejściowego.

```
0100 ;VALUE
0110 ;
0120 COX =   $94
0130 EXPSX = $A2E1
0140 INCPRC = $A28C
0150 LOMEM = $80
0160 PCNTC = $9D
0170 SAVCPM = $A21B
0180 SXTAB = $A605
0190 ;
0200     *=  $A29B
0210 ;
0220     CMP #$0F
0230     BEQ SAV
0240     BCS EXPSX
0250     CMP #$0D
0260     BNE CSX
0270     JSR INCPRC
0280     JMP EXPSX+3
0290 CSX PLA
0300     PLA
0310     LDA # <SXTAB-1
0320     PHA
0330     LDA # >SXTAB-1
0340     PHA
0350     JMP SAVCPM
0360 SAV JSR INCPRC
0370     LDY #$00
0380     LDA (PCNTC),Y
0390     LDY COX
0400     DEY
0410     STA (LOMEM),Y
0420     CLC
0430     RTS
```

Nazwy wszystkich legalnych operatorów i funkcji Atari Basic są zapisane w tablicy OPFN. Jest ona zbliżona do tablicy nazw instrukcji, lecz nie zawiera wektorów do tablicy składni. Niektóre znajdujące się w niej operatory dotyczące zmiennych różnych typów są powtórzone. Umożliwia to wybranie odpowiedniego wariantu procedury przy realizacji programu.

```
0100 ;OPerator & Function Names
0110 ;
0120     *=  $A7DE
0130 ;
0140     .CBYTE $02
0150     .CBYTE $00
0160     .CBYTE ","
0170     .CBYTE "$"
0180     .CBYTE ":"
0190     .CBYTE ";"
0200     .BYTE $9B      ;EOL
0210     .CBYTE "GOTO"
0220     .CBYTE "GOSUB"
0230     .CBYTE "TO"
0240     .CBYTE "STEP"
0250     .CBYTE "THEN"
0260     .CBYTE "#"
0270     .CBYTE "<="    ;
0280     .CBYTE "<>"    ;
0290     .CBYTE ">="    ;operatory
0300     .CBYTE "<"     ;liczbowe
0310     .CBYTE ">"     ;
0320     .CBYTE "="     ;
0330     .CBYTE "^"
0340     .CBYTE "*"
0350     .CBYTE "+"
0360     .CBYTE "-"
0370     .CBYTE "/"
0380     .CBYTE "NOT"
0390     .CBYTE "OR"
0400     .CBYTE "AND"
0410     .CBYTE "("     ;nawiasy
0420     .CBYTE ")"     ;zwykłe
0430     .CBYTE "="     ;LET liczbowe
0440     .CBYTE "="     ;LET tekstowe
0450     .CBYTE "<="    ;
0460     .CBYTE "<>"    ;
0470     .CBYTE ">="    ;operatory
0480     .CBYTE "<"     ;tekstowe
0490     .CBYTE ">"     ;
0500     .CBYTE "="     ;
0510     .CBYTE "+"     ;znak
0520     .CBYTE "-"     ;znak
0530     .CBYTE "("     ;wyr. tekstowe
0540     .CBYTE $00     ;zm. indeks.
0550     .CBYTE $00     ;wym. zm. ind.
0560     .CBYTE "("     ;wyr. funkc.
0570     .CBYTE "("     ;wym. zm. tekst.
0580     .CBYTE ","     ;w indeksie zm.
0590     .CBYTE "STR$"
0600     .CBYTE "CHR$"
0610     .CBYTE "USR"
0620     .CBYTE "ASC"
0630     .CBYTE "VAL"
0640     .CBYTE "LEN"
```

```
0650     .CBYTE "ADR"
0660     .CBYTE "ATN"
0670     .CBYTE "COS"
0680     .CBYTE "PEEK"
0690     .CBYTE "SIN"
0700     .CBYTE "RND"
0710     .CBYTE "FRE"
0720     .CBYTE "EXP"
0730     .CBYTE "LOG"
0740     .CBYTE "CLOG"
0750     .CBYTE "SQR"
0760     .CBYTE "SGN"
0770     .CBYTE "ABS"
0780     .CBYTE "INT"
0790     .CBYTE "PADDLE"
0800     .CBYTE "STICK"
0810     .CBYTE "PTRIG"
0820     .CBYTE "STRIG"
0830     .BYTE $00
```

Powyższa tablica jest przeszukiwana przez procedurę EXPSX, która służy do kontroli składni wyrażeń arytmetycznych, logicznych i tekstowych. Znaleziony numer operatora lub funkcji zapisywany jest do bufora tokenizacji. Poprawny wynik tej operacji jest sygnalizowany skasowaniem bitu Carry.

```
0100 ;EXPression SyntaX
0110 ;
0120 AUXBR = $AF
0130 BHLD1 = $B0
0140 CIX =   $F2
0150 INBSS = $DBA1
0160 OPFN =  $A7DE
0170 PCNTC = $9D
0180 RECNAM = $A454
0190 SAVTKN = $A2C4
0200 STMNUM = $B2
0210 TMPIX = $B3
0220 ;
0230     *=  $A2E1
0240 ;
0250     JSR INBSS
0260     LDA CIX
0270     CMP TMPIX
0280     BEQ CHK
0290     STA TMPIX
0300     LDA # >OPFN
0310     LDY # <OPFN
0320     LDX #$00
0330     JSR RECNAM
0340     BCS NFD
0350     STX STMNUM
0360     LDA AUXBR
0370     ADC #$10
0380     STA BHLD1
0390 CHK LDY #$00
0400     LDA (PCNTC),Y
0410     CMP BHLD1
0420     BEQ SAV
0430     CMP #$44
0440     BNE ERR
```

```
0450     LDA BHLD1
0460     CMP #$44
0470     BCC ERR
0480 SAV JSR SAVTKN
0490     LDX STMNUM
0500     STX CIX
0510     CLC
0520     RTS
0530 NFD LDA #$00
0540     STA BHLD1
0550 ERR SEC
0560     RTS
```

2.2.3. Wykonanie instrukcji

Jeżeli wprowadzony wiersz jest w trybie bezpośrednim, to procedura SYNTAX kończy się skokiem do PRCSTM. Jej zadaniem jest w tym wypadku wykonanie instrukcji zawartych we wprowadzonym wierszu. Procedura ta jest także wywoływana przez XRUN w celu wykonania programu znajdującego się w pamięci komputera.

```
0100 ;PRoCeed STateMent
0110 ;
0120 BUFIX = $9F
0130 EXESTM = $A97E
0140 GHISTM = $A9E1
0150 GIRQST = $A9F2
0160 GSTMLN = $B819
0170 INIX =  $A8
0180 NXTSTM = $A9D0
0190 OUTIX = $A7
0200 STMCUR = $8A
0210 WAITIN = $A05D
0220 XEND =  $B78C
0230 XSTOP = $B792
0240 ;
0250     *=  $A95E
0260 ;
0270 PRCSTM JSR GSTMLN
0280 STAT JSR GIRQST
0290     BEQ STOP
0300     LDY OUTIX
0310     CPY BUFIX
0320     BCS NEXT
0330     LDA (STMCUR),Y
0340     STA OUTIX
0350     TYA
0360     INY
0370     LDA (STMCUR),Y
0380     INY
0390     STY INIX
0400     JSR EXESTM
0410     NOP
0420     JMP STAT
0430 ;
0440     *=  $A989
0450 ;
0460 NEXT LDY #$01
0470     LDA (STMCUR),Y
0480     BMI WAIT
```

```
0490     LDA BUFIX
0500     JSR NXTSTM
0510     JSR GHISTM
0520     BPL PRCSTM
0530     JMP XEND
0540 STOP JMP XSTOP
0550 WAIT JMP WAITIN
```

Pierwszą czynnością jest odczytanie długości wykonywanego wiersza. Uzyskana wartość służy do zliczania tokenów i umożliwia zakończenie realizacji wiersza po osiągnięciu jego końca. Odczyt długości wiersza następuje przez wywołanie procedury GSTMLN. Przy pomocy wektora STMCUR (STateMent CURrent address) trzeci token, którego wartość określa długość wiersza, jest pobierany z tabeli instrukcji i zapisywany do licznika BUFIX (BUFfer IndeX). Indeks następnego tokena jest zapisywany w liczniku OUTIX.

```
0100 BUFIX = $9F
0110 FNDCST = $A9A2
0120 OUTIX = $A7
0130 STMCUR = $8A
0140 ;
0150     *=  $B816
0160 ;
0170 ;Find STatement & Get LeNgth
0180 ;
0190 FSTGLN JSR FNDCST
0200 ;
0210 ;Get STateMent LeNgth
0220 ;
0230 GSTMLN LDY #$02
0240     LDA (STMCUR),Y
0250     STA BUFIX
0260     INY
0270     STY OUTIX
0280     RTS
```

Następnie sprawdzany jest przez procedurę GIRQST rejestr statusu przerwań. Jeśli sygnalizuje on naciśnięcie klawisza BREAK, to wykonywanie wiersza jest przerywane skokiem do procedury XSTOP. Naciśnięcie BREAK jest więc równoważne umieszczeniu w wierszu instrukcji STOP. Teraz porównywane są zawartości liczników BUFIX i OUTIX. Gdy są one różne, to wykonywanie wiersza jest kontynuowane. Następny odczytany token oznacza długość instrukcji, a właściwie indeks jej ostatniego tokena. Jest on zapisywany do licznika OUTIX. Z kolei do akumulatora pobierany jest token instrukcji, a indeks następnego tokena do odczytu jest umieszczany w rejestrze INIX (INput IndeX). Odpowiednia procedura realizacyjna instrukcji jest wywoływana przez procedurę EXESTM i pętla się powtarza od sprawdzenia klawisza BREAK.

Zrównanie zawartości rejestrów BUFIX i OUTIX oznacza zakończenie wykonywania całego wiersza. Jeżeli był to wiersz w trybie bezpośrednim, to wykonywany jest skok do etykiety WAITIN i komputer oczekuje na następne polecenia. W trybie programowym poprzez wywołanie procedury NXTSTM przepisywany jest do rejestru STMCUR adres kolejnego wiersza. Starszy bajt tego wiersza jest odczytywany przez procedurę GHISTM. Wartość $80 sygnalizuje, że następny wiersz

jest w trybie bezpośrednim, a więc cały program został już wykonany. W takim przypadku wykonywany jest skok do procedury XEND, jeśli zaś następny wiersz programu znajduje się w pamięci, to ponownie wykonywana jest procedura PRCSTM.

```
0100 ;Get HIgh byte of STateMent
0110 ;
0120 STMCUR = $8A
0130 ;
0140     *=  $A9E1
0150 ;
0160 GHISTM LDY #$01
0170     LDA (STMCUR),Y
0180 ;
0190 ;eXecute REM statement
0200 ;
0210 XREM RTS
```

Właściwe wywołanie procedury realizującej żądaną instrukcję jest przeprowadzane przez procedurę EXESTM według wartości tokenu przekazanej w akumulatorze. Zastosowano tu popularny w systemie Atari sposób: adres procedury jest umieszczany na stosie, po czym wykonywany jest rozkaz RTS. Powrót z procedury wykonawczej następuje do miejsca wywołania EXESTM.

```
0100 ;EXEcute STateMent
0110 ;
0120 STVTAB = $A9FA
0130 ;
0140     *=  $A97E
0150 ;
0160     ASL A
0170     TAX
0180     LDA STVTAB,X
0190     PHA
0200     LDA STVTAB+1,X
0210     PHA
0220     RTS
```

Adres procedury wykonawczej jest odczytywany z tablicy wektorów STVTAB przy wykorzystaniu podwojonej wartości tokena instrukcji (adresy są dwubajtowe). Tablica wektorów musi więc zawierać adresy procedur w kolejności zgodnej z tablicą nazw instrukcji STNAME.

```
0100 ;STatement Vectors TABle
0110 ;
0120 SNTXER = $B912
0130 XBYE =  $A9E6
0140 XCLR =  $B766
0150 XCLOAD = $BB64
0160 XCLOSE = $BC22
0170 XCOLOR = $BA1F
0180 XCONT = $B7B5
0190 XCSAVE = $BBD1
0200 XDEG =  $B28D
0210 XDIM =  $B206
0220 XDRAW = $BA27
0230 XDOS =  $A9EC
0240 XEND =  $B78C
0250 XENTER = $BAC5
```

```
0260 XFOR =  $B67D
0270 XGET =  $BC85
0280 XGOSUB = $B6D2
0290 XGOTO = $B6D5
0300 XGRAPH = $BA46
0310 XIF =   $B778
0320 XINPUT = $B33E
0330 XLET =  $AADA
0340 XLIST = $B4B5
0350 XLOAD = $BAFB
0360 XLOCAT = $BC9E
0370 XLPRNT = $B496
0380 XNEW =  $A00C
0390 XNEXT = $B700
0400 XNOTE = $BC3D
0410 XON =   $B7E4
0420 XOPEN = $BBF2
0430 XPLOT = $BA6C
0440 XPOINT = $BC54
0450 XPOKE = $B278
0460 XPOP =  $B83E
0470 XPOS =  $BA0C
0480 XPRINT = $B3DA
0490 XPUT =  $BC78
0500 XRAD =  $B291
0510 XREAD = $B2AE
0520 XREM =  $A9E5
0530 XRSTR = $B296
0540 XRTRN = $BDA8
0550 XRUN =  $B74C
0560 XSAVE = $BB6D
0570 XSETC = $B9AD
0580 XSOUND = $B9D3
0590 XSTAT = $BC2F
0600 XSTOP = $B792
0610 XTRAP = $B7D8
0620 XXIO =  $BBEC
0630 ;
0640     *=  $A9FA
0650 ;
0660     .DBYTE XREM-1    ;REM
0670     .DBYTE XREM-1    ;DATA
0680     .DBYTE XINPUT-1  ;INPUT
0690     .DBYTE XCOLOR-1  ;COLOR
0700     .DBYTE XLIST-1   ;LIST
0710     .DBYTE XENTER-1  ;ENTER
0720     .DBYTE XLET-1    ;LET
0730     .DBYTE XIF-1     ;IF
0740     .DBYTE XFOR-1    ;FOR
0750     .DBYTE XNEXT-1   ;NEXT
0760     .DBYTE XGOTO-1   ;GOTO
0770     .DBYTE XGOTO-1   ;GO TO
0780     .DBYTE XGOSUB-1  ;GOSUB
0790     .DBYTE XTRAP-1   ;TRAP
0800     .DBYTE XBYE-1    ;BYE
0810     .DBYTE XCONT-1   ;CONT
0820     .DBYTE XDIM-1    ;COM
0830     .DBYTE XCLOSE-1  ;CLOSE
0840     .DBYTE XCLR-1    ;CLR
0850     .DBYTE XDEG-1    ;DEG
0860     .DBYTE XDIM-1    ;DIM
```

```
0870     .DBYTE XEND-1    ;END
0880     .DBYTE XNEW-1    ;NEW
0890     .DBYTE XOPEN-1   ;OPEN
0900     .DBYTE XLOAD-1   ;LOAD
0910     .DBYTE XSAVE-1   ;SAVE
0920     .DBYTE XSTAT-1   ;STATUS
0930     .DBYTE XNOTE-1   ;NOTE
0940     .DBYTE XPOINT-1  ;POINT
0950     .DBYTE XXIO-1    ;XIO
0960     .DBYTE XON-1     ;ON
0970     .DBYTE XPOKE-1   ;POKE
0980     .DBYTE XPRINT-1  ;PRINT
0990     .DBYTE XRAD-1    ;RAD
1000     .DBYTE XREAD-1   ;READ
1010     .DBYTE XRSTR-1   ;RESTORE
1020     .DBYTE XRTRN-1   ;RETURN
1030     .DBYTE XRUN-1    ;RUN
1040     .DBYTE XSTOP-1   ;STOP
1050     .DBYTE XPOP-1    ;POP
1060     .DBYTE XPRINT-1  ;?
1070     .DBYTE XGET-1    ;GET
1080     .DBYTE XPUT-1    ;PUT
1090     .DBYTE XGRAPH-1  ;GRAPHICS
1100     .DBYTE XPLOT-1   ;PLOT
1110     .DBYTE XPOS-1    ;POSITION
1120     .DBYTE XDOS-1    ;DOS
1130     .DBYTE XDRAW-1   ;DRAWTO
1140     .DBYTE XSETC-1   ;SETCOLOR
1150     .DBYTE XLOCAT-1  ;LOCATE
1160     .DBYTE XSOUND-1  ;SOUND
1170     .DBYTE XLPRNT-1  ;LPRINT
1180     .DBYTE XCSAVE-1  ;CSAVE
1190     .DBYTE XCLOAD-1  ;CLOSE
1200     .DBYTE XLET-1    ;opuszczone LET
1210     .DBYTE SNTXER-1  ;ERROR
```

Rozdział 3

OBSŁUGA BŁĘDÓW

Zanim przystąpimy do analizy poszczególnych instrukcji Atari Basic należy zapoznać się ze sposobem obsługi błędów. Idea głównej procedury realizującej sygnalizację i obsługę błędów interpretera jest zbliżona do zastosowanej w DOS 2.0 (zob. "Mapa pamięci Atari XL/XE. Dyskowe systemy operacyjne").

```
0100 ;Errors routine
0110 ;
0120 CLNN =  $A0
0130 DSPFLG = $02FE
0140 DSPLIN = $B993
0150 ERRCOD = $B9
0160 ERRSAV = $C3
0170 FDEXST = $B6E0
0180 FR0 =   $D4
0190 GHISTM = $A9E1
0200 POKADR = $95
0210 PRTRET = $BD79
0220 PUTSTM = $B66F
0230 PUTTXT = $B567
0240 RSTCHN = $BD5B
0250 SAVCLN = $B7A6
0260 STMCUR = $8A
0270 SYNTAX = $A060
0280 TRAPLN = $BC
0290 ;
0300     *=  $B90A
0310 ;
0320 ;LOAD file ERror
0330 LOADER INC ERRCOD
0340 ;DeViCe Number ERror
0350 DVCNER INC ERRCOD
0360 ;PRogram too LonG ERror
0370 PRLGER INC ERRCOD
0380 ;invalid CHARacter ERror
0390 CHARER INC ERRCOD
0400 ;SyNTaX ERror
0410 SNTXER INC ERRCOD
0420 ;RETurn ERror
0430 RETER INC ERRCOD
0440 ;GOSub Line ERror
0450 GOSLER INC ERRCOD
0460 ;Line Too LonG ERror
0470 LTLGER INC ERRCOD
0480 ;No matching FOR ERror
0490 NFORER INC ERRCOD
0500 ;Line Not FounD ERror
0510 LNFDER INC ERRCOD
0520 ;OVerflow/UNderflow ERror
0530 OVUNER INC ERRCOD
0540 ;STack OVerflow ERror
0550 STOVER INC ERRCOD
0560 ;DIMension ERror
0570 DIMER INC ERRCOD
```

```
0580 ;Bad INPut ERror
0590 BINPER INC ERRCOD
0600 ;bad LINe Number ERror
0610 LINNER INC ERRCOD
0620 ;Out of DATa ERror
0630 ODATER INC ERRCOD
0640 ;String LENgth ERror
0650 SLENER INC ERRCOD
0660 ;Too Many VaRiables ERror
0670 TMVRER INC ERRCOD
0680 ;Bad VALue ERror
0690 BVALER INC ERRCOD
0700 ;INSufficient Memory ERror
0710 INSMER INC ERRCOD
0720 ;SUCCess
0730 SUCC INC ERRCOD
0740 ;
0750 ;GET ERRor code
0760 ;
0770 GETERR LDA #$00
0780     STA DSPFLG
0790     JSR SAVCLN
0800     LDA TRAPLN+1
0810     BMI PRNT
0820     STA CLNN+1
0830     LDA TRAPLN
0840     STA CLNN
0850     LDA #$80
0860     STA TRAPLN+1
0870     LDA ERRCOD
0880     STA ERRSAV
0890     LDA #$00
0900     STA ERRCOD
0910     JMP FDEXST
0920 PRNT JSR PRTRET
0930     LDA #$37
0940     JSR PUTSTM
0950     LDA ERRCOD
0960     STA FR0
0970     LDA #$00
0980     STA FR0+1
0990     JSR DSPLIN
1000 ;
1010 ;Display STop MeSsaGe
1020 ;
1030 DSTMSG JSR GHISTM
1040     BMI DRM
1050     LDA # <LNMSG
1060     STA POKADR
1070     LDA # >LNMSG
1080     STA POKADR+1
1090     JSR PUTTXT
1100     LDY #$01
1110     LDA (STMCUR),Y
1120     STA FR0+1
1130     DEY
1140     LDA (STMCUR),Y
1150     STA FR0
1160     JSR DSPLIN
1170 DRM JSR PRTRET
1180     LDA #$00
```

```
1190     STA ERRCOD
1200     JSR RSTCHN
1210     JMP SYNTAX
1220 ;
1230     *=  $B9A4
1240 ;
1250 LNMSG .CBYTE " AT LINE "
```

Miejsce rozpoczęcia procedury jest wybierane w zależności od rodzaju błędu. Kod błędu otrzymuje odpowiednią wartość przez kolejne zwiększanie zawartości rejestru ERRCOD (ERRor CODe). Właściwa procedura rozpoczyna się od wyzerowania rejestru DSPFLG (DiSPlay FLaG), który kontroluje sposób wyświetlania znaków na ekranie. Następnie wywoływana jest procedura SAVCLN. Odczytuje ona starszy bajt numeru wiersza, a gdy jest on mniejszy od $80 (tryb programowy), przepisuje numer aktualnie wykonywanego wiersza programu do rejestru STOPLN (STOP LiNe number). W obu przypadkach SAVCLN kończy się skokiem do procedury RSTCHN.

```
0100 ;SAVe Current LiNe
0110 ;
0120 GHISTM = $A9E1
0130 RSTCHN = $BD5B
0140 STMCUR = $8A
0150 STOPLN = $BA
0160 ;
0170     *=  $B7A6
0180 ;
0190     JSR GHISTM
0200     BMI END
0210     STA STOPLN+1
0220     DEY
0230     LDA (STMCUR),Y
0240     STA STOPLN
0250 END JMP RSTCHN
```

Tu numer aktualnie używanego kanału wejścia/wyjścia jest ustawiany na zero, czyli na kanał IOCB obsługujący edytor ekranowy. Uzyskiwane jest to przez wyzerowanie rejestrów IOCHN (Input/Output CHaNnel) oraz ACHNN (Auxiliary CHaNnel Number).

```
0100 ;ReSeT CHaNnel registers
0110 ;
0120 ACHNN = $B4
0130 IOCHN = $B5
0140 ;
0150     *=  $BD5B
0160 ;
0170     LDA #$00
0180     STA ACHNN
0190     STA IOCHN
0200     RTS
```

Dalszy przebieg obsługi błędu zależy od zawartości rejestru TRAPLN (TRAP LiNe number). Jeśli znajduje się tam poprawny numer wiersza (mniejszy od $8000), to jest on przepisywany do rejestru CLNN (Current LiNe Number), a do starszego bajtu TRAPLN wpisywana jest wartość $80. Następnie kod błędu z ERRCOD jest przenoszony do ERRSAV (ERRor code SAVe register), a ERRCOD jest zerowany. Po wykonaniu tych operacji następuje skok do procedury FDEXST

(wewnątrz XGOSUB) i realizacja programu jest kontynuowana.

Jeśli rejestr TRAPLN nie zawiera poprawnego numeru wiersza, to najpierw wyświetlany jest znak końca wiersza (przy pomocy procedury PRTRET będącej częścią PRTPRM). Teraz w akumulatorze umieszczany jest kod tokena błędu ($37) i wywoływana jest procedura PUTSTM. Odszukuje ona, korzystając z pomocniczych procedur FNDPEL i ADDPAD, adres początkowy słowa kluczowego instrukcji w tabeli nazw STNAME. W tym przypadku jest to adres napisu "ERROR-". Adres ten jest zapisywany w rejestrze POKADR i następuje bezpośrednie przejście do procedury PTMSG.

```
0100 ;PUT STateMent name
0110 ;
0120 AUXBR = $AF
0130 FNDPEL = $B53E
0140 PTMSG2 = $B586
0150 STNAME = $A49F
0160 ;
0170     *=  $B66F
0180 ;
0190     STA AUXBR
0200     LDX #$02
0210     LDA # >STNAME
0220     LDY # <STNAME
0230     JSR FNDPEL
0240     JMP PTMSG2
```

Procedura FNDPEL odszukuje nazwę w odpowiedniej tablicy, której adres przekazany został w akumulatorze i rejestrze Y. Korzysta przy tym z zasady, że każda nazwa (instrukcji, operatora, funkcji i zmiennej) kończy się znakiem, którego najstarszy bit jest ustawiony. Zwiększanie zawartości rejestru POKADR jest wykonywane przez ADDPAD.

```
0100 ;FiND Program ELement
0110 ;
0120 ADDPAD = $B557
0130 AUXBR = $AF
0140 POKADR = $95
0150 SETPAD = $B562
0160 STPTR = $AA
0170 ;
0180     *=  $B53E
0190 ;
0200     STX STPTR
0210     JSR SETPAD
0220 ELP LDY STPTR
0230     DEC AUXBR
0240     BMI ADDPAD
0250 ILP LDA (POKADR),Y
0260     BMI NXT
0270     INY
0280     BNE ILP
0290 NXT INY
0300     JSR ADDPAD
0310     JMP ELP

0100 POKADR = $95
0110 ;
```

```
0120     *=  $B557
0130 ;
0140 ;ADDition Poke ADdress
0150 ;
0160 ADDPAD CLC
0170     TYA
0180     ADC POKADR
0190     STA POKADR
0200     TAY
0210     LDA POKADR+1
0220     ADC #$00
0230 ;
0240 ;SET Poke ADdress
0250 ;
0260 SETPAD STA POKADR+1
0270     STY POKADR
0280     RTS
```

Wyświetlenie odnalezionej nazwy jest wykonywane przez procedurę PTMSG. Przy jej wywołaniu od etykiety PTMSG1 przed nazwą wyświetlana jest jeszcze spacja ($20). Niezależnie od miejsca wywołania po wyświetlonej nazwie również umieszczana jest spacja. Sama nazwa jest wyświetlana poprzez procedurę PUTTXT.

```
0100 ;PuT MeSsaGe
0110 ;
0120 PRPCHN = $BA99
0130 PUTTXT = $B567
0140 ;
0150     *=  $B581
0160 ;
0170 PTMSG1 LDA #$20
0180     JSR PRPCHN
0190 PTMSG2 JSR PUTTXT
0200     LDA #$20
0210     JMP PRPCHN
```

Korzystając z zawartego w rejestrze POKADR adresu procedura PUTTXT wyświetla napis znak po znaku, aż do napotkania RETURN lub znaku z ustawionym najstarszym bitem. Wykorzystywana przy tym procedura PRPCHN jest opisana w rozdziale 6.4.

```
0100 ;PUT TeXT
0110 ;
0120 AUXBR = $AF
0130 POKADR = $95
0140 PRPCHN = $BA99
0150 ;
0160     *=  $B567
0170 ;
0180     LDY #$FF
0190     STY AUXBR
0200 LOOP INC AUXBR
0210     LDY AUXBR
0220     LDA (POKADR),Y
0230     PHA
0240     CMP #$9B
0250     BEQ PUT
0260     AND #$7F
0270     BEQ NXT
```

```
0280 PUT JSR PRPCHN
0290 NXT PLA
0300     BPL LOOP
0310     RTS
```

Po wyświetleniu napisu "ERROR-" do rejestru FR0 przepisywany jest kod błędu z ERRCOD i wywoływana jest procedura DSPLIN. Dokonuje ona zamiany dwubajtowej liczby całkowitej na ciąg znaków ASCII (przy pomocy IFP i FASC), a adres tego ciągu umieszcza w rejestrze POKADR. Następnie wykonywany jest skok do procedury PUTTXT, dzięki czemu kod błędu jest wyświetlany na ekranie.

```
0100 ;DiSPlay LINe number
0110 ;
0120 FASC =  $D8E6
0130 IFP =   $D9AA
0140 INBUFP = $F3
0150 POKADR = $95
0160 PUTTXT = $B567
0170 ;
0180     *=  $B993
0190 ;
0200     JSR IFP
0210     JSR FASC
0220     LDA INBUFP
0230     STA POKADR
0240     LDA INBUFP+1
0250     STA POKADR+1
0260     JMP PUTTXT
```

Jeżeli błąd wystąpił podczas wykonywania wiersza w trybie programowym (numer wiersza mniejszy od $8000), to przez wywołanie PUTTXT wyświetlany jest napis "AT LINE". Następnie po przepisaniu do FR0 numeru aktualnego wiersza procedura DSPLIN wyświetla ten numer na ekranie.

Niezależnie od trybu pracy ostatnim wyświetlanym znakiem jest koniec wiersza (RETURN). Obsługa błędu kończy się wyzerowaniem rejestru ERRCOD i ponownym wywołaniem RSTCHN. Po wykonaniu wszystkich opisanych operacji interpreter przechodzi do procedury SYNTAX i oczekuje na polecenia od użytkownika.

Rozdział 4

INSTRUKCJA PRZYPISANIA

W każdym programie najczęściej wykonywana jest instrukcja przypisania (LET). Od niej zaczniemy więc opis działania poszczególnych procedur wykonawczych Atari Basic. Ponieważ jest ona często wykorzystywana przez procedury pozostałych instrukcji, a sama wykorzystuje procedury operatorów i funkcji, to jej opis został umieszczony w oddzielnym rozdziale.

```
0100 ;eXecute LET statement
0110 ;
0120 EXEOP = $AB18
0130 LOMEM = $80
0140 OPCT =  $AC35
0150 PUTVAR = $ABB2
0160 RECVAL = $AB36
0170 RSTPTR = $AB26
0180 STIX =  $A9
0190 TIX =   $AC
0200 TOX =   $AB
0210 ;
0220     *=  $AADA
0230 ;
0240 XLET JSR RSTPTR
0250 LOOP JSR RECVAL
0260     BCS OPER
0270     JSR PUTVAR
0280     BMI LOOP
0290 OPER STA TOX
0300     TAX
0310     LDA OPCT-$10,X
0320     LSR A
0330     LSR A
0340     LSR A
0350     LSR A
0360     STA TIX
0370 GETT LDY STIX
0380     LDA (LOMEM),Y
0390     TAX
0400     LDA OPCT-$10,X
0410     AND #$0F
0420     CMP TIX
0430     BCC SAVT
0440     TAX
0450     BEQ $AB35   ;RTS
0460 ;
0470 ;PRoCeed OPerator
0480 ;
0490 PRCOP LDA (LOMEM),Y
0500     INC STIX
0510     JSR EXEOP
0520     JMP GETT
0530 SAVT LDA TOX
0540     DEY
0550     STA (LOMEM),Y
0560     STY STIX
0570     JMP LOOP
```

Pierwszym krokiem jest odpowiednie ustawienie liczników i innych rejestrów. W tym celu wywoływana jest procedura RSTPTR. Zeruje ona rejestry STPTR (STack PoinTeR), BHLD1 i BHLD2 (Basic HoLD register) oraz wpisuje wartość $FF do licznika STIX (STack IndeX). Ponadto na koniec bufora tokenizacji wpisywana jest wartość $11.

```
0100 ;ReSeT PoinTeRs
0110 ;
0120 BHLD1 = $B0
0130 BHLD2 = $B1
0140 LOMEM = $80
0150 STIX =  $A9
0160 STPTR = $AA
0170 ;
0180     *=  $AB26
0190 ;
0200     LDY #$FF
0210     LDA #$11
0220     STA (LOMEM),Y
0230     STY STIX
0240     INY
0250     STY BHLD1
0260     STY STPTR
0270     STY BHLD2
0280 ;
0290 ;eXecute PLUS sign
0300 ;
0310 XPLS RTS
```

Teraz rozpoczyna się główna pętla procedury przypisania XLET. Wywoływana jest tu procedura RECVAL, której zadaniem jest rozpoznanie wartości do przypisania. Najpierw odczytywany jest token do rozpoznania. Wartość tego tokena większa od $7F oznacza zmienną i powoduje skok do etykiety VARBL umieszczonej na końcu procedury. Wartość $0F oznacza stałą tekstową, a $0E - stałą liczbową. W tych przypadkach wykonywane są odpowiednie fragmenty procedury. Każdy inny token powoduje opuszczenie RECVAL z ustawionym bitem Carry, co sygnalizuje, że badany token nie oznacza wartości (stałej lub zmiennej).

```
0100 ;RECognize VALue
0110 ;
0120 CALPC = $AC1E
0130 FR0 =   $D4
0140 INIX =  $A8
0150 PCNTC = $9D
0160 STMCUR = $8A
0170 VARN =  $D3
0180 VART =  $D2
0190 ;
0200     *=  $AB36
0210 ;
0220 RECVAL LDY INIX
0230     INC INIX
0240     LDA (STMCUR),Y
0250     BMI VARBL
0260     CMP #$0F
0270     BCC NUMBER
0280     BEQ TEXT
0290     RTS
0300 ;
```

```
0310 ;NUMBER
0320 ;
0330 NUMBER LDX #$00
0340 NXT1 INY
0350     LDA (STMCUR),Y
0360     STA FR0,X
0370     INX
0380     CPX #$06
0390     BCC NXT1
0400     INY
0410     LDA #$00
0420     TAX
0430     BEQ STORE
0440 TEXT INY
0450     LDA (STMCUR),Y
0460     LDX #STMCUR
0470 ;
0480 ;STore TeXT
0490 ;
0500 STTXT STA FR0+2
0510     STA FR0+4
0520     INY
0530     TYA
0540     CLC
0550     ADC $00,X
0560     STA FR0
0570     LDA #$00
0580     STA FR0+3
0590     STA FR0+5
0600     ADC $01,X
0610     STA FR0+1
0620     TYA
0630     ADC FR0+2
0640     TAY
0650     LDX #$00
0660     LDA #$83
0670 STORE STA VART
0680     STX VARN
0690     STY INIX
0700     CLC
0710     RTS
0720 ;
0730 ;VARiaBLe
0740 ;
0750 VARBL JSR CALPC
0760 NXT2 LDA (PCNTC),Y
0770     STA VART,Y
0780     INY
0790     CPY #$08
0800     BCC NXT2
0810     CLC
0820     RTS
```

Jeśli rozpoznana została stała liczbowa, to jej sześć bajtów jest przepisywane z tablicy instrukcji do rejestru operacji zmiennoprzecinkowych FR0. Po rozpoznaniu stałej tekstowej jej długość jest zapisywana do trzeciego i piątego bajtu rejestru FR0, a w dwóch pierwszych bajtach zapisywany jest adres pierwszego znaku stałej. Taki format zapisu odpowiada dokładnie zapisowi zmiennej tekstowej w tablicy wartości zmiennych. Na zakończenie do rejestru VARN (VARiable Number)

wpisywane jest zero, co oznacza stałą, a do rejestru VART (VARiable Type) - $00 dla stałej liczbowej lub $83 dla stałej tekstowej.

W przypadku zmiennej najpierw wywoływana jest procedura CALPC, która oblicza adres zmiennej w tablicy wartości. W tym celu numer zmiennej (bez najstarszego bitu) jest mnożony przez osiem (długość wpisu w tablicy wartości zmiennych) i dodawany do adresu tej tablicy. Uzyskany wynik umieszczany jest w liczniku PCNTC.

```
0100 ;CALculate Program Counter
0110 ;
0120 PCNTC = $9D
0130 VVTP =  $86
0140 ;
0150     *=  $AC1E
0160 ;
0170     LDY #$00
0180     STY PCNTC+1
0190     ASL A
0200     ASL A
0210     ROL PCNTC+1
0220     ASL A
0230     ROL PCNTC+1
0240     CLC
0250     ADC VVTP
0260     STA PCNTC
0270     LDA VVTP+1
0280     ADC PCNTC+1
0290     STA PCNTC+1
0300     RTS
```

Następnie według obliczonego adresu wartość zmiennej jest odczytywana z tablicy i przepisywana do rejestru operacji zmiennoprzecinkowych. Po wykonaniu tej czynności w rejestrze VART znajduje się kod typu zmiennej, a w rejestrze VARN - jej numer. Rejestr FR0 zawiera wartość zmiennej w przypadku prostej zmiennej liczbowej lub parametry zmiennej w przypadku zmiennej tablicowej.

W przypadku napotkania wartości w instrukcji przypisania wywoływana jest procedura PUTVAR. Przepisuje ona znalezioną wartość z rejestrów VART, VARN i FR0 (razem osiem bajtów) do bufora tokenizacji. Odpowiednio zwiększany jest przy tym licznik STPTR. Jeśli nastąpi przepełnienie bufora, co jest sprawdzane na początku procedury, to następuje skok do STOVER i sygnalizowany jest błąd (STack OVerflow ERror). Wynika z tego, że jednocześnie można zapisać w buforze 256/8=32 wartości. Nie jest to jednak prawda, gdyż część bufora zajmują tokeny operatorów i funkcji, a ponadto podczas obliczania wyrażenia niektóre wartości i tokeny mogą być usuwane z bufora jeszcze przed zapisaniem następnych.

```
0100 ;PUT VARiable
0110 ;
0120 LOMEM = $80
0130 STIX =  $A9
0140 STOVER = $B920
0150 STPTR = $AA
0160 VART =  $D2
```

```
0170 ;
0180     *=  $ABB2
0190 ;
0200     INC STPTR
0210     LDA STPTR
0220     ASL A
0230     ASL A
0240     ASL A
0250     CMP STIX
0260     BCS ERR
0270     TAY
0280     DEY
0290     LDX #$07
0300 LOOP LDA VART,X
0310     STA (LOMEM),Y
0320     DEY
0330     DEX
0340     BPL LOOP
0350     RTS
0360 ERR JMP STOVER
```

Po przepisaniu wartości przez procedurę PUTVAR pętla XLET jest powtarzana od wywołania RECVAL. Ponieważ składnia instrukcji jest sprawdzana podczas wprowadzania wiersza, to niemożliwe jest rozpoznanie bezpośrednio po sobie dwóch wartości.

