Przewodnik programisty

(rev. 1.1)

Spis treści

Wstęp

Niniejszy podręcznik przeznaczony jest dla koderów mających ochotę pisać programy aplikacyjne i systemowe dla SpartaDOS X. Autorzy zakładają, że Czytelnik ma orientację w pisaniu programów na Atari, oraz jest zaawansowanym użytkownikiem SpartaDOS X, wobec czego wiadomości zawarte w podręczniku „Dyskowy System Operacyjny SpartaDOS X. Podręcznik Użytkownika” oraz suplemencie „SpartaDOS X v. 4.40” tutaj pomijamy jako oczywiste. Nadto zakłada się też, że pojęcia w rodzaju „PORTB” nie wymagają objaśnień.

Listingi w asemblerze napisane są pseudokodem o składni zgodnej z asemblerem MAE. Jest ona najbardziej zbliżona do składni asemblera MAC/65, szczegółami tylko różni się też od składni używanej przez crossassembler MADS. Programista zechce na własną rękę przystosować je sobie do ulubionego asemblera.

Życzymy przyjemnej lektury

DLT Ltd.

Rozdział 1: symbole

Co to jest symbol

Podręcznik programowania pod SpartaDOS X dobrze jest zacząć od objaśnienia pojęcia *symbolu*, z jakim Czytelnik będzie się spotykał podczas lektury.

Symbol w SpartaDOS X jest to rodzaj rozbudowanego wskaźnika. Zawiera on informację, gdzie w pamięci komputera znajduje się dany obiekt: zmienna, tablica lub procedura. Jeden symbol odpowiada jednemu takiemu obiektowi. Wszystkich symboli jest około setki, połączone są w listę. Zapewnia ona dostęp do najważniejszych – z punktu widzenia programisty – procedur i struktur wewnętrznych SpartaDOS X, oraz do sterowników i nakładek, gdyż programy rezydentne mogą, naturalnie, dołączać do listy własne symbole.

Najważniejszą zaletą takiego rozwiązania jest to, że procedury systemowe, dzięki temu, że są wskazywane przez symbole, nie są przypisane na stałe do konkretnych adresów. Adresy te mogą się zmieniać z wersji na wersję DOS-u, a mimo to programy będą działać. Co więcej, dzięki zdefiniowaniu nowego symbolu o nazwie takiej samej, jak jeden z już istniejących, nakładka może łatwo przejąć pośrednictwo pomiędzy programem, a procedurą systemową, jaką on wywołuje.

Struktura symbolu (SYMBOL)

Dokładny opis struktury symbolu nie jest wprawdzie do niczego potrzebny, bo programy nie korzystają z nich w sposób wymagający od programisty jej znajomości. Niemniej informacja ta może przynajmniej posłużyć zaspokojeniu ciekawości Czytelnika.

Pojedynczy symbol zajmuje 13 bajtów, składają się nań kolejno:

+$00-$01: wskaźnik do następnego symbolu (2 bajty)

+$02-$09: nazwa symbolu (8 znaków ASCII)

+$0A: bajt kontrolny

+$0B-$0C: adres wskazywany przez symbol (2 bajty)

Nazwa symbolu spełnia podobne warunki, co nazwa pliku: jest to do ośmiu znaków ASCII, przy czym są to na ogół duże litery alfabetu i znak podkreślenia. Gdy nazwa ma mniej niż osiem znaków, dopełniona jest spacjami.

Bajt kontrolny zawiera w dwóch najstarszych bitach indeks pamięci wskazywanej przez symbol: jest to 0 dla pamięci głównej oraz 2 dla dodatkowej. Sześć młodszych bitów to tzw. PID (*Program Identifier*): numer programu, do którego należy symbol. Zero oznacza tu bibliotekę systemową, o której jeszcze będzie mowa.

Funkcje biblioteki pozwalające na dostęp do listy symboli opisane są w jednym z dalszych rozdziałów.

UWAGA: w chwili uruchomienia programu komendą X lista symboli staje się NIEDOSTĘPNA.

Rozdział 2: format bloków binarnych SpartaDOS

Rodzaje bloków binarnych

Atari DOS zna tylko jeden rodzaj bloku pliku binarnego, jest to segment z sześciobajtowym nagłówkiem zaczynającym się (opcjonalnie zresztą) od sygnatury $FFFF. Format ten jest znany, wobec powyższego jego opis zostanie pominięty.

SpartaDOS wyróżnia w sumie siedem rodzajów bloków binarnych. Jednym z nich jest, naturalnie, powyżej wspomniany segment Atari DOS. Nierelokowalny segment SpartaDOS ma identyczną strukturę, jedynie sygnatura nagłówka jest inna: $FFFA zamiast $FFFF. Zmiana sygnatury ma na celu uniemożliwienie odczytu takich binariów przez inne DOS-y, gdyż format ten przeznaczony jest dla plików systemowych (sterowników itp.) SpartaDOS.

Pozostałe pięć rodzajów to:

1) blok rezerwacji pamięci
2) relokowalny blok binarny
3) blok aktualizacji adresów wewnętrznych programu (tzw. *fixupy*)
4) wykazy symboli wymaganych przez program
5) wykazy symboli definiowanych przez program

Blok rezerwacji pamięci

Blok rezerwacji pamięci powoduje, jak poucza sama nazwa, zarezerwowanie przez loader wskazanej ilości wskazanej pamięci. Blok taki składa się wyłącznie z nagłówka liczącego osiem bajtów:

+$00-$01: sygnatura $FFFE
+$02: numer kolejny bloku

+$03: bajt kontrolny o wartości $80 dodać indeks pamięci
+$04-$05: przesunięcie względem początku programu
+$06-$07: liczba bajtów do zarezerwowania

Pamięć zostaje zarezerwowana nad wskaźnikiem wolnej pamięci (czyli np. nad MEMLO w pamięci głównej). Rodzaj pamięci, główna czy dodatkowa, jest wskazany przez indeks znajdujący się w bajcie kontrolnym. O indeksach pamięci więcej napisano w rozdziale „Gospodarka pamięcią”.

Relokowalny blok binarny

Relokowalny blok binarny ma nagłówek w zasadzie identyczny jak blok rezerwacji pamięci.

+$00-$01: sygnatura $FFFE
+$02: numer kolejny bloku
+$03: bajt kontrolny o wartości $00 dodać indeks pamięci
+$04-$05: przesunięcie względem początku programu
+$06-$07: liczba bajtów do załadowania

Oba typy bloków różnią się tylko dwiema rzeczami: po pierwsze, bajt kontrolny w nagłówku bloku relokowalnego ma zgaszony bit 7; jest to sygnał dla loadera, że za nagłówkiem znajdują się dane do wczytania. Druga rzecz to właśnie ten fakt. Dane binarne są ładowane pod adres wskazany przez wskaźnik wolnej pamięci. Rodzaj pamięci, główna czy dodatkowa, jest wskazany przez indeks zapisany w bajcie kontrolnym.

Blok fixupów

Blok aktualizacji adresów wewnętrznych, tzw. *fixupów*, bloku relokowalnego ma pięciobajtowy nagłówek:

+$00-$01: sygnatura $FFFD
+$02: numer kolejny bloku, którego dotyczy fixupowanie
+$03-$04: dwa bajty zarezerwowane

Kolejne bajty to przesunięcia wewnątrz bloku, pierwsze w stosunku do adresu załadowania bloku, każde następne w stosunku do poprzedniego. Innymi słowy, bajt taki oznacza „zwiększ adres o tyle a tyle bajtów i wykonaj fixup”. Fixupowanie polega na tym, że adres ładowania bloku jest każdorazowo dodawany do dwubajtowego słowa znajdującego się pod adresem wskazanym przez bajt przesunięcia.