Gdy badany token nie oznacza wartości, to jest zapisywany do rejestru TOX (Temporary Output indeX). Według niego jest następnie odczytywany z tablicy OPCT kod, którego starsza połowa jest umieszczana w rejestrze TIX (Temporary Input indeX). Teraz według licznika STIX odczytywany jest token z bufora (w pierwszym przejściu jest to zawsze $11). Według tego tokena ponownie pobierany jest kod z OPCT, a jego młodsza połowa jest porównywana z zawartością rejestru TIX. Jeżeli zawartość TIX jest większa, to token operacji jest zapisywany na końcu bufora (licznik STIX jest przy tym zmniejszany). W przeciwnym przypadku procedura XLET kończy się, jeśli w akumulatorze jest wartość zero. Trzeba tu zwrócić uwagę na sposób wykorzystania bufora tokenizacji. Od jego początku kolejno są umieszczane wartości, a od końca tokeny operacji. Licznik STPTR wskazuje przy tym liczbę zapisanych wartości, a licznik STIX - token pierwszej operacji do wykonania. Tablica OPCT jest skonstruowana w taki sposób, że w odpowiednim momencie następuje przejście do części oznaczonej etykietą PRCOP. Wartości w tej tablicy wyznaczają jednocześnie priorytet operatorów i funkcji.

```
0100 ;OPerator's Coefficients Table
0110 ;
0120     *=  $AC35
0130 ;
0140     .BYTE $00   ;niewykorzystany
0150     .BYTE $00   ;niewykorzystany
0160     .BYTE $00   ;,
0170     .BYTE $00   ;$
0180     .BYTE $00   ;: -koniec instrukcji
0190     .BYTE $00   ;;
0200     .BYTE $00   ;EOL -koniec wiersza
0210     .BYTE $00   ;GOTO
0220     .BYTE $00   ;GOSUB
0230     .BYTE $00   ;TO
0240     .BYTE $00   ;STEP
```

```
0250     .BYTE $00   ;THEN
0260     .BYTE $00   ;#
0270     .BYTE $88   ;<=  |
0280     .BYTE $88   ;<>  |
0290     .BYTE $88   ;>=  |operatory
0300     .BYTE $88   ;<   |liczbowe
0310     .BYTE $88   ;>   |
0320     .BYTE $88   ;=   |
0330     .BYTE $CC   ;^
0340     .BYTE $AA   ;*
0350     .BYTE $99   ;+
0360     .BYTE $99   ;-
0370     .BYTE $AA   ;/
0380     .BYTE $DD   ;NOT
0390     .BYTE $55   ;OR
0400     .BYTE $66   ;AND
0410     .BYTE $F2   ;(
0420     .BYTE $4E   ;)
0430     .BYTE $F1   ;= -LET liczbowe
0440     .BYTE $F1   ;= -LET tekstowe
0450     .BYTE $EE   ;<=  |
0460     .BYTE $EE   ;<>  |
0470     .BYTE $EE   ;>=  |operatory
0480     .BYTE $EE   ;<   |tekstowe
0490     .BYTE $EE   ;>   |
0500     .BYTE $EE   ;=   |
0510     .BYTE $DD   ;+ -znak
0520     .BYTE $DD   ;- -znak
0530     .BYTE $F2   ;( -wyr. tekstowe
0540     .BYTE $F2   ;( -zm. indeksowana
0550     .BYTE $F2   ;( -wym. zm. ind.
0560     .BYTE $F2   ;( -wyr. funkcyjne
0570     .BYTE $F2   ;( -wym. zm. tekst.
0580     .BYTE $43   ;, -w indeksie
0590     .BYTE $F2   ;STR$
0600     .BYTE $F2   ;CHR$
0610     .BYTE $F2   ;USR
0620     .BYTE $F2   ;ASC
0630     .BYTE $F2   ;VAL
0640     .BYTE $F2   ;LEN
0650     .BYTE $F2   ;ADR
0660     .BYTE $F2   ;ATN
0670     .BYTE $F2   ;COS
0680     .BYTE $F2   ;PEEK
0690     .BYTE $F2   ;SIN
0700     .BYTE $F2   ;RND
0710     .BYTE $F2   ;FRE
0720     .BYTE $F2   ;EXP
0730     .BYTE $F2   ;LOG
0740     .BYTE $F2   ;CLOG
0750     .BYTE $F2   ;SQR
0760     .BYTE $F2   ;SGN
0770     .BYTE $F2   ;ABS
0780     .BYTE $F2   ;INT
0790     .BYTE $F2   ;PADDLE
0800     .BYTE $F2   ;STICK
0810     .BYTE $F2   ;PTRIG
0820     .BYTE $F2   ;STRIG
```

Po rozpoznaniu operatora o priorytecie wyższym niż poprzednie, jego token jest odczytywany z bufora tokenizacji, a licznik STIX jest zwiększany. Teraz w celu wykonania operacji lub funkcji wywoływana jest procedura EXEOP. Działanie jej jest identyczne, jak wcześniej opisanej procedury EXESTM. Adres procedury wykonawczej jest jednak w tym przypadku pobierany z tabeli wektorów OPVTAB. Zmniejszenie wartości tokena o $1D jest konieczne dla pominięcia operatorów, które są integralnymi częściami instrukcji.

```
0100 ;EXEcute OPerator
0110 ;
0120 OPVTAV = $AA6A
0130 ;
0140     *=  $AB18
0150 ;
0160     SEC
0170     SBC #$1D
0180     ASL A
0190     TAX
0200     LDA OPVTAB,X
0210     PHA
0220     LDA OPVTAB+1,X
0230     PHA
0240     RTS
```

Także tablica OPVTAB jest zbudowana identycznie jak STVTAB. Z podanego wyżej powodu zawiera ona jedynie wektory operacji i funkcji, których tokeny mają wartości od $1D do $54.

```
0100 ;OPerator Vectors TABle
0110 ;
0120 XABS =  $B099
0130 XADD =  $BDFA
0140 XADR =  $B007
0150 XAND =  $ACCF
0160 XASC =  $AFFD
0170 XATN =  $B118
0180 XCHR =  $B052
0190 XCLOG = $B13D
0200 XCOM =  $AD64
0210 XCOS =  $B10F
0220 XDIV =  $AC8C
0230 XEPAR = $AD66
0240 XEQU =  $ACC9
0250 XEXP =  $B14A
0260 XFRE =  $AFD6
0270 XGEQ =  $ACC2
0280 XGRT =  $ACB9
0290 XINT =  $B0C8
0300 XLEN =  $AFB5
0310 XLEQ =  $ACA3
0320 XLES =  $ACB2
0330 XLOG =  $B121
0340 XMIN =  $AC95
0350 XMUL =  $AC83
0360 XNEQ =  $ACAC
0370 XNLET = $AD4A
0380 XNOT =  $ACE4
0390 XOR =   $ACD9
0400 XPADDL = $B00D
0410 XPDIM = $AD6D
```

```
0420 XPEEK = $AFCC
0430 XPIXV = $AD71
0440 XPLS =  $AB35
0450 XPSEX = $AE11
0460 XPTRIG = $B015
0470 XRND =  $B076
0480 XRPW =  $B15E
0490 XSGN =  $AD04
0500 XSIN =  $B106
0510 XSLET = $AE8E
0520 XSQR =  $B153
0530 XSTICK = $B011
0540 XSTR =  $B034
0550 XSTRIG = $B019
0560 XSUB =  $AC7A
0570 XUSR =  $B0A5
0580 XVAL =  $AFEB
0590 ;
0600     *=  $AA6A
0610 ;
0620     .DBYTE XLEQ-1   ;<= |
0630     .DBYTE XNEQ-1   ;<> |
0640     .DBYTE XGEQ-1   ;>= |operatory
0650     .DBYTE XLES-1   ;<  |liczbowe
0660     .DBYTE XGRT-1   ;>  |
0670     .DBYTE XEQU-1   ;=  |
0680     .DBYTE XRPW-1   ;^
0690     .DBYTE XMUL-1   ;*
0700     .DBYTE XADD-1   ;+
0710     .DBYTE XSUB-1   ;-
0720     .DBYTE XDIV-1   ;/
0730     .DBYTE XNOT-1   ;NOT
0740     .DBYTE XOR-1    ;OR
0750     .DBYTE XAND-1   ;AND
0760     .DBYTE XPLS-1   ;(
0770     .DBYTE XEPAR-1  ;)
0780     .DBYTE XNLET-1  ;= -liczbowy
0790     .DBYTE XSLET-1  ;= -tekstowy
0800     .DBYTE XLEQ-1   ;<= |
0810     .DBYTE XNEQ-1   ;<> |
0820     .DBYTE XGEQ-1   ;>= |operatory
0830     .DBYTE XLES-1   ;<  |tekstowe
0840     .DBYTE XGRT-1   ;>  |
0850     .DBYTE XEQU-1   ;=  |
0860     .DBYTE XPLS-1   ;+ znak
0870     .DBYTE XMIN-1   ;- znak
0880     .DBYTE XPSEX-1  ;( -wyr. tekst.
0890     .DBYTE XPIXV-1  ;( -zm. indeks.
0900     .DBYTE XPDIM-1  ;( -wym. zm. ind.
0910     .DBYTE XEPAR-1  ;( -wyr. funkc.
0920     .DBYTE XPDIM-1  ;( -wym. zm. tekst.
0930     .DBYTE XCOM-1   ;, -indeks
0940     .DBYTE XSTR-1   ;STR$
0950     .DBYTE XCHR-1   ;CHR$
0960     .DBYTE XUSR-1   ;USR
0970     .DBYTE XASC-1   ;ASC
0980     .DBYTE XVAL-1   ;VAL
0990     .DBYTE XLEN-1   ;LEN
1000     .DBYTE XADR-1   ;ADR
1010     .DBYTE XATN-1   ;ATN
1020     .DBYTE XCOS-1   ;COS
```

```
1030     .DBYTE XPEEK-1  ;PEEK
1040     .DBYTE XSIN-1   ;SIN
1050     .DBYTE XRND-1   ;RND
1060     .DBYTE XFRE-1   ;FRE
1070     .DBYTE XEXP-1   ;EXP
1080     .DBYTE XLOG-1   ;LOG
1090     .DBYTE XCLOG-1  ;CLOG
1100     .DBYTE XSQR-1   ;SQR
1110     .DBYTE XSGN-1   ;SGN
1120     .DBYTE XABS-1   ;ABS
1130     .DBYTE XINT-1   ;INT
1140     .DBYTE XPADDL-1 ;PADDLE
1150     .DBYTE XSTICK-1 ;STICK
1160     .DBYTE XPTRIG-1 ;PTRIG
1170     .DBYTE XSTRIG-1 ;STRIG
```

Rozdział 5

PROCEDURY OPERATORÓW I FUNKCJI

Opisana w poprzednim rozdziale instrukcja przypisania wykorzystuje wiele różnorodnych operatorów i funkcji. Sposób wywołania wszystkich procedur wykonawczych operatorów i funkcji jest jednakowy (zob. opis procedury EXEOP). Elementy składowe instrukcji przypisania można podzielić na sześć podstawowych grup: operatory arytmetyczne, operatory logiczne, operatory relacji, operatory przypisania, nawiasy oraz funkcje. Są one kolejno opisane w tym rozdziale.

5.1. Operatory arytmetyczne

Do operatorów arytmetycznych zaliczamy operatory dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/) i potęgowania (^) oraz znaki plus (+) i minus (-). Najprostszym operatorem jest znak plus. Nie zmienia on nic w wartości liczby i cała procedura wykonawcza ogranicza się do rozkazu RTS. Znajduje się on w procedurze RSTPTR, która została opisana w poprzednim rozdziale (str. 64).

Wykonanie operatora znaku minus (-), który zmienia znak liczby, jest realizowane przez procedurę XMIN. Liczba do operacji jest pobierana z bufora tokenizacji przez procedurę GETVAR. Następnie zmieniana jest wartość najstarszego bitu w pierwszym bajcie liczby FP, co powoduje zmianę znaku tej liczby. XMIN kończy się skokiem do procedury PUTVAR, która zapisuje wynik ponownie do bufora tokenizacji.

```
0100 ;eXecute MINus sign
0110 ;
0120 FR0 =   $D4
0130 GETVAR = $ABE9
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $AC95
0170 ;
0180     JSR GETVAR
0190     LDA FR0
0200     BEQ EXIT
0210     EOR #$80
0220     STA FR0
0230 EXIT JMP PUTVAR
```

Działanie procedury GETVAR jest odwrotne do PUTVAR. Wartość odczytana z licznika STPTR i zmniejszona o jeden wskazuje - po pomnożeniu przez osiem - ostatni bajt ostatniej liczby w buforze. Według tej wartości osiem bajtów liczby jest przepisywane z bufora do rejestrów VART, VARN i FR0.

```
0100 ;GET VARiable
0110 ;
0120 LOMEM = $80
0130 STPTR = $AA
0140 VART =  $D2
```

```
0150 ;
0160     *=  $ABE9
0170 ;
0180     LDA STPTR
0190     DEC STPTR
0200     ASL A
0210     ASL A
0220     ASL A
0230     TAY
0240     DEY
0250     LDX #$07
0260 LOOP LDA (LOMEM),Y
0270     STA VART,X
0280     DEY
0290     DEX
0300     BPL LOOP
0310     RTS
```

5.1.1. Dodawanie

Operacja dodawania (+) jest realizowana przez procedurę XADD. Składa się ona jedynie z trzech wywołań innych procedur. GETVRS pobiera z bufora dwie liczby, BADD wykonuje ich dodawanie, a PUTVAR umieszcza rezultat ponownie w buforze.

```
0100 ;eXecute ADDition
0110 ;
0120 BADD =  $AD26
0130 GETVRS = $ABFD
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $BDFA
0170 ;
0180     JSR GETVRS
0190     JSR BADD
0200     JMP PUTVAR
```

Procedura GETVRS służy do odczytania dwóch liczb dla operacji arytmetycznej i jest złożeniem dwóch procedur GETVAR. Pierwsze wywołanie GETVAR pobiera jedną liczbę z bufora tokenizacji i umieszcza ją w rejestrze FR0. Liczba ta jest przepisywana przez procedurę FMOV01 (wchodzi ona w skład pakietu zmiennoprzecinkowego) do rejestru FR1. Kończący procedurę GETVRS skok do GETVAR powoduje pobranie drugiej liczby do rejestru FR0.

```
0100 ;GET VaRiableS
0110 ;
0120 FMOV01 = $DDB6
0130 GETVAR = $ABE9
0140 ;
0150     *=  $ABFD
0160 ;
0170     JSR GETVAR
0180     JSR FMOV01
0190     JMP GETVAR
```

Zadaniem procedury BADD jest jedynie wywołanie procedury zmiennoprzecinkowej FADD. Użycie BADD zamiast bezpośredniego wywołania FADD ma na celu umożliwienie poprawnej sygnalizacji błędu powstałego podczas dodawania. Podobne jest znaczenie procedur BSUB, BMUL

i BDIV. Zakończenie operacji zmiennoprzecinkowej z ustawionym bitem Carry oznacza wystąpienie błędu. W takich przypadkach wykonywany jest skok do procedury OVUNER (OVerflow/UNderflow ERror).

```
0100 FADD =  $DA66
0110 FDIV =  $DB28
0120 FMUL =  $DADB
0130 FSUB =  $DA60
0140 OVUNER = $B91E
0150 ;
0160     *=  $AD26
0170 ;
0180 ;Basic ADDition
0190 ;
0200 BADD JSR FADD
0210     BCS ERR
0220     RTS
0230 ;
0240 ;Basic SUBtraction
0250 ;
0260 BSUB JSR FSUB
0270     BCS ERR
0280     RTS
0290 ;
0300 ;Basic MULtiplication
0310 ;
0320 BMUL JSR FMUL
0330     BCS ERR
0340     RTS
0350 ;
0360 ;Basic DIVision
0370 ;
0380 BDIV JSR FDIV
0390     BCS ERR
0400     RTS
0410 ERR JSR OVUNER
```

5.1.2. Odejmowanie

Operacja odejmowania (-) jest realizowana identycznie, jak dodawania. Po odczytaniu dwóch liczb przez GETVRS odejmowanie jest wykonywane przez BSUB (zob. wyżej). Uzyskany wynik jest zapisywany w buforze tokenizacji przez procedurę PUTVAR.

```
0100 ;eXecute SUBtraction
0110 ;
0120 BSUB =  $AD2C
0130 GETVRS = $ABFD
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $AC7A
0170 ;
0180     JSR GETVRS
0190     JSR BSUB
0200     JMP PUTVAR
```

5.1.3. Mnożenie

Także przebieg operacji mnożenia (*) jest analogiczny do przebiegu poprzednio opisanych operacji. W celu realizacji obliczenia jest tu wywoływana procedura BMUL.

```
0100 ;eXecute MULtiplication
0110 ;
0120 BMUL =  $AD32
0130 GETVRS = $ABFD
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $AC83
0170 ;
0180     JSR GETVRS
0190     JSR BMUL
0200     JMP PUTVAR
```

5.1.4. Dzielenie

Ostatnią z prostych operacji arytmetycznych jest dzielenie (/) wykonywane przez procedurę XDIV. Jedyną różnicą w stosunku do poprzednio opisanych procedur wykonawczych jest zastosowanie do realizacji obliczenia procedury BDIV.

```
0100 ;eXecute DIVision
0110 ;
0120 BDIV =  $AD38
0130 GETVRS = $ABFD
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $AC8C
0170 ;
0180     JSR GETVRS
0190     JSR BDIV
0200     JMP PUTVAR
```

5.1.5. Potęgowanie

Znacznie bardziej skomplikowanym działaniem jest podnoszenie do potęgi (oznaczone operatorem ^). Jest ono wykonywane przez procedurę XRPW. Na jej początku sprawdzane są wartości liczb. Gdy wykładnik jest równy zero, to przez skok do procedury ONEP (wewnątrz XLEQ - zob. rozdział 5.3.1) wynik jest ustalany na jeden. Jeśli liczba potęgowana jest równa zero, to dla ujemnej wartości wykładnika sygnalizowana jest niepoprawna wartość (skok do procedury VALER w XSQR - zob. rozdział 5.2.4), zaś dla pozostałych wykładników wynikiem jest zero (następuje bezpośredni skok do procedury PUTVAR).

```
0100 BMUL =  $AD32
0110 CONVFN = $BA76
0120 EXP10 = $DDCC
0130 FR0 =   $D4
0140 FR1 =   $E0
0150 GETVRS = $ABFD
0160 LOG10 = $DED1
0170 ONEP =  $ACF0
0180 OVUNER = $B91E
0190 PUTVAR = $ABB2
```

```
0200 VALER = $B15B
0210 ZTEMP2 = $F7
0220 ;
0230     *=  $B15E
0240 ;
0250 ;eXecute Raise to PoWer function
0260 ;
0270 XRPW JSR GETVRS
0280     LDA FR1
0290     BEQ ONE
0300     ROL A
0310     LDY FR0
0320     BNE EXE
0330     BCS VALER
0340 ;
0350 ;PUT RESult
0360 ;
0370 PUTRES JMP PUTVAR
0380 ONE JMP ONEP
0390 EXE LDX #FR0
0400     JSR CONVFN
0410     ROR A
0420     PHA
0430     LDX #FR1
0440     JSR CONVFN
0450     TYA
0460     BPL POS
0470     AND #$7F
0480     STA FR0
0490     BCS GOOD
0500     PLA
0510     BCC VALER
0520 GOOD LDA FR1
0530     BPL BPS
0540     CLC
0550 BPS PHP
0560     LDX ZTEMP2
0570     CPX #$05
0580     BCS PLS
0590     LDA FR1+1,X
0600     ROR A
0610     BCC PLS
0620     LDA #$80
0630     BNE MIN
0640 POS LDA FR1
0650     BPL SST
0660     CLC
0670 SST PHP
0680 PLS LDA #$00
0690 MIN PHA
0700     LDX #$05
0710 LP1 LDA FR1,X
0720     PHA
0730     DEX
0740     BPL LP1
0750     JSR LOG10
0760     LDX #$00
0770     LDY #$05
0780 LP2 PLA
0790     STA FR1,X
0800     INX
```

```
0810     DEY
0820     BPL LP2
0830     JSR BMUL
0840     JSR EXP10
0850     BCS ERR
0860     PLA
0870     ORA FR0
0880     STA FR0
0890     PLP
0900     PLA
0910     BPL PUTRES
0920     BCC PUTRES
0930     LDX #FR0
0940     JSR CONVFN
0950     BCS PUTRES
0960     LDA FR0
0970     SEC
0980     AND #$7F
0990     SBC #$3F
1000     CMP #$06
1010     BCS PUT
1020     TAX
1030     TAY
1040     SED
1050     SEC
1060 LP3 LDA FR0,X
1070     ADC #$00
1080     STA FR0,X
1090     DEX
1100     BNE LP3
1110     CLD
1120     BCC NXT
1130     INC FR0
1140     INC FR0+1
1150 NXT INY
1160     CPY #$06
1170     BCS PUT
1180     STX FR0,Y
1190     BCC NXT
1200 PUT JMP PUTVAR
1210 ERR JSR OVUNER
```

Dla wszystkich pozostałych wartości liczby potęgowanej i wykładnika przeprowadzane jest obliczenie. Przed jego opisaniem trzeba jednak wyjaśnić działanie pomocniczej procedury CONVFN. Wymaga ona jako danej wejściowej adresu rejestru FRn w rejestrze X. Jeżeli wskazany rejestr zawiera liczbę ujemną, to procedura CONVFN jest przerywana ze skasowanym bitem Carry. Dla liczb dodatnich ich wykładnik jest umieszczany w rejestrze ZTEMP2 (Zeropage TEMPorary register), a do akumulatora są odczytywane dwie pierwsze cyfry różne od zera stojące poniżej pozycji setek. Na przykład dla liczby 1234 będzię to 34. Jeśli nie ma takich cyfr, to bit Carry jest ustawiany.

```
0100 ;CONVert Floating Number
0110 ;
0120 ZTEMP1 = $F5
0130 ZTEMP2 = $F7
0140 ;
0150     *=  $BA76
0160 ;
```

```
0170     SEC
0180     LDA $00,X
0190     AND #$7F
0200     SBC #$40
0210     BCC END
0220     STA ZTEMP1
0230     STA ZTEMP2
0240     TXA
0250     ADC ZTEMP1
0260     INX
0270     INX
0280     INX
0290     INX
0300     INX
0310     INX
0320     STX ZTEMP1
0330     TAX
0340 LOOP INX
0350     CPX ZTEMP1
0360     BCS END
0370     LDA $00,X
0380     BEQ LOOP
0390 END RTS
```

Procedura CONVFN jest najpierw wywoływana dla liczby potęgowanej, a zawartość akumulatora podzielona przez dwa jest zapisywana na stosie. Przy drugim wywołaniu CONVFN sprawdzany jest wykładnik potęgi. Teraz jest wyznaczany znak wyniku, przy czym sposób obliczenia zależy od znaku potęgowanej liczby. Następnie (w liniach 700-850 na wydruku) przeprowadzane jest właściwe obliczenie potęgi według wzoru:

$$A\text{^}B=10\text{^}(\log_{10}A*B)$$

W tym miejscu procedura XRPW kończy się dla następujących przypadków: ujemny wykładnik, ujemna liczba potęgowana lub brak cyfry różnej od zera poniżej pozycji setek dla dodatniej liczby potęgowanej. Obliczenia są kontynuowane tylko dla cyfr znaczących mniejszych od setek, jeśli liczba potęgowana jest mniejsza od 10,000,000,000 (100^5).

5.2. Funkcje

Ważnym elementem interpretera są procedury obliczania różnorodnych funkcji. Ich wykorzystanie przy pisaniu własnych programów w języku maszynowym pozwala zaoszczędzić dużo pracy. Programując zaś w Basicu trzeba znać ograniczenia tych procedur, aby uniknąć błędów powodowanych przez niepoprawne wartości lub zbyt przybliżone wyniki. Atari Basic posiada 24 funkcje, których opis znajduje się poniżej, poza funkcją SGN opisaną razem z operatorami relacji.

5.2.1. Funkcja ABS

Funkcja ABS zwraca moduł czyli wartość bezwzględną podanej liczby. Dla uzyskania takiego efektu wystarczy skasowanie najstarszego bitu w pierwszym bajcie liczby. Czynność ta jest wykonywana przez procedurę XABS. Liczba poddawana tej operacji jest pobierana z bufora przez procedurę GETVAR, a ponownie zapisywana przez PUTVAR.

```
0100 ;eXecute ABS function
0110 ;
0120 FR0 =   $D4
0130 GETVAR = $ABE9
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $B099
0170 ;
0180     JSR GETVAR
0190     LDA FR0
0200     AND #$7F
0210     STA FR0
0220     JMP PUTVAR
```

5.2.2. Funkcja RND

Funkcja RND zwraca wartość losową z przedziału <0,1) czyli liczbę o wartości większej lub równej zero i mniejszej od jeden. Realizowana jest ona przez procedurę XRND. Najpierw stała RNDC o wartości 65536 jest przepisywana do rejestru FR1. Znajdujące się tu wywołanie procedury GETVAR jest konieczne dla usunięcia z bufora argumentu funkcji, który nie ma żadnego znaczenia. Następnie dwa przypadkowe bajty z rejestru sprzętowego RANDOM są umieszczane w rejestrze FR0 i zamieniane na liczbę zmiennoprzecinkową z zakresu od 0 do 65535. Liczba ta jest dzielona przy pomocy procedury BDIV przez stałą 65536. Na końcu uzyskany wynik jest przepisywany do bufora przez procedurę PUTVAR.

```
0100 ;eXecute RND function
0110 ;
0120 BDIV =  $AD38
0130 FLD1R = $DD98
0140 FR0 =   $D4
0150 GETVAR = $ABE9
0160 IFP =   $D9AA
0170 PUTVAR = $ABB2
0180 RANDOM = $D20A
0190 ;
0200     *=  $B076
0210 ;
0220     LDX # <RNDC
0230     LDY # >RNDC
0240     JSR FLD1R
0250     JSR GETVAR
0260     LDY RANDOM
0270     STY FR0
0280     LDY RANDOM
0290     STY FR0+1
0300     JSR IFP
0310     JSR BDIV
0320     JMP PUTVAR
0330 ;
0340 ;RaNDom Constant
0350 ;
0360 RNDC .BYTE $42,$06,$55
0370     .BYTE $36,$00,$00
```

Trzeba zwrócić uwagę na to, że argument funkcji RND nie ma żadnego znaczenia pomimo, iż jest umieszczany w rejestrze FR0. Procedura IFP zamienia na liczbę zmiennoprzecinkową tylko dwa pierwsze bajty tego rejestru, a więc tylko te, które zostały odczytane z RANDOM.

5.2.3. Funkcja INT

Funkcja INT zwraca część całkowitą podanego argumentu, przy czym liczba jest zawsze zaokrąglana do mniejszej wartości. Oznacza to, że INT(5.5)=5, lecz INT(-5.5)=-6! Procedura realizacyjna tej operacji - XINT - jest bardzo zbliżona do procedur operacji arytmetycznych. Składa się ona z wywołań trzech procedur: odczytu liczby (GETVAR), wykonania obliczenia (BINT) i zapisu wyniku (PUTVAR).

```
0100 ;eXecute INT function
0110 ;
0120 BINT =  $B0D2
0130 GETVAR = $ABE9
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $B0C8
0170 ;
0180     JSR GETVAR
0190     JSR BINT
0200     JMP PUTVAR
```

Zasadnicze obliczenia w funkcji INT są wykonywane przez procedurę BINT. Rozpoczyna się ona od obliczenia pozycji przecinka w liczbie poddawanej przekształceniu. Następnie wszystkie cyfry po przecinku są zerowane, przy czym ich dotychczasowe wartości są sumowane w akumulatorze. Jeżeli liczba była dodatnia (FR0 mniejsze od $80) lub całkowita ujemna (akumulator równy zero), to BINT jest opuszczana przez skok do procedury normalizującej format - NFR0. Niecałkowita wartość liczby ujemnej wymaga jeszcze odjęcia jedności od wyniku (dokładnie jest to dodanie -1). Po tej operacji normalizacja formatu nie jest już potrzebna.

```
0100 ;Basic INTeger routine
0110 ;
0120 BADD =  $AD26
0130 FR0 =   $D4
0140 FR1 =   $E0
0150 NFR0 =  $DC00
0160 ZFR0 =  $DA44
0170 ;
0180     *=  $B0D1
0190 ;
0200     LDA FR0
0210     AND #$7F
0220     SEC
0230     SBC #$3F
0240     BPL EXE
0250     LDA #$00
0260 EXE TAX
0270     LDA #$00
0280     TAY
0290 LOOP CPX #$05
0300     BCS ADD
0310     ORA FR0+1,X
0320     STY FR0+1,X
```

```
0330     INX
0340     BNE LOOP
0350 ADD LDX FR0
0360     BPL NRM
0370     TAX
0380     BEQ NRM
0390     LDX #FR1
0400     JSR ZFR0+2
0410     LDA #$C0
0420     STA FR1
0430     LDA #$01
0440     STA FR1+1
0450     JSR BADD
0460     RTS
0470 NRM JMP NFR0
```

5.2.4. Funkcja SQR

Funkcja SQR zwraca wartość pierwiastka kwadratowego podnego argumentu. Także tutaj zastosowano wywołanie zasadniczej procedury wykonawczej SQR po odczytaniu liczby przez GETVAR. Natomiast poprawność wyniku jest rozpoznawana w procedurze XSQR przez sprawdzenie bitu Carry. Błąd jest sygnalizowany przez BVALER (Bad VALue ERror), zaś sukces powoduje skok do PUTRES (znajduje się tam tylko rozkaz skoku JMP PUTVAR). Obie użyte tu etykiety - RESULT i VALER - są wykorzystywane przez procedury innych funkcji, co pozwala na zastąpienie rozkazów skoków bezwzględnych (JMP i JSR) rozkazami skoków względnych (B..). Każda taka zamiana daje jeden bajt oszczędności.

```
0100 ;eXecute SQR function
0110 ;
0120 BVALER = $B92E
0130 GETVAR = $ABE9
0140 PUTRES = $B16C
0150 SQR =   $BF43
0160 ;
0170     *=  $B153
0180 ;
0190 XSQR JSR GETVAR
0200     JSR SQR
0210 ;
0220 ;RESULT
0230 ;
0240 RESULT BCC PUTRES
0250 ;
0260 ;VALue ERror
0270 ;
0280 VALER JSR BVALER
```

Ze względu na znaczny stopień skomplikowania procedury SQR jej opis zostanie pominięty, a dociekliwi Czytelnicy mogą dokonać analizy samodzielnie. Należy jednak zauważyć, że poza wykorzystywaniem standardowych procedur obliczeń zmiennoprzecinkowych (FADD, FSUB, FMUL i FDIV) oraz przemieszczeń liczb FP (FLD.., FST.. i FMOV..) są tu jeszcze używane dwie tablice: TLOG i SQRC. Pierwsza z nich zawiera współczynniki logarytmowania i znajduje się w pakiecie FP.

```
0100 DIGRT = $F1
```

```
0110 ESIGN = $EF
0120 FADD =  $DA66
0130 FDIV =  $DB28
0140 FLD0R = $DD89
0150 FLD1R = $DD98
0160 FMOV01 = $DDB6
0170 FMUL =  $DADB
0180 FPSCR = $05E6
0190 FPSCR1 = $05EC
0200 FR0 =   $D4
0210 FR1 =   $E0
0220 FST0R = $DDA7
0230 FSUB =  $DA60
0240 SQRC =  $BAF1
0250 TLOG =  $DF66
0260 ;
0270     *=  $BF41
0280 ;
0290 ERR SEC
0300     RTS
0310 ;
0320 ;SQuare Root routine
0330 ;
0340 SQR LDA #$00
0350     STA DIGRT
0360     LDA FR0
0370     BMI ERR
0380     CMP #$3F
0390     BEQ LBL1
0400     CLC
0410     ADC #$01
0420     STA DIGRT
0430     STA FR1
0440     LDA #$01
0450     STA FR1+1
0460     LDX #$04
0470     LDA #$00
0480 NXT1 STA FR1+2,X
0490     DEX
0500     BPL NXT1
0510     JSR FDIV
0520 LBL1 LDA #$06
0530     STA ESIGN
0540     LDX # <FPSCR
0550     LDY # >FPSCR
0560     JSR FST0R
0570     JSR FMOV01
0580     LDX # <SQRC
0590     LDY # >SQRC
0600     JSR FLD0R
0610     JSR FSUB
0620     LDX # <FPSCR
0630     LDY # >FPSCR
0640     JSR FLD1R
0650     JSR FMUL
0660 LOOP LDX # <FPSCR1
0670     LDY # >FPSCR1
0680     JSR FST0R
0690     JSR FMOV01
0700     LDX # <FPSCR
0710     LDY # >FPSCR
```

```
0720     JSR FLD0R
0730     JSR FDIV
0740     LDX # <FPSCR1
0750     LDY # >FPSCR1
0760     JSR FLD1R
0770     JSR FSUB
0780     LDX # <TLOG+6
0790     LDY # >TLOG+6
0800     JSR FLD1R
0810     JSR FMUL
0820     LDA FR0
0830     BEQ LBL2
0840     LDX # <FPSCR1
0850     LDY # >FPSCR1
0860     JSR FLD1R
0870     JSR FADD
0880     DEC ESIGN
0890     BPL LOOP
0900 LBL2 LDX # <FPSCR1
0910     LDY # >FPSCR1
0920     JSR FLD0R
0930     LDA DIGRT
0940     BEQ END
0950     SEC
0960     SBC #$40
0970     CLC
0980     ROR A
0990     CLC
1000     ADC #$40
1010     AND #$7F
1020     STA FR1
1030     LDA DIGRT
1040     ROR A
1050     LDA #$01
1060     BCC LBL3
1070     LDA #$10
1080 LBL3 STA FR1+1
1090     LDX #$04
1100     LDA #$00
1110 NXT2 STA FR1+2,X
1120     DEX
1130     BPL NXT2
1140     JSR FMUL
1150 END RTS
```

Druga tablica, umieszczona w interpreterze Atari Basic zawiera stałą o wartości 2 ($2*100^0$). Stała ta jest wykorzystywana jako wartość stopnia pierwiastkowania.

```
0100 ;SQuare Root Constant
0110 ;
0120     *=  $BAF1
0130 ;
0140     .BYTE $40,$02
0150     .BYTE $00,$00
0160     .BYTE $00,$00
```

Konieczna jest tu jeszcze jedna dygresja. Uważny Czytelnik powinien zauważyć niezgodność adresu procedury SQR z podanym w pierwszej części "Mapy" ("Podstawowe procedury systemu operacyjnego"). Prawdziwy jest oczywiście adres podany tu, choć we wszystkich dostępnych

źródłach jest inaczej. Podawany tam bowiem adres procedury SQR jest adresem w interpreterze Basic Revision B!, a więc w poprzedniej wersji. Uwaga ta dotyczy także pozostałych procedur zmiennoprzecinkowych: SIN, COS i ATAN.

5.2.5. Funkcja EXP

Funkcja EXP zwraca wartość podniesienia liczby e (2.71828183) do potęgi podanej jako argument. Do realizacji tej funkcji wykorzystywana jest przez procedurę XEXP standardowa procedura pakietu FP - EXP. Przedtem jednak argument funkcji jest pobierany z bufora przez wywołanie GETVAR. Uzyskany wynik jest natomiast przesyłany do procedury RESULT, gdzie dokonywana jest ocena jego poprawności i zapisanie do bufora.

```
0100 ;eXecute EXP function
0110 ;
0120 EXP =   $DDC0
0130 GETVAR = $ABE9
0140 RESULT = $B159
0150 ;
0160     *=  $B14A
0170 ;
0180     JSR GETVAR
0190     JSR EXP
0200     JMP RESULT
```

5.2.6. Funkcje LOG i CLOG

Dla podanego argumentu funkcja LOG zwraca wartość logarytmu naturalnego (przy podstawie e), a funkcja CLOG - logarytmu dziesiętnego (przy podstawie 10). Wykonanie tych funkcji jest przeprowadzane przez procedury XLOG i XCLOG. Są one niemal identyczne, a różni je tylko wywoływana procedura pakietu FP: dla XLOG jest to LOG, zaś dla XCLOG - LOG10. Najpierw liczba do logarytmowania jest odczytywana z bufora przez GETVAR i gdy jest równa zero, to niepoprawna wartość jest sygnalizowana przez skok do VALER. W przeciwnym przypadku wywoływana jest odpowiednia procedura zmiennoprzecinkowa. Od tego miejsca przebieg obu procedur (XLOG i XCLOG) jest wspólny.

```
0100 ;eXecute LOG function
0110 ;
0120 FR0 =   $D4
0130 GETVAR = $ABE9
0140 LOG =   $DECD
0150 LOG10 = $DED1
0160 PUTRES = $B16C
0170 VALER = $B15B
0180 ;
0190     *=  $B121
0200 ;
0210 XLOG JSR GETVAR
0220     LDA FR0
0230     BEQ VALER
0240     JSR LOG
0250 ;
0260 ;STore LOGarithm
0270 ;
0280 STLOG BCS VALER
```

```
0290     LDA FR0
0300     EOR #$3B
0310     BNE PUTRES
0320     LDA FR0+1
0330     AND #$F8
0340     BNE PUTRES
0350     STA FR0
0360     BEQ PUTRES
0370 ;
0380 ;eXecute CLOG function
0390 ;
0400 XCLOG JSR GETVAR
0410     LDA FR0
0420     BEQ VALER
0430     JSR LOG10
0440     JMP STLOG
```

Przede wszystkim ustawiony bit Carry powoduje sygnalizację błędu przez skok do VALER. Następnie uzyskany rezultat jest przetwarzany do poprawnej postaci i procedura jest opuszczana skokiem do PUTRES.