Bajt przesunięcia wskazuje zawsze dwubajtowe słowo – a więc program relokowalny NIE MOŻE zawierać wartości któregoś ze swoich wewnętrznych adresów podzielonej na młodszy i starszy bajt ulokowane oddzielnie. Konstrukcje w rodzaju:

lda #<adres
ldx #>adres

czy tablice grupujące oddzielnie młodsze i starsze bajty wskaźników są wykluczone.

Takie znaczenie ma każda wartość mniejsza od $FC. Wartości od $FC do $FF to dodatkowe kody sterujące:

$FF: zwiększenie adresu o 250 bajtów (nic poza tym nie jest robione)
$FE: zmiana numeru bloku fixupowanego na wartość nast. bajtu
$FD: zmiana początku bloku na adres zawarty w nast. słowie, i fixup
$FC: znacznik końca bloku

Wykazy wymaganych symboli

Jest to lista symboli, które muszą być zdefiniowane w systemie, żeby ładowany program mógł działać. Brak któregoś powoduje przerwanie ładowania i błąd nr 154 (Loader: Symbol not defined).

+$00-$01: sygnatura $FFFB
+$02-$09: nazwa symbolu (8 znaków)
+$0A-$0B: dwa bajty zarezerwowane

Dalej następują bajty przesunięć oraz kontrolne tak samo, jak w bloku fixupów. Adres wskazywany przez symbol jest dodawany do dwubajtowego słowa znajdującego się pod adresem wskazanym przez bajt przesunięcia.

Definicja symbolu

Blok definicji symbolu powoduje dodanie przez loader nowego symbolu do globalnej listy symboli. Struktura:

$00-$01: sygnatura $FFFC
$02: numer bloku programu, gdzie zdefiniowany jest symbol
$03-$04: przesunięcie
$05-$0C: nazwa symbolu

Adres wskazywany przez nowy symbol to adres ładowania bloku o numerze wymienionym w bajcie $02, plus przesunięcie zawarte w bajtach $03-$04.

Rozdział 3: gospodarka pamięcią

Użytkowanie pamięci przez SpartaDOS

SpartaDOS X rozróżnia następujące rodzaje pamięci:

1) pamięć główna: podstawowe 48k RAM od MEMLO do MEMTOP-u

2) rozszerzenie pamięci: dodatkowe 14k RAM („pod ROM-em", obszary $C000-$CFFF i $D800-$FFFF) w komputerach 800XL i 65XE, lub dodatkowy RAM (obszar $4000-$7FFF) w ilości do 1 MB w komputerach 130XE, lub do 2 MB w komputerach Atari 800. W przypadku braku tej pamięci zamiast niej przydzielana jest pamięć główna.

3) własny moduł ROM: 128k ROM w obszarze $A000-$BFFF.

System dla własnych potrzeb zajmuje pamięć jak następuje:

1) kernel w pamięci głównej, od $0700 do MEMLO.
2) sterowniki systemowe w pamięci dodatkowej
3) biblioteka systemowa w module ROM
4) urządzenie CAR: tamże

Wspominana już wcześniej *biblioteka systemowa* zawiera zbiór procedur pośredniczących pomiędzy programami użytkownika a kernelem DOS-u, oraz jeden program aplikacyjny – mianowicie *SpartaDOS Formatter*. Biblioteka zajmuje dwa banki (po 8k) modułu ROM w SpartaDOS X 4.20, oraz 3 banki w SpartaDOS X 4.40. Działaniu i użytkowaniu biblioteki poświęcona jest główna część niniejszego podręcznika.

Główny moduł biblioteki (bank nr 1) jest normalnie obecny w obszarze $A000-$BFFF, może jednak zostać odłączony i zastąpiony przez pamięć RAM, jeśli program uruchamiany jest poleceniem X. Zwykle dzieje się tak w przypadku zwykłych binariów przeznaczonych dla Atari DOS-u (tych z nagłówkiem $FFFF).