5.2.7. Funkcja SIN

Funkcja SIN zwraca wartość sinusa podanego argumentu. Procedura wykonawcza tej funkcji - XSIN - ma standardową strukturę: odczyt argumentu (GETVAR), wykonanie obliczenia (SIN) oraz sprawdzenie i zapisanie wyniku (RESULT).

```
0100 ;eXecute SIN function
0110 ;
0120 GETVAR = $ABE9
0130 RESULT = $B159
0140 SIN =   $BE05
0150 ;
0160     *=  $B106
0170 ;
0180     JSR GETVAR
0190     JSR SIN
0200     JMP RESULT
```

Zasadnicza procedura obliczeniowa jest wspólna dla obu funkcji trygonometrycznych (SIN i COS). Różna jest tylko wartość początkowa w tej procedurze. Do rejestru FCHRFLG (First CHaRacter FLaG) jest wpisywana wartość $01 dla cosinusa, $02 dla sinusa liczby ujemnej i $04 dla sinusa liczby dodatniej. Szczegółowy opis tej procedury zostanie pominięty. Trzeba jednak zwrócić uwagę na dwie tablice wykorzystywane przez nią. Tablica TRIGC zawiera wartość 1.

```
0100 ;TRIGonometric's Constant
0110 ;
0120     *=  $BECF
0130 ;
0140     .BYTE $40,$01,$00
0150     .BYTE $00,$00,$00

0100 DIGRT = $F1
0110 FCHRFLG = $F0
0120 FDIV =  $DB28
0130 FLD0R = $DD89
```

```
0140 FLD1R = $DD98
0150 FMOV01 = $DDB6
0160 FMUL =  $DADB
0170 FPSCR = $05E6
0180 FR0 =   $D4
0190 FR1 =   $E0
0200 FST0R = $DDA7
0210 FSUB =  $DA60
0220 PLYEVL = $DD40
0230 RADFLG = $FB
0240 TRIGC = $BECF
0250 TSCC =  $BE9F
0260 ;
0270     *=  $BE03
0280 ;
0290 ERR SEC
0300     RTS
0310 ;
0320 ;SINus routine
0330 ;
0340 SIN LDA #$04
0350     BIT FR0
0360     BPL EXE
0370     LDA #$02
0380     BNE EXE
0390 ;
0400 ;COSinus routine
0410 ;
0420 COS LDA #$01
0430 EXE STA FCHRFLG
0440     LDA FR0
0450     AND #$7F
0460     STA FR0
0470     LDA #<TSCC+$1F
0480     CLC
0490     ADC RADFLG
0500     TAX
0510     LDY #>TSCC+$1F
0520     JSR FLD1R
0530     JSR FDIV
0540     BCC CNT
0550     RTS
0560 CNT LDA FR0
0570     AND #$7F
0580     SEC
0590     SBC #$40
0600     BMI LBL2
0610     CMP #$04
0620     BPL ERR
0630     TAX
0640     LDA FR0+1,X
0650     STA DIGRT
0660     AND #$10
0670     BEQ LBL1
0680     LDA #$02
0690 LBL1 CLC
0700     ADC DIGRT
0710     AND #$03
0720     ADC FCHRFLG
0730     STA FCHRFLG
0740     STX DIGRT
```

```
0750     JSR FMOV01
0760     LDX DIGRT
0770     LDA #$00
0780 LOOP STA FR1+2,X
0790     INX
0800     CPX #$03
0810     BCC LOOP
0820     JSR FSUB
0830 LBL2 LSR FCHRFLG
0840     BCC LBL3
0850     JSR FMOV01
0860     LDX # <TRIGC
0870     LDY # >TRIGC
0880     JSR FLD0R
0890     JSR FSUB
0900 LBL3 LDX # <FPSCR
0910     LDY >FPSCR
0920     JSR FST0R
0930     JSR FMOV01
0940     JSR FMUL
0950     BCS ERR
0960     LDA #$06
0970     LDX # <TSCC
0980     LDY # >TSCC
0990     JSR PLYEVL
1000     LDX # <FPSCR
1010     LDY # >FPSCR
1020     JSR FLD1R
1030     JSR FMUL
1040     LSR FSCHFLG
1050     BCC END
1060     CLC
1070     LDA FR0
1080     BEQ END
1090     EOR #$80
1100     STA FR0
1110 END RTS
```

Druga tablica (TSCC - stosowana tylko do obliczania funkcji sinus i cosinus) zawiera siedem wartości dla procedury przeliczania wielomianowego PLYEVL (znajduje się ona w pakiecie FP - zob. "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego"). Są to kolejno liczby: $-3.55149939*10^{-6}$, $1.60442752*10^{-4}$, $-4.681754355*10^{-3}$, 0.0796926239, -0.6459640867, 1.57079632 i 90.

```
0100 ;Table: SIN & COS Constants
0110 ;
0120 *= $BE9F 0130 ;
0140 .BYTE $BD,$03,$55
0150 .BYTE $14,$99,$39
0160 .BYTE $3E,$01,$60
0170 .BYTE $44,$27,$52
0180 .BYTE $BE,$46,$81
0190 .BYTE $75,$43,$55
0200 .BYTE $3F,$07,$96
0210 .BYTE $92,$62,$39
```

```
0220 .BYTE $BF,$64,$59
0230 .BYTE $64,$08,$67
0240 .BYTE $40,$01,$57
0250 .BYTE $07,$96,$32
0260 .BYTE $40,$90,$00
0270 .BYTE $00,$00,$00
```

Książka "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego" podaje adresy procedur SIN i COS zawartych w poprzedniej wersji interpretera Atari Basic (Revision B). Uwaga ta dotyczy także pozostałych procedur zmiennoprzecinkowych: SQR i ATAN.

5.2.8. Funkcja COS

Funkcja COS zwraca wartość cosinusa podanego argumentu. Jej procedura wykonawcza - XCOS - jest niemal identyczna z procedurą funkcji SIN (zob. wyżej).

```
0100 ;eXecute COS function
0110 ;
0120 COS =   $BE0F
0130 GETVAR = $ABE9
0140 RESULT = $B159
0150 ;
0160     *=  $B10F
0170 ;
0180     JSR GETVAR
0190     JSR COS
0200     JSR RESULT
```

5.2.9. Funkcja ATN

Funkcja ATN zwraca wartość arcus tangens podanego argumentu. Także tu procedura wykonawcza - XATN - jest niemal identyczna z procedurą funkcji SIN (zob. wyżej).

```
0100 ;eXecute ATN function
0110 ;
0120 ATAN =  $BED5
0130 GETVAR = $ABE9
0140 RESULT = $B159
0150 ;
0160     *=  $B118
0170 ;
0180     JSR GETVAR
0190     JSR ATN
0200     JSR RESULT
```

Odmienna jest natomiast właściwa procedura obliczeniowa ATAN. Ponieważ jej struktura jest bardzo skomplikowana i zawiera wiele kolejnych przekształceń matematycznych, to opis zostanie pominięty.

```
0100 ;Arcus TANgent routine
0110 ;
0120 DIGRT = $F1
0130 FADD =  $DA66
0140 FCHRFLG = $F0
0150 FDIV =  $DB28
0160 FLD1R = $DD98
```

```
0170 FMOV01 = $DDB6
0180 FMUL =  $DADB
0190 FPSCR = $05E6
0200 FR0 =   $D4
0210 FST0R = $DDA7
0220 PLYEVL = $DD40
0230 RADFLG = $FB
0240 RSQT =  $DE95
0250 TATAN = $DFAE
0260 TATNC = $BEC9
0270 ;
0280     *=  $BED5
0290 ;
0300     LDA #$00
0310     STA FCHRFLG
0320     STA DIGRT
0330     LDA FR0
0340     AND #$7F
0350     CMP #$40
0360     BMI LBL1
0370     LDA FR0
0380     AND #$80
0390     STA FCHRFLG
0400     INC DIGRT
0410     LDA #$7F
0420     AND FR0
0430     STA FR0
0440     LDX # <TATAN+$3C
0450     LDY # >TATAN+$3C
0460     JSR RSQT
0470 LBL1 LDX # <FPSCR
0480     LDY # >FPSCR
0490     JSR FST0R
0500     JSR FMOV01
0510     JSR FMUL
0520     BCS END
0530     LDA #$0B
0540     LDX # <TATAN
0550     LDY # >TATAN
0560     JSR PLYEVL
0570     BCS END
0580     LDX # <FPSCR
0590     LDY # >FPSCR
0600     JSR FLD1R
0610     JSR FMUL
0620     BCS END
0630     LDA DIGRT
0640     BEQ LBL2
0650     LDX # <TATAN+$42
0660     LDY # >TATAN+$42
0670     JSR FLD1R
0680     JSR FADD
0690     LDA FCHRFLG
0700     ORA FR0
0710     STA FR0
0720 LBL2 LDA RADFLG
0730     BEQ END
0740     LDX # <TATNC
0750     LDY # >TATNC
0760     JSR FLD1R
0770     JSR FDIV
```

```
0780 END RTS
```

Wykorzystywana przez procedurę ATAN tablica TATNC zawiera stałą liczbową o wartości 0.01745329.

```
0100 ;Table: ATN's Constant
0110 ;
0120     *=  $BEC9
0130 ;
0140     .BYTE $3F,$01,$74
0150     .BYTE $53,$29,$25
```

Książka "Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego" podaje adres procedury ATAN zawartej w poprzedniej wersji interpretera Atari Basic (Revision B). Uwaga ta dotyczy także pozostałych procedur zmiennoprzecinkowych: SQR, SIN i COS.

5.2.10. Funkcja CHR$

Funkcja CHR$ zwraca znak, którego kod ASCII został podany jako argument, jej wynikiem jest więc stała tekstowa o długości jednego znaku. Argument pobrany przez procedurę GETVAR jest zamieniany przy pomocy GETINT na dwubajtową liczbę całkowitą, a jej młodszy bajt przepisywany jest do bufora LBUFF (Line BUFFer). Starszy bajt jest przy tym całkowicie pomijany, więc argument większy od 255 ($FF) da w rezultacie znak, którego kod ASCII jest równy reszcie z dzielenia argumentu przez 256. Trzeba jednak pamiętać, że argument większy od 65535 ($FFFF) spowoduje błąd podczas realizacji procedury GETINT.

Ponieważ wynik jest zapisywany zawsze w to samo miejsce bufora LBUFF, to jednocześnie można wykonać tylko jedną funkcję CHR$. Na przykład, porównanie CHR$(A)=CHR$(B) będzie zawsze prawdziwe, gdyż przed wykonaniem porównania nowa wartość funkcji CHR$ skasuje poprzednią.

```
0100 ;eXecute CHR$ function
0110 ;
0120 FR0 =   $D4
0130 GETINT = $AD41
0140 GETVAR = $ABE9
0150 LBUFF = $0580
0160 PUTVAR = $ABB2
0170 VARN =  $D3
0180 VART =  $D2
0190 ;
0200     *=  $B052
0210 ;
0220     JSR GETVAR
0230     JSR GETINT
0240     LDA FR0
0250     STA LBUFF+$40
0260     LDA # >LBUFF+$40
0270     STA FR0+1
0280     LDA # <LBUFF+$40
0290     STA FR0
0300     LDA #$01
0310     STA FR0+2
0320 ;
```

```
0330 ;STRing TYPe
0340 ;
0350 STRTYP LDA #$00
0360     STA FR0+3
0370     STA VARN
0380     LDA #$83
0390     STA VART
0400     JMP PUTVAR
```

Adres znaku w buforze LBUFF umieszczany jest teraz w dwóch pierwszych bajtach rejestru FR0, a w dwóch następnych wartość 1, która oznacza długość stałej tekstowej. Odpowiednio do rejestru VARN wpisywane jest zero, a do VART - $83, co oznacza stałą tekstową. Procedura XCHR kończy się skokiem do PUTVAR, a więc zapisaniem wyniku do bufora tokenizacji.

5.2.11. Funkcja STR$

Funkcja STR$ zamienia podany argument liczbowy na ciąg znaków ASCII. Procedura wykonawcza XSTR pobiera liczbę przez GETVAR i przy pomocy procedury FASC zamienia ją na ciąg znaków. Adres bufora zawierającego ten ciąg jest wpisywany do rejestru FR0 (dwa pierwsze bajty). Następnie obliczana jest długość ciągu, a wynik umieszczany jest w trzecim bajcie FR0. Teraz następuje skok do etykiety STRTYP (wewnątrz procedury XCHR), gdzie zapisywany jest numer i typ wartości.

Ta funkcja może być użyta w wyrażeniu tylko jeden raz, gdyż podobnie jak XCHR, umieszcza wynik zawsze w tym samym miejscu. Ponadto trzeba uważać przy jednoczesnym użyciu funkcji STR$ i CHR$, gdyż obie korzystają z bufora LBUFF. Jeśli ciąg wynikowy funkcji STR$ będzie miał długość większą od 64 znaków, to zajmie miejsce przeznaczone na wynik funkcji CHR$. Ten rezultat, który był wcześniej obliczony, będzie więc niepoprawny.

```
0100 ;eXecute STR$ function
0110 ;
0120 FASC =  $D8E6
0130 FR0 =   $D4
0140 GETVAR = $ABE9
0150 INBUFP = $F3
0160 STRTYP = $B069
0170 ;
0180     *=  $B034
0190 ;
0200     JSR GETVAR
0210     JSR FASC
0220     LDA INBUFP
0230     STA FR0
0240     LDA INBUFP+1
0250     STA FR0+1
0260     LDY #$FF
0270 LOOP INY
0280     LDA (INBUFP),Y
0290     BPL LOOP
0300     AND #$7F
0310     STA (INBUFP),Y
0320     INY
0330     STY FR0+2
0340     BNE STRTYP
```

5.2.12. Funkcja USR

Funkcja USR powoduje wywołanie procedury w języku maszynowym od adresu podanego jako argument. Ponadto dalsze argumenty - jeśli są - zostają przekazane jako parametry do tej procedury. Do realizacji tej funkcji służy procedura XUSR. Zasadnicza operacja odczytu parametrów i wywołania procedury maszynowej użytkownika jest wykonywana przez PHDAT. Przekazany przez nią wynik jest zamieniany na liczbę zmiennoprzecinkową (przez IFP) i przy pomocy procedury PUTVAR zapisywany w buforze.

```
0100 ;eXecute USR function
0110 ;
0120 IFP =   $D9AA
0130 PHDAT = $B0AE
0140 PUTVAR = $ABB2
0150 ;
0160     *=  $B0A5
0170 ;
0180     JSR PHDAT
0190     JSR IFP
0200     JMP PUTVAR
```

Procedura PHDAT rozpoczyna się od odczytania z rejestru BHLD1 liczby parametrów funkcji USR i umieszczenia jej w liczniku ACNT1 (Auxiliary CouNTer 1). Następnie w pętli sterowanej stanem tego licznika wywoływana jest procedura GETNUM, która odczytuje kolejne parametry. Są one odkładane na stosie w kolejności bajtów młodszy/starszy. Na końcu wpisywana jest na stos liczba parametrów, natomiast ostatnia odczytana liczba pozostaje w rejestrze FR0. Liczba ta jest wykorzystywana jako adres skoku kończącego procedurę PHDAT.

```
0100 ;PusH user's DATa
0110 ;
0120 ACNT1 = $C6
0130 BHLD1 = $B0
0140 FR0 =   $D4
0150 GETNUM = $ABDA
0160 ;
0170     *=  $B0AE
0180 ;
0190     LDA BHLD1
0200     STA ACNT1
0210 LOOP JSR GETNUM
0220     DEC ACNT1
0230     BMI EXIT
0240     LDA FR0
0250     PHA
0260     LDA FR0+1
0270     PHA
0280     JMP LOOP
0290 EXIT LDA BHLD1
0300     PHA
0310     JMP (FR0)
```

Wspomniana procedura GETNUM jest częścią składową LETNUM. Przez wywołanie XLET wartość wyrażenia jest obliczana i umieszczana w buforze. Teraz (od etykiety GETNUM) z bufora

pobierana jest liczba zmiennoprzecinkowa, którą procedura GETINT zamienia na dwubajtową liczbę całkowitą.

```
0100 GETINT = $AD41
0110 GETVAR = $ABE9
0120 XLET =  $AADA
0130 ;
0140     *=  $ABD7
0150 ;
0160 ;LET NUMber
0170 ;
0180 LETNUM JSR XLET
0190 ;
0200 ;GET NUMber
0210 ;
0220 GETNUM JSR GETVAR
0230     JMP GETINT
```

5.2.13. Funkcja LEN

Funkcja LEN zwraca wartość długości ciągu podanego jako argument. Wykonująca tą operację procedura XLEN najpierw odczytuje przy pomocy procedury FTARV wartość z bufora tokenizacji i umieszcza ją w rejestrze FR0. Procedura FTARV jest wywoływana zamiast bezpośredniego wywołania GETVAR, ponieważ sprawdza ponadto poprawność argumentu. Następnie wartość określająca aktualną długość ciągu jest przepisywana z trzeciego i czwartego bajtu rejestru FR0 do jego dwóch pierwszych bajtów. Przepisanie zawartości akumulatora i rejestru Y do FR0 zostało oznaczone etykietą STNUM i jest wykorzystywane także przez inne procedury.

```
0100 ;eXecute LEN function
0110 ;
0120 FR0 =   $D4
0130 FTARV = $AB90
0140 IFP =   $D9AA
0150 PUTVAR = $ABB2
0160 VARN =  $D3
0170 VART =  $D2
0180 ;
0190     *=  $AFB5
0200 ;
0210 XLEN JSR FTARV
0220     LDA FR0+2
0230     LDY FR0+3
0240 ;
0250 ;STore NUMber
0260 ;
0270 STNUM STA FR0
0280     STY FR0+1
0290 ;
0300 ;CHange TYPe
0310 ;
0320 CHTYP JSR IFP
0330 ;
0340 ;NUMber TYPe
0350 ;
0360 NUMTYP LDA #$00
0370     STA VART
0380     STA VARN
```

```
0390     JMP PUTVAR
```

Teraz liczba całkowita jest zamieniana na zmienno- przecinkową (jest to oznaczone przez CHTYP). Po wpisaniu do rejestrów VART i VARN wartości oznaczających stałą liczbową procedura XLEN kończy się skokiem do PUTVAR, gdzie wynik przepisywany jest do bufora.

W celu odczytania adresu ciągu będącego argumentem procedura FTARV najpierw wywołuje GETVAR. Jeśli zawartość rejestru VART oznacza stałą tekstową, to procedura FTARV się kończy. Skasowany bit 0 tego rejestru sygnalizuje natomiast, że zmienna nie była zadeklarowana i powoduje skok do procedury błędu DIMER (DIMension ERror).
0100 ;Find Table or ARray Value 0110 ; 0120 DIMER = $B922 0130 FR0 = $D4 0140 GETVAR = $ABE9 0150 STARP = $8C 0160 VART = $D2 0170 ; 0180 *= $AB90 0190 ; 0200 JSR GETVAR 0210 LDA #$02 0220 BIT VART 0230 BNE END 0240 ORA VART 0250 STA VART 0260 ROR A 0270 BCC ERR 0280 CLC 0290 LDA FR0 0300 ADC STARP 0310 STA FR0 0320 TAY 0330 LDA FR0+1 0340 ADC STARP+1 0350 STA FR0+1 0360 END RTS 0370 ERR JSR DIMER
Gdy zmienna jest poprawna, to przez dodanie zawartości STARP i FR0 obliczany jest jej adres w pamięci. Ta część procedury jest zbędna przy wykonywaniu funkcji XLEN, a wykorzystuje ją tylko funkcja XADR (zob. niżej).

5.2.14. Funkcja FRE

Funkcja FRE zwraca wartość określającą wielkość pozostałej jeszcze, wolnej pamięci RAM. Procedura wykonawcza tej funkcji - XFRE - rozpoczyna się od wywołania GETVAR w celu odczytania z bufora argumentu, który nie ma jednak żadnego znaczenia. Właściwe obliczenie polega na odjęciu zawartości rejestrów MEMTOP (MEMory TOP) i BMEMHI (Basic MEMory HIgh address). Uzyskany rezultat określa liczbę bajtów pomiędzy górną granicą obszaru pamięci zajmowanego przez program w Basicu i dolną granicą programu ANTIC-a. Doprowadzenie wyniku odejmowania, który jest dwubajtową liczbą całkowitą do poprawnej postaci, jest osiągane przez skok do procedury CHTYP (wewnątrz XLEN).

```
0100 ;eXecute FRE function
0110 ;
0120 BMEMHI = $90
0130 CHTYP = $AFC0
0140 FR0 =   $D4
0150 GETVAR = $ABE9
0160 MEMTOP = $02E5
0170 ;
0180     *=  $AFD6
0190 ;
0200     JSR GETVAR
0210     SEC
0220     LDA MEMTOP
0230     SBC BMEMHI
0240     STA FR0
0250     LDA MEMTOP8+1
0260     SBC BMEMHI+1
0270     STA FR0+1
0280     JMP CHTYP
```

Przy wykorzystywaniu wyniku funkcji FRE do określenia, czy pozostaje wystarczający obszar pamięci na umieszczenie dodatkowych danych lub procedur w języku maszynowym, trzeba pamiętać, że górna granica programu Basica jest ruchoma. Każde otwarcie pętli FOR/NEXT zajmuje 16 bajtów, a każde wywołanie procedury GOSUB/RETURN - cztery bajty. Dostępny obszar pamięci należy więc zmniejszyć o liczbę ustaloną w zależności od maksymalnej liczby jednocześnie aktywnych pętli i procedur.

5.2.15. Funkcja PEEK

Funkcja PEEK zwraca wartość zapisaną w komórce pamięci, której adres został podany jako argument. W tym celu procedura GETNUM odczytuje ze stosu argument i zamienia go na liczbę całkowitą. Liczba ta jest następnie wykorzystywana jako adres odczytu zawartości komórki pamięci. Doprowadzenie wyniku do postaci zmiennoprzecinkowej i zapisanie w buforze jest dokonywane przez procedurę STNUM wchodzącą w skład XLEN (zob. wyżej).

```
0100 ;eXecute PEEK function
0110 ;
0120 FR0 =   $D4
0130 GETNUM = $ABDA
0140 STNUM = $AFBC
0150 ;
0160     *=  $AFCC
0170 ;
0180     JSR GETNUM
0190     LDY #$00
0200     LDA (FR0),Y
0210     JMP STNUM
```

5.2.16. Funkcja ADR

Funkcja ADR zwraca wartość określającą adres pierwszego znaku ciągu podanego jako argument. Procedura realizująca tą operację zawiera jedynie wywołanie FTARV i skok do CHTYP. Opisywana już wcześniej (zob. 5.2.13) procedura FTARV umieszcza w dwóch pierwszych bajtach rejestru FR0 adres podanego ciągu. Adres ten jest przekształcany do postaci stałej liczbowej w procedurze CHTYP, która znajduje się wewnątrz XLEN.

```
0100 ;eXecute ADR function
0110 ;
0120 CHTYP = $AFC0
0130 FTARV = $AB90
0140 ;
0150     *=  $B007
0160 ;
0170     JSR FTARV
0180     JMP CHTYP
```

5.2.17. Funkcja ASC

Funkcja ASC zwraca wartość kodu ATASCII pierwszego znaku ciągu podanego jako argument, jest więc odwrotnością funkcji CHR$. Jej procedura wykonawcza XASC przebiega podobnie do procedury XADR. Jednak do CHTYP jest przekazywany nie adres, lecz odczytany według tego adresu kod pierwszego znaku ciągu.

```
0100 ;eXecute ASC function
0110 ;
0120 FR0 =   $D4
0130 FTARV = $AB90
0140 STNUM = $AFBC
0150 ;
0160     *=  $AFFD
0170 ;
0180     JSR FTARV
0190     LDY #$00
0200     LDA (FR0),Y
0210     JMP STNUM
```

5.2.18. Funkcja VAL

Funkcja VAL zwraca wartość uzyskaną z zamiany na liczbę cyfr ciągu ASCII podanego jako argument i jest funkcją odwrotną dla STR$. Realizuje ją procedura XVAL, która najpierw odczytuje informacje o ciągu przy pomocy GFILSP, a następnie zamienia odnaleziony ciąg na liczbę zmiennoprzecinkową przez wywołanie procedury AFP z pakietu FP. Wywołanie procedury PSTMAD jest tu konieczne tylko dla skasowania znacznika MEOLFLG (Modified EOL FLaG) ustawionego przez procedurę GFILSP. Poprawny wynik otrzymuje odpowiedni format i jest zapisywany na stosie przez skok do procedury NUMTYP, która znajduje się wewnątrz XLEN. Jeśli podany ciąg nie daje się zamienić na liczbę, to wykonywany jest skok do CHARER i sygnalizowany jest błąd (CHARacter ERror).

```
0100 ;eXecute VAL function
0110 ;
0120 AFP =   $D800
0130 CHARER = $B910
0140 CIX =   $F2
0150 GFILSP = $BD7D
0160 NUMTYP = $AFC3
0170 PSTMAD = $BD9D
0180 ;
0190     *=  $AFEB
0200 ;
0210     JSR GFILSP
0220     LDA #$00
0230     STA CIX
0240     JSR AFP
0250     JSR PSTMAD
0260     BCC NUMTYP
0270     JSR CHARER
```

Procedura GFILSP najpierw odszukuje podany ciąg przy pomocy FTARV, a następnie wpisuje jego adres do wektora INBUFP. Ponadto do rejestru CSTAD przepisywany jest starszy bajt adresu ciągu i młodszy bajt jego długości. Przed opuszczeniem procedury znak końca wiersza (RETURN)

jest jeszcze umieszczany na końcu ciągu, a jeśli jego długość jest większa od 256 bajtów, to jako 256 znak.

```
0100 ;Get FILe SPecification
0110 ;
0120 CSTAD = $97
0130 FR0 =   $D4
0140 FTARV = $AB90
0150 INBUFP = $F3
0160 MEOLFLG = $92
0170 ;
0180     *=  $BD7D
0190 ;
0200     JSR FTARV
0210     LDA FR0
0220     STA INBUFP
0230     LDA FR0+1
0240     STA INBUFP+1
0250     LDY FR0+2
0260     LDX FR0+3
0270     BEQ PUT
0280     LDY #$FF
0290 PUT LDA (INBUFP),Y
0300     STA CSTAD
0310     STY CSTAD+1
0320     LDA #$9B
0330     STA (INBUFP),Y
0340     STA MEOLFLG
0350     RTS
```

5.2.19. Funkcje manipulatorów

Atari Basic posiada cztery dodatkowe funkcje, które służą do odczytu położenia manipulatorów. Funkcja STICK zwraca liczbę określającą pozycję joysticka, a STRIG - stan jego przycisku. Podobnie wynik funkcji PADDLE określa stan potencjometru, a PTRIG - jego przycisku. Argumentem tych funkcji jest numer manipulatora: 0 lub 1 dla joysticka oraz liczba od 0 do 3 dla potencjometru. Działanie wszystkich tych funkcji polega na odczycie zawartości odpowiedniego rejestru. Rejestr jest wybierany przez zsumowanie jego numeru i odległości od rejestru pierwszego manipulatora (PADDLE0). Przekształcenia i zapisania wyniku dokonuje procedura STNUM.

```
0100 BVALER = $B92E
0110 FR0 =   $D4
0120 GETNUM = $ABDA
0130 PADDL0 = $0270
0140 STNUM = $AFBC
0150 ;
0160     *=  $B00D
0170 ;
0180 ;eXecute PADDLE function
0190 ;
0200 XPADDL LDA #$00
0210     BEQ GET
0220 ;
0230 ;eXecute STICK function
0240 ;
0250 XSTICK LDA #$08
0260     BNE GET
```

```
0270 ;
0280 ;eXecute PTRIG function
0290 ;
0300 XPTRIG LDA #$0C
0310     BNE GET
0320 ;
0330 ;eXecute STRIG function
0340 ;
0350 XSTRIG LDA #$14
0360 GET PHA
0370     JSR GETNUM
0380     LDA FR0+1
0390     BNE ERR
0400     LDA FR0
0410     PLA
0420     CLC
0430     ADC FR0
0440     TAX
0450     LDA PADDL0,X
0460     LDY #$00
0470     BEQ STNUM
0480 ERR JSR BVALER
```

5.3. Inne operatory

Poza opisanymi wyżej operatorami i funkcjami Atari Basic posiada jeszcze dodatkowe operatory. Umożliwiają one wykonanie porównań, operacji logicznych, zmianę priorytetu działań itd. W wielu przypadkach operatory te są oznaczane jednakowymi symbolami, na co była już wcześniej zwracana uwaga.

5.3.1. Operatory relacji i logiczne

Operatory logiczne i operatory relacji zwracają wartości PRAWDA lub FAŁSZ. W Atari Basic wartości logicznej FAŁSZ odpowiada wartość liczbowa zero, zaś wartości PRAWDA - jeden. Interpreter Basica zawiera następujące operatory relacji: mniejsze lub równe (<=), większe lub równe (>=), mniejsze (<), większe (>), różne (<>) i równe (=). Operatorami logicznymi są zaś AND (iloczyn logiczny), OR (alternatywa) i NOT (negacja). Wszystkie procedury wykonawcze tych operatorów są ze sobą ściśle powiązane, ponadto dołączona jest do nich procedura XSGN realizująca funkcję SGN.

```
0100 ;Execute compare operators
0110 ;
0120 AF1 =   $DA48
0130 CMPVAR = $AD11
0140 FR0 =   $D4
0150 FR1 =   $E0
0160 GETVAR = $ABE9
0170 GETVRS = $ABFD
0180 PUTVAR = $ABB2
0190 VART =  $D2
0200 ;
0210     *=  $ACA3
0220 ;
0230 ;eXecute Less or EQual
0240 ;
0250 XLEQ JSR CMPVAR
```

```
0260     BMI ONEP
0270     BEQ ONEP
0280     BPL ZERO
0290 ;
0300 ;eXecute Not EQual
0310 ;
0320 XNEQ JSR CMPVAR
0330     JMP RESLTZ
0340 ;
0350 ;eXecute LESs than
0360 ;
0370 XLES JSR CMPVAR
0380     BMI ONEP
0390     BPL ZERO
0400 ;
0410 ;eXecute GReaTer than
0420 ;
0430 XGRT JSR CMPVAR
0440     BMI ZERO
0450     BEQ ZERO
0460     BPL ONEP
0470 ;
0480 ;eXecute Greater or EQual
0490 ;
0500 XGEQ JSR CMPVAR
0510     BMI ZERO
0520     BPL ONEP
0530 ;
0540 ;eXecute EQUal
0550 ;
0560 XEQU JSR CMPVAR
0570     JMP RESLTO
0580 ;
0590 ;eXecute AND operator
0600 ;
0610 XAND JSR GETVRS
0620     LDA FR0
0630     AND FR1
0640     JMP RESLTZ
0650 ;
0660 ;eXecute OR operator
0670 ;
0680 XOR JSR GETVRS
0690     LDA FR0
0700     ORA FR1
0710 ;
0720 ;RESuLT Zero
0730 ;
0740 RESLTZ BEQ ZERO
0750     BNE ONEP
0760 ;
0770 ;eXecute NOT operator
0780 ;
0790 XNOT JSR GETVAR
0800     LDA FR0
0810 ;
0820 ;RESuLT One
0830 ;
0840 RESLTO BEQ ONEP
0850 ;
0860 ;result ZERO
```

```
0870 ;
0880 ZERO LDA #$00
0890     TAY
0900     BEQ STRES
0910 ;
0920 ;result ONE Positive
0930 ;
0940 ONEP LDA #$40
0950 ;
0960 ;result ONE Negative
0970 ;
0980 ONEN LDY #$01
0990 ;
1000 ;STore RESult
1010 ;
1020 STRES STA FR0
1030     STY FR0+1
1040     LDX #FR0+2
1050     LDY #$04
1060     JSR AF1
1070     STA VART
1080 END JMP PUTVAR
1090 ;
1100 ;eXecute SGN function
1110 ;
1120 XSGN JSR GETVAR
1130     LDA FR0
1140     BEQ END
1150     BPL ONEP
1160     LDA #$C0
1170     BMI ONEN
```

Określenie relacji między dwoma argumantami rozpoczyna się zawsze od wywołania procedury CMPVAR, która dokonuje porównania odpowiednich wartości, a wynik sygnalizuje w odpowiednich bitach statusu procesora. Zależnie od rodzaju porównania i stanu bitów Negative i Zero wykonywane są skoki do właściwych fragmentów procedury, gdzie ustalane są wartości dwóch pierwszych bajtów wyniku. Pozostałe bajty rejestru FR0 są zerowane przez procedurę AF1 i rezultat przepisywany jest do bufora przez procedurę PUTVAR.

Nieco inaczej przebiega obliczenie wyniku operacji logicznych. Procedury GETVRS (dla AND i OR) lub GETVAR (dla NOT) pobierają z bufora argumenty, na których pierwszym bajcie wykonywana jest odpowiednia operacja logiczna. Oznacza to, że operacje logiczne Atari Basic rozróżniają tylko dwa stany: zero i niezero. Opracowanie wyniku i jego zapisanie są analogiczne jak dla operatorów relacji.

W opisywanym zespole procedur znajduje się również procedura XSGN, realizująca funkcję SGN, która zwraca znak podanego argumentu. W zależności od wartości najstarszego bitu liczby pobranej przez GETVAR następuje skok do odpowiedniej etykiety i zapisanie wyniku: 0, 1 lub -1.

Samo porównanie wartości jest przeprowadzane przez procedurę CMPVAR. Najpierw odczytuje ona token operatora. Jeśli jest to token operacji na wartościach tekstowych, to wykonywany jest skok do procedury CMPSTR. W przeciwnym przypadku porównywane liczby są odczytywane z

bufora i, przez wywołanie BSUB, odejmowane od siebie. Znak wyniku tego działania jest umieszczany w akumulatorze i procedura się kończy.

```
0100 ;CoMPare VARiables
0110 ;
0120 BSUB =  $AD2C
0130 CMPSTR = $AF6C
0140 FR0 =   $D4
0150 GETVRS = $ABFD
0160 LOMEM = $80
0170 STIX =  $A9
0180 ;
0190     *=  $AD11
0200 ;
0210     LDY STIX
0220     DEY
0230     LDA (LOMEM),Y
0240     CMP #'/
0250     BCC NUM
0260     JMP CMPSTR
0270 NUM JSR GETVRS
0280 ;
0290 ;GET SUBtract
0300 ;
0310 GETSUB JSR BSUB
0320     LDA FR0
0330     RTS
```

W przypadku porównania ciągów tekstowych, najpierw - przez dwukrotne wywołanie procedury FTARV - odczytywane są informacje o nich. Następnie rozpoczyna się pętla, w której kolejno porównywany jest znak po znaku. Jeśli znaki są równe, to adresy kontrolowanych znaków są zwiększane i pętla się powtarza. Gdy znaki są różne, to wynik porównania jest zapisywany do akumulatora (jak w przypadku porównania liczb) i procedura się kończy. Pętla jest także przerywana, gdy wszystkie znaki są jednakowe, a został osiągnięty koniec jednego lub obu ciągów. Jeśli obu, to sygnalizowana jest ich równość. W przeciwnym przypadku ciąg dłuższy zostaje uznany za większy. Licznikiem jest tu długość ciągu zapisana w trzecim i czwartym bajcie rejestrów FR0 i FR1.

```
0100 ;CoMPare STRings
0110 ;
0120 FMOV01 = $DDB6
0130 FR0 =   $D4
0140 FR1 =   $E0
0150 FTARV = $AB90
0160 ;
0170     *=  $AF6C
0180 ;
0190 CMPSTR JSR FTARV
0200     JSR FMOV01
0210     JSR FTARV
0220 LOOP LDX #FR0+2
0230     JSR DECREG
0240     PHP
0250     LDX #FR1+2
0260     JSR DECREG
0270     BEQ CST
0280     PLP
```

```
0290     BEQ MNS
0300     LDY #$00
0310     LDA (FR0),Y
0320     CMP (FR1),Y
0330     BEQ IF0
0340     BCC MNS
0350 PLS LDA #$01
0360     RTS
0370 MNS LDA #$80
0380     RTS
0390 CST PLP
0400     BNE PLS
0410     RTS
0420 IF0 INC FR0
0430     BNE IF1
0440     INC FR0+1
0450 IF1 INC FR1
0460     BNE LOOP
0470     INC FR1+1
0480     BNE LOOP
0490 ;
0500 ;DECrease REGister
0510 ;
0520 DECREG LDA $00,X
0530     BNE SKIP
0540     LDA $01,X
0550     BEQ END
0560     DEC $01,X
0570 SKIP DEC $00,X
0580     TAY
0590 END RTS
```

5.3.2. Operatory przypisania

W interpreterze Atari Basic występują dwa operatory przypisania (oba oznaczone znakiem "="): przypisanie liczbowe i przypisanie tekstowe. Służą one odpowiednio do nadania wartości zmiennym liczbowym i tekstowym. Pomimo jednakowego symbolu mają one inne tokeny i są realizowane przez inne procedury.

Procedurą wykonawczą przypisania liczbowego jest XNLET. Przypisanie liczby musi być ostatnią operacją w obliczaniu wyrażenia. Dlatego najpierw sprawdzana jest wartość licznika STIX. Jeśli jest ona różna od $FF, to oznacza zmienną indeksowaną i konieczność obliczenia indeksów. W takim razie rejestr BHLD2 otrzymuje wartość $80 i procedura jest przerywana. W przeciwnym przypadku obie wartości są pobierane z bufora przez GETVRS, a następnie obliczona wartość liczby jest przepisywana do rejestru zmiennej liczbowej. Teraz XNLET kończy się skokiem do procedury SAVVAL.

```
0100 ;eXecute Number LET mark
0110 ;
0120 BHLD2 = $B1
0130 FR0 =   $D4
0140 FR1 =   $E0
0150 GETVRS = $ABFD
0160 SAVVAL = $AC0C
0170 STIX =  $A9
```

```
0180 ;
0190     *=  $AD4A
0200 ;
0210     LDA STIX
0220     CMP #$FF
0230     BNE EXIT
0240     JSR GETVRS
0250     LDX #$05
0260 LOOP LDA FR1,X
0270     STA FR0,X
0280     DEX
0290     BPL LOOP
0300     JMP SAVVAL
0310 EXIT LDA #$80
0320     STA BHLD2
0330     RTS
```

W tej procedurze według numeru zmiennej z rejestru VARN obliczany jest jej adres w tablicy wartości zmiennych (przez wywołanie CALPC). Cała zmienna (rejestry VARN, VART i FR0 - razem osiem bajtów) jest przepisywana w pętli do obliczonego miejsca i procedura się kończy.

```
0100 ;SAVe VALue
0110 ;
0120 CALPC = $AC1E
0130 PCNTC = $9D
0140 VARN =  $D3
0150 VART =  $D2
0160 ;
0170     *=  $AC0C
0180 ;
0190     LDA VARN
0200     JSR CALPC
0210     LDX #$00
0220 LOOP LDA VART,X
0230     STA (PCNTC),Y
0240     INY
0250     INX
0260     CPX #$08
0270     BCC LOOP
0280     RTS
```

Znacznie bardziej skomplikowana jest procedura XSLET realizująca operację przypisania tekstowego. Najpierw wywoływana jest procedura FTARV i odczytany przez nią adres ciągu jest umieszczany w rejestrze OLDMHI, a jego długość w rejestrze MRANGE. Jeśli jest to ostatni operator wyrażenia, to przez ponowne wywołanie FTARV odczytywana jest maksymalna długość ciągu. Jeśli nie, to wywołanie procedury PRCOP powoduje obliczenie początkowego adresu przypisania. Przekroczenie zadeklarowanej długości nie powoduje błędu, a nadmiar znaków jest po prostu pomijany. Następnie obliczany jest adres w tablicy STARP, pod którym powinny znaleźć się znaki ciągu, i odpowiedni obszar pamięci jest przepisywany przez procedurę MOVMEM. Jeżeli wszystkie czynności przebiegły poprawnie, to po zapisaniu nowej długości ciągu wykonywany jest skok do procedury SAVVAL. Próba przypisania wartości niezadeklarowanej zmiennej tekstowej powoduje natomiast ustawienie bitu Carry przed końcem procedury.

```
0100 ;eXecute String LET mark
0110 ;
```

```
0120 BHLD2 = $B1
0130 FR0 =   $D4
0140 FTARV = $AB90
0150 MOVMEM = $A944
0160 MRANGE = $A2
0170 NEWMHI = $9B
0180 OLDMHI = $99
0190 PRCOP = $AB04
0200 SAVVAL = $AC0C
0210 STARP = $8C
0220 STIX =  $A9
0230 VARBL = $AB81
0240 VARN =  $D3
0250 ZTEMP3 = $F9
0260 ;
0270     *=  $AE8E
0280 ;
0290     JSR FTARV
0300     LDA FR0
0310     STA OLDMHI
0320     LDA FR0+1
0330     STA OLDMHI+1
0340     LDA FR0+2
0350     STA MRANGE
0360     LDY FR0+3
0370     STY MRANGE+1
0380     LDY STIX
0390     CPY #$FF
0400     BEQ MAX
0410     LDA #$80
0420     STA BHLD2
0430     JSR PRCOP
0440     LDA FR0+3
0450     LDY FR0+2
0460     ROL BHLD2
0470     BCS BPS
0480 MAX JSR FTARV
0490     LDA FR0+5
0500     LDY FR0+4
0510 BPS CMP MRANGE+1
0520     BCC STR
0530     BNE ADD
0540     CPY MRANGE
0550     BCS ADD
0560 STR STA MRANGE+1
0570     STY MRANGE
0580 ADD CLC
0590     LDA FR0
0600     ADC MRANGE
0610     TAY
0620     LDA FR0+1
0630     ADC MRANGE+1
0640     TAX
0650     SEC
0660     TYA
0670     SBC STARP
0680     STA ZTEMP3
0690     TXA
0700     SBC STARP+1
0710     STA ZTEMP3+1
0720     SEC
```

```
0730     LDA #$00
0740     SBC MRANGE
0750     STA MRANGE
0760     SEC
0770     LDA OLDMHI
0780     SBC MRANGE
0790     STA OLDMHI
0800     LDA OLDMHI+1
0810     SBC $00
0820     STA OLDMHI+1
0830     SEC
0840     LDA FR0
0850     SBC MRANGE
0860     STA NEWMHI
0870     LDA FR0+1
0880     SBC #$00
0890     STA NEWMHI+1
0900     JSR MOVMEM
0910     LDA VARN
0920     JSR VARBL
0930     SEC
0940     LDA ZTEMP3
0950     SBC FR0
0960     TAY
0970     LDA ZTEMP3+1
0980     SBC FR0+1
0990     TAX
1000     LDA #$02
1010     AND BHLD2
1020     BEQ SAVE
1030     LDA #$00
1040     STA BHLD2
1050     CPX FR0+3
1060     BCC EXIT
1070     BNE SAVE
1080     CPY FR0+2
1090     BCS SAVE
1100 EXIT RTS
1110 SAVE STY FR0+2
1120     STX FR0+3
1130     JMP SAVVAL
```

5.3.3. Operatory nawiasów

Ważną rolę w wyrażeniach pełnią nawiasy. Służą one do wydzielenia indeksów w instrukcjach deklaracji zmiennych tablicowych (indeksowych i tekstowych) oraz przypisania wartości tym zmiennym. Także w tym przypadku jednakowym oznaczeniom odpowiadają różne tokeny i procedury wykonawcze.

Napotkanie tokena oznaczającego nawias zmiennej indeksowanej powoduje wywołanie procedury XPIXV. Niemal identyczny jest w początkowym fragmencie przebieg procedury wywoływanej po rozpoznaniu tokena nawiasu w instrukcji deklarującej zmienną tablicową - XPDIM.

W przypadku deklaracji na początku procedury XPDIM w rejestrze BHLD2 umieszczana jest

wartość $40, która służy później do rozpoznania rodzaju nawiasu. Następnie z bufora pobierane są przy pomocy GETNUM liczby określające wymiar lub indeks zmiennej. Jeśli przy tym zawartość rejestru BHLD1 jest różna od zera, to są to dwie liczby, a gdy jest równa zero, to tylko jedna. Do rejestru CSTAD wpisywany jest pierwszy wymiar (jeśli istnieje) lub zero. Drugi wymiar przepisywany jest zawsze do rejestru ZTEMP1. Przekroczenie przez jeden z wymiarów wartości $7FFF powoduje sygnalizację błędu przez skok do procedury DIMER (DIMension ERror). Następnie odczytywana jest nazwa zmiennej (przez wywołanie GETVAR). Teraz możliwe są trzy sytuacje. Jeżeli jest to deklaracja, to procedura się kończy przez RTS. Jeżeli jest to nawias indeksu zmiennej, która nie była przedtem deklarowana (skasowany bit 0 rejestru VART), to sygnalizowany jest błąd. Dalsze kontynuowanie procedury następuje jedynie w przypadku nawiasu zmiennej indeksowanej, która została wcześniej deklarowana.

```
0100 AUXBR = $AF
0110 BHLD1 = $B0
0120 BHLD2 = $B1
0130 CSTAD = $97
0140 DIMER = $B922
0150 EVZTMP = $AF48
0160 FR0 =   $D4
0170 GETNUM = $ABDA
0180 GETVAR = $ABE9
0190 PUTVAR = $ABB2
0200 STARP = $8C
0210 STPTR = $AA
0220 VART =  $D2
0230 ZT1ADD = $AF3D
0240 ZT1ML6 = $AF31
0250 ZTEMP1 = $F5
0260 ;
0270     *=  $AD6D
0280 ;
0290 ;eXecute Parenthese of DIMension
0300 ;
0310 XPDIM LDA #$40
0320     STA BHLD2
0330 ;
0340 ;eXecute Parenthese of IndeX Variable
0350 ;
0360 XPIXV BIT BHLD2
0370     BPL GET1
0380     LDA STPTR
0390     STA AUXBR
0400     DEC STPTR
0410 GET1 LDA #$00
0420     TAY
0430     CMP BHLD1
0440     BEQ GET2
0450     DEC BHLD1
0460     JSR GETNUM
0470     LDA FR0+1
0480     BMI ERR
0490     LDY FR0
0500 GET2 STA CSTAD+1
0510     STY CSTAD
0520     JSR GETNUM
0530     LDA FR0
0540     STA ZTEMP1
```

```
0550     LDA FR0+1
0560     BMI ERR
0570     STA ZTEMP1+1
0580     JSR GETVAR
0590     BIT BHLD2
0600     BVC CVT
0610     LDA #$00
0620     STA BHLD2
0630     RTS
0640 CVT ROR VART
0650     BCS FDIM
0660 ERR JSR DIMER
0670 FDIM LDA ZTEMP1+1
0680     CMP FR0+3
0690     BCC SDIM
0700     BNE ERR
0710     LDA ZTEMP1
0720     CMP FR0+2
0730     BCS ERR
0740 SDIM LDA CSTAD+1
0750     CMP FR0+5
0760     BCC MEM
0770     BNE ERR
0780     LDA CSTAD
0790     CMP FR0+4
0800     BCS ERR
0810 MEM JSR EVZTMP
0820     LDA CSTAD
0830     LDY CSTAD+1
0840     JSR ZT1ADD
0850     JSR ZT1ML6
0860     LDA FR0
0870     LDY FR0+1
0880     JSR ZT1ADD
0890     LDA STARP
0900     LDY STARP+1
0910     JSR ZT1ADD
0920     BIT BHLD2
0930     BPL SAV
0940     LDA AUXBR
0950     STA STPTR
0960     JSR GETVAR
0970     LDY #$05
0980 NXT1 LDA FR0,Y
0990     STA (ZTEMP1),Y
1000     DEY
1010     BPL NXT1
1020     INY
1030     STY BHLD2
1040     RTS
1050 SAV LDY #$05
1060 NXT2 LDA (ZTEMP1),Y
1070     STA FR0,Y
1080     DEY
1090     BPL NXT2
1100     INY
1110     STY VART
1120     JMP PUTVAR
```

Odczytanie odpowiedniej wartości zmiennej indeksowanej rozpoczyna się od porównania podanych wymiarów z wymiarami zadeklarowanymi. Przekroczenie tych ostatnich powoduje także skok do DIMER. Jeśli dotąd wszystko przebiegło poprawnie, to wywoływana jest procedura EVZTMP, która przelicza zawartość rejestru ZTEMP1, aby umożliwić odszukanie w pamięci właściwego elementu.

```
0100 ;EValuate ZTEMP registers
0110 ;
0120 FRE =   $DA
0130 ZTEMP1 = $F5
0140 ZTEMP2 = $F7
0150 ZTEMP3 = $F9
0160 ;
0170     *=  $AF48
0180 ;
0190     LDA #$00
0200     STA ZTEMP2
0210     STA ZTEMP2+1
0220     LDY #$10
0230 LOOP LDA ZTEMP1
0240     LSR A
0250     BCC SKIP
0260     CLC
0270     LDX #$FE
0280 NXT1 LDA ZTEMP3,X
0290     ADC FRE,X
0300     STA ZTEMP3,X
0310     INX
0320     BNE NXT1
0330 SKIP LDX #$03
0340 NXT2 ROR ZTEMP1,X
0350     DEX
0360     BPL NXT2
0370     DEY
0380     BNE LOOP
0390     RTS
```

Następnie, przez kolejne wywołania procedur ZT1ML6 i ZT1ADD z różnymi parametrami w akumulatorze i rejestrze Y, adres elementu zmiennej jest obliczany jako suma adresu w tablicy wartości tej zmiennej, indeksu tablicy od początku tablicy STARP oraz adresu początkowego STARP.

```
0100 ZTEMP1 = $F5
0110 ;
0120     *=  $AF31
0130 ;
0140 ;ZTEMP1 MuLtiple by 6
0150 ;
0160 ZT1ML6 ASL ZTEMP1
0170     ROL ZTEMP1+1
0180     LDY ZTEMP1+1
0190     LDA ZTEMP1
0200     ASL ZTEMP1
0210     ROL ZTEMP1+1
0220 ;
0230 ;ZTEMP1 ADDition
0240 ;
0250 ZT1ADD CLC
```

```
0260     ADC ZTEMP1
0270     STA ZTEMP1
0280     TYA
0290     ADC ZTEMP1+1
0300     STA ZTEMP1+1
0310     RTS
```

Gdy element został już odszukany, to dalsze czynności zależą od zawartości rejestru BHLD2. Wartość ujemna oznacza zapis elementu, a dodatnia jego odczyt. W pierwszym przypadku liczba z rejestru FR0 jest przepisywana pod wskazany adres i procedura kończy się przez RTS. Podobnie w drugim przypadku liczba przepisywana jest z odnalezionego miejsca do rejestru FR0, lecz na końcu następuje skok do procedury PUTVAR, która zapisuje uzyskany wynik do bufora.

Token nawiasu wyrażenia tekstowego powoduje wywołanie procedury XPSEX. Znaczna jej część jest przeznaczona na sprawdzenie zgodności wymiarów użytych ciągów tekstowych. Sposób porównania jest przy tym regulowany przez rejestry BHLD1 i BHLD2. Pierwszy z nich zawiera liczbę wartości użytych do określenia fragmentu ciągu, a drugi rodzaj operacji - zapis czy odczyt (jak w XPIXV). Przy zapisie podana długość ciągu jest porównywana z zadeklarowaną (piąty i szósty bajt rejestru FR0), a przy odczycie - z długością aktualną (trzeci i czwarty bajt FR0). Jeśli długość ciągu okaże się niepoprawna, to błąd jest sygnalizowany przez procedurę SLENER (String LENgth ERror).

Na zakończenie procedura FTARV odczytuje adres ciągu w pamięci. Po dodaniu adresu elementu w ciągu otrzymany bezwzględny adres jest zapisywany do FR0. Przy odczycie jest to już koniec, a przy zapisie następuje skok do PUTVAR, gdzie współrzędne elementu są przepisywane do bufora.

```
0100 ;eXecute Parenthesis of String Expression
0110 ;
0120 BHLD1 = $B0
0130 BHLD2 = $B1
0140 CSTAD = $97
0150 FR0 =   $D4
0160 FTARV = $AB90
0170 GETSLN = $AE81
0180 GETVAR = $ABE9
0190 PUTVAR = $ABB2
0200 SLENER = $B92A
0210 ZTEMP1 = $F5
0220 ;
0230     *=  $AE11
0240 ;
0250     LDA BHLD1
0260     BEQ GLN
0270     JSR GETSLN
0280     STY CSTAD+1
0290     STA CSTAD
0300 GLN JSR GETSLN
0310     SEC
0320     SBC #$01
0330     STA ZTEMP1
0340     TYA
0350     SBC #$00
0360     STA ZTEMP1+1
```

```
0370     JSR GETVAR
0380     LDA BHLD2
0390     BPL SCD
0400     ORA BHLD1
0410     STA BHLD2
0420     LDY FR0+5
0430     LDA FR0+4
0440     JMP CONT
0450 SCD LDA FR0+2
0460     LDY FR0+3
0470 COND LDX BHLD1
0480     BEQ SBZ
0490     DEC BHLD1
0500     CPY CSTAD+1
0510     BCC ERR
0520     BNE RAD
0530     CMP CSTAD
0540     BCC ERR
0550 RAD LDY CSTAD+1
0560     LDA CSTAD
0570 SBZ SEC
0580     SBC ZTEMP1
0590     STA FR0+2
0600     TAX
0610     TYA
0620     SBC ZTEMP1+1
0630     STA FR0+3
0640     BCC ERR
0650     TAY
0660     BNE ADZ
0670     TXA
0680     BEQ ERR
0690 ADZ JSR FTARV+3
0700     CLC
0710     LDA FR0
0720     ADC ZTEMP1
0730     STA FR0
0740     LDA FR0+1
0750     ADC ZTEMP1+1
0760     STA FR0+1
0770     BIT BHLD2
0780     BPL PUT
0790     RTS
0800 PUT JMP PUTVAR
0810 ERR JSR SLENER
```

W opisanej powyżej procedurze odczyt indeksu długości ciągu jest wykonywany przez pomocniczą procedurę GETSLN. Przez wywołanie GETNUM pobiera ona z bufora liczbę całkowitą i sprawdza jej wartość. Jeśli liczba ta jest równa zero (oba bajty zerowe), to przez skok do SLENER sygnalizowany jest błąd.

```
0100 ;GET String LeNgth
0110 ;
0120 FR0 =   $D4
0130 GETNUM = $ABDA
0140 ;
0150     *=  $AE81
0160 ;
0170     JSR GETNUM
0180     LDA FR0
```

```
0190     LDY FR0+1
0200     BNE END
0210     TAX
0220     BEQ $AE7E   ;JSR SLENER
0230 END RTS
```

Ostatnia procedura jest wspólna dla przecinka oddzielającego indeksy zmiennej tablicowej (XCOM) oraz dla nawiasu zamykającego (XEPAR). Zdejmuje ona jedynie ze stosu dwie wartości (adres powrotny) i kończy się skokiem do procedury PRCOP wywołującej realizacyjną procedurę operatora lub funkcji. W przypadku przecinka zwiększana jest jeszcze zawartość rejestru BHLD1, który określa liczbę wartości użytych dla wskazania indeksu zmiennej. Należy tu zwrócić uwagę, że procedura XCOM jest wykonywana TYLKO dla przecinka w indeksie zmiennej tablicowej, a nie dla przecinków zastosowanych w innych miejscach.

```
0100 BHLD1 = $B0
0110 PRCOP = $AB04
0120 STIX =  $A9
0130 ;
0140     *=  $AD64
0150 ;
0160 ;eXecute index COMma
0170 ;
0180 XCOM INC BHLD1
0190 ;
0200 ;eXecute End PARenthesis
0210 ;
0220 XEPAR LDY STIX
0230     PLA
0240     PLA
0250     JMP PRCOP
```

Rozdział 6

PROCEDURY WYKONAWCZE INSTRUKCJI

Opisywana w rozdziale 2 procedura EXSTM w celu wykonania instrukcji Basica pobiera z tablicy wektorów STVTAB adres odpowiedniej procedury realizacyjnej, a następnie wywołuje ją. Po opisie wszystkich pozostałych elementów programu nadeszła więc pora na przedstawienie procedur wykonawczych poszczególnych instrukcji zaimplementowanych w interpreterze Atari Basic. Procedury te zostały podzielone na grupy w zależności od funkcji spełnianych w programie przez te instrukcje.

6.1. Instrukcje kontroli programu

Instrukcjami kontroli programu nazywamy takie, które sterują bezpośrednio pracą interpretera, a także realizacją programu. Należą do nich instrukcje: NEW, BYE, DOS, END, STOP, CONT, RUN i CLR.

6.1.1. Instrukcja NEW

Instrukcja NEW kasuje program w Basicu zawarty w pamięci komputera oraz wszystkie wykorzystywane przez niego zmienne i ich wartości. Jest ona częścią składową procedury zimnego startu CDST i została opisana razem z nią. Analiza tej procedury ujawnia, że wszystkie dane programu po jej wykonaniu nadal pozostają w pamięci. Skasowane zostają tylko wskazujące je wektory. Odzyskanie w całości skasowanego programu jest jednak niemożliwe, gdyż wpisanie kolejnej instrukcji zniszczy jego część. Pewne fragmenty można wszakże odtworzyć.

6.1.2. Instrukcja BYE

Instrukcja BYE powoduje zakończenie pracy interpretera i przejście do wbudowanego programu testującego. Jej procedura wykonawcza - XBYE - składa się tylko z wywołania procedury CLALL, która zamyka wszystkie kanały IOCB (oprócz edytora) i wyłącza dźwięk oraz skoku do programu testującego poprzez tablicę skoków JMPTAB.

```
0100 ;eXecute BYE command
0110 ;
0120 CLALL = $BD45
0130 JTSTROM = $E471
0140 ;
0150     *=  $A9E6
0160 ;
0170     JSR CLALL
0180     JMP JTSTROM
```

6.1.3. Instrukcja DOS

Także instrukcja DOS przerywa pracę interpretera, lecz sterowanie komputerem przekazuje do programu, którego adres zapisany jest w rejestrze DOSVEC. Przy pracy ze stacją dysków znajduje

się tam adres DOS-u (zob. "Mapa pamięci Atari XL/XE. Dyskowe systemy operacyjne"). Jeśli nie ma stacji, to rejestr DOSVEC zawiera adres programu testującego, więc działanie instrukcji DOS jest analogiczne jak BYE. Procedura wykonawcza - XDOS - jest również zbliżona do XBYE, a różni się tylko końcowym skokiem.

```
0100 ;eXecute DOS command
0110 ;
0120 CLALL = $BD45
0130 DOSVEC = $0A
0140 ;
0150     *=  $A9EC
0160 ;
0170     JSR CLALL
0180     JMP (DOSVEC)
```

6.1.4. Instrukcja END

Instrukcja END przerywa działanie programu, a następnie zamyka wszystkie kanały dźwięku i komunikacji z urządzeniami zewnętrznymi. Jest to realizowane w procedurze XEND przez skok do gorącego startu interpretera. Przedtem jednak przez wywołanie procedury SAVCLN w rejestrze STOPLN (STOP LiNe number) zapamiętywany jest numer aktualnego wiersza programu.

```
0100 ;eXecute END statement
0110 ;
0120 SAVCLN = $B7A6
0130 WRMST = $A050
0140 ;
0150     *=  $B78C
0160 ;
0170     JSR SAVCLN
0180     JMP WRMST
```

6.1.5. Instrukcja STOP

Instrukcja STOP także przerywa działanie programu, jednak nie wyłącza dźwięku i nie zamyka kanałów I/O. Efekt jej działania jest niemal identyczny z efektem wystąpienia błędu w programie. Procedura wykonawcza tej instrukcji - XSTOP - jest także wywoływana po naciśnięciu klawisza BREAK.

Realizacja instrukcji STOP rozpoczyna się od zapamiętania w rejestrze STOPLN aktualnego numeru wiersza (przez SAVCLN) oraz wyświetlenia na ekranie znaku końca wiersza RETURN (przez PRTRET). Następnie w rejestrze POKADR umieszczany jest adres komunikatu "STOPPED", który jest wyświetlany po wywołaniu procedury PUTTXT. Dalsza część komunikatu (numer wiersza w trybie programowym lub tylko RETURN w trybie bezpośrednim) jest wyświetlana po przekazaniu sterowania do wnętrza procedury obsługi błędu poprzez skok do etykiety DSTMSG.

```
0100 ;eXecute STOP statement
0110 ;
0120 DSTMSG = $B968
0130 POKADR = $95
0140 PRTRET = $BD79
```

```
0150 PUTTXT = $B567
0160 SAVCLN = $B7A6
0170 STMSG = $A5FD
0180 ;
0190     *=  $B792
0200 ;
0210     JSR SAVCLN
0220     JSR PRTRET
0230     LDA # <STMSG
0240     STA POKADR
0250     LDA # >STMSG
0260     STA POKADR+1
0270     JSR PUTTXT
0280     JMP DSTMSG
```

Tekst komunikatu o przerwaniu pracy programu jest umieszczony w pamięci za tablicą nazw instrukcji STNAME i oznaczony etykietą STMSG.

```
0100 ;STopped MeSsaGe
0110 ;
0120     *=  $A5FD
0130 ;
0140     .CBYTE "STOPPED "
```

6.1.6. Instrukcja CONT

Wznowienie realizacji przerwanego programu jest wykonywane przez instrukcję CONT, a właściwie jej procedurę - XCONT. Instrukcja CONT podana w trybie programowym, co jest sprawdzane przez pierwsze wywołanie GHISTM, powoduje jedynie ustawienie rejestrów kanału I/O na kanał edytora (poprzez RSTCHN). W trybie bezpośrednim z rejestru STOPLN numer wiersza jest przepisywany do CLNN (Current LiNe Number) i procedura FNDCST odszukuje ten wiersz. Jeśli jest on w trybie bezpośrednim, to procedura jest przerywana skokiem do gorącego startu interpretera.

```
0100 ;eXecute CONT statement
0110 ;
0120 CLNN =  $A0
0130 FNDCST = $A9A2
0140 GHISTM = $A9E1
0150 GLNLEN = $A9DC
0160 GSTMLN = $B819
0170 NXTSTM = $A9D0
0180 STOPLN = $BA
0190 ;
0200     *=  $B7B5
0210 ;
0220     JSR GHISTM
0230     BPL $B7B2   ;JMP RSTCHN
0240     LDA STOPLN
0250     STA CLNN
0260     LDA STOPLN+1
0270     STA CLNN+1
0280     JSR FNDCST
0290     JSR GHISTM
0300     BMI $B775   ;JMP WRMST
0310     JSR GLNLEN
0320     JSR NXTSTM
```

```
0330     JSR GHISTM
0340     BMI $B775   ;JMP WRMST
0350     JMP GSTMLN
```

Po odnalezieniu aktualnego wiersza programu odczytywana jest przy pomocy GLNLEN jego długość i procedura NXTSTM znajduje następny wiersz. Jeżeli jest on w trybie bezpośrednim, to także wykonywany jest skok do WRMST. W przeciwnym przypadku następuje skok do GSTMLN i realizacja programu jest kontynuowana. Jeśli zatrzymanie programu nastąpiło w środku wiersza, to dalsze instrukcje w tym wierszu NIE będą wykonane.

6.1.7. Instrukcje CLR i RUN

Instrukcja CLR kasuje deklaracje zmiennych tablicowych oraz zeruje pozostałe zmienne wykorzystywane w programie. Ponadto usuwa ze stosu bieżącego wszelkie informacje o procedurach i pętlach FOR/NEXT. Instrukcja RUN powoduje natomiast uruchomienie programu znajdującego się w pamięci komputera lub odczytanie programu z urządzenia zewnętrznego i jego uruchomienie. Procedura wykonawcza instrukcji CLR (oznaczona etykietą XCLR) jest częścią składową procedury XRUN, która realizuje instrukcję RUN. Z tego powodu zostały one opisane łącznie.

```
0100 ;eXecute RUN statement
0110 ;
0120 CLNN =  $A0
0130 CLRVV = $B8B9
0140 DATAD = $B6
0150 DATALN = $B7
0160 FSTGLN = $B816
0170 GETIX = $B904
0180 GHISTM = $A9E1
0190 LDPRGM = $BAF7
0200 RSMEMT = $B8A8
0210 RSTBRG = $B8F1
0220 WRMST = $A050
0230 ;
0240     *=  $B74C
0250 ;
0260 XRUN JSR GETIX
0270     BCS RUN
0280     JSR LDPRGM
0290 RUN NOP
0300     LDA #$00
0310     STA CLNN
0320     STA CLNN+1
0330     JSR FSTGLN
0340     JSR GHISTM
0350     BMI END
0360     JSR RSTBRG
0370 ;
0380 ;eXecute CLR statement
0390 ;
0400 XCLR JSR CLRVV
0410     JSR RSMEMT
0420     LDA #$00
0430     STA DATALN
0440     STA DATALN+1
0450     STA DATAD
```

```
0460     RTS
0470 END JMP WRMST
```

Najpierw rozpoznawany jest rodzaj instrukcji RUN, to znaczy, czy program do uruchomienia znajduje się już w pamięci, czy musi być najpierw odczytany. W tym celu porównywane są przez procedurę GETIX indeksy wejściowy (INIX) i wyjściowy (OUTIX). Jeśli indeks wejściowy jest większy, to wywoływana jest procedura LDPRGM (jest ona fragmentem XLOAD), która odczytuje program ze wskazanego urządzenia.

```
0100 ;GET IndeX
0110 ;
0120 INIX =  $A8
0130 OUTIX = $A7
0140 ;
0150     *=  $B904
0160 ;
0170     LDX INIX
0180     INX
0190     CPX OUTIX
0200     RTS
```

Sposób uruchomienia programu jest bardzo prosty. Najpierw zerowany jest rejestr CLNN, a następnie procedura FSTGLN odszukuje wiersz o najniższym numerze. Jeśli jest to wiersz w trybie bezpośrednim (numer $8000), to procedura jest przerywana skokiem do WRMST. Jeżeli nie, to kanały I/O są zamykane, a rejestry Basica zerowane przez procedurę RSTBRG.

Teraz rozpoczyna się procedura XCLR. Na początku zerowane są wszystkie zmienne liczbowe oraz informacje o zmiennych tekstowych i tablicowych. Wykonuje to procedura CLRVV, która wypełnia zerami prawie całą tablicę wartości zmiennych. Nienaruszone pozostają tylko bajty określające numery i typy zmiennych.

```
0100 ;CLeaR Variable Value
0110 ;
0120 STMTAB = $88
0130 VVTP =  $86
0140 ZTEMP1 = $F5
0150 ;
0160     *=  $B8B9
0170 ;
0180     LDX VVTP
0190     STX ZTEMP1
0200     LDY VVTP+1
0210     STY ZTEMP1+1
0220 LOOP LDX ZTEMP1+1
0230     CPX STMTAB+1
0240     BCC EXEC
0250     LDX ZTEMP1
0260     CPX STMTAB
0270     BCC EXEC
0280     RTS
0290 EXEC LDY #$00
0300     LDA (ZTEMP1),Y
0310     AND #$FE
0320     STA (ZTEMP1),Y
0330     LDY #$02
```

```
0340     LDX #$06
0350     LDA #$00
0360 NEXT STA (ZTEMP1),Y
0370     INY
0380     DEX
0390     BNE NEXT
0400     LDA ZTEMP1
0410     CLC
0420     ADC #$08
0430     STA ZTEMP1
0440     LDA ZTEMP1+1
0450     ADC #$00
0460     STA ZTEMP1+1
0470     BNE LOOP
```

Kolejnym krokiem jest skasowanie tablicy zmiennych tablicowych i tekstowych oraz stosu bieżącego. Ta operacja jest wykonywana przez procedurę RSMEMT. Polega ona na przepisaniu wektora STARP (STring and ARray Pointer) do wektorów RUNSTK (RUNtime STacK), BMEMHI (Basic MEMory HIgh) oraz APPMHI (APPlication Memory HIgh - wykorzystywany przez system operacyjny).

```
0100 ;ReStore MEMory Top
0110 ;
0120 APPMHI = $0E
0130 BMEMHI = $90
0140 RUNSTK = $8E
0150 STARP = $8C
0160 ;
0170     *=  $B8A8
0180 ;
0190     LDA STARP
0200     STA RUNSTK
0210     STA BMEMHI
0220     STA APPMHI
0230     LDA STARP+1
0240     STA RUNSTK+1
0250     STA BMEMHI+1
0260     STA APPMHI+1
0270     RTS
```

Ostatnim etapem jest odtworzenie stanu początkowego liczników danych zapisanych w instrukcjach DATA. Uzyskuje się to przez wpisanie zer do rejestrów DATAD (DATa ADdress) i DATALN (DATA LiNe number).

Uruchomienie programu następuje zawsze od wiersza o najmniejszym numerze. Jest to jednak możliwe do ominięcia przy pomocy krótkiej procedury maszynowej. Musi ona umieścić w rejestrze CLNN numer wiersza, od którego będzie rozpoczęte wykonywanie programu, a następnie wykonać skok do wnętrza XRUN do adresu $B75B.

6.2. Instrukcje pomocnicze

Kolejna grupa instrukcji pełni rolę pomocniczą, lecz prawidłowe działanie programu jest bez nich niemożliwe lub przynajmniej utrudnione. Znajdują się tu także instrukcje, których nie można zaliczyć do żadnej z pozostałych grup. Są to: REM, DATA, DEG, RAD, RESTORE, DIM i POKE.

6.2.1. Instrukcje REM i DATA

Instrukcja REM zawiera komentarz wpisywany przez programistę w celu zwiększenia przejrzystości programu lub podania istotnych informacji. Natomiast w instrukcji DATA są umieszczane dane dla instrukcji READ. Obie te instrukcje są w trakcie wykonywania programu pomijane przez interpreter. Procedura wykonawcza tych instrukcji ogranicza się więc tylko do rozkazu RTS. Rozkaz ten znajduje się w procedurze GHISTM, która została opisana w rozdziale 2.

6.2.2. Instrukcje DEG i RAD

Instrukcje DEG i RAD ustalają podstawę do obliczeń funkcji trygonometrycznych. DEG powoduje traktowanie wszystkich argumentów kątowych, jako podanych w stopniach, zaś RAD - jako podanych w radianach. Odpowiednio procedury wykonawcze tych instrukcji umieszczają w rejestrze RADFLG (RADians FLaG) wartości sześć lub zero.

```
0100 RADFLG = $FB
0110 ;
0120     *=  $B28D
0130 ;
0140 ;eXecute DEG statement
0150 ;
0160 XDEG LDA #$06
0170     BNE EXE
0180 ;
0190 ;eXecute RAD statement
0200 ;
0210 XRAD LDA #$00
0220 EXE STA RADFLG
0230     RTS
```

6.2.3. Instrukcja RESTORE

Instrukcja RESTORE ustawia znacznik odczytu danych z instrukcji DATA na początek wiersza, którego numer został podany jako argument. Jeśli instrukcja nie ma argumentu, to znacznik jest ustawiany na pierwszy wiersz programu. Operacja ta jest przeprowadzana przez procedurę XRSTR. W każdym przypadku najpierw zerowany jest rejestr DATAD, co ustawia znacznik na pierwszej danej w odpowiedniej instrukcji DATA.

```
0100 ;eXecute RESTORE statement
0110 ;
0120 DATAD = $B6
0130 DATALN = $B7
0140 FR0 =   $D4
0150 GETIX = $B904
0160 GLNNUM = $ABCD
0170 ;
0180     *=  $B296
0190 ;
0200     LDA #$00
0210     STA DATAD
0220     JSR GETIX
0230     BCC GLN
0240     TAY
0250     BEQ SLN
0260 GLN JSR GLNNUM
```

```
0270     LDA FR0+1
0280     LDY FR0
0290 SLN STA DATALN+1
0300     STY DATALN
0310     RTS
```

Następnie przez wywołanie procedury GETIX sprawdzane są indeksy (podobnie jak w XRUN). Jeśli instrukcja posiada argument, to zostaje on odczytany przez procedurę GLNNUM i przepisany do rejestru DATALN. W przeciwnym przypadku rejestr DATALN jest zerowany.

```
0100 ;Get LiNe NUMber
0110 ;
0120 FR0 =   $D4
0130 LETNUM = $ABD7
0140 LINNER = $B926
0150 ;
0160     *=  $ABCD
0170 ;
0180     JSR LETNUM
0190     LDA FR0+1
0200     BPL $ABC9   ;RTS
0210     JMP LINNER
```

Procedura GLNNUM wykorzystuje do odczytu dwubajtowej wartości procedurę LETNUM. Po sprawdzeniu, czy uzyskany wynik nie przekracza dopuszczalnej wartości ($7FFF), GLNNUM się kończy. Jeśli jednak wartość nie jest poprawna, to przez skok do procedury LINNER sygnalizowany jest błąd (LINe Number ERror).

6.2.4. Instrukcja DIM

Instrukcja DIM służy do deklarowania wymiarów zmiennych tablicowych i tekstowych. Atari Basic wymaga zadeklarowania wszystkich takich zmiennych przed ich użyciem. Do realizacji tej instrukcji przeznaczona jest procedura XDIM.

Ponieważ instrukcja DIM może zawierać deklaracje kilku zmiennych, to procedura XDIM przebiega w pętli, która jest sterowana licznikami INIX i OUTIX. Każde przejście pętli rozpoczyna się od odczytania typu zmiennej. Jeśli była ona wcześniej deklarowana (ustawiony bit 0 rejestru VART), to błąd sygnalizowany jest przez skok do DIMER.

```
0100 ;eXecute DIM statement
0110 ;
0120 CSTAD = $97
0130 DIMER = $B922
0140 EVZTMP = $AF48
0150 FR0 =   $D4
0160 INIX =  $A8
0170 INSEL = $A87A
0180 OUTIX = $A7
0190 RUNSTK = $8E
0200 SAVVAL = $AC0C
0210 STARP = $8C
0220 VART =  $D2
0230 XLET =  $AADA
0240 ZFR0 =  $DA44
```

```
0250 ZT1ML6 = $AF31
0260 ZTEMP1 = $F5
0270 ;
0280     *=  $B206
0290 ;
0300 XDIM LDY INIX
0310     CPY OUTIX
0320     BCC EXE
0330     RTS
0340 EXE JSR XLET
0350     LDA VART
0360     ROR A
0370     BCC NUM
0380 ERR JSR DIMER
0390 NDM SEC
0400     ROL A
0410     STA VART
0420     BMI STRNG
0430     LDY ZTEMP1
0440     LDX ZTEMP1+1
0450     INY
0460     BNE COL
0470     INX
0480     BMI ERR
0490 COL STY FR0+2
0500     STX FR0+3
0510     STY ZTEMP1
0520     STX ZTEMP1+1
0530     LDY CSTAD
0540     LDX CSTAD+1
0550     INY
0560     BNE ROW
0570     INX
0580     BMI ERR
0590 ROW STY FR0+4
0600     STX FR0+5
0610     JSR EVZTMP
0620     JST ZT1ML6
0630     LDY ZTEMP1
0640     LDA ZTEMP1+1
0650     BMI ERR
0660     BPL INS
0670 STRNG LDA #$00
0680     STA FR0+2
0690     STA FR0+3
0700     LDY ZTEMP1
0710     STY FR0+4
0720     LDA ZTEMP1+1
0730     STA FR0+5
0740     BNE INS
0750     CPY #$00
0760     BEQ ERR
0770 INS LDX #RUNSTK
0780     JSR INSEL+2
0790     SEC
0800     LDA CSTAD
0810     SBC STARP
0820     STA FR0
0830     LDA CSTAD+1
0840     SBC STARP+1
0850     STA FR0+1
```

```
0860     JSR SAVVAL
0870     JMP XDIM
```

Po ustawieniu najmłodszego bitu rejestru VART sprawdzany jest jego najstarszy bit, który wskazuje typ zmiennej. Zależnie od jego stanu wybierany jest odpowiedni wariant procedury (skasowany - zmienna liczbowa indeksowana czyli zmienna tablicowa, ustawiony - zmienna tekstowa).