Przydział pamięci RAM dla poszczególnych elementów systemu następuje w zależności od tego, czy jest dostępna, oraz od tego, co wybrał użytkownik w pliku CONFIG.SYS. Możliwe są tu cztery kombinacje:

1) użytkowanie pamięci głównej: USE NONE w CONFIG.SYS. Wszystkie komponenty DOS-u ładowane są do pamięci głównej począwszy od adresu $0700. Ta konfiguracja wybierana jest automatycznie w przypadku komputerów Atari 800 wyposażonych w nie więcej niż 48k RAM.

2) użytkowanie pamięci „pod ROM-em": USE OSRAM w CONFIG.SYS. Kernel zajmuje pamięć od $0700 do MEMLO, pozostałe komponenty DOS-u ładowane są do pamięci w obszarze $E400-$FFFF, obszar $D800-$DFFF zajmowany jest na dane (od wersji 4.40 – wcześniej był wolny), obszar $C000-$CFFF pozostaje wolny, na resztę systemu przydzielana jest pamięć główna. Ta konfiguracja wybierana jest automatycznie w przypadku komputerów Atari 800XL i Atari 65XE wyposażonych w nie więcej niż 64k RAM.

3) użytkowanie pamięci „pod ROM-em": USE OSRAM / DEVICE SPARTA OSRAM w CONFIG.SYS. Jak wyżej, z tym że pamięć $C000-$CFFF zostaje zużytkowana na bufory DOS-u odpowiednio zmniejszając zajętość pamięci głównej.

4) użytkowanie pamięci bankowanej: kernel ładowany jest w obszar od adresu $0700 do MEMLO, dodatkowe komponenty DOS-u, dane i bufory zajmują jeden bank pamięci dodatkowej znajdującej się w obszarze $4000-$7FFF. Ta konfiguracja wybierana jest automatycznie, gdy komputer Atari 800 jest w ogóle wyposażony w takie rozszerzenie

(typu Axlon, do 2 MB RAM w bankach), albo gdy komputer XL/XE ma ponad 128k RAM.

Pamięć dodatkowa rozróżniana jest przez DOS na bank systemowy, w którym rezydują sterowniki, przede wszystkim procedura SPARTA.SYS, oraz całą resztę. Procedury DOS-u zapewniają łatwy dostęp tylko do banku systemowego. Jest to uzasadnione tym, że pamięć przydzielona dla systemu może znajdować się w obszarze głównego RAM-u, pod ROM-em lub w pamięci bankowanej, podłączenie odpowiedniej pamięci system bierze więc na siebie. Natomiast „cała reszta" to rozszerzenie typu 130XE (albo Axlon). SpartaDOS oferuje tu pewne wsparcie – o czym niżej – jednak przełączenie banków program musi wykonać na własną rękę przez ingerencję w odpowiednie rejestry I/O.

Rozpoznawanie konfiguracji pamięci

Nawet programy zasadniczo nieprzeznaczone dla SpartaDOS X mogą być zainteresowane w rozpoznaniu bieżącej konfiguracji pamięci tego DOS-u, a zwłaszcza, które banki pamięci rozszerzonej są przezeń zajęte. Część danych na ten temat zawarta jest w tablicy COMTAB wskazywanej przez wektor DOSVEC ($0A-$0B), oraz przez symbol COMTAB.

Przestroga: należy wystrzegać się traktowania wartości DOSVEC jako stałej. Adres wskazywany przez ten wektor różni się w różnych wersjach SpartaDOS (a nawet w różnych wersjach SpartaDOS X) i nie ma gwarancji, że nie zmieni się w przyszłości. Stałe są tylko przesunięcia (offsety) względem wskazywanego adresu, tak jak podano poniżej.

COMTAB+$1D (NBANKS) zawiera liczbę wolnych banków pamięci dodatkowej typu 130XE lub (na Atari 800) Axlon. Gdy znajduje się tu zero, oznacza to, że pamięć dodatkowa albo nie istnieje, albo jest

całkowicie zajęta przez komponenty DOS-u, tj. sterowniki, ramdyski itp.

COMTAB+$1E (BANKFLG): gdy tu jest $FF, system załadowany jest do pamięci bankowanej (USE