W przypadku zmiennej liczbowej indeksowanej jej wymiary są odczytywane z rejestrów ZTEMP1 oraz CSTAD i po zwiększeniu o jeden umieszczane w odpowiednich bajtach rejestru FR0. Zwiększenie jest konieczne, gdyż elementy zmiennej są numerowane od zera. Na tym etapie ponownie (uczyniła to już procedura XPDIM) sprawdzane jest, czy wartości wymiarów są mniejsze od $8000. Etap ten kończy się obliczeniem obszaru pamięci niezbędnego dla zapisania wszystkich elementów zmiennej.

Dla zmiennej tekstowej procedura jest nieco prostsza, gdyż może ona mieć tylko jeden wymiar. Jako druga dana wpisywana jest aktualna długość ciągu, a więc zero. Ponadto nie trzeba obliczać przestrzeni zajmowanej przez zmienną.

Ostatnia faza pętli jest również wspólna dla zmiennych obu typów. Przez wywołanie procedury INSEL rezerwowany jest w tablicy STARP obszar pamięci dla zmiennej. Aktualne parametry zmiennej są zapisywane w tablicy wartości (VVT) przez procedurę SAVVAL. Ponieważ procedura INSEL (zob. rozdział 1) nie zeruje rezerwowanego obszaru, to odczyt elementu zmiennej liczbowej da zupełnie przypadkowy wynik. Próba odczytu elementu zmiennej tekstowej spowoduje natomiast błąd długości ciągu (SLENER).

6.2.5. Instrukcja POKE

Instrukcja POKE umieszcza liczbę podaną jako drugi argument w komórce pamięci, której adres jest pierwszym argumentem. Realizuje ją procedura XPOKE. Najpierw procedura LETNUM odczytuje dwubajtową liczbę całkowitą, która stanowi pierwszy argument. Jest ona przepisywana do rejestru POKADR (POKe ADdRess). Jednobajtowa wartość drugiego argumentu jest odczytywana z bufora przez procedurę GETBYT. Zostaje ona następnie umieszczona we wskazanym miejscu pamięci.

```
0100 ;eXecute POKE statement
0110 ;
0120 FR0 =   $D4
0130 GETBYT = $ABE0
0140 LETNUM = $ABD7
0150 POKADR = $95
0160 ;
0170     *=  $B278
0180 ;
0190     JSR LETNUM
0200     LDA FR0
0210     STA POKADR
0220     LDA FR0+1
```

```
0230     STA POKADR+1
0240     JSR GETBYT
0250     LDA FR0
0260     LDY #$00
0270     STA (POKADR),Y
0280     RTS
```

Procedura GETBYT działa dwustopniowo. Przez wywołanie GLNNUM odczytuje liczbę dwubajtową mniejszą od $8000 (dla liczb większych GLNNUM wskazuje błąd), a jeśli starszy bajt jest niezerowy, to sygnalizuje błąd przez skok do BVALER (Bad VALue ERror).

```
0100 ;GET BYTe
0110 ;
0120 BVALER = $B92E
0130 GLNNUM = $ABCD
0140 ;
0150     *=  $ABE0
0160 ;
0170     JSR GLNNUM
0180     BNE ERR
0190     RTS
0200 ERR JSR BVALER
```

6.3. Instrukcje strukturalne

Podstawowy kształt programu jest tworzony przez instrukcje strukturalne. Umożliwiają one realizację skoków, procedur i pętli oraz warunkowe wykonanie innych instrukcji. Do instrukcji strukturalnych w Atari Basic należą: POP, TRAP, GOTO, GOSUB, RETURN, FOR, NEXT, ON i IF.

6.3.1. Instrukcja POP

Instrukcja POP zdejmuje ze stosu bieżącego informacje o ostatniej procedurze GOSUB/RETURN lub pętli FOR/NEXT. Jest to konieczne przy opuszczaniu procedury lub pętli przez skok. Instrukcja ta jest realizowana przez procedurę XPOP.

```
0100 ;eXecute POP statement
0110 ;
0120 BMEMHI = $90
0130 CLNN =  $A0
0140 DELEL = $A8F7
0150 RUNSTK = $8E
0160 STMNUM = $B2
0170 ;
0180     *=  $B83E
0190 ;
0200     LDA RUNSTK+1
0210     CMP BMEMHI+1
0220     BCC POP
0230     LDA RUNSTK
0240     CMP BMEMHI
0250     BCS $B83D   ;RTS
0260 POP LDA #$04
0270     LDX #BMEMHI
0280     JSR DELEL
0290     LDY #$03
```

```
0300     LDA (BMEMHI),Y
0310     STA STMNUM
0320     DEY
0330     LDA (BMEMHI),Y
0340     STA CLNN+1
0350     DEY
0360     LDA (BMEMHI),Y
0370     STA CLNN
0380     DEY
0390     LDA (BMEMHI),Y
0400     BEQ END
0410     PHA
0420     LDA #$0C
0430     LDX #BMEMHI
0440     JSR DELEL
0450     PLA
0460 END CLC
0470     RTS
```

Procedura XPOP musi najpierw sprawdzić, czy na stosie jest coś zapisane, w przeciwnym bowiem przypadku uległaby zniszczeniu część tablicy zmiennych indeksowanych. Następnie przez wywołanie procedury DELEL wektor BMEMHI jest obniżany o cztery miejsca. Odczytane stamtąd wartości są przepisywane kolejno do rejestru numeru instrukcji STMNUM (STateMent NUMber) oraz numeru wiersza CLNN (Current LiNe Number). Ostatnia wartość pozwala na rozpoznanie rodzaju informacji zapisanej na stosie. Jeśli jest to zero, a więc informacja dotyczy procedury GOSUB, to procedura XPOP się kończy. W przeciwnym przypadku są to dane pętli FOR/NEXT (teraz już zbędne) i przez ponowne wywołanie DELEL wektor BMEMHI jest zmniejszany jeszcze o dwanaście miejsc.

6.3.2. Instrukcja TRAP

Instrukcja TRAP ustala numer wiersza programu, od którego będzie kontynuowana praca po wystąpieniu błędu. Jej procedura wykonawcza - XTRAP - jest bardzo prosta. Numer wiersza odczytany przy pomocy LETNUM umieszczany jest w rejestrze TRAPLN (TRAP LiNe number) i to wszystko.

```
0100 ;eXecute TRAP statement
0110 ;
0120 FR0 =   $D4
0130 LETNUM = $ABD7
0140 TRAPLN = $BC
0150 ;
0160     *=  $B7D8
0170 ;
0180     JSR LETNUM
0190     LDA FR0
0200     STA TRAPLN
0210     LDA FR0+1
0220     STA TRAPLN+1
0230     RTS
```

6.3.3. Instrukcje GOTO i GOSUB

Instrukcje GOTO (lub GO TO) i GOSUB służą do wykonywania skoków do innych miejsc programu. Dodatkowo GOSUB powoduje zapamiętanie adresu aktualnej instrukcji, co umożliwia potem powrót. W związku z tym ich procedury realizacyjne - XGOTO i XGOSUB - różnią się tylko jednym rozkazem, który wywołuje procedurę zapisu parametrów bieżącej instrukcji.

```
0100 CLNN =  $A0
0110 FNDCST = $A9A2
0120 FR0 =   $D4
0130 GLNNUM = $ABCD
0140 LNFDER = $B91C
0150 PRCSTM = $A95E
0160 RSCSTM = $B6F0
0170 SAVSTM = $B6F9
0180 ;
0190     *=  $B6D2
0200 ;
0210 ;eXecute GOSUB statement
0220 ;
0230 ;
0240 XGOSUB JSR SAVSTM
0250 ;
0260 ;eXecute GOTO statement
0270 ;
0280 XGOTO JSR GLNNUM
0290 ;
0300 ;GO to LINE
0310 ;
0320 GOLINE LDA FR0+1
0330     STA CLNN+1
0340     LDA FR0
0350     STA CLNN
0360 ;
0370 ;FinD & EXecute STatement
0380 ;
0390 FDEXST JSR FNDCST
0400     BCS ERR
0410     PLA
0420     PLA
0430     JMP PRCSTM
0440 ERR JSR RSCSTM
0450     JSR LNFDER
```

Odczytanie numeru wiersza, do którego będzie wykonany skok realizuje procedura GLNNUM. Numer ten jest przepisywany do rejestru CLNN i procedura FNDCST odszukuje go w pamięci. Następnie ze stosu procesora zdejmowany jest adres powrotny dla XGOTO lub XGOSUB i po wykonaniu skoku do PRCSTM realizowany jest wskazany wiersz programu. Jeżeli nie ma w programie wiersza o podanym numerze, to przez wywołanie procedury RSCSTM odtwarzany jest aktualny numer wiersza i przez skok do LNFDER sygnalizowany jest błąd (Line Not FounD ERror).

```
0100 ;ReStore Current STateMent
0110 ;
0120 SAVCUR = $BE
0130 STMCUR = $8A
0140 ;
```

```
0150     *=  $B6F0
0160 ;
0170     LDA SAVCUR
0180     STA STMCUR
0190     LDA SAVCUR+1
0200     STA STMCUR+1
0210     RTS
```

Podczas realizacji instrukcji GOSUB dodatkowo jest wywoływana procedura SAVSTM, która zapisuje parametry aktualnego wiersza na stosie bieżącym Basica. Składa się ona z wywołań dwóch innych procedur, przy czym PHSTK jest fragmentem procedury XFOR (rozdział 6.3.5).

```
0100 ;SAVe current STateMent
0110 ;
0120 PHSTK = $B6B5
0130 SAVIX = $B883
0140 ;
0150     *=  $B6F9
0160 ;
0170     JSR SAVIX
0180     LDA #$00
0190     BEQ PHSTK
```

Druga procedura - SAVIX - przepisuje tylko aktualny stan licznika wejściowego INIX (INput IndeX) do pomocniczego rejestru TMPIX (TeMPorary IndeX).

```
0100 ;SAVe IndeX
0110 ;
0120 INIX =  $A8
0130 TMPIX = $B3
0140 ;
0150     *=  $B883
0160 ;
0170     LDY INIX
0180     STY TMPIX
0190     RTS
```

6.3.4. Instrukcja RETURN

Instrukcja RETURN jest odwrotnością instrukcji GOSUB i powoduje powrót do niej, a dalszy przebieg programu jest wykonywany od instrukcji następującej po GOSUB lub ON/GOSUB. Jej procedura realizacyjna - XRTRN - jest znacznie bardziej uniwersalna i stosowana jest także dla odtworzenia aktualnego wiersza programu po wykonaniu instrukcji READ i LIST. Rozpoczyna się ona od wywoływania opisanej wyżej procedury XPOP, aż do natrafienia w ostatnim zdejmowanym ze stosu bajcie na wartość równą zero. Jeśli zaś na stosie brak takich danych, to przez skok do RETER sygnalizowany jest błąd (RETurn ERror).

```
0100 ;eXecute RETURN statement
0110 ;
0120 FSTGLN = $B816
0130 GOSLER = $B916
0140 OUTIX = $A7
0150 RETER = $B914
0160 RSCSTM = $B6F0
0170 STMCUR = $8A
0180 STMNUM = $B2
```

```
0190 XPOP =  $B83E
0200 ;
0210     *=  $BDA8
0220 ;
0230 XRTRN JSR XPOP
0240     BCS ERR
0250     BNE XRTRN
0260     JSR RETURN
0270     CMP #$0C
0280     BEQ END
0290     CMP #$1E
0300     BEQ END
0310     CMP #$04
0320     BEQ END
0330     CMP #$22
0340     BEQ END
0350 ;
0360 ;Bad RETurn LiNe
0370 ;
0380 BRETLN JSR RSCSTM
0390     JSR GOSLER
0400 ERR JSR RETER
0410 ;
0420 ;RETURN
0430 ;
0440 RETURN JSR FSTGLN
0450     BCS BRETLN
0460     LDY STMNUM
0470     DEY
0480     LDA (STMCUR),Y
0490     STA OUTIX
0500     INY
0510     LDA (STMCUR),Y
0520 END RTS
```

Następnie przez wywołanie RETURN odtwarzany jest stan licznika wyjściowego OUTIX oraz adres ostatniego wykonanego tokena, który jest ponadto odczytywany. Jeśli jest to token jednej z dozwolonych tu instrukcji - GOSUB, ON, LIST lub READ, to procedura się kończy. W przeciwnym przypadku procedura RSCSTM odtwarza adres instrukcji RETURN i skokiem do GOSLER wskazywany jest błąd (GOSub Line ERror).

6.3.5. Instrukcja FOR

Instrukcja FOR rozpoczyna pętlę, w której wielokrotnie wykonywany jest zestaw innych instrukcji. Do realizacji tej instrukcji służy procedura XFOR.

```
0100 ;eXecute FOR statement
0110 ;
0120 BMTUP = $B871
0130 FR0 =   $D4
0140 GETIX = $B904
0150 LETVAR = $AC06
0160 PHREG = $B888
0170 PLSTK = $B823
0180 SAVIX = $B883
0190 SAVMHI = $C4
0200 STMCUR = $8A
0210 TMPIX = $B3
```

```
0220 VARN =  $D3
0230 XLET =  $AADA
0240 ZFR0 =  $DA44
0250 ;
0260     *=  $B67D
0270 ;
0280     JSR SAVIX
0290     JSR XLET
0300     LDA VARN
0310     ORA #$80
0320     PHA
0330     JSR PLSTK
0340     LDA #$0C
0350     JSR BMTUP
0360     JSR LETVAR
0370     LDX #FR0
0380     LDY #$00
0390     JSR PHREG
0400     JSR ZFR0
0410     LDA #$01
0420     STA FR0+1
0430     LDA #$40
0440     STA FR0
0450     JSR GETIX
0460     BCS PHN
0470     JSR LETVAR
0480 PHN LDX #FR0
0490     LDY #$06
0500     JSR PHREG
0510     PLA
0520 ;
0530 ;PusH on STacK
0540 ;
0550 PHSTK PHA
0560     LDA #$04
0570     JSR BMTUP
0580     PLA
0590     LDY #$00
0600     STA (SAVMHI),Y
0610     LDA (STMCUR),Y
0620     INY
0630     STA (SAVMHI),Y
0640     LDA (STMCUR),Y
0650     INY
0660     STA (SAVMHI),Y
0670     LDX TMPIX
0680     DEX
0690     TXA
0700     INY
0710     STA (SAVMHI),Y
0720     RTS
```

Rozpoczyna się ona od zapisania aktualnego stanu licznika INIX przez procedurę SAVIX oraz ustalenia przy pomocy procedury XLET stanu początkowego zmiennej sterującej (licznika pętli). Numer tej zmiennej jest odkładany na stos i wywoływana jest procedura PLSTK.

```
0100 ;PulL from STacK
0110 ;
0120 ACNT2 = $C7
0130 BMEMHI = $90
```

```
0140 SAVMHI = $C4
0150 SAVTST = $B87A
0160 XPOP =  $B83E
0170 ;
0180     *=  $B823
0190 ;
0200     STA ACNT2
0210     JSR SAVTST
0220 POP JSR XPOP
0230     BCS MEM
0240     BEQ MEM
0250     CMP ACNT2
0260     BEQ END
0270     BNE POP
0280 MEM LDA SAVMHI
0290     STA BMEMHI
0300     LDA SAVMHI+1
0310     STA BMEMHI+1
0320 END RTS
```

Zapisuje ona numer zmiennej do pomocniczego rejestru ACNT2 (Auxiliary CouNTer) oraz przepisuje aktualny wektor górnej granicy pamięci do rejestru SAVMHI (przy pomocy procedury SAVTST). Następnie przez kolejne wywołania XPOP przeszukiwany jest stos bieżący. Znalezienie danych dla procedury GOSUB lub osiągnięcie końca stosu przerywa to przeszukiwanie i powoduje odtworzenie wektora BMEMHI. Natomiast bez odtwarzania tego wektora procedura PLSTK kończy się w przypadku znalezienia na stosie zmiennej o tym samym numerze, co znajdujący się w rejestrze ACNT2.

```
0100 ;SAVe Top of STack
0110 ;
0120 BMEMHI = $90
0130 SAVMHI = $C4
0140 ;
0150     *=  $B87A
0160 ;
0170     LDX BMEMHI
0180     STX SAVMHI
0190     LDX BMEMHI+1
0200     STX SAVMHI+1
0210     RTS
```

Teraz w akumulatorze umieszczona zostaje wartość $0C i wywoływana jest procedura BMTUP. Najpierw zapisuje ona aktualny stan BMEMHI, a potem podnosi go przy pomocy procedury INSEL o wartość z akumulatora, czyli o 12 bajtów, przez co uzyskuje się miejsce na umieszczenie parametrów zmiennej sterującej pętli (wartość początkowa, wartość końcowa i krok).

```
0100 ;Basic Memory Top UP
0110 ;
0120 BMEMHI = $90
0130 INSEL = $A87A
0140 SAVTST = $B87A
0150 ;
0160     *=  $B871
0170 ;
0180     JSR SAVTST
0190     TAY
```

```
0200     LDX #BMEMHI
0210     JMP INSEL
```

Graniczna wartość licznika pętli jest ustalana przy pomocy procedury LETVAR. Przez kolejne wywołania XLET i GETVAR oblicza ona wartość tej granicy i przepisuje uzyskany wynik z bufora wejściowego do rejestru FR0.

```
0100 ;LET VARiable
0110 ;
0120 GETVAR = $ABE9
0130 XLET =  $AADA
0140 ;
0150     *=  $AC06
0160 ;
0170     JSR XLET
0180     JMP GETVAR
```

Obliczona wartość graniczna zmiennej sterującej jest przepisywana przez procedurę PHREG z rejestru FR0 do odpowiedniego miejsca stosu bieżącego. Używana jest w tym celu zawartość rejestru SAVMHI, gdyż w rejestrze BMEMHI znajduje się już nowy adres szczytu zajętej pamięci.

```
0100 ;PusH REGister
0110 ;
0120 ACNT1 = $C6
0130 SAVMHI = $C4
0140 ;
0150     *=  $B888
0160 ;
0170     LDA #$06
0180     STA ACNT1
0190 LOOP LDA $00,X
0200     STA (SAVMHI),Y
0210     INX
0220     INY
0230     DEC ACNT1
0240     BNE LOOP
0250     RTS
```

Następnie w rejestrze FR0 umieszczana jest standardowa wartość kroku pętli (1). Jeśli procedura GETIX wskaże, że instrukcja FOR ma jeszcze dalszy ciąg, to liczba ta jest zastępowana przez wartość kroku odczytaną przez procedurę LETVAR. Kolejne wywołanie PHREG umieszcza odpowiednią wartość kroku pętli na stosie bieżącym.

Na zakończenie przepisywane są na stos parametry instrukcji FOR. Są to kolejno: numer zmiennej użytej do sterowania pętli, numer wiersza zawierającego instrukcję FOR oraz indeks wskazujący początek instrukcji następującej po FOR. Ponieważ szczyt stosu jest przy tym ponownie podnoszony o cztery bajty, więc razem dla zapisania wszystkich informacji zostało wykorzystane 16 bajtów.

6.3.6. Instrukcja NEXT

Instrukcja NEXT stanowi taką parę dla instrukcji FOR, jak RETURN dla GOSUB. Poza odtworzeniem adresu początkowego pętli trzeba tu jeszcze dokonać zmiany stanu licznika pętli i

sprawdzić jego wartość. Do wykonania tych czynności służy procedura XNEXT. Jej początek jest podobny jak w procedurze XFOR. Tu jednak nieodnalezienie na stosie zmiennej o podanym numerze powoduje skok do procedury NFORER i zasygnalizowanie przez nią błędu (No matching FOR ERror).

```
0100 ;eXecute NEXT statement
0110 ;
0120 ACNT2 = $C7
0130 BADD =  $AD26
0140 BMTUP = $B871
0150 BRETLN = $BDC2
0160 FR1 =   $E0
0170 GETSUB = $AD20
0180 INIX =  $A8
0190 NFORER = $B91A
0200 PLREG = $B897
0210 RETURN = $BDCB
0220 SAVVAL = $AC0C
0230 STMCUR = $8A
0240 VARBL = $AB81
0250 XPOP =  $B83E
0260 ;
0270     *=  $B700
0280 ;
0290     LDY INIX
0300     LDA (STMCUR),Y
0310     STA ACNT2
0320 POP JSR XPOP
0330     BCS ERR
0340     BEQ ERR
0350     CMP ACNT2
0360     BNE POP
0370     LDY #$06
0380     JSR PLREG
0390     LDA FR1
0400     PHA
0410     LDA ACNT2
0420     JSR VARBL
0430     JSR BADD
0440     JSR SAVVAL
0450     LDY #$00
0460     JSR PLREG
0470     PLA
0480     BPL BPS
0490     JSR GETSUB
0500     BPL EXIT
0510     RTS
0520 BPS JSR GETSUB
0530     BEQ EXIT
0540     BMI EXIT
0550 END RTS
0560 EXIT LDA #$10
0570     JSR BMTUP
0580     JSR RETURN
0590     CMP #$08
0600     BEQ END
0610     JMP BRETLN
0620 ERR JSR NFORER
```

Po odnalezieniu parametrów właściwej instrukcji FOR wywoływana jest procedura PLREG (odwrotna do PHREG). Pobiera ona ze stosu wartość kroku i dodaje go do aktualnej wartości zmiennej sterującej, która została odczytana przez procedurę VARBL. Uzyskany rezultat jest ponownie zapisywany przy pomocy procedury SAVVAL.

```
0100 ;PulL REGister
0110 ;
0120 ACNT1 = $C6
0130 FR1 =   $E0
0140 BMEMHI = $90
0150 ;
0160     *=  $B897
0170 ;
0180     LDA #$06
0190     STA ACNT1
0200     LDX #FR1
0210 LOOP LDA (BMEMHI),Y
0220     STA $00,X
0230     INX
0240     INY
0250     DEC ACNT1
0260     BNE LOOP
0270     RTS
```

Następne wywołanie PLREG powoduje odczytanie ze stosu granicznej wartości zmiennej sterującej. Warto przy tym zauważyć, że dla uproszczenia wartość odczytana przez PLREG jest zawsze umieszczana w rejestrze FR1. Po wykonaniu odejmowania przez procedurę GETSUB dalsze postępowanie zależy od znaku kroku pętli i znaku uzyskanego rezultatu. Jeśli została osiągnięta lub przekroczona graniczna wartość zmiennej sterującej, to procedura XNEXT kończy się rozkazem RTS i dalej wykonywana jest instrukcja następująca po NEXT. W przeciwnym wypadku przy użyciu procedury RETURN odczytywany jest adres pierwszej instrukcji pętli i tam przekazywane jest sterowanie.

6.3.7. Instrukcja IF

Instrukcja IF umożliwia warunkowe wykonanie innych instrukcji, ewentualnie skoku, jeśli zamiast instrukcji umieszczony w niej będzie numer wiersza. Jej procedura realizacyjna - XIF - jest bardzo prosta.

```
0100 ;eXecute IF statement
0110 ;
0120 BUFIX = $9F
0130 FR0 =   $D4
0140 GETIX = $B904
0150 LETVAR = $AC06
0160 OUTIX = $A7
0170 XGOTO = $B6D5
0180 ;
0190     *=  $B778
0200 ;
0210     JSR LETVAR
0220     LDA FR0+1
0230     BEQ MIX
0240     JSR GETIX
0250     BCS END
```

```
0260     JMP XGOTO
0270 MIX LDA BUFIX
0280     STA OUTIX
0290 END RTS
```

Przede wszystkim przez wywołanie procedury LETVAR obliczane jest wyrażenie warunkowe. Jeśli wynikiem jest zero, to po zrównaniu liczników BUFIX i OUTIX procedura się kończy. W takim przypadku wykonywanie programu jest kontynuowane od początku następnego wiersza. Jeśli wyrażenie warunkowe jest prawdziwe (wynik różny od zera), to procedura GETIX porównuje stany liczników bufora. Napotkanie końca wiersza powoduje skok do procedury XGOTO, a w przeciwnym wypadku wykonywane są dalsze instrukcje znajdujące się w tym samym wierszu.

6.3.8. Instrukcja ON

Instrukcja ON jest wielokrotną warunkową instrukcją skoku i posiada dwie odmiany - ON/GOTO i ON/GOSUB. W zależności od wartości wyrażenia wykonywany jest skok do wiersza, którego numer jest umieszczony na odpowiedniej pozycji w instrukcji. Procedurą realizacyjną tej instrukcji jest XON.

Rozpoczyna się ona odczytaniem wartości wyrażenia przy pomocy procedury GETBYT. Wynika z tego, że wyrażenie musi dawać wynik całkowity, mniejszy od 256, gdyż w innym przypadku wystąpi błąd. Zerowa wartość tego wyrażenia powoduje natychmiastowe opuszczenie procedury XON. Następnie sprawdzany jest rodzaj instrukcji ON. Jeśli jest to ON/GOSUB, to adres powrotny zapamiętywany jest przez wywołanie procedury SAVSTM.

Teraz w pętli kolejno odczytywane są numery wierszy i zmniejszana jest obliczona wartość wyrażenia (przepisana do rejestru TMPIX). Wyzerowanie tego rejestru oznacza napotkanie właściwego numeru wiersza i powoduje wykonanie skoku przez przejście do środka procedury XGOSUB w miejsce oznaczone etykietą GOLINE. Jeśli wartość wyrażenia jest większa od liczby numerów wierszy zawartych w instrukcji ON, to procedura XON jest opuszczana bez wykonania skoku. W takim przypadku adres powrotny dla ON/GOSUB jest jeszcze zdejmowany ze stosu przez procedurę XPOP.

```
0100 ;eXecute ON statement
0110 ;
0120 FR0 =   $D4
0130 GETBYT = $ABE0
0140 GETIX = $B904
0150 GLNNUM = $ABCD
0160 GOLINE = $B6D8
0170 INIX =  $A8
0180 SAVIX = $B883
0190 SAVSTM = $B6F9
0200 STMCUR = $8A
0210 TMPIX = $B3
0220 XPOP =  $B83E
0230 ;
0240     *=  $B7E4
0250 ;
0260     JSR SAVIX
```

```
0270     JSR GETBYT
0280     LDA FR0
0290     BEQ END
0300     LDY INIX
0310     DEY
0320     LDA (STMCUR),Y
0330     CMP #$17
0340     PHP
0350     BEQ SKIP
0360     JSR SAVSTM+3
0370 SKIP LDA FR0
0380     STA TMPIX
0390 NEXT JSR GLNNUM
0400     DEC TMPIX
0410     BEQ JUMP
0420     JSR GETIX
0430     BCC NEXT
0440     PLP
0450     BEQ END
0460     JSR XPOP
0470 END RTS
0480 JUMP PLP
0490     JMP GOLINE
```

6.4. Obsługa komunikacji

Wiele procedur interpretera komunikuje się z urządzeniami zewnętrznymi. Przeważnie są to procedury wykonawcze instrukcji wejścia/wyjścia, lecz nie tylko. Do realizacji tego celu przeznaczony jest zestaw procedur obsługujących komunikację z urządzeniami zewnętrznymi. Ze względu na wykorzystywanie przez liczne procedury realizujące instrukcje Basica zestaw ten został wydzielony w osobnym rozdziale.

Podstawową procedurą komunikacji z urządzeniami zewnętrznymi jest PRPCHN. Przygotowuje i przeprowadza ona operacje wejścia/wyjścia, korzystając przy tym z innych procedur pomocniczych. Przed jej wywołaniem w rejestrze IOCHN (I/O CHannel Number) trzeba umieścić numer wykorzystywanego kanału IOCB, a w rejestrze IOCMD (I/O CoMmanD) kod rozkazu wykonywanej operacji. Ponadto, jeśli jest to operacja zapisu, w akumulatorze powinien znajdować się przesyłany znak. Przy wywoływaniu tej procedury od etykiety PRPDVC trzeba dodatkowo umieścić w rejestrze X numer kanału IOCB.

```
0100 ENDIO = $BCBB
0110 ICAX1 = $034A
0120 ICAX1Z = $2A
0130 ICAX2 = $034B
0140 ICAX2Z = $2B
0150 IOCAL = $BAB2
0160 IOCHN = $B5
0170 SAVDVC = $BAC0
0180 ;
0190     *=  $BA99
0200 ;
0210 ;PRePare CHaNnel
0220 ;
0230 PRPCHN LDX IOCHN
0240 ;
```

```
0250 ;PRePare DeViCe
0260 ;
0270 PRPDVC PHA
0280     JSR SAVDVC
0290     LDA ICAX1,X
0300     STA ICAX1Z
0310     LDA ICAX2,X
0320     STA ICAX2Z
0330     PLA
0340     TAY
0350     JSR IOCAL
0360     TYA
0370     JMP ENDIO+3
```

Na początku, po odczytaniu numeru kanału z rejestru IOCHN i odłożeniu przesyłanego znaku na stosie, wywoływana jest procedura SAVDVC. Zapisuje ona numer kanału i przelicza go na postać wymaganą przez system operacyjny (zob. "Mapa pamięci Atari XL/XE. Procedury wejścia/wyjścia"), a wynik zwraca w rejestrze X. Następnie parametry transmisji z odpowiedniego bloku IOCB przepisywane są na stronę zerową. Teraz przesyłany znak jest zdejmowany ze stosu do rejestru Y i wywoływana jest procedura IOCAL.

```
0100 ;Input/Output routine CAL1
0110 ;
0120 ICPUTB = $0346
0130 ;
0140     *=  $BAB2
0150 ;
0160     LDA ICPUTB+1,X
0170     PHA
0180     LDA ICPUTB,X
0190     PHA
0200     TYA
0210     LDY #$92
0220     RTS
```

Jej zadaniem jest uruchomienie odpowiedniej procedury wykonującej operację I/O. W tym celu adres tej procedury jest przepisywany na stos, a przesyłany znak do akumulatora, zaś do rejestru Y zapisywany jest kod nieistniejącej operacji (Function Not Implemented). Wykonanie teraz rozkazu RTS powoduje zdjęcie ze stosu adresu procedury transmisji i tym samym wykonanie operacji. Po jej zakończeniu status operacji przepisywany jest z rejestru Y do akumulatora. Procedura PRPCHN kończy się skokiem do specjalnej procedury kończącej operacje wejścia/wyjścia - ENDIO.

Wspomniana wcześniej procedura SAVDVC jest fragmentem nieco większej procedury SAVCMD. Umieszcza ona jedynie zawartość akumulatora w rejestrze IOCMD (tylko przy wywołaniu od SAVCMD) oraz zawartość rejestru X w IODVC (I/O DeViCe). Potem następuje bezpośredni skok do procedury MLTCHN.

```
0100 IOCMD = $C0
0110 IODVC = $C1
0120 MLTCHN = $BCAF
0130 ;
0140     *=  $BABE
0150 ;
0160 ;SAVe CoMmanD
```

```
0170 ;
0180 SAVCMD STA IOCMD
0190 ;
0200 ;SAVe DeViCe number
0210 ;
0220 SAVDVC STX IODVC
0230     JMP MLTCHN
```

Także i ta procedura jest częścią większej - SETDVC. System operacyjny Atari wymaga, aby przy wywołaniu procedur I/O w rejestrze X znajdował się numer bloku IOCB pomnożony przez 16. Numer ten jest więc odczytywany z rejestru IODVC i przez czterokrotne przesunięcie w lewo doprowadzany do właściwej postaci. Zbyt duży numer kanału powoduje skok do DVCNER i sygnalizację błędu (DeViCe Number ERror).

```
0100 DVCNER = $B90C
0110 IODVC = $C1
0120 IXGDAT = $BD07
0130 ;
0140     *=  $BCA8
0150 ;
0160 ;SET DeViCe
0170 ;
0180 SETDVC JSR IXGDAT
0190     STA IODVC
0200     BEQ ERR
0210 ;
0220 ;MuLTiple CHannel Number
0230 ;
0240 MLTCHN LDA IODVC
0250     ASL A
0260     ASL A
0270     ASL A
0280     ASL A
0290     TAX
0300     BPL $BD06   ;RTS
0310 ERR JSR DVCNER
```

Przy wywołaniu tej procedury od etykiety SETDVC ustalany jest najpierw numer kanału. W tym celu wywoływana jest procedura IXGDAT, która przy pomocy GLNNUM pobiera z bufora wejściowego liczbę. Liczba ta zostaje następnie umieszczona w rejestrze IODVC. Także tu niepoprawny (zerowy) numer kanału powoduje zasygnalizowanie błędu przez skok do DVCNER.

```
0100 FR0 =   $D4
0110 GLNNUM = $ABCD
0120 INIX =  $A8
0130 ;
0140     *=  $BD07
0150 ;
0160 ;Increase indeX & Get DATa
0170 ;
0180 IXGDAT INC INIX
0190 ;
0200 ;GET DATa
0210 ;
0220 GETDAT JSR GLNNUM
0230     LDA FR0
0240     RTS
```

Procedura kończąca operacje I/O służy do kontroli ich poprawnego wykonania i wstępnej obsługi ewentualnych błędów. W tym celu najpierw wywoływana jest procedura GETST, która odczytuje z odpowiedniego bloku IOCB status wykonanej operacji. Jeśli jest on mniejszy od $80, czyli przebieg operacji był poprawny, to procedura ENDIO kończy się rozkazem RTS (w tym przypadku znajduje się on w innej procedurze).

```
0100 ;END I/O operation
0110 ;
0120 CDST =  $A000
0130 CLCHN = $BCF7
0140 DSPFLG = $02FE
0150 ERRCOD = $B9
0160 GETERR = $B934
0170 GETST = $BD00
0180 IODVC = $C1
0190 IRQSTAT = $11
0200 LOADFLG = $CA
0210 PROMPT = $C2
0220 RSTCHN = $BD5B
0230 WRMST2 = $A053
0240 ;
0250     *=  $BCBB
0260 ;
0270     JSR GETST
0280     BPL $BD06   ;RTS
0290     LDY #$00
0300     STY DSPFLG
0310     CMP #$80
0320     BNE GST
0330     STY IRQSTAT
0340     LDA LOADFLG
0350     BEQ $BD06   ;RTS
0360     JMP CDST
0370 GST LDY IODVC
0380     CMP #$88
0390     BEQ EXIT
0400 PST STA ERRCOD
0410     CPY #$07
0420     BNE RST
0430     JSR CLCHN
0440 RST JSR RSTCHN
0450     JMP GETERR
0460 EXIT CPY #$07
0470     BNE PST
0480     LDX #$5D
0490     CPX PROMPT
0500     BNE PST
0510     JSR CLCHN
0520     JMP WRMST2
```

```
0100 ;GET STatus
0110 ;
0120 ICSTAT = $0343
0130 MLTCHN = $BCAF
0140 ;
0150     *=  $BD00
0160 ;
0170     JSR MLTCHN
0180     LDA ICSTAT,X
```

```
0190     RTS
```

Jeśli status wskazuje niepoprawny przebieg operacji, to po wyzerowaniu znacznika DSPFLG (DiSPlay FLaG) jest on porównywany z kodem przerwania przez naciśnięcie klawisza BREAK ($80). Taka przyczyna błędu jest zapisywana w rejestrze IRQSTAT (IRQ STATus shadow register). Jeżeli operacja I/O dotyczyła odczytu programu z pamięci masowej, co wskazuje niezerowa zawartość rejestru LOADFLG (LOADing FLaG), to następuje ponowne zainicjowanie interpretera przez skok do CDST. W przeciwnym przypadku procedura ENDIO kończy się rozkazem RTS.

Następnie sprawdzane jest, czy status sygnalizuje napotkanie końca pliku ($88). Jeśli tak, to w przypadku odczytu programu zamykany jest kanał 7 (poprzez CLCHN) i wykonywany jest skok do gorącego startu, a więc interpreter przechodzi do trybu bezpośredniego bez sygnalizowania błędu. Każdy inny status błędu powoduje zamknięcie kanału (z wyjątkiem kanału IOCB 0), ustawienie wyjścia na kanał 0 (przez procedurę RSTCHN) i sygnalizowanie błędu przez skok do procedury GETERR (zob. rozdział 3).

Zamknięcie kanału IOCB jest wykonywane przez procedurę CLCHN. Ponieważ numer kanału jest ustalany przez procedurę MLTCHN, to zawsze zamykany jest kanał o numerze zapisanym w rejestrze IODVC. Kanał IOCB 0 jest wykorzystywany przez edytor ekranowy i jego zamknięcie jest niemożliwe. Zerowy numer kanału powoduje więc przerwanie procedury rozkazem RTS. W przeciwnym przypadku do akumulatora wpisywany jest kod operacji CLOSE ($0C) i następuje skok do procedury CIOEXE.

```
0100 ;CLose CHaNnel
0110 ;
0120 CIOEXE = $BD2B
0130 MLTCHN = $BCAF
0140 ;
0150     *=  $BCF7
0160 ;
0170     JSR MLTCHN
0180     BEQ $BD06   ;RTS
0190     LDA #$0C
0200     BNE CIOEXE
```

Etykieta CIOEXE oznacza końcowy fragment procedury BFLN. Rozpoczyna się ona od ustalenia długości bufora wykorzystywanego podczas operacji I/O. Długość tego bufora jest zależna od miejsca rozpoczęcia procedury. Przy wywołaniu od BFLN1 długość bufora wynosi $FF, od BFLN2 - zero (znak przesyłany jest więc w akumulatorze), od BFLN3 - długość przepisywana jest z rejestru Y (mniejsza od $0100), zaś przy wywołaniu od BFLN4 długość ustalana jest według zawartości akumulatora (starszy bajt) i rejestru Y (młodszy bajt).

Teraz (od etykiety SIBUFA) jako adres bufora jest przepisywany wektor zawarty w rejestrze INBUFP (INput BUFfer Pointer). Następnie z rejestru IOCMD odczytywany jest do akumulatora kod rozkazu operacji, która ma być wykonana (etykieta CMDEXE). Kod ten jest umieszczany w rejestrze ICCMND (IOCB CoMmaND register) odpowiedniego bloku IOCB. To miejsce jest właśnie oznaczone etykietą CIOEXE. Cała procedura kończy się bezpośrednim skokiem do

centralnej procedury wejścia/wyjścia systemu operacyjnego - CIOMAIN (zob. "Mapa pamięci Atari XL/XE. Procedury wejścia/wyjścia.").

```
0100 ICBUFA = $0344
0110 ICBUFL = $0348
0120 ICCMND = $0342
0130 INBUFP = $F3
0140 IOCMD = $C0
0150 JCIOMAIN = $E456
0160 ;
0170     *=  $BD0F
0180 ;
0190 ;set BuFfer LeNgth
0200 ;
0210 BFLN1 LDY #$FF
0220     BNE BFLN3
0230 BFLN2 LDY #$00
0240 BFLN3 LDA #$00
0250 BFLN4 STA ICBUFL+1,X
0260     TYA
0270     STA ICBUFL,X
0280 ;
0290 ;Set Input BUFfer Address
0300 ;
0310 SIBUFA LDA INBUFP+1
0320     LDY INBUFP
0330     STA INBUFA+1,X
0340     TYA
0350     STA ICBUFA,X
0360 ;
0370 ;CoMmanD EXEcute
0380 ;
0390 CMDEXE LDA IOCMD
0400 ;
0410 ;Central I/O EXEcute
0420 ;
0430 CIOEXE STA ICCMND,X
0440     JMP JCIOMAIN
```

6.5. Instrukcje dźwiękowe i graficzne

W grupie instrukcji wejścia/wyjścia najczęściej używane są instrukcje dźwiękowe i graficzne. Dlatego też zostały one wydzielone w osobnym rozdziale. Są to instrukcje: SOUND, SETCOLOR, COLOR, POSITION, PLOT, DRAWTO i GRAPHICS. Istnieją jeszcze inne instrukcje działające na ekranie, lecz ze względu na ich wykorzystanie przy współpracy z urządzeniami zewnętrznymi znajdują się one w następnym rozdziale.

6.5.1. Instrukcja SOUND

Instrukcja SOUND powoduje uruchomienie wskazanego generatora dźwięku. Praca generatora trwa do następnej przeznaczonej dla niego instrukcji SOUND albo do instrukcji NEW, END lub RUN. Instrukcja SOUND wymaga czterech parametrów, które oznaczają kolejno: numer generatora, okres generatora, rodzaj zniekształceń i głośność dźwięku. Do realizacji tej instrukcji służy procedura XSOUND.

```
0100 ;eXecute SOUND statement
```

```
0110 ;
0120 AUDC1 = $D201
0130 AUDCTL = $D208
0140 AUDF1 = $D200
0150 FR0 =   $D4
0160 GETBYT = $ABE0
0170 LETNUM = $ABD7
0180 SKCTL = $D20F
0190 ;
0200     *=  $B9D3
0210 ;
0220     JSR GETBYT
0230     LDA FR0
0240     CMP #$04
0250     BCS $B9D0   ;JMP BVALER
0260     ASL A
0270     PHA
0280     LDA #$00
0290     STA AUDCTL
0300     LDA #$03
0310     STA SKCTL
0320     JSR LETNUM
0330     PLA
0340     PHA
0350     TAX
0360     LDA FR0
0370     STA AUDF1,X
0380     JSR LETNUM
0390     LDA FR0
0400     ASL A
0410     ASL A
0420     ASL A
0430     ASL A
0440     PHA
0450     JSR LETNUM
0460     PLA
0470     TAY
0480     PLA
0490     TAX
0500     TYA
0510     CLC
0520     ADC FR0
0530     STA AUDC1,X
0540     RTS
```

Jako pierwszy pobierany jest przez procedurę GETNUM numer generatora. Jeśli jest on większy od 3, to procedura jest przerywana skokiem do BVALER, gdzie sygnalizowany jest błąd. Poprawna wartość jest mnożona przez dwa i odkładana na stosie. Następnie ustawiane są początkowe stany rejestrów AUDCTL (AUDio ConTroL) oraz SKCTL (Serial/Keyboard ConTroL). Wywołana teraz procedura LETNUM odczytuje wartość okresu generatora, która jest wpisywana przy użyciu odtworzonego ze stosu numeru do rejestru AUDF (AUDio Frequerency) odpowiedniego generatora. Kolejne wywołanie LETNUM pobiera kod zniekształceń dźwięku. Jest on mnożony przez 16 i odkładany na stos. Trzecie i ostatnie wywołanie LETNUM daje wartość głośności dźwięku. Do tej wartości dodawany jest zdjęty ze stosu kod zniekształceń - cztery młodsze bity określają więc głośność, a cztery starsze zniekształcenie. Wynik umieszczany jest w rejestrze AUDC (AUDio Control) odpowiedniego generatora.

6.5.2. Instrukcja SETCOLOR

Zadaniem instrukcji SETCOLOR jest ustawienie barwy i jasności koloru w jednym z rejestrów koloru obrazu. Do tego celu potrzebne są trzy parametry: numer rejestru koloru oraz barwa i jasność koloru. Procedura realizacyjna tej instrukcji - XSETC - jest bardzo podobna do procedury XSOUND.

```
0100 ;eXecute SETCOLOR statement
0110 ;
0120 BVALER = $B92E
0130 COLFP0S = $02C4
0140 FR0 =   $D4
0150 GETBYT = $ABE0
0160 LETNUM = $ABD7
0170 ;
0180     *=  $B9AD
0190 ;
0200     JSR GETBYT
0210     LDA FR0
0220     CMP #$05
0230     BCS ERR
0240     PHA
0250     JSR LETNUM
0260     LDA FR0
0270     ASL A
0280     ASL A
0290     ASL A
0300     ASL A
0310     PHA
0320     JSR LETNUM
0330     PLA
0340     CLC
0350     ADC FR0
0360     TAY
0370     PLA
0380     TAX
0390     TYA
0400     STA COLPF0S,X
0410     RTS
0420 ERR JSR BVALER
```

Rozpoczyna się ona również od pobrania przez GETBYT numeru rejestru, sprawdzenia jego wartości i odłożenia jej na stos. Następnie przez dwukrotne wywołanie LETNUM odczytywane są dwa pozostałe parametry. Są one sumowane podobnie jak w procedurze XSOUND - kod barwy zajmuje cztery starsze bity, a kod jasności cztery młodsze. Uzyskany wynik jest zapisywany według numeru pobranego ze stosu do odpowiedniego rejestru koloru obrazu COLPF (COLor of Play Field - rejestry 0-3) lub tła COLBAK (COLor of BAcKground - rejestr 4).

6.5.3. Instrukcja COLOR

Instrukcja COLOR ustala numer rejestru koloru lub kod znaku dla następnych instrukcji graficznych. Numer koloru jest jednak inny niż użyty w instrukcji SETCOLOR. Rejestrom 0-3 odpowiadają tu bowiem numery 1-4, zaś rejestrowi 4 (tło) numer 0 (poza trybami graficznymi 9, 10 i 11, które są sterowane przez GTIA). Instrukcja COLOR jest realizowana przez procedurę XCOLOR. Pobiera ona przy pomocy procedury LETNUM liczbę i zapisuje ją do rejestru COLOR,

z którego korzystają inne procedury graficzne. Wartość tej liczby nie jest tu sprawdzana i może mieć dowolną wartość mniejszą od $FFFF (większe wartości spowodują błąd w procedurze LETNUM). Zapisywany jest jednak tylko młodszy bajt kodu, więc w rezultacie maksymalny numer koloru lub kod znaku wynosi $FF.

```
0100 ;eXecute COLOR statement
0110 ;
0120 COLOR = $C8
0130 FR0 =   $D4
0140 LETNUM = $ABD7
0150 ;
0160     *=  $BA1F
0170 ;
0180     JSR LETNUM
0190     LDA FR0
0200     STA COLOR
0210     RTS
```

6.5.4. Instrukcja POSITION

Instrukcja POSITION umieszcza kursor na podanej pozycji. Jest ona realizowana przez procedurę XPOS. Najpierw pozioma pozycja kursora odczytywana jest przy pomocy LETNUM i jej oba bajty przepisywane są do rejestru COLCRS (COLumn of CuRSor). Następnie wywoływana jest procedura GETBYT odczytująca pozycję pionową, która przepisywana jest do rejestru ROWCRS (ROW of CuRSor). Odczyt współrzędnych jest wykonywany przez różne procedury, gdyż maksymalna pozycja pozioma wynosi 319 (liczba dwubajtowa), zaś maksymalna pozycja pionowa 191, a więc jeden bajt.

```
0100 ;eXecute POSITION statement
0110 ;
0120 COLCRS = $55
0130 FR0 =   $D4
0140 GETBYT = $ABE0
0150 LETNUM = $ABD7
0160 ROWCRS = $54
0170 ;
0180     *=  $BA0C
0190 ;
0200     JSR LETNUM
0210     LDA FR0
0220     STA COLCRS
0230     LDA FR0+1
0240     STA COLCRS+1
0250     JSR GETBYT
0260     LDA FR0
0270     STA ROWCRS
0280     RTS
```

6.5.5. Instrukcja PLOT

Instrukcja PLOT umieszcza znak (w trybie tekstowym) lub punkt (w trybie bitowym) we wskazanym miejscu ekranu. Numer koloru lub kod znaku pobierany jest przy tym z rejestru COLOR. Operację tą wykonuje procedura XPLOT. Najpierw przez wywołanie procedury XPOS kursor ustawiany jest na odpowiedniej pozycji. Następnie do akumulatora pobierany jest numer

koloru lub kod znaku, a do rejestru X numer IOCB 6, który służy do przeprowadzania operacji graficznych. Potem wykonywany jest skok do procedury PRPDVC, która przygotowuje i przeprowadza (przy pomocy procedur systemu operacyjnego) operację zapisu punktu lub znaku na ekranie.

```
0100 ;eXecute PLOT statement
0110 ;
0120 COLOR = $C8
0130 PRPDVC = $BA9B
0140 XPOS =  $BA0C
0150 ;
0160     *=  $BA6C
0170 ;
0180     JSR XPOS
0190     LDA COLOR
0200     LDX #$06
0210     JMP PRPDVC
```

6.5.6. Instrukcja DRAWTO

Instrukcja DRAWTO rysuje linię od aktualnej pozycji kursora do pozycji o podanych współrzędnych. Procedura realizacyjna XDRAW rozpoczyna się, podobnie jak XPLOT, od wywołania XPOS i odczytania kodu znaku lub numeru koloru. Ta ostatnia wartość jest teraz przepisywana do rejestru ATACHR (ATASCII CHaRacter). Następnie w akumulatorze umieszczany jest kod operacji rysowania linii ($11), a w rejestrze X numer kanału IOCB ($06). Wartości te są doprowadzane do właściwej postaci i zapisywane do odpowiednich rejestrów RAM przez wywołanie procedury SAVCMD. Po ustawieniu parametrów pomocniczych w rejestrach ICAX1 i ICAX2 (IOCB AuXiliary register) operacja jest wykonywana przez wywołanie procedury CMDEXE. Po jej przeprowadzeniu XDRAW kończy się skokiem do ENDIO, gdzie kontrolowany jest jeszcze status wykonanej operacji.

```
0100 ;eXecute DRAWTO statement
0110 ;
0120 ATACHR = $02FB
0130 CMDEXE = $BD29
0140 COLOR = $C8
0150 ENDIO = $BCBB
0160 ICAX1 = $034A
0170 ICAX2 = $034B
0180 SAVCMD = $BABE
0190 XPOS =  $BA0C
0200 ;
0210     *=  $BA27
0220 ;
0230     JSR XPOS
0240     LDA COLOR
0250     STA ATACHR
0260     LDA #$11
0270     LDX #$06
0280     JSR SAVCMD
0290     LDA #$0C
0300     STA ICAX1,X
0310     LDA #$00
0320     STA ICAX2,X
0330     JSR CMDEXE
```

```
0340     JMP ENDIO
```

6.5.7. Instrukcja GRAPHICS

Instrukcja GRAPHICS służy do wyboru trybu graficznego. Polega to na otwarciu kanału IOCB dla ekranu (Screen) i jest wykonywane przez instrukcję XGRAPH. Wszystkie operacje na ekranie są standardowo wykonywane poprzez kanał szósty, więc najpierw do rejestru IODVC wpisywany jest numer tego kanału i wywoływana jest procedura CLCHN, która go zamyka. Następnie wymagany numer trybu graficznego jest odczytywany przez procedurę LETNUM, a adres nazwy urządzenia SCRNAM jest przepisywany do wektora bufora INBUFP. Teraz numer trybu jest pobierany z rejestru FR0 i umieszczany w akumulatorze, a jego bity 4-7 w rejestrze Y (określają one rodzaj otwarcia urządzenia). Sama operacja otwarcia przeprowadzana jest przez procedurę PRPOPN, a kontrola statusu tej operacji przez ENDIO.

```
0100 ;eXecute GRAPHICS statement
0110 ;
0120 CLCHN = $BCF7
0130 ENDIO = $BCBB
0140 FR0 =   $D4
0150 INBUFP = $F3
0160 IODVC = $C1
0170 LETNUM = $ABD7
0180 PRPOPN = $BBD8
0190 ;
0200     *=  $BA46
0210 ;
0220     LDX #$06
0230     STX IODVC
0240     JSR CLCHN
0250     JSR LETNUM
0260     LDX # <SCRNAM
0270     LDY # >SCRNAM
0280     STX INBUFP
0290     STY INBUFP+1
0300     LDX #$06
0310     LDA FR0
0320     AND #$F0
0330     EOR #$1C
0340     TAY
0350     LDA FR0
0360     JSR PRPOPN
0370     JMP ENDIO
0380 ;
0390 SCRNAM .BYTE "S:",$9B
```

Procedura PRPOPN ustala kod wykonywanej operacji na $03 (OPEN) i zapisuje go do odpowiednich rejestrów przy pomocy SAVCMD. Następnie umieszcza przekazane parametry otwarcia ekranu w rejestrach ICAX1 i ICAX2. Wywołanie systemowej procedury I/O, która otwiera ekran, odbywa się za pośrednictwem SIBUFA (dodatkowo ustala ona adres bufora). Procedura PRPOPN kończy się bezpośrednim skokiem do STBV, gdzie odtwarzana jest poprzednia wartość wektora INBUFP.

```
0100 ;PRePare for OPeN
0110 ;
```

```
0120 ICAX1 = $034A
0130 ICAX2 = $034B
0140 SAVCMD = $BABE
0150 SIBUFA = $BD1E
0160 STBV =  $DA51
0170 ;
0180     *=  $BBD8
0190 ;
0200     PHA
0210     LDA #$03
0220     JSR SAVCMD
0230     PLA
0240     STA ICAX2,X
0250     TYA
0260     STA ICAX1,X
0270     JSR SIBUFA
0280     JMP STBV
```

6.6. Instrukcje wejścia/wyjścia

Ostatnią, najliczniejszą grupę instrukcji Atari Basic stanowią instrukcje wejścia/wyjścia. Służą one do komunikacji komputera z urządzeniami zewnętrznymi, w tym także z edytorem i ekranem. Są to następujące instrukcje: CLOSE, OPEN, XIO, STATUS, PUT, GET, LOCATE, PRINT, LPRINT, POINT, NOTE, INPUT, READ, SAVE, CSAVE, LOAD, CLOAD, ENTER i LIST.

6.6.1. Instrukcja CLOSE

Instrukcja CLOSE zamyka podany kanał wejścia/wyjścia, a więc służy do zakończenia komunikacji przez odpowiedni blok IOCB (zob. "Mapa pamięci Atari XL/XE. Procedury wejścia/wyjścia."). Jest ona realizowana przez procedurę wykonawczą XCLOSE.

```
0100 CMDEXE = $BD2B
0110 ENDIO = $BCBB
0120 IOCMD = $C0
0130 SETDVC = $BCA8
0140 ;
0150     *=  $BC22
0160 ;
0170 ;eXecute CLOSE statement
0180 ;
0190 XCLOSE LDA #$0C
0200 ;
0210 ;PRoCeed I/O
0220 ;
0230 PRCIO STA IOCMD
0240     JSR SETDVC
0250 ;
0260 ;I/O OPERation
0270 ;
0280 IOOPER JSR CMDEXE
0290     JMP ENDIO
```

Procedura ta umieszcza w rejestrze IOCMD (I/O CoMmanD) kod operacji zamknięcia kanału, a następnie przez wywołanie SETDVC ustala numer kanału według wartości odczytanej z bufora wejściowego. Operacja jest realizowana przez wywołanie procedury CMDEXE, a sprawdzenie statusu i obsługę ewentualnego błędu wykonuje procedura ENDIO.

Dwie dodatkowe etykiety - PRCIO i IOOPER - pozwalają na wykorzystanie tej procedury do innych operacji I/O. Po wywołaniu od PRCIO na kanale, którego numer zawiera rejestr X, wykonywana jest operacja, której kod umieszczony jest w akumulatorze. Realizacja operacji I/O, której parametry zostały już wcześniej ustalone, następuje po wywołaniu procedury od etykiety IOOPER.

6.6.2. Instrukcje XIO i OPEN

Instrukcja XIO jest uniwersalną instrukcją wejścia/wyjścia i służy do wykonania dowolnej operacji, której kod zostanie podany jako pierwszy parametr. Kolejnymi parametrami są: numer kanału, dwa parametry pomocnicze i nazwa pliku lub urządzenia. Instrukcja OPEN dokonuje otwarcia kanału IOCB do komunikacji z urządzeniem zewnętrznym. Jej składnia jest taka sama jak instrukcji XIO, z wyjątkiem pierwszego parametru (numer kanału, dwa parametry pomocnicze i nazwa pliku lub urządzenia).

Procedury wykonawcze tych instrukcji - XXIO i XOPEN - mają wspólny przebieg, a różnią się jedynie etapem początkowym. XXIO rozpoczyna się od wywołania procedury GETDAT, która odczytuje kod operacji do wykonania. Na początku procedury XOPEN kod ten jest ustalany na $03 (OPEN). Teraz - od zapisania kodu w rejestrze IOCMD - obie procedury przebiegają wspólnie. Następnie procedura SETDVC odczytuje z bufora numer wykorzystywanego kanału IOCB. Dwie dalsze wartości (parametry pomocnicze) są odczytywane przez kolejne wywołania procedury GETDAT i umieszczane w akumulatorze (pierwszy parametr) i rejestrze Y (drugi parametr). Warto przy tym zwrócić uwagę, że zarówno procedura SETDVC, jak i GETDAT przyjmują bez sygnalizowania błędu dane dwubajtowe, lecz uwzględniany jest tylko młodszy bajt danej.

```
0100 BFLN1 = $BD0F
0110 ENDIO = $BCBB
0120 GETDAT = $BD09
0130 GFILSP = $BD7D
0140 ICAX1 = $034A
0150 ICAX2 = $034B
0160 IOCMD = $C0
0170 MLTCHN = $BCAF
0180 PSTMAD = $BD9D
0190 SETDVC = $BCA8
0200 STBV =  $DA51
0210 CLET =  $AADA
0220 ;
0230     *=  $BBEC
0240 ;
0250 ;eXecute XIO statement
0260 ;
0270 XXIO JSR GETDAT
0280     JMP EXE
0290 ;
0300 ;eXecute OPEN statement
0310 ;
0320 XOPEN LDA #$03
0330 EXE STA IOCMD
0340     JSR SETDVC
```

```
0350     JSR GETDAT
0360     PHA
0370     JSR GETDAT
0380     TAY
0390     PLA
0400 ;
0410 ;OPeN CHaNnel
0420 ;
0430 OPNCHN PHA
0440     TYA
0450     PHA
0460     JSR XLET
0470     JSR GFILSP
0480     JSR MLTCHN
0490     PLA
0500     STA ICAX2,X
0510     PLA
0520     STA ICAX1,X
0530     JSR BFLN1
0540     JSR PSTMAD
0550     JSR STBV
0560     JMP ENDIO
```

Realizacja operacji I/O rozpoczyna się od etykiety OPNCHN. Najpierw wartości przekazane w akumulatorze i rejestrze Y są odkładane na stosie i wywoływana jest procedura XLET. Według obliczonego przez nią adresu procedura GFILSP odczytuje specyfikację pliku lub urządzenia. Następnie poprzez MLTCHN obliczany jest poprawny numer bloku IOCB i zdjęte ze stosu parametry dodatkowe przepisywane są do rejestrów ICAX1 i ICAX2. Ustalenie pozostałych parametrów i wykonanie operacji jest przeprowadzane bezpośrednio przez procedurę BFLN. Przed kończącym procedurę skokiem do ENDIO odtwarzany jest jeszcze przy pomocy STBV wektor bufora wejściowego.

6.6.3. Instrukcja STATUS

Instrukcja STATUS odczytuje status podanego urządzenia lub pliku, a jego wartość przypisuje wskazanej zmiennej. Jej procedurą wykonawczą jest XSTAT.

```
0100 ;eXecute STATUS statement
0110 ;
0120 CIOEXE = $BD2B
0130 GETST = $BD00
0140 SAVBYT = $BD31
0150 SETDVC = $BCA8
0160 ;
0170     *=  $BC2F
0180 ;
0190     JSR SETDVC
0200     LDA #$0D
0210     JSR CIOEXE
0220     JSR GETST
0230     JMP SAVBYT
```

Po odczytaniu przez procedurę SETDVC numeru kanału, do akumulatora wpisywany jest kod operacji STATUS ($0D). Operacja jest realizowana przez wywołanie procedury CIOEXE, a jej status odczytuje procedura GETST. Uzyskany wynik jest zapisywany przy pomocy procedury

SAVBYT.

Najpierw odkłada ona na stosie przekazaną w akumulatorze wartość (a przy wywołaniu od etykiety SAVWRD także wartość z rejestu Y). Następnie przez wywołanie procedury LETVAR odczytywana jest wskazana zmienna. Zdjęte ze stosu wartości wpisywane są do jej dwóch pierwszych bajtów i przez wywołanie procedury IFP zamieniane na liczbę zmiennoprzecinkową. Procedura XSTAT kończy się skokiem do SAVVAL, gdzie uzyskana liczba jest przepisywana do tablicy wartości zmiennych.

```
0100 FR0 =   $D4
0110 IFP =   $D9AA
0120 LETVAR = $AC06
0130 SAVVAL = $AC0C
0140 ;
0150     *=  $BD31
0160 ;
0170 ;SAVe BYTe
0180 ;
0190 SAVBYT LDY #$00
0200 ;
0210 ;SAVe WoRD
0220 ;
0230 SAVWRD PHA
0240     TYA
0250     PHA
0260     JSR LETVAR
0270     PLA
0280     STA FR0+1
0290     PLA
0300     STA FR0
0310     JSR IFP
0320     JMP SAVVAL
```

6.6.4. Instrukcja PUT

Zadaniem instrukcji PUT jest zapisanie we wskazanym kanale IOCB podanej wartości. Czynność tą wykonuje procedura XPUT, która rozpoczyna się od ustalenia przy pomocy SETDVC odpowiedniego numeru kanału.

```
0100 ;eXecute PUT statement
0110 ;
0120 FR0 =   $D4
0130 IODVC = $C1
0140 LETNUM = $ABD7
0150 PRPDVC = $BA9B
0160 SETDVC = $BCA8
0170 ;
0180     *=  $BC78
0190 ;
0200     JSR SETDVC
0210     JSR LETNUM
0220     LDA FR0
0230     LDX IODVC
0240     JMP PRPDVC
```

Następnie w akumulatorze umieszczany jest odczytany przez procedurę LETNUM bajt do zapisu, zaś w rejestrze X numer kanału pobrany z IODVC (I/O DeViCe). Procedura XPUT kończy się bezpośrednim skokiem do PRPDVC, gdzie dokonuje się realizacja zapisu i kontrola statusu tej operacji.

6.6.5. Instrukcje GET i LOCATE

Zadaniem instrukcji GET jest odczytanie ze wskazanego kanału IOCB wartości i przypisanie jej podanej zmiennej. Instrukcja LOCATE wykonuje podobne zadanie, lecz odczyt następuje zawsze z ekranu i z podanej pozycji kursora. Procedury wykonawcze tych instrukcji - XGET i XLOCAT - są więc również podobne.

```
0100 BFLN3 = $BD15
0110 ENDIO = $BCBB
0120 INBUFP = $F3
0130 IOCMD = $C0
0140 SAVBYT = $BD31
0150 SAVDVC = $BAC0
0155 SETDVC = $BCA8
0160 STBV =  $DA51
0170 XPOS =  $BA0C
0180 ;
0190     *=  $BC85
0200 ;
0210 ;eXecute GET statement
0220 ;
0230 XGET JSR STBV
0240     JSR SETDVC
0250 EXE LDA #$07
0260     STA IOCMD
0270     LDY #$01
0280     JSR BFLN3
0290     JSR ENDIO
0300     LDY #$00
0310     LDA (INBUFP),Y
0320     JMP SAVBYT
0330 ;
0340 ;eXecute LOCATE statement
0350 ;
0360 XLOCAT JSR XPOS
0370     LDX #$06
0380     JSR SAVDVC
0390     BNE EXE
```

Procedura XGET na początku ustawia przy pomocy STBV adres bufora dla operacji I/O oraz numer kanału IOCB (przez wywołanie SETDVC). Natomiast procedura XLOCAT najpierw przez wywołanie XPOS umieszcza kursor na odpowiedniej pozycji ekranu, a następnie przy pomocy SAVDVC ustala numer kanału na 6. Dalej obie procedury przebiegają identycznie. Do rejestru IOCMD wpisywany jest kod operacji GET BYTE ($07), a procedura BFLN ustala długość bufora na jeden znak. Po sprawdzeniu poprawności operacji przy pomocy ENDIO odczytana liczba jest umieszczana w akumulatorze. Procedury XGET i XLOCAT są opuszczane skokiem do SAVBYT, która przypisuje otrzymaną wartość wskazanej zmiennej.

6.6.6. Instrukcja PRINT

Instrukcja PRINT służy do zapisywania dowolnych wartości na urządzeniu zewnętrznym. Numer tego urządzenia powinien być podany jako pierwszy parametr i oznaczony znakiem "#". W przypadku braku numeru urządzenia interpreter przyjmuje, że zapis ma być dokonany do kanału IOCB o numerze 0 czyli do edytora. W instrukcji PRINT poza wartościami mogą znajdować się separatory, które służą do nadania odpowiedniej formy zapisywanej informacji. Separatorami typi są: średnik (;), przecinek (,), dwukropek (:) i znak końca wiersza (EOL - End Of Line). Procedura wykonawcza XPRINT, która realizuje tą instrukcję, rozpoznaje kolejno w pętli elementy (wartości i separatory) oraz realizuje ich zapis. Przed rozpoczęciem pętli skok tabulacji (liczba znaków między sąsiednimi pozycjami tabulacji) jest przepisywany z rejestru PTABW (Position TABulate Width) do AUXBR, a licznik wyjściowy COX (Current Output indeX) jest zerowany.

Pierwszą czynnością w każdym przejściu pętli jest odczyt tokena do zapisu i jego rozpoznanie. Jeśli nie jest to token żadnego separatora, to wywoływane są procedury XLET i GETVAR, które kolejno obliczają wartość do zapisu i pobierają ją z bufora wejściowego. Rodzaj wartości (liczbowa czy tekstowa) jest rozpoznawany według najstarszego bitu rejestru VART.

Wartości liczbowe są najpierw doprowadzane do standardowego formatu, w którym przedstawiane jest dziewięć cyfr znaczących. Po zamianie tej liczby na ciąg znaków ASCII (przez procedurę FASC) kolejne znaki są odczytywane z bufora i zapisywane przy pomocy procedury RSTHIB (zob. niżej). Ostatni znak liczby jest sygnalizowany przez ustawienie w nim najstarszego bitu. Po jego zapisaniu następuje skok do początku głównej pętli (PRNT).

Przed zapisem wartości tekstowej wywoływana jest procedura FTARV, która odczytuje parametry ciągu. Następnie kolejne znaki ciągu są zapisywane przez procedurę INOUTX. Jako licznik znaków służą w tym przypadku trzeci i czwarty bajt informacji o ciągu, które zawierają jego aktualną długość. Po zapisaniu całego żądanego tekstu następuje powrót do pętli głównej.

Rozpoznanie tokena numeru kanału ($1C) powoduje wywołanie procedury IXGDAT. Odczytany przez nią numer IOCB jest przepisywany do rejestru IOCHN i procedura XPRINT przechodzi do rozpoznawania następnego tokena.

Średnik (;) powoduje zapisanie następnej wartości bezpośrednio po ostatnio zapisanej. Po odczytaniu jego tokena ($15) sprawdzany jest następny token. Jeżeli jest to znak inny niż dwukropek (:) lub koniec wiersza (znak RETURN), to powtarzana jest główna pętla procedury. W przeciwnym przypadku zerowany jest rejestr IOCHN (odtwarza to standardowe urządzenie wyjścia - edytor) i procedura XPRINT się kończy.

W każdym innym - niż opisany powyżej - przypadku rozpoznanie tokena kończącego instrukcję PRINT (dwukropek - $14 lub koniec wiersza - $16) powoduje wywołanie procedury INOUTX (wewnątrz RSTHIB), która zapisuje znak RETURN ($9B). Po wyzerowaniu rejestru IOCHN procedura XPRINT także się kończy.

```
0100 ;eXecute PRINT statement
0110 ;
0120 AUXBR = $AF
0130 CIX =   $F2
0140 COX =   $94
0150 FASC =  $D8E6
0160 FR0 =   $D4
0170 FTARV = $AB90
0180 GETVAR = $ABE9
0190 INBUFP = $F3
0200 INIX =  $A8
0210 INOUTX = $B491
0220 IOCHN = $B5
0230 IXGDAT = $BD07
0240 PTABW = $C9
0250 RSTHIB = $B48F
0260 STMCUR = $8A
0270 VART =  $D2
0280 XLET =  $AADA
0290 ;
0300     *=  $B3DA
0310 ;
0320     LDA PTABW
0330     STA AUXBR
0340     LDA #$00
0350     STA COX
0360 PRNT LDY INIX
0370     LDA (STMCUR),Y
0380     CMP #$12          ,
0390     BEQ COM
0400     CMP #$16          EOL
0410     BEQ EOL
0420     CMP #$14          :
0430     BEQ EOL
0440     CMP #$15          ;
0450     BEQ SCL
0460     CMP #$1C          #
0470     BEQ DVC
0480     JSR XLET
0490     JSR GETVAR
0500     DEC INIX
0510     BIT VART
0520     BMI ARR
0530     LDA FR0+1
0540     CMP #$10
0550     BCC ASC
0560     LDA FR0+5
0570     AND #$F0
0580     STA FR0+5
0590 ASC JSR FASC
0600     LDA #$00
0610     STA CIX
0620 DIG LDY CIX
0630     LDA (INBUFP),Y
0640     PHA
0650     INC CIX
0660     JSR RSTHIB
0670     PLA
0680     BPL DIG
0690     BMI PRNT
0700 ARR JSR FTARV+3
```

```
0710     LDA #$00
0720     STA CIX
0730 LOOP LDA FR0+2
0740     BNE DCN
0750     DEC FR0+3
0760     BMI PRNT
0770 DCN DEC FR0+2
0780     LDY CIX
0790     LDA (FR0),Y
0800     INC CIX
0810     BNE NXT
0820     INC FR0+1
0830 NXT JSR INOUTX
0840     JMP LOOP
0850 COM LDY COX
0860     INY
0870     CPY AUXBR
0880     BCC TAB
0890     CLC
0900     LDA PTABW
0910     ADC AUXBR
0920     STA AUXBR
0930     BCC COM
0940 TAB LDY COX
0950     CPY AUXBR
0960     BCS SCL
0970     LDA #$20
0980     JSR RSTHIB
0990     JMP TAB
1000 EOL JMP RET           EOL & :
1010 DVC JSR IXGDAT        #
1020     STA IOCHN
1030     DEC INIX
1040     JMP PRNT
1050 SCL INC INIX          ;
1060     LDY INIX
1070     LDA (STMCUR),Y
1080     CMP #$16          EOL
1090     BEQ END
1100     CMP #$14          :
1110     BEQ END
1120     JMP PRNT
1130 RET LDA #$9B
1140     JSR INOUTX
1150 END LDA #$00
1160     STA IOCHN
1170     RTS
```

Token przecinka ($12) nakazuje przejście do następnej pozycji tabulacji. Jest to realizowane przy pomocy pętli, w której procedura RSTHIB zapisuje spacje (po jednej w każdym przejściu pętli). Licznikiem tej pętli jest rejestr AUXBR, który zawiera wielokrotność kroku tabulacji (przepisanego z rejestru PTABW) zmniejszoną o aktualną pozycję kursora w zapisywanym wierszu. Gdy wszystkie wymagane spacje zostaną już zapisane, następuje przeskok do fragmentu procedury XPRNT realizującego token średnika (;).

Procedura RSTHIB (i jej fragment oznaczony etykietą INOUTX) zwiększa stan licznika wyjściowego COX i przekazuje znak podany w akumulatorze do procedury PRPCHN. W ten

sposób znak z akumulatora zostaje wysłany do urządzenia zewnętrznego. Przy wywołaniu od RSTHIB dodatkowo w zapisywanym znaku kasowany jest najstarszy bit.

```
0100 COX =   $94
0110 PRPCHN = $BA99
0120 ;
0130     *=  $B48F
0140 ;
0150 ;ReSeT HIgh Bit
0160 ;
0170 RSTHIB AND #$7F
0180 ;
0190 ;INcrease OUTput indeX
0200 ;
0210 INOUTX INC COX
0220     JMP PRPCHN
```

6.6.7. Instrukcja LPRINT

Instrukcja LPRINT ma podobne zadanie jak PRINT, lecz w tym przypadku zapis wykonywany jest zawsze na drukarkę i przez kanał IOCB 7. Realizuje to procedura XLPRNT. Rozpoczyna ją ustawienie wektora INBUFP na nazwę urządzenia PRTNAM (drukarka - P:) i wpisanie do rejestru IOCHN numeru kanału ($07). Następnie urządzenie jest otwierane do zapisu przez wywołanie procedury PRPOPN z wartością $08 (zapis) w rejestrze Y i $00 w akumulatorze. Po sprawdzeniu poprawności otwarcia przez procedurę ENDIO właściwy zapis realizowany jest przez XPRINT. Po wykonaniu operacji wykorzystywany kanał IOCB jest zamykany przez procedurę CLCHN.

```
0100 ;eXecute LPRINT statement
0110 ;
0120 CLCHN = $BCF7
0130 ENDIO = $BCBB
0140 INBUFP = $F3
0150 IOCHN = $B5
0160 PRPOPN = $BBD8
0170 XPRINT = $B3DA
0180 ;
0190     *=  $B496
0200 ;
0210     LDA # <PRTNAM
0220     STA INBUFP
0230     LDA # >PRTNAM
0240     STA INBUFP+1
0250     LDX #$07
0260     STX IOCHN
0270     LDA #$00
0280     LDY #$08
0290     JSR PRPOPN
0300     JSR ENDIO
0310     JSR XPRINT
0320     JMP CLCHN
0330 ;
0340 PRTNAM .BYTE "P:",$9B
```

6.6.8. Instrukcja POINT

Instrukcja POINT ustawia głowicę zapisująco-odczytującą stacji dysków na podanym bajcie we wskazanym sektorze. Jej procedurą wykonawczą jest XPOINT, która po ustaleniu potrzebnych parametrów wywołuje systemową procedurę CIOMAIN.

```
0100 ;eXecute POINT statement
0110 ;
0120 FR0 =   $D4
0130 GLNNUM = $ABCD
0140 ICAX3 = $034C
0150 ICAX4 = $034D
0160 ICAX5 = $034E
0170 IOCMD = $C0
0180 IOOPER = $BC29
0190 MLTCHN = $BCAF
0200 SETDVC = $BCA8
0210 ;
0220     *=  $BC54
0230 ;
0240     JSR SETDVC
0250     JSR GLNNUM
0260     JSR MLTCHN
0270     LDA FR0
0280     STA ICAX3,X
0290     LDA FR0+1
0300     STA ICAX4,X
0310     JSR GLNNUM
0320     JSR MLTCHN
0330     LDA FR0
0340     STA ICAX5,X
0350     LDA #$25
0360     STA IOCMD
0370     BNE IOOPER
```

Pierwszym parametrem jest numer kanału IOCB, który jest pobierany z bufora wejściowego przez procedurę SETDVC. Pozostałe parametry odczytywane są przez dwukrotne wywołanie procedury GLNNUM, przy czym procedura MLTCHN służy każdorazowo do obliczenia poprawnego adresu IOCB. Odczytane wartości są umieszczane w rejestrach ICAX3 i ICAX4 (numer sektora) oraz ICAX5 (numer bajtu w sektorze). Po tych przygotowaniach do rejestru IOCMD wpisywany jest kod operacji POINT ($25) i wykonywany jest skok do procedury IOOPER, która wywołuje systemową procedurę I/O.

6.6.9. Instrukcja NOTE

Odwrotnością POINT jest instrukcja NOTE. Odczytuje ona pozycję głowicy stacji dysków i uzyskane rezultaty przypisuje dwóm wskazanym zmiennym. Procedura wykonawcza tej instrukcji - XNOTE - jest także odwróceniem procedury XPOINT.

```
0100 ;eXecute NOTE statement
0110 ;
0120 ICAX3 = $034C
0130 ICAX4 = $034D
0140 ICAX5 = $034E
0150 MLTCHN = $BCAF
0160 PRCIO = $BC24
```

```
0170 SAVBYT = $BD31
0180 SAVWRD = $BD33
0190 ;
0200     *=  $BC3D
0210 ;
0220     LDA #$26
0230     JSR PRCIO
0240     LDA ICAX3,X
0250     LDY ICAX4,X
0260     JSR SAVWRD
0270     JSR MLTCHN
0280     LDA ICAX5,X
0290     JMP SAVBYT
```

Rozpoczyna ją wywołanie procedury PRCIO z umieszczonym w akumulatorze kodem operacji NOTE ($26). Numer sektora, w którym aktualnie znajduje się głowica stacji dysków, odczytywany jest z rejestrów ICAX3 i ICAX4, a następnie przekazywany do odpowiedniej zmiennej przez procedurę SAVWRD. Analogicznie, procedura SAVBYT przypisuje zmiennej numer bajtu odczytany z rejestru ICAX5.

6.6.10. Instrukcje INPUT i READ

Instrukcje INPUT i READ służą do odczytu informacji, przy czym INPUT dokonuje odczytu z urządzenia zewnętrznego (w tym także z edytora), zaś READ odczytuje dane zawarte w instrukcjach DATA. Odczytane dane są przypisywane umieszczonym w instrukcji zmiennym. W obu przypadkach dozwolony jest jednoczesny (w jednej instrukcji) odczyt dowolnej liczby danych dowolnego typu. Procedury wykonawcze tych instrukcji są ze sobą mocno powiązane i dlatego też muszą być opisywane razem.

Procedura wykonawcza instrukcji READ - XREAD - rozpoczyna się od wpisania adresu wiersza z instrukcją DATA jako wektora bufora wejściowego. W tym celu aktualny numer wiersza jest zapamiętywany przy pomocy procedury SAVSTM, a zawartość rejestru DATALN (DATA LiNe Number) jest przepisywana do CLNN (Current LiNe Number). Następnie procedura FNDCST odszukuje ten wiersz w pamięci, a jego adres z rejestru STMCUR (StaTeMent CURrent address) jest przepisywany do wektora INBUFP (INput BUFfer Pointer). Teraz wywołanie procedury XRTRN odtwarza adres aktualnie wykonywanego wiersza z instrukcją READ. Podczas tych operacji stan licznika wejściowego INIX jest przechowywany na stosie. Trzeba tu zwrócić uwagę, że rejestr DATALN w momencie uruchomienia programu zawiera wartość $00, a później może być zmieniony tylko przez instrukcję READ (zob. dalej) i RESTORE (zob. rozdział 6.2.3).

Teraz przez kilkakrotne wywoływanie procedury NXTDAT i stanowiącej jej fragment SENDDT odczytywane są parametry znalezionego wiersza. Są to kolejno: dwa bajty numeru wiersza przepisywane do DATALN, długość wiersza przepisywana do pierwszego bajtu rejestru ZTEMP1 i długość instrukcji przepisywana do drugiego bajtu ZTEMP1. Procedura NXTDAT nie jest tu jedank wykorzystywana zgodnie ze swoim przeznaczeniem. Zasadniczo służy ona do poszukiwania końca odczytywanej danej, bowiem w rezultacie jej wywołania bit Zero statusu procesora jest ustawiany,

gdy odczytanym znakiem jest przecinek (,) lub koniec wiersza (EOL). W każdym innym przypadku bit Zero pozostaje skasowany.

```
0100 CIX =   $F2
0110 INBUFP = $F3
0120 ;
0130     *=  $B32D
0140 ;
0150 ;NeXT DATa
0160 ;
0170 NXTDAT INC CIX
0180 ;
0190 ;Search for END of DaTa
0200 ;
0210 SENDDT LDY CIX
0220     LDA (INBUFP),Y
0230     CMP #',
0240     CLC
0250     BEQ END
0260     CMP #$9B
0270 END RTS
```

Ostatnie w tej fazie wywołanie NXTDAT zwraca token pierwszej instrukcji w wierszu. Jeżeli nie jest to instrukcja DATA (token $01), poszukiwany jest token następnej instrukcji w tym samym wierszu. Po dotarciu do końca wiersza zerowany jest znacznik DATAD i przeszukiwany jest następny wiersz programu, aż do napotkania wiersza o numerze przekraczającym $7FFF. W takim przypadku przez skok do procedury ODATER sygnalizowany jest błąd (Out of DATa ERror). Następny wiersz poszukiwany jest także wtedy, gdy stan licznika DATAD przekracza wartość pierwszego bajtu rejestru ZTEMP1. Po znalezieniu danej, która ma zostać odczytana, w rejestrze SXFLG (SyntaX FLaG) umieszczana jest wartość $40, która pozwoli później rozpoznać rodzaj wykonywanej instrukcji.

```
0100 ACHNN = $B4
0110 AFP =   $D800
0120 BINPER = $B924
0130 CIX =   $F2
0140 CLNN =  $A0
0150 DATAD = $B6
0160 DATALN = $B7
0170 FNDCST = $A9A2
0180 GETIX = $B904
0190 GIRQST = $A9F2
0200 INBUFP = $F3
0210 INIX =  $A8
0220 INPREC = $BDE4
0230 IXGDAT = $BD07
0240 NXTDAT = $B32D
0250 ODATER = $B928
0260 PROMPT = $C2
0270 PUTVAR = $ABB2
0280 RECVAL = $AB36
0290 RSTPTR = $AB26
0300 SAVSTM = $B6F9
0310 SAVVAL = $AC0C
0320 SENDDT = $B32F
0330 STBV =  $DA51
0340 STMCUR = $8A
0610     STA INIX
0620 LOOP LDY #$00
0630     STY CIX
0640     JSR SENDDT
0650     STA DATALN
0660     JSR NXTDAT
0670     STA DATALN+1
0680     JSR NXTDAT
0690     STA ZTEMP1
0700 NXT1 JSR NXTDAT
0710     STA ZTEMP1+1
0720     JSR NXTDAT
0730     EOR #$01
0740     BEQ SDT
0750     LDY ZTEMP1+1
0760     CPY ZTEMP1
0770     BCS BPS1
0780     DEY
0790     STY CIX
0800     BCC NXT1
0810 BPS1 STY CIX
0820     DEC CIX
0830 GET LDY #$01
0840     LDA (INBUFP),Y
0850     BMI RERR
```

```
0350 STTXT = $AB5C
0360 SXFLG = $A6
0370 VART =  $D2
0380 XRTRN = $BDA8
0390 XSLET = $AE8E
0400 XSTOP = $B792
0410 ZTEMP1 = $F5
0420 ;
0430     *=  $B2AE
0440 ;
0450 ;eXecute READ statement
0460 ;
0470 XREAD LDA INIX
0480     PHA
0490     JSR SAVSTM
0500     LDA DATALN
0510     STA CLNN
0520     LDA DATALN+1
0530     STA CLNN+1
0540     JSR FNDCST
0550     LDA STMCUR
0560     STA INBUFP
0570     LDA STMCUR+1
0580     STA INBUFP+1
0590     JSR XRTRN
0600     PLA

1120 ;
1130 ;eXecute INPUT statement
1140 ;
1150 XINPUT LDA #'?
1160     STA PROMPT
1170     JSR RECVAL
1180     DEC INIX
1190     BCC INP
1200     JSR IXGDAT
1210     STA ACHNN
1220 INP JSR STBV
1230     JSR INPREC
1240     JSR GIRQST
1250     BEQ BREAK
1260     LDY #$00
1270     STY SXFLG
1280     STY CIX
1290 RTV JSR RECVAL
1300     INC INIX
1310     LDA VART
1320     BMI TEXT
1330     JSR AFP
1340     BCS IERR
1350     JSR SENDDT
1360     BNE IERR
1370     JSR SAVVAL
1380     JMP CKE
1390 BREAK JMP XSTOP
1400 IERR LDA #$00
1410     STA ACHNN
1420     JSR BINPER
1430 TEXT JSR RSTPTR
1440     JSR PUTVAR
1450     DEC CIX

0860     SEC
0870     LDA CIX
0880     ADC INBUFP
0890     STA INBUFP
0900     LDA #$00
0910     STA DATAD
0920     ADC INBUFP+1
0930     STA INBUFP+1
0940     BCC LOOP
0950 SDT STA ZTEMP1
0960 NDT LDA ZTEMP1
0970     CMP DATAD
0980     BCS NLN
0990 NXT2 JSR NXTDAT
1000     BNE NXT2
1010     BCS GET
1020     INC ZTEMP1
1030     BNE NDT
1040 NLN LDA #$40
1050     STA SXFLG
1060     INC CIX
1070     BCS RTV
1080 ;
1090     *=  $B33B
1100 ;
1110 RERR JSR ODATER

1480     LDX #$FF
1490 NST INX
1500     JSR NXTDAT
1510     BNE NST
1520     BCS FND
1530     BIT SXFLG
1540     BVC NST
1550 FND LDY ZTEMP1
1560     LDA INIX
1570     PHA
1580     TXA
1590     LDX #INBUFP
1600     JSR STTXT
1610     PLA
1620     STA INIX
1630     JSR XSLET+3
1640 CKE BIT SXFLG
1650     BVC CKI
1660     INC DATAD
1670     JSR GETIX
1680     BCS END
1690     JSR SENDDT
1700     BCC BPS2
1710     JMP GET
1720 CKI JSR GETIX
1730     BCC NXGT
1740 END JSR STBV
1750     LDA #$00
1760     STA ACHNN
1770     RTS
1780 NXGT JSR SENDDT
1790     BCC BPS2
1800     JMP INP
1810 BPS2 INC CIX
```

```
1460     LDA CIX             1820     JMP RTV
1470     STA ZTEMP1
```

Odmienny jest początek procedury wykonawczej instrukcji INPUT - XINPUT. Najpierw w rejestrze PROMPT (PROMPT character) umieszczany jest kod znaku zapytania (?) i wywoływana jest procedura RECVAL. Rozpoznaje ona wartość stanowiącą pierwszy parametr instrukcji. Jeśli jest to znak "#", to następująca po nim liczba jest odczytywana przez procedurę IXGDAT i przepisywana do rejestru ACHNN (Auxiliary CHaNnell Number). Następnie odtwarzany jest przy pomocy STBV wektor bufora wejściowego i wywoływana jest procedura INPREC. Pobiera ona rekord (ciąg znaków zakończony przez RETURN) z urządzenia zewnętrznego. Jeśli urządzeniem tym jest edytor (IOCB 0), to najpierw na ekranie jest umieszczany znak z rejestru PROMPT.

Po zakończeniu INPREC wywoływana jest procedura GIRQST, która sprawdza, czy wykonywana operacja nie została przerwana przez naciśnięcie klawisza BREAK. W takim przypadku wykonywany jest skok do procedury XSTOP, która przerywa działanie programu. Po poprawnym odczytaniu rekordu do rejestru SXFLG wpisywana jest wartość $00, która oznacza instrukcję INPUT (zob. wyżej).

Niektóre źródła podają, że przez zmianę wartości w rejestrze PROMPT można zmienić znak wyświetlany przez instrukcję INPUT. W świetle tego, co zostało wyżej napisane, jest to niemożliwe, gdyż przed wykonaniem INPUT do rejestru PROMPT wpisywany jest kod znaku zapytania. Taka zmiana wymaga więc zmiany w samym interpreterze Atari Basic.

```
0100 ACHNN = $B4
0110 BFLN1 = $BD0F
0120 ENDIO = $BCBB
0130 PROMPT = $C2
0140 PRPCHN = $BA99
0150 SAVCMD = $BABE
0160 ;
0170     *=  $BDDB
0180 ;
0190 ;PUT RETurn
0200 ;
0210 PUTRET LDX ACHNN
0220     BNE GETREC
0230     LDA #$9B
0240     JSR PRPCHN
0250 ;
0260 ;INPut RECord
0270 ;
0280 INPREC LDX ACHNN
0290     BNE GETREC
0300     LDA PROMPT
0310     JSR PRPCHN
0320 ;
0330 ;GET RECord
0340 ;
0350 GETREC LDX ACHNN
0360     LDA #$05
0370     JSR SAVCMD
0380     JSR BFLN1
0390     JMP ENDIO
```

Od tego miejsca przebieg obu procedur jest prawie jednakowy. Najpierw przez wywołanie procedury RECVAL rozpoznawany jest typ zmiennej. Zmienna liczbowa jest zamieniana przy pomocy procedury AFP z ciągu cyfr w kodzie ASCII na postać zmiennoprzecinkową. Gdy podczas tej zamiany wystąpi błąd lub następny znak nie oznacza końca danej, to procedura jest przerywana skokiem do BINPER, gdzie sygnalizowany jest błąd (Bad INPut ERror). Poprawna wartość jest natomiast przypisywana odpowiedniej zmiennej przez wywołanie procedury SAVVAL.

W przypadku zmiennej tekstowej jej parametry są umieszczane przy pomocy procedury PUTVAR w buforze wyjściowym odtworzonym przez wywołanie RSTPTR. Następnie jest odszukiwany koniec zmiennej przez kolejne wywołania procedury NXTDAT, przy czym dla instrukcji INPUT znakiem końca danej jest tylko znak RETURN (EOL), zaś dla READ także przecinek. Teraz cały ciąg jest przepisywany do bufora wejściowego przez procedurę STTXT, a potem przypisywany odpowiedniej zmiennej przez wywołanie XSLET.

Teraz procedura GETIX sprawdza, czy jest to koniec instrukcji, w przypadku instrukcji READ jest ponadto zwiększany licznik DATAD (DATa ADdress). Jeżeli cała instrukcja została wykonana, to procedura STBV odtwarza wektor bufora, zerowany jest rejestr ACHNN, a procedury XREAD i XINPUT kończą się rozkazem RTS. Jeśli konieczne są następne dane, to dla instrukcji INPUT są one ponownie pobierane z podanego urządzenia (skok do etykiety INP). Kolejna dana dla instrukcji READ jest odczytywana, gdy koniec poprzedniej był oznaczony przecinkiem (skok do etykiety RTV). W innym przypadku poszukiwana jest następna instrukcja DATA (skok do GET).

6.6.11. Instrukcje SAVE i CSAVE

Instrukcja SAVE służy do zapisu stokenizowanego programu w Atari Basic na dowolnym urządzeniu zewnętrznym. Podobna do niej instrukcja CSAVE dokonuje tego zapisu zawsze na magnetofon. Procedury wykonawcze tych instrukcji - XSAVE i XCSAVE - różnią się jedynie sposobem otwarcia kanału 7, a ich dalszy przebieg (procedura SAVE) jest identyczny. Procedury te (a także procedury wykorzystywane przez inne instrukcje) korzystają z kanału IOCB 7. Należy więc unikać używania tego kanału we własnych programach.

```
0100 ;OPeN CHannel #7
0110 ;
0120 CIOEXE = $BD2B
0130 IOCMD = $C0
0140 IODVC = $C1
0150 MLTCHN = $BCAF
0160 OPNCHN = $BC02
0170 ;
0180     *=  $BAD7
0190 ;
0200     PHA
0210     LDY #$07
0220     STY IODVC
0230     JSR MLTCHN
0240     LDA #$0C
0250     JSR CIOEXE
0260     LDY #$03
0270     STY IOCMD
```

```
0280     PLA
0290     LDY #$00
0300     JSR OPNCHN
0310     LDA #$07
0320     RTS
```

Procedura XSAVE po umieszczeniu w akumulatorze wartości $08, która wskazuje zapis, wywołuje procedurę OPNCH7. Tu najpierw kanał IOCB 7 jest zamykany, a dopiero potem ponownie otwierany do zapisu. Ponieważ jako drugi parametr pomocniczy (do rejestru ICAX2) jest przesyłana do procedury OPNCHN wartość $00, to w przypadku magnetofonu zapis będzie się odbywał z długimi przerwami. Wykonywane tu zamknięcie kanału przed jego otwarciem pozwala ustrzec się przed ewentualnymi błędami, których wystąpienie jest możliwe przy stosowaniu instrukcji CSAVE (zob. niżej).

```
0100 ;OPeN CASsette
0110 ;
0120 ENDIO = $BCBB
0130 INBUFP = $F3
0140 PRPOPN = $BBD8
0150 ;
0160     *=  $BBB4
0170 ;
0180     NOP
0190     NOP
0200     PHA
0210     LDX # <CASNAM
0220     STX INBUFP
0230     LDX # >CASNAM
0240     STX INBUFP+1
0250     LDX #$07
0260     PLA
0270     TAY
0280     LDA #$80
0290     JSR PRPOPN
0300     JSR ENDIO
0310     LDA #$07
0320     RTS
0330 ;
0340 CASNAM .BYTE "C:",$9B
```

Procedura XCSAVE także umieszcza w akumulatorze wartość $08, lecz następnie wywołuje procedurę OPNCAS. Nie zamyka ona kanału 7 przed jego otwarciem, co może być przyczyną błędów, jeśli kanał ten jest wykorzystywany w programie. Sama operacja otwarcia jest przeprowadzona podobnie jak w opisanej wcześniej procedurze XLPRNT (instrukcja LPRINT). Trzeba tu jednak zwrócić uwagę na wartość $80 przekazywaną do PRPOPN. Jest ona tam następnie przepisywana do rejestru ICAX2 i powoduje zapis z krótkimi przerwami pomiędzy rekordami na kasecie.

```
0100 BFLN3 = $BD15
0110 BFLN4 = $BD17
0120 CLCHN = $BCF7
0130 ENDIO = $BCBB
0140 INBUFP = $F3
0150 IOCMD = $C0
0160 LBUFF = $0580
```

```
0170 LOMEM = $80
0180 MLTCHN = $BCAF
0190 OPNCAS = $BBB4
0200 OPNCH7 = $BAD7
0210 RUNSTK = $8E
0220 VNTP =  $82
0230 ;
0240     *=  $BB6D
0250 ;
0260 ;eXecute SAVE statement
0270 ;
0280 XSAVE LDA #$08
0290     JSR OPNCH7
0300 ;
0310 ;SAVE routine
0320 ;
0330 SAVE LDA #$0B
0340     STA IOCMD
0350     LDX #LOMEM
0360 LOOP SEC
0370     LDA $00,X
0380     SBC LOMEM
0390     STA [LBUFF-LOMEM],X
0400     INX
0410     LDA $00,X
0420     SBC LOMEM+1
0430     STA [LBUFF-LOMEM],X
0440     INX
0450     CPX #RUNSTK
0460     BCC LOOP
0470     JSR MLTCHN
0480     LDY #$0E
0490     JSR BFLN3
0500     JSR ENDIO
0510 ;
0520 ;PRoGraM area TRansmit
0530 ;
0540 PRGMTR JSR MLTCHN
0550     LDA VNTP
0560     STA INBUFP
0570     LDA VNTP+1
0580     STA INBUFP+1
0590     LDY LBUFF+$0D
0600     DEY
0610     TYA
0620     LDY LBUFF+$0C
0630     JSR BFLN4
0640     JSR ENDIO
0650     JMP CLCHN
0660 ;
0670     *=  $BBD1
0680 ;
0690 ;eXecute CSAVE statement
0700 ;
0710 XCSAVE LDA #$08
0720     JSR OPNCAS
0730     BNE SAVE
```

Procedura SAVE po zapisaniu do rejestru IOCMD kodu operacji PUT BYTE ($0B), realizuje zapisanie programu w dwóch etapach. W pierwszym etapie zapisywane są względne wartości

wektorów stosowanych przez interpreter (LOMEM, VNTP, VNTD, VVTP, STMTAB, STMCUR i STARP). Wartości względne są uzyskiwane przez odjęcie od każdego wektora wielkości LOMEM. Dzięki temu program będzie działał poprawnie po odczycie, niezależnie od aktualnej wartości dolnej granicy dostępnej pamięci RAM. Czyni to program całkowicie relokowalnym i pozwala na zainstalowanie przed odczytem programu DOS-u lub dodatkowych procedur pomocniczych w języku maszynowym (tzw. akcesoriów). Po przepisaniu tych wektorów do bufora LBUFF (Line BUFFer) są one zapisywane na urządzenie przez wywołanie procedury BFLN. Poprawność tego zapisu jest sprawdzana przy pomocy procedury ENDIO.

W drugim etapie zapisywana jest cała treść programu wraz ze zmiennymi (tablica nazw zmiennych, tablica wartości zmiennych i tablica instrukcji). W tym celu jako adres bufora do zapisu podawana jest zawartość wektora VNTP, a długość bufora określana jest według zmniejszonej o jeden względnej wartości wektora STARP. Tak określona zawartość bufora jest zapisywana przez ponowne wywołanie procedury BFLN. Po sprawdzeniu poprawności wykonanej operacji przez ENDIO procedura SAVE kończy się skokiem do CLCHN, gdzie wykorzystywany kanał IOCB jest zamykany.

6.6.12. Instrukcje LOAD i CLOAD

Instrukcje LOAD i CLOAD służą do odczytu stokenizowanego programu i stanowią odpowiedniki instrukcji SAVE i CSAVE. Poza punktami wywołań XLOAD i XCLOAD ich procedura wykonawcza posiada jeszcze etykietę LDPRGM. Służy ona do realizacji odczytu przez instrukcję RUN w przypadku uruchamiania programu z urządzenia zewnętrznego. Rodzaj instrukcji jest rozpoznawany według wartości odkładanej na stosie na początku procedury ($00 dla LOAD i CLOAD, $FF dla RUN).

```
0100 BFLN3 = $BD15
0110 ENDIO = $BCBB
0120 IOCMD = $C0
0130 LBUFF = $0580
0140 LOADER = $B90A
0150 LOADFLG = $CA
0160 LOMEM = $80
0170 MEMTOP = $02E5
0180 MLTCHN = $BCAF
0190 OPNCAS = $BBB4
0200 OPNCH7 = $BAD7
0210 PRGMTR = $BB98
0220 PRLGER = $B90E
0230 STARP = $8C
0240 VNTP =  $82
0250 WRMST = $A050
0260 XCLR =  $B766
0270 ;
0280     *=  $BAF7
0290 ;
0300 ;LoaD PRoGraM
0310 ;
0320 LDPRGM LDA #$FF
0330     BNE EXE
0340 ;
```

```
0350 ;eXecute LOAD statement
0360 ;
0370 XLOAD LDA #$00
0380 EXE PHA
0390     LDA #$04
0400     JSR OPNCH7
0410     PLA
0420 ;
0430 ;LOAD routine
0440 ;
0450 LOAD PHA
0460     LDA #$07
0470     STA IOCMD
0480     STA LOADFLG
0490     JSR MLTCHN
0500     LDY #$0E
0510     JSR BFLN3
0520     JSR ENDIO
0530     LDA LBUFF
0540     ORA LBUFF+1
0550     BNE ERR
0560     LDX #STARP
0570 LOOP CLC
0580     LDA LOMEM
0590     ADC [LBUFF-LOMEM],X
0600     PHP
0610     CLC
0620     ADC #$00
0630     TAY
0640     LDA LOMEM+1
0650     ADC [LBUFF-LOMEM+1],X
0660     PLP
0670     ADC #$00
0680     CMP MEMTOP+1
0690     BCC WRT
0700     BNE PLG
0710     CPY MEMTOP
0720     BCC WRT
0730 PLG JMP PRLGER
0740 WRT STA $01,X
0750     STY $00,X
0760     DEX
0770     DEX
0780     CPX #VNTP
0790     BCS LOOP
0800     JSR PRGMTR
0810     JSR XCLR
0820     LDA #$00
0830     STA LOADFLG
0840     PLA
0850     BEQ WRM
0860     RTS
0870 WRM JMP WRMST
0880 ERR LDA #$00
0890     STA LOADFLG
0900     JSR LOADER
0910 ;
0920 ;eXecute CLOAD statement
0930 ;
0940 XCLOAD LDA #$04
0950     JSR OPNCAS
```

```
0960     LDA #$00
0970     BEQ LOAD
```

Podobnie jak przy zapisie wykonywanie instrukcji rozpoczyna się od otwarcia kanału IOCB 7 przez odpowiednie procedury (OPNCH7 lub OPNCAS), lecz w tym przypadku do odczytu - wartość $04 w akumulatorze. Następnie do rejestru IOCMD wpisywany jest kod operacji GET BYTE ($07), a znacznik LOADFLG (LOADin FLaG) otrzymuje wartość niezerową. Znacznik ten pozwala po wystąpieniu błędu na rozpoznanie, czy sam odczyt był poprawny.

Teraz przez wywołanie BFLN odczytywany jest pierwszy blok informacji, czyli względne wartości wektorów. Jeśli dwa pierwsze bajty nie są zerami (LOMEM-LOMEM=0), to procedura jest przerywana skokiem do LOADER i sygnalizowany jest błąd (LOAD file ERror). Po dodaniu aktualnej zawartości LOMEM odczytane wektory są przepisywane do odpowiednich rejestrów. Wartość każdego z nich jest przy tym porównywana z wektorem MEMTOP, a jeśli go przekracza, to przez skok do procedury PRLGER sygnalizowany jest błąd (PRogram too LonG ERror).

Druga faza (odczyt treści programu) jest realizowana przez wywołanie procedury PRGMTR, która jest fragmentem SAVE. Dla zapewnienia poprawnej pracy programu jest teraz wywoływana procedura XCLR kasująca wartości i deklaracje wszystkich zmiennych. Po zakończeniu odczytu zerowany jest znacznik LOADFLG. Sposób opuszczenia procedury LOAD zależy od wartości zdjętej ze stosu. Zero (instrukcja LOAD lub CLOAD) powoduje skok do WRMST i gorący start interpretera. Inne wartości wskazują na wywołanie z procedury XRUN i powrót do niej jest wykonywany przez rozkaz RTS.

6.6.13. Instrukcja ENTER

Instrukcja ENTER pozwala na odczyt z urządzenia zewnętrznego programu zapisanego w postaci pliku znaków ASCII, to znaczy niestokenizowanego. Ponieważ jej procedura wykonawcza - XENTER - nie zmienia wektorów interpretera, to odczytywany program jest dołączany do znajdującego się już w pamięci.

```
0100 ;eXecute ENTER statement
0110 ;
0120 ACHNN = $B4
0130 OPNCH7 = $BAD7
0140 SYNTAX = $A060
0150 ;
0160     *=  $BAC5
0170 ;
0180     LDA #$04
0190     JSR OPNCH7
0200     STA ACHNN
0210     JMP SYNTAX
```

Sam przebieg procedury jest bardzo prosty. Najpierw kanał 7 jest otwierany do odczytu przez wywołanie procedury OPNCH7, a jego numer umieszczany jest w rejestrze ACHNN (Auxiliary CHaNnel Number). Po tym procedura kończy się skokiem do SYNTAX. W ten sposób następuje

wprowadzanie kolejnych wierszy programu na normalnych zasadach, lecz z wybranego urządzenia zewnętrznego zamiast z klawiatury.

6.6.14. Instrukcja LIST

Instrukcja LIST służy do zapisu programu w postaci niestokenizowanej (plik znaków ASCII) na dowolnym urządzeniu zewnętrznym. Zapisywany może być cały program lub fragment określony podanymi numerami wierszy. Jeśli nie zostanie podane urządzenie do zapisu, to jest on wykonywany do edytora (na ekran w trybie zero). Instrukcja LIST jest realizowana przez procedurę XLIST.

Rozpoczyna się ona od przygotowania standardowych parametrów instrukcji. Numer pierwszego zapisywanego wiersza w rejestrze CLNN (Current LiNe Number) jest ustalany na zero, zaś ostatniego - w rejestrze MAXLN (MAXimal LiNe number) - na $7FFF. Ponadto w rejestrze DSPFLG umieszczana jest wartość różna od zera, co pozwala na wyświetlanie na ekranie znaków specjalnych. Na zakończenie tej fazy przy pomocy procedury PRPCHN wyświetlany jest znak końca wiersza (RETURN) i wywoływana jest procedura SAVSTM, która zapisuje parametry aktualnie wykonywanego wiersza.

Teraz rozpoznawane są kolejne parametry instrukcji LIST (jeśli istnieją). Jeżeli pierwsza wartość - odczytana przy pomocy procedury LETVAR - jest tekstowa, to oznacza ona urządzenie, na którym ma być wykonany zapis. W takim przypadku wywoływana jest procedura OPNWRT, która otwiera do zapisu kanał IOCB 7. Jej przebieg jest tak prosty, że nie wymaga żadnego komentarza.

```
0100 ;OPeN for WRiTe
0110 ;
0120 IOCHN = $B5
0130 OPNCH7 = $BAD7
0140 ;
0150     *=  $BACF
0160 ;
0170     LDA #$08
0180     JSR OPNCH7
0190     STA IOCHN
0200     RTS
```

Kolejna liczba (koniecznie) odczytana przez procedurę GLNNUM stanowi numer pierwszego z zapisywanych wierszy programu i jest przepisywana do rejestru CLNN. Ta sama wartość jest umieszczana w rejestrze MAXLN, gdy w instrukcji LIST nie ma więcej parametrów. W przeciwnym przypadku numer ostatniego zapisywanego wiersza (MAXLN) jest odczytywany przez ponowne wywołanie GLNNUM. Teraz według zawartości CLNN procedura FNDCST odszukuje w pamięci wiersz o takim numerze lub - jeśli go nie ma - o najmniejszym wyższym numerze i rozpoczyna się główna pętla procedury.

```
0100 ;eXecute LIST statement
0110 ;
0120 CLCHN = $BCF7
0130 CLNN =  $A0
0140 DSPFLG = $02FE
```

```
0150 FNDCST = $A9A2
0160 FR0 =   $D4
0170 GHISTM = $A9E1
0180 GIRQST = $A9F2
0190 GLNLEN = $A9DC
0200 GLNNUM = $ABCD
0210 INIX =  $A8
0220 IOCHN = $B5
0230 LETVAR = $AC06
0240 LSTPLN = $B58E
0250 MAXLN = $AD
0260 NXTSTM = $A9D0
0270 OPNWRT = $BACF
0280 OUTIX = $A7
0290 PRPCHN = $BA99
0300 SAVSTM = $B6F9
0310 STMCUR = $8A
0320 VART =  $D2
0330 XRTRN = $BDA8
0340 ;
0350     *=  $B4B5
0360 ;
0370     LDY #$00
0380     STY CLNN
0390     STY CLNN+1
0400     DEY
0410     STY MAXLN
0420     LDA #$7F
0430     STA MAXLN+1
0440     STA DSPFLG
0450     LDA #$9B
0460     JSR PRPCHN
0470     JSR SAVSTM
0480 EXE LDY INIX
0490     INY
0500     CPY OUTIX
0510     BCS CST
0520     LDA INIX
0530     PHA
0540     JSR LETVAR
0550     PLA
0560     STA INIX
0570     LDA VART
0580     BPL LIN
0590     JSR OPNWRT
0600     JMP EXE
0610 LIN JSR GLNNUM
0620     STA CLNN+1
0630     LDA FR0
0640     STA CLNN
0650     LDY INIX
0660     CPY OUTIX
0670     BEQ MAX
0680     JSR GLNNUM
0690 MAX LDA FR0
0700     STA MAXLN
0710     LDA FR0+1
0720     STA MAXLN+1
0730 CST JSR FNDCST
0740 NXTLN JSR GHISTM
0750     BMI END
```

```
0760     LDY #$01
0770     LDA (STMCUR),Y
0780     CMP MAXLN+1
0790     BCC LST
0800     BNE END
0810     DEY
0820     LDA (STMCUR),Y
0830     CMP MAXLN
0840     BCC LST
0850     BNE END
0860 LST JSR LSTPLN
0870     JSR GIRQST
0880     BEQ END
0890     JSR GLNLEN
0900     JSR NXTSTM
0910     JMP NXTLN
0920 END LDA IOCHN
0930     BEQ EXIT
0940     JSR CLCHN
0950     LDA #$00
0960     STA IOCHN
0970 EXIT STA DSPFLG
0980     JMP XRTRN
```

Przede wszystkim sprawdzane jest ewentualne osiągnięcie wyznaczonego zakresu wierszy programu. Pętla jest przerywana, jeśli wiersz do zapisu ma numer większy od zawartego w rejestrze MAXLN lub większy od $7FFF (czyli jest w trybie bezpośrednim). W takim przypadku następuje przejście do końcowej fazy procedury XLIST. Jeżeli zapis był wykonywany na kanał inny niż IOCB 0, to kanał ten jest zamykany przez wywołanie CLCHN, a rejestr IOCHN jest zerowany. Zerowany jest również - niezależnie od użytego IOCB - rejestr DSPFLG. Potem procedura XLIST jest opuszczana skokiem do XRTRN, gdzie odtwarzany jest numer wiersza zawierającego instrukcję LIST i jej adres w tym wierszu.

Jeśli wiersz do zapisu mieści się w wyznaczonym zakresie, to jego zapis wykonuje procedura LSTPLN (zob. niżej). Następnie przy pomocy GIRQST sprawdzany jest stan klawisza BREAK. Gdy został on naciśnięty, procedura XLIST także jest przerywana. Jeśli nie, to przez wywołanie GLNLEN i NXTSTM znajdowany jest kolejny wiersz programu i pętla się powtarza.

Poszczególne wiersze programu są zapisywane na wskazanym urządzeniu zewnętrznym przez procedurę LSTPLN. Najpierw odczytuje ona numer zapisywanego wiersza i zamienia go na ciąg znaków ASCII. Adres tego ciągu jest umieszczany w rejestrze POKADR i wywoływana jest procedura PTMSG2, która dokonuje zapisu numeru.

```
0100 ;LiST Program LiNe
0110 ;
0120 BUFIX = $9F
0130 FASC =  $D8E6
0140 FR0 =   $D4
0150 IFP =   $D9AA
0160 INBUFP = $F3
0170 INIX =  $A8
0180 OUTIX = $A7
0190 POKADR = $95
```

```
0200 PRTLCN = $B5C2
0210 PTMSG2 = $B586
0220 STMCUR = $8A
0230 ;
0240     *=  $B58E
0250 ;
0260 LSTPLN LDY #$00
0270     LDA (STMCUR),Y
0280     STA FR0
0290     INY
0300     LDA (STMCUR),Y
0310     STA FR0+1
0320     JSR IFP
0330     JSR FASC
0340     LDA INBUFP
0350     STA POKADR
0360     LDA INBUFP+1
0370     STA POKADR+1
0380     JSR PTMSG2
0390 ;
0400 ;PRinT Program LiNe
0410 ;
0420 PRTPLN LDY #$02
0430     LDA (STMCUR),Y
0440     STA BUFIX
0450     INY
0460 LOOP LDA (STMCUR),Y
0470     STA OUTIX
0480     INY
0490     STY INIX
0500     JSR PRTLCN
0510     LDY OUTIX
0520     CPY BUFIX
0530     BCC LOOP
0540     RTS
```

Teraz długość wiersza jest przepisywana do rejestru BUFIX (BUFfer IndeX) - będzie ona stanowić licznik tokenów wiersza. Następny token, który określa długość instrukcji umieszczany jest w liczniku OUTIX (OUTput IndeX), zaś numer kolejnego tokena liczony od początku wiersza w liczniku INIX (INput IndeX). Zawartość wiersza jest zapisywana kolejno instrukcja po instrukcji przez procedurę PRTLCN w pętli sterowanej wymienionymi wyżej licznikami.

Procedura zapisu zawartości wiersza PRTLCN jest sterowana przez pomocniczą procedurę INCIX oraz będącą jej fragmentem CMPIX. Procedura ta porównuje stany liczników INIX i OUTIX. Jeżeli rezultat sygnalizuje zakończenie zapisywanej instrukcji, to, po zdjęciu ze stosu adresu powrotnego, procedura kończy się rozkazem RTS. Powoduje to powrót bezpośrednio do procedury LSTPLN z pominięciem miejsca wywołania w PRTLCN. W przeciwnym razie kolejny token jest odczytywany z wiersza i przekazywany w akumulatorze do procedury PRTLCN.

```
0100 INIX =  $A8
0110 OUTIX = $A7
0120 STMCUR = $8A
0130 ;
0140     *=  $B661
0150 ;
0160 ;INCrease IndeX
```

```
0170 ;
0180 INCIX INC INIX
0190 ;
0200 ;CoMPare IndeX
0210 ;
0220 CMPIX LDY INIX
0230     CPY OUTIX
0240     BCS EXIT
0250     LDA (STMCUR),Y
0260     RTS
0270 EXIT PLA
0280     PLA
0290     RTS
```

Przy zapisie instrukcji pierwszy odczytany token porównywany jest najpierw z kodem opuszczonej instrukcji LET ($36). Jego rozpoznanie powoduje od razu przejście do następnej fazy procedury. Każdy inny token powoduje wywołanie procedury PUTSTM, która zapisuje słowo kluczowe tej instrukcji. Jeśli była to instrukcja REM ($00) lub DATA ($01) albo token oznaczał błąd składni ($37), to pozostała zawartość wiersza jest zapisywana bez żadnych zmian.

Dla pozostałych instrukcji odczytywany jest następny token. Jego wartość większa od $7F oznacza numer zmiennej. Nazwa zmiennej o takim numerze jest odszukiwana w tablicy nazw zmiennych przez procedurę FNDPEL i zapisywana przy pomocy PUTTXT. Jeżeli koniec instrukcji nie został jeszcze osiągnięty, to rozpoznawany jest następny token.

Wartość mniejsza od $0F (możliwa jest tylko $0E) sygnalizuje stałą liczbową. Stała ta jest odczytywana przez procedurę NUMBER i zamieniana na ciąg znaków ASCII. Zapis tego ciągu jest wykonywany przy użyciu procedury PUTTXT, identycznie jak numer wiersza w procedurze LSTPLN.

Wartość tokena równa $0F wskazuje stałą tekstową. Najpierw długość tej stałej jest przepisywana do rejestru AUXBR i procedura PRPCHN zapisuje znak cudzysłowu ("). Teraz przy wykorzystaniu rejestru AUXBR jako licznika kolejne znaki stałej są zapisywane przez procedurę PRPCHN. Na końcu PRPCHN zapisuje jeszcze jeden znak cudzysłowu.

```
0100 ;PRinT Line CoNtens
0110 ;
0120 AUXBR = $AF
0130 CHKLTR = $A3EC
0140 CMPIX = $B663
0150 FASC =  $D8E6
0160 FNDPEL = $B53E
0170 INBUFP = $F3
0180 INCIX = $B661
0190 INIX =  $A8
0200 NUMBER = $AB45
0210 OPFN =  $A7DE
0220 POKADR = $95
0230 PRPCHN = $BA99
0240 PTMSG1 = $B581
0250 PUTSTM = $B66F
0260 PUTTXT = $B567
```

```
0270 VNTP =  $82
0280 ;
0290     *=  $B5C2
0300 ;
0310     JSR CMPIX
0320     CMP #$36
0330     BEQ CHCK
0340     JSR PUTSTM
0350     JSR CMPIX
0360     CMP #$37
0370     BEQ PUT
0380     CMP #$02
0390     BCS CHCK
0400 PUT JSR INCIX
0410     JSR PRPCHN
0420     JMP PUT
0430 CHCK JSR INCIX
0440     BPL NUM
0450     AND #$7F
0460     STA AUXBR
0470     LDX #$00
0480     LDA VNTP+1
0490     LDY VNTP
0500     JSR FNDPEL
0510     JSR PUTTXT
0520     CMP #['(+$80]
0530     BNE CHCK
0540     JSR INCIX
0550     JMP CHCK
0560 NUM CMP #$0F
0570     BEQ STR
0580     BCS OPER
0590     JSR NUMBER
0600     DEC INIX
0610     JSR FASC
0620     LDA INBUFP
0630     STA POKADR
0640     LDA INBUFP+1
0650     STA POKADR+1
0660 PRNT JSR PUTTXT
0670     JMP CHCK
0680 STR JSR INCIX
0690     STA AUXBR
0700     LDA #'"
0710     JSR PRPCHN
0720     LDA AUXBR
0730     BEQ END
0740 CHR JSR INCIX
0750     JSR PRPCHN
0760     DEC AUXBR
0770     BNE CHR
0780 END LDA #'"
0790     JSR PRPCHN
0800     JMP CHCK
0810 OPER SEC
0820     SBC #$10
0830     STA AUXBR
0840     LDX #$00
0850     LDA # >OPFN
0860     LDY # <OPFN
0870     JSR FNDPEL
```

```
0880     JSR CMPIX
0890     CMP #$3D
0900     BCS PRNT
0910     LDY #$00
0920     LDA (POKADR),Y
0930     AND #$7F
0940     JSR CHKLTR
0950     BCS PRNT
0960     JSR PTMSG1
0970     JMP CHCK
```

Każdy token, inny niż dotychczas opisane, oznacza operator lub funkcję. Jest on zmniejszany o $10 i procedura FNDPEL odszukuje odpowiednią nazwę w tablicy OPFN. Znalezione operatory (oprócz logicznych) są zapisywane przez procedurę PTMSG1, a nazwy funkcji i operatory logiczne przez PUTTXT.

Po zapisaniu każdego elementu instrukcji następuje przejście do początku głównej pętli (oznaczonego etykietą CHCK) procedury PRTCLN i rozpoznanie kolejnego tokena. Opuszczenie procedury jest realizowane wyłącznie w opisany wcześniej sposób poprzez procedurę INCIX.

DODATKI

Dodatek A

Adresy procedur Basica i OS

```
$A000 - CDST - zimny start interpretera
$A00C - XNEW - wykonanie instrukcji NEW
$A050 - WRMST - gorący start interpretera
$A053 - WRMST2 - gorący start interpretera
$A05D - WAITIN - oczekiwanie na rekord
$A060 - SYNTAX - tokenizacja i kontrola składni
$A0B1 - DCDSTM - rozpoznanie instrukcji
$A0FB - MOVTKN - przemieszczenie bloku tokenów
$A186 - DELPLN - skasowanie wiersza programu
$A19A - PLINEN - rozpoznanie numeru wiersza
$A1BE - STMSX - kontrola składni instrukcji
$A208 - PHADR - zapisanie na stosie adresu procedury
$A21B - SAVCPM - zapisanie stanu liczników
$A28C - INCPRC - zwiększenie licznika programu
$A293 - GTCCHR - odczyt kodu z tablicy składni
$A29B - VALUE - rozpoznanie wartości
$A2C4 - SAVTKN - zapis tokena do bufora
$A2CF - SCDSTM - zapis końca instrukcji IF
$A2DB - MOVLIN - przepisanie wiersza do bufora tokenizacji
$A2E1 - EXPSX - kontrola składni wyrażenia
$A320 - NUMVAR - tokenizacja zmiennej liczbowej
$A324 - STRVAR - tokenizacja zmiennej tekstowej
$A3E8 - CMPLTR - rozpoznanie znaku nazwy
$A3EC - CHKLTR - rozpoznanie litery z akumulatora
$A3F5 - NUMCNS - tokenizacja stałej liczbowej
$A41C - STRCNS - tokenizacja stałej tekstowej
$A454 - RECNAM - wyszukiwanie nazwy w tablicy
$A482 - NXTNAM - wyszukiwanie następnej nazwy
$A49F - STNAME - tablica nazw instrukcji
$A5F5 - ERMSG - meldunek błędu "ERROR"
$A5FD - STMSG - meldunek zatrzymania "STOPPED"
$A605 - SXTAB - tablica składni instrukcji
$A7DE - OPFN - tablica nazw operatorów i funkcji
$A87A - INSEL - przemieszczenie bloku pamięci w górę
$A8F7 - DELEL - przemieszczenie bloku pamięci w dół
$A944 - MOVMEM - przemieszczenie bloku pamięci
$A95E - PRCSTM - wykonanie wiersza programu
$A97E - EXESTM - wykonanie instrukcji Basica
$A9A2 - FNDCST - wyszukanie aktualnego wiersza
$A9D0 - NXTSTM - wyszukanie następnego wiersza
$A9DC - GLNLEN - odczytanie długości wiersza
$A9E1 - GHISTM - odczytanie starszego bajtu numeru wiersza
$A9E5 - XREM - wykonanie instrukcji REM i DATA
$A9E6 - XBYE - wykonanie instrukcji BYE
$A9EC - XDOS - wykonanie instrukcji DOS
$A9F2 - GIRQST - odczyt statusu przerwań IRQ
$A9FA - STVTAB - tablica wektorów procedur instrukcji
$AA6A - OPVTAV - tablica wektorów procedur operatorów
$AADA - XLET - wykonanie instrukcji LET
$AB04 - PRCOP - realizacja procedury operatora
$AB18 - EXEOP - wykonanie procedury operatora
```

$AB26 - RSTPTR - skasowanie rejestrów liczników
$AB35 - XPLS - wykonanie operatora plus (znak)
$AB36 - RECVAL - rozpoznanie rodzaju wartości
$AB45 - NUMBER - odczytanie liczby
$AB5C - STTXT - zapisanie tekstu
$AB81 - VARBL - odczytanie wartości zmiennej
$AB90 - FTARV - obliczenie adresu zmiennej tablicowanej
$ABB2 - PUTVAR - zapisanie wartości zmiennej
$ABCD - GLNNUM - odczytanie i sprawdzenie numeru wiersza
$ABD7 - LETNUM - obliczenie wartości wyrażenia
$ABDA - GETNUM - odczyt liczby całkowitej z bufora
$ABE0 - GETBYT - odczytanie liczby jednobajtowej z bufora
$ABE9 - GETVAR - odczytanie wartości zmiennej
$ABFD - GETVRS - odczytanie wartości dwóch zmiennych
$AC06 - LETVAR - obliczenie i odczyt wartości zmiennej
$AC0C - SAVVAL - przepisanie wartości według wektora
$AC1E - CALPC - obliczenie adresu zmiennej
$AC35 - OPCT - tablica współczynników operatorów
$AC7A - XSUB - wykonanie operatora odejmowania
$AC83 - XMUL - wykonanie operatora mnożenia
$AC8C - XDIV - wykonanie operatora dzielenia
$AC95 - XMIN - wykonanie operatora minus (znak)
$ACA3 - XLEQ - wykonanie operatora "mniejsze lub równe"
$ACAC - XNEQ - wykonanie operatora "różne"
$ACB2 - XLES - wykonanie operatora "mniejsze"
$ACB9 - XGRT - wykonanie operatora "większe"
$ACC2 - XGEQ - wykonanie operatora "większe lub równe"
$ACC9 - XEQU - wykonanie operatora "równe"
$ACCF - XAND - wykonanie operatora AND
$ACD9 - XOR - wykonanie operatora OR
$ACE0 - RESLTZ - ustalenie wyniku operacji logicznej
$ACE4 - XNOT - wykonanie operatora NOT
$ACE9 - RESLTO - ustalenie wyniku operacji logicznej
$ACEB - ZERO - ustalenie wyniku operacji na zero
$ACF0 - ONEP - ustalenie wyniku operacji na jeden
$ACF2 - ONEN - ustalenie wyniku operacji na minus jeden
$ACF4 - STRES - zapisanie wyniku operacji porównania
$AD04 - XSGN - wykonanie funkcji SGN
$AD11 - CMPVAR - porównanie wartości dwóch zmiennych
$AD20 - GETSUB - obliczenie różnicy liczb z FR0 i FR1
$AD26 - BADD - wywołanie procedury dodawania
$AD2C - BSUB - wywołanie procedury odejmowania
$AD32 - BMUL - wywołanie procedury mnożenia
$AD38 - BDIV - wywołanie procedury dzielenia
$AD41 - GETINT - zamiana liczby na całkowitą
$AD4A - XNLET - wykonanie operatora "przypisanie liczby"
$AD64 - XCOM - wykonanie operatora indeksowego
$AD66 - XEPAR - wykonanie operatora "koniec nawiasu"
$AD6D - XPDIM - wykonanie operatora "nawias wymiaru"
$AD71 - XPIXV - wykonanie operatora "nawias indeksu"
$AE11 - XPSEX - wykonanie operatora "nawias indeksu"
$AE81 - GETSLN - odczytanie długości ciągu
$AE8E - XSLET - wykonanie operatora "przypisanie tekstu"
$AF31 - ZT1ML6 - mnożenie przez sześć zawartości ZTEMP1
$AF3D - ZT1ADD - dodawanie akumulatora i rejestru Y do ZTEMP1
$AF48 - EVZTMP - przeliczanie zawartości rejestru ZTEMP1
$AF6C - CMPSTR - porównanie wartości tekstowych
$AFA7 - DECREG - zmniejszenie wartości wektorów Basica
$AFB5 - XLEN - wykonanie funkcji LEN
$AFBC - STNUM - zapisanie liczby całkowitej do FR0
$AFC0 - CHTYP - zamiana liczby całkowitej na FP

$AFC3 - NUMTYP - ustawienie typu zmiennej liczbowej
$AFCC - XPEEK - wykonanie funkcji PEEK
$AFD6 - XFRE - wykonanie funkcji FRE
$AFEB - XVAL - wykonanie funkcji VAL
$AFFD - XASC - wykonanie funkcji ASC
$B007 - XADR - wykonanie funkcji ADR
$B00D - XPADDL - wykonanie funkcji PADDLE
$B011 - XSTICK - wykonanie funkcji STICK
$B015 - XPTRIG - wykonanie funkcji PTRIG
$B019 - XSTRIG - wykonanie funkcji STRIG
$B034 - XSTR - wykonanie funkcji STR$
$B052 - XCHR - wykonanie funkcji CHR$
$B069 - STRTYP - ustawienie typu zmiennej tekstowej
$B076 - XRND - wykonanie funkcji RND
$B093 - RNDC - stała do obliczania funkcji RND
$B099 - XABS - wykonanie funkcji ABS
$B0A5 - XUSR - wykonanie funkcji USR
$B0AE - PHDAT - zapis na stosie parametrów funkcji USR
$B0C8 - XINT - wykonanie funkcji INT
$B0D1 - BINT - procedura obliczenia części całkowitej
$B106 - XSIN - wykonanie funkcji SIN
$B10F - XCOS - wykonanie funkcji COS
$B118 - XATN - wykonanie funkcji ATN
$B121 - XLOG - wykonanie funkcji LOG
$B12B - STLOG - opracowanie wyniku operacji logarytmowania
$B13D - XCLOG - wykonanie funkcji CLOG
$B14A - XEXP - wykonanie funkcji EXP
$B153 - XSQR - wykonanie funkcji SQR
$B159 - RESULT - zapis wyniku w buforze tokenizacji
$B15B - VALER - sygnalizowanie niepoprawnej wartości
$B15E - XRPW - wykonanie operatora potęgowania
$B16C - PUTRES - zapisanie wyniku operacji do bufora
$B206 - XDIM - wykonanie instrukcji DIM
$B278 - XPOKE - wykonanie instrukcji POKE
$B28D - XDEG - wykonanie instrukcji DEG
$B291 - XRAD - wykonanie instrukcji RAD
$B296 - XRSTR - wykonanie instrukcji RESTORE
$B2AE - XREAD - wykonanie instrukcji READ
$B32D - NXTDAT - odszukanie następnej pozycji w instrukcji DATA
$B32F - SENDDT - odszukanie końca pozycji w instrukcji DATA
$B33E - XINPUT - wykonanie instrukcji INPUT
$B3DA - XPRINT - wykonanie instrukcji PRINT
$B48F - RSTHIB - skasowanie najstarszego bitu akumulatora
$B491 - INOUTX - zwiększenie licznika bufora wyjściowego
$B496 - XLPRNT - wykonanie instrukcji LPRINT
$B4B2 - PRTNAM - nazwa drukarki - "P:"
$B4B5 - XLIST - wykonanie instrukcji LIST
$B53E - FNDPEL - odszukanie elementu programu
$B557 - ADDPAD - zwiększenie stanu rejestru POKADR
$B562 - SETPAD - ustawienie stanu rejestru POKADR
$B567 - PUTTXT - wyświetlenie tekstu na ekranie
$B581 - PTMSG1 - wyświetlenie raportu na ekranie
$B586 - PTMSG2 - wyświetlenie raportu na ekranie
$B58E - LSTPLN - wylistowanie wiersza programu
$B5AA - PRTPLN - wyświetlenie wiersza programu
$B5C2 - PRTLCN - wyświetlenie zawartości wiersza programu
$B661 - INCIX - zwiększenie stanu licznika bufora wejściowego
$B663 - CMPIX - porównanie stanu liczników bufora
$B66F - PUTSTM - wyświetlenie słowa instrukcji
$B67D - XFOR - wykonanie instrukcji FOR
$B6B5 - PHSTK - zapis parametrów aktualnej instrukcji na stosie

$B6D2 - XGOSUB - wykonanie instrukcji GOSUB
$B6D5 - XGOTO - wykonanie instrukcji GOTO
$B6D8 - GOLINE - przejście do wskazanego wiersza programu
$B6E0 - FDEXST - odszukanie i wykonanie wiersza programu
$B6F0 - RSCSTM - odtworzenie adresu aktualnego wiersza
$B6F9 - SAVSTM - zachowanie adresu aktualnego wiersza
$B700 - XNEXT - wykonanie instrukcji NEXT
$B74C - XRUN - wykonanie instrukcji RUN
$B766 - XCLR - wykonanie instrukcji CLR
$B778 - XIF - wykonanie instrukcji IF
$B78C - XEND - wykonanie instrukcji END
$B792 - XSTOP - wykonanie instrukcji STOP
$B7A6 - SAVCLN - zapisanie aktualnego numeru wiersza
$B7B5 - XCONT - wykonanie instrukcji CONT
$B7D8 - XTRAP - wykonanie instrukcji TRAP
$B7E4 - XON - wykonanie instrukcji ON
$B816 - FSTGLN - odszukanie instrukcji i odczyt jej długości
$B819 - GSTMLN - odczyt długości instrukcji
$B823 - PLSTK - odczyt parametrów aktualnej instrukcji ze stosu
$B83E - XPOP - wykonanie instrukcji POP
$B871 - BMTUP - podniesienie górnej granicy pamięci Basica
$B87A - SAVTST - zapisanie górnej granicy pamięci Basica
$B883 - SAVIX - zapisanie indeksu w rejestrze tymczasowym
$B888 - PHREG - zapis aktualnego stanu rejestrów na stosie
$B897 - PLREG - odczyt aktualnego stanu rejestrów ze stosu
$B8A8 - RSMEMT - zerowanie tablicy STARP i stosu bieżącego
$B8B9 - CLRVV - zerowanie tablicy wartości zmiennych
$B8F1 - RSTBRG - zerowanie rejestrów Basica
$B904 - GETIX - odczyt i porównanie indeksów bufora
$B90A - LOADER - błąd podczas odczytu pliku
$B90C - DVCNER - błąd: zbyt duży numer urządzenia
$B90E - PRLGER - błąd: zbyt długi program
$B910 - CHARER - błąd: niepoprawny znak ciągu
$B912 - SNTXER - błąd składni
$B914 - RETER - błąd: instrukcja RETURN bez GOSUB
$B916 - GOSLER - błąd: brak instrukcji GOSUB lub FOR
$B918 - LTLGER - błąd: zbyt długi wiersz programu
$B91A - NFORER - błąd: instrukcja NEXT bez FOR
$B91C - LNFDER - błąd: brak wiersza o podanym numerze
$B91E - OVUNER - błąd: liczba zbyt duża lub zbyt mała
$B920 - STOVER - błąd: przepełnienie stosu bieżącego
$B922 - DIMER - błąd: zła deklaracja wymiaru zmiennej
$B924 - BINPER - błąd: niepoprawna wartość dla INPUT lub READ
$B926 - LINNER - błąd: niepoprawny numer wiersza
$B928 - ODATER - błąd: brak danych dla instrukcji READ
$B92A - SLENER - błąd: zła długość zmiennej tekstowej
$B92C - TMVRER - błąd: zbyt dużo zmiennych
$B92E - BVALER - błąd: użyta niepoprawna wartość
$B930 - INSMER - błąd: niedostateczny rozmiar pamięci
$B932 - SUCC - operacja wykonana poprawnie
$B934 - GETERR - odczyt kodu błędu
$B968 - DSTMSG - wyświetlenie raportu przerwania programu
$B993 - DSPLIN - wyświetlenie numeru wiersza
$B9A4 - LNMSG - napis "AT LINE"
$B9AD - XSETC - wykonanie instrukcji SETCOLOR
$B9D3 - XSOUND - wykonanie instrukcji SOUND
$BA0C - XPOS - wykonanie instrukcji POSITION
$BA1F - XCOLOR - wykonanie instrukcji COLOR
$BA27 - XDRAW - wykonanie instrukcji DRAWTO
$BA46 - XGRAPH - wykonanie instrukcji GRAPHICS
$BA69 - SCRNAM - nazwa ekranu - "S:"

$BA6C - XPLOT - wykonanie instrukcji PLOT
$BA76 - CONVFN - przekształcenie liczby rzeczywistej
$BA99 - PRPCHN - przygotowanie kanału IOCB
$BA9B - PRPDVC - przygotowanie urządzenia do operacji I/O
$BAB2 - IOCAL - wywołanie procedury wykonującej operację I/O
$BABE - SAVCMD - zapisanie kodu rozkazu operacji I/O
$BAC0 - SAVDVC - zapisanie numeru urządzenia dla operacji I/O
$BAC5 - XENTER - wykonanie instrukcji ENTER
$BACF - OPNWRT - otwarcie kanału IOCB 7 do zapisu
$BAD7 - OPNCH7 - otwarcie kanału IOCB 7
$BAF1 - SQRC - stała do obliczania funkcji SQR
$BAF7 - LDPRGM - odczyt programu przed uruchomieniem
$BAFB - XLOAD - wykonanie instrukcji LOAD
$BB04 - LOAD - procedura odczytu programu z urządzenia
$BB64 - XCLOAD - wykonanie instrukcji CLOAD
$BB6D - XSAVE - wykonanie instrukcji SAVE
$BB72 - SAVE - procedura zapisu programu na urządzenie
$BB98 - PRGMTR - procedura transmisji stokenizowanego programu
$BBB4 - OPNCAS - przygotowanie operacji otwarcia magnetofonu
$BBCE - CASNAM - nazwa magnetofonu - "C:"
$BBD1 - XCSAVE - wykonanie instrukcji CSAVE
$BBD8 - PRPOPN - przygotowanie operacji otwarcia kanału IOCB
$BBEC - XXIO - wykonanie instrukcji XIO
$BBF2 - XOPEN - wykonanie instrukcji OPEN
$BC02 - OPNCHN - otwarcie kanału IOCB dla operacji I/O
$BC22 - XCLOSE - wykonanie instrukcji CLOSE
$BC24 - PRCIO - przygotowanie operacji wejścia/wyjścia
$BC29 - IOOPER - przeprowadzenie operacji wejścia/wyjścia
$BC2F - XSTAT - wykonanie instrukcji STATUS
$BC3D - XNOTE - wykonanie instrukcji NOTE
$BC54 - XPOINT - wykonanie instrukcji POINT
$BC78 - XPUT - wykonanie instrukcji PUT
$BC85 - XGET - wykonanie instrukcji GET
$BC9E - XLOCAT - wykonanie instrukcji LOCATE
$BCA8 - SETDVC - ustalenie numeru urządzenia dla operacji I/O
$BCAF - MLTCHN - mnożenie numeru kanału IOCB
$BCBB - ENDIO - procedura końcowa operacji I/O
$BCF7 - CLCHN - zamknięcie kanału IOCB
$BD00 - GETST - odczyt statusu wykonanej operacji I/O
$BD07 - IXGDAT - zwiększenie licznika i odczyt danych
$BD09 - GETDAT - odczyt bajtu danych
$BD0F - BFLN1 - ustalenie długości bufora dla operacji I/O
$BD13 - BFLN2 - ustalenie długości bufora dla operacji I/O
$BD15 - BFLN3 - ustalenie długości bufora dla operacji I/O
$BD17 - BFLN4 - ustalenie długości bufora dla operacji I/O
$BD1E - SIBUFA - ustalenie adresu bufora dla operacji I/O
$BD29 - CMDEXE - odczyt kodu i wykonanie operacji I/O
$BD2B - CIOEXE - wywołanie systemowej procedury operacji I/O
$BD31 - SAVBYT - zapisanie wartości bajtu w zmiennej
$BD33 - SAVWRD - zapisanie wartości dwubajtowej w zmiennej
$BD45 - CLALL - zamknięcie wszystkich kanałów IOCB
$BD5B - RSTCHN - skasowanie rejestrów kanału IOCB
$BD62 - PRTPRM - wyświetlenie napisu "READY"
$BD72 - READYP - napis "READY"
$BD79 - PRTRET - wyświetlenie znaku końca wiersza
$BD7D - GFILSP - odczytanie adresu zmiennej tablicowej
$BD9D - PSTMAD - zapis adresu instrukcji
$BDA8 - XRTRN - wykonanie instrukcji RETURN
$BDC2 - BRETLN - błąd: brak wiersza z instrukcją GOSUB
$BDCB - RETURN - odtworzenie adresu instrukcji w wierszu
$BDDB - PUTRET - zapis znaku końca wiersza do edytora

```
$BDE4 - INPREC - zapis znaku z rejestru PROMPT do edytora
$BDED - GETREC - odczyt rekordu z urządzenia
$BDFA - XADD - wykonanie operatora dodawania
$BE05 - SIN - obliczenie wartości sinusa liczby
$BE0F - COS - obliczenie wartości cosinusa liczby
$BE9F - TSCC - tablica stałych dla funkcji sinus i cosinus
$BEC9 - TATNC - stała do obliczania funkcji arcus tangens
$BECF - TRIGC - stała do obliczeń trygonometrycznych
$BED5 - ATAN - obliczenie wartości arcus tangens liczby
$BF43 - SQR - obliczenie pierwiastka kwadratowego
$BFF0 - CART - blok informacji o cartridge'u
$BFFA - CARTRUN - adres uruchomienia cartridge'a
$BFFC - CARTINS - znacznik zainstalowania cartridge'a
$BFFD - CARTOPT - rejestr rodzaju cartridge'a
$BFFE - CARTINI - adres inicjowania cartridge'a
$D800 - AFP - zamiana ciągu ASCII na liczbę FP
$D8E6 - FASC - zamiana liczby FP na ciąg ASCII
$D9AA - IFP - zamiana liczby całkowitej na FP
$D9D2 - FPI - zamiana liczby FP na całkowitą
$DA44 - ZFR0 - zerowanie FR0
$DA48 - AF1 - zerowanie wg rejestru X
$DA51 - STBV - zapis wektora bufora
$DA60 - FSUB - odejmowanie liczb FP
$DA66 - FADD - dodawanie liczb FP
$DADB - FMUL - mnożenie liczb FP
$DB28 - FDIV - dzielenie liczb FP
$DBA1 - INBSS - przeszukiwanie bufora
$DBAF - ADBT - zamiana znaku ASCII na kod BCD
$DC00 - NFR0 - poprawienie formatu liczby FP
$DD40 - PLYEVL - przeliczanie wielomianowe
$DD89 - FLD0R - zapis liczby FP do FR0 według X,Y
$DD98 - FLD1R - zapis liczby FP do FR1 według X,Y
$DDA7 - FST0R - zapis liczby FP z FR0 według X,Y
$DDB6 - FMOV01 - przepisanie z FR0 do FR1
$DDC0 - EXP - potęgowanie o podstawie e
$DDCC - EXP10 - potęgowanie o podstawie 10
$DE95 - RSQT - iloraz różnicowy
$DECD - LOG - logarytm naturalny
$DED1 - LOG10 - logarytm dziesiętny
$DF66 - TLOG - tabela współczynników logarytmowania
$DFAE - TATAN - tabela współczynników funkcji arctg
$E456 - JCIOMAIN - skok do CIOMAIN
$E480 - JTSTROM - skok do TSTROM
```

Dodatek B

Rejestry Basica i OS w pamięci RAM

```
  $08 - WARMST - znacznik gorącego startu
  $0A - DOSVEC - wektor startowy programu dyskowego
  $0E - APPMHI - najwyższy adres RAM zajęty przez program
  $11 - IRQSTAT - rejestr-cień IRQST
  $2A - ICAX1Z - rejestr pomocniczy ZIOCB
  $2B - ICAX2Z - rejestr pomocniczy ZIOCB
  $54 - ROWCRS - pionowa pozycja kursora
  $55 - COLCRS - pozioma pozycja kursora
  $80 - LOMEM - wektor bufora wejściowego tokenizacji
  $82 - VNTP - wektor początku tablicy nazw zmiennych
  $84 - VNTD - wektor końca tablicy nazw zmiennych
```

$86 - VVTP - wektor początku tablicy wartości zmiennych
$88 - STMTAB - wektor początku tablicy instrukcji
$8A - STMCUR - wektor aktualnie wykonywanego wiersza
$8C - STARP - wektor tablicy zmiennych tablicowanych
$8E - RUNSTK - wektor stosu bieżącego Basica
$90 - BMEMHI - koniec obszaru pamięci zajętego przez Basic
$92 - MOELFLG - zmodyfikowany znacznik końca wiersza
$94 - COX - bieżący indeks wyjściowy tokenizacji
$95 - POKADR - wektor pomocniczy różnych procedur Basica
$97 - CSTAD - adres aktualnego wiersza programu
$99 - OLDMHI - adres przemieszczanego bloku pamięci
$9B - NEWMHI - adres docelowy przemieszczenia bloku pamięci
$9D - PCNTC - licznik programu
$9F - BUFIX - indeks wejściowy bufora składni
$A0 - CLNN - numer aktualnego wiersza programu
$A2 - MRANGE - rozmiar przemieszczanego bloku pamięci
$A4 - LENPEL - rozmiar elementu programu
$A6 - SXFLG - znacznik rodzaju wiersza przy kontroli składni
$A7 - OUTIX - indeks wyjściowy Basica
$A8 - INIX - indeks wejściowy Basica
$A9 - STIX - indeks stosu wyrażeń
$AA - STPTR - licznik stosu
$AB - TOX - pomocniczy indeks wyjściowy
$AC - TIX - pomocniczy indeks wejściowy
$AD - MAXLN - numer ostatniego wiersza programu
$AF - AUXBR - pomocniczy rejestr Basica
$B0 - BHLD1 - pomocniczy rejestr Basica
$B1 - BHLD2 - pomocniczy rejestr Basica
$B2 - STMNUM - numer aktualnej instrukcji
$B3 - TMPIX - pomocniczy indeks tymczasowy
$B4 - ACHNN - pomocniczy rejestr numeru kanału I/O
$B5 - IOCHN - rejestr numeru kanału IOCB dla operacji I/O
$B6 - DATAD - indeks kolejnej pozycji w instrukcji DATA
$B7 - DATALN - numer kolejnego wiersza z instrukcją DATA
$B9 - ERRCOD - rejestr obliczania kodu błędu
$BA - STOPLN - numer wiersza zatrzymania programu
$BC - TRAPLN - numer wiersza dla skoku po błędzie
$BE - SAVCUR - tymczasowy rejestr numeru wiersza programu
$C0 - IOCMD - kod rozkazu wykonania operacji I/O
$C1 - IODVC - numer urządzenia wykonującego operację I/O
$C2 - PROMPT - kod znaku wyświetlanego przez instrukcję INPUT
$C3 - ERRSAV - kod ostatnio rozpoznanego błędu
$C4 - SAVMHI - tymczasowy rejestr granicy zajętej pamięci
$C6 - ACNT1 - pomocniczy licznik interpretera Basica
$C7 - ACNT2 - pomocniczy licznik interpretera Basica
$C8 - COLOR - rejestr aktualnie używanego koloru
$C9 - PTABW - liczba znaków między pozycjami tabulacji
$CA - LOADFLG - znacznik zakończenia operacji odczytu
$CB-$D1 - niewykorzystane - przeznaczone dla użytkownika
$D2 - VART - rejestr rodzaju zmiennej lub stałej
$D3 - VARN - rejestr numeru zmiennej lub stałej
$D4 - FR0 - zerowy rejestr liczb FP
$DA - FRE - dodatkowy rejestr liczb FP
$E0 - FR1 - pierwszy rejestr liczb FP
$EF - ESIGN - rejestr znaku wykładnika
$F0 - FCHRFLG - znacznik pierwszego znaku liczby
$F1 - DIGRT - liczba cyfr po przecinku
$F2 - CIX - indeks znaku w buforze
$F3 - INBUFP - adres bufora wejściowego
$F5 - ZTEMP1 - rejestr tymczasowy
$F7 - ZTEMP2 - rejestr tymczasowy

```
  $F9 - ZTEMP3 - rejestr tymczasowy
  $FB - RADFLG - znacznik operacji trygonometrycznych
  $FC - FLPTR - adres liczby FP
  $FE - FPTR2 - adres liczby FP
$0100 - STACK - stos mikroprocesora 6502
$0270 - PADDL0 - stan potencjometru 0, rejestr-cień POT0
$0271 - PADDL1 - stan potencjometru 1, rejestr-cień POT1
$0272 - PADDL2 - stan potencjometru 2, rejestr-cień POT2
$0273 - PADDL3 - stan potencjometru 3, rejestr-cień POT3
$0278 - JSTICK0 - położenie joysticka 0
$0279 - JSTICK1 - położenie joysticka 1
$027C - PTRIG0 - przycisk potencjometru 0
$027D - PTRIG1 - przycisk potencjometru 1
$027E - PTRIG0 - przycisk potencjometru 2
$027F - PTRIG1 - przycisk potencjometru 3
$0284 - TRIG0S - przycisk joysticka 0, rejestr-cień TRIG0
$0285 - TRIG1S - przycisk joysticka 1, rejestr-cień TRIG1
$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
$02E5 - MEMTOP - adres górnej granicy wolnej pamięci RAM
$02E7 - MEMLO - adres dolnej granicy wolnej pamięci RAM
$02FB - ATACHR - kod ATASCII znaku
$02FE - DSPFLG - znacznik wyświetlania znaków kontrolnych
$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
$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
$0480 - BINIX - indeks wejściowy bufora kontroli składni
$0481 - BOUTIX - indeks wyjściowy bufora kontroli składni
$0482 - BPRCNT - licznik programu do kontroli składni
$0580 - LBUFF - bufor wyjściowy operacji FP
$05E6 - FPSCR - rejestr pomocniczy operacji FP
$05EC - FPSCR1 - rejestr pomocniczy operacji FP
$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)
$D20A - RANDOM - rejestr liczby losowej (O)
$D20F - SKCTL - rejestr kontroli złącza szeregowego (Z)
```

Dodatek C

Zmienne systemowe

```
CARTINS
    0 - cartridge zainstalowany
    nie 0 - brak cartridge'a w gnieździe

CARTOPT
bit 0 - wstępny odczyt z dyskietki (0 = zabroniony)
bit 2 - 1 = inicjowanie i uruchomienie cartridge'a
        0 = tylko inicjowanie
bit 7 - rodzaj cartridge'a (1 = diagnostyczny)
pozostałe bity niewykorzystane

DSPFLG
    0 - wykonywanie znaków kontrolnych
    nie 0 - wyświetlanie znaków kontrolnych

LOADFLG
    0 - poprawny przebieg ostatniej operacji odczytu programu
    nie 0 - operacja odczytu programu nie została zakończona

SXFLG
bit 6 - błąd składni we wprowadzanym wierszu (1 = wystąpił)
bit 7 - tryb  pracy interpretera Atari Basic (0 = programowy, 1
= bezpośredni)

VART
bit 0 - deklaracja zmiennej tablicowej (1 = deklarowana)
bit 1 - oznaczenie stałej tekstowej (1 = stała, tylko jako $83)
bity 2-5 - niewykorzystane
bit 6 - zmienna indeksowana (1 = wartość z indeksem)
bit 7 - typ zmiennej (0 = liczbowa, 1 = tekstowa)
```

Dodatek D

Słownik terminów informatycznych

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.

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. W Atari procedury zmiennoprzecinkowe zawarte są w pakiecie FP.

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 $3BF.

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 pamięci.

OS

Operating System - system operacyjny, zestw procedur zapisanych przeważnie w pamięci ROM, które sterują pracą komputera i jego współpracą z urządzeniami zewnętrznymi.

pakiet FP

Obszar pamięci ROM komputerów Atari XL/XE zawierający wbudowany zestaw procedur wykonujących operacje na liczbach zmiennoprzecinkowych. Nazwa ta określa także sam zestaw procedur FP.

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.

strona

Część pamięci komputera o wielkości 256 bajtów. Starszy bajt adresu wskazuje zawsze numer strony, a młodszy - komórkę na stronie.

token

Jednobajtowy kod instrukcji, operatora, funkcji lub innego elementu programu w języku Basic. Stosowanie tokenów do zapisu programu zmniejsza wykorzystywany obszar pamięci i oszczędza nośniki pamięci zewnętrznych.

tryb bezpośredni

Tryb pracy interpretera Atari Basic, w którym każdy wprowadzony wiersz jest wykonywany natychmiast po zatwierdzeniu go znakiem końca wiersza EOL (jeśli jest bez błędów). W tym trybie wykonywane są wszystkie wiersze podane bez numeru.

tryb programowy

Tryb pracy interpretera Atari Basic, w którym każdy wprowadzony wiersz jest po zatwierdzeniu go znakiem końca wiersza EOL zapisywany w pamięci w postaci stokenizowanej. Wiersze wpisywane w tym trybie muszą posiadać numer kolejny o wartości z zakresu $0000-$7FFF (0-32767).

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	44288	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, 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. Praca zbiorowa, redakcja: Migut Wiesław: Atari BASIC, KAW, Warszawa, 1987.
6. Ruszczyc Jan: Asembler 6502, SOETO, Warszawa, 1987.
7. Zientara Wojciech: PEEK-POKE 2 (opracowanie), B.U.K. "GLAD", Warszawa, 1987.
8. Zientara Wojciech: Mapa pamięci Atari XL/XE. Podstawowe procedury systemu operacyjnego, SOETO, Warszawa, 1988.
9. Zientara Wojciech: Mapa pamięci Atari XL/XE. Procedury wejścia/wyjścia, SOETO, Warszawa, 1988.

10. Zientara Wojciech: Mapa pamięci Atari XL/XE. Dyskowe systemy operacyjne, SOETO, Warszawa, 1988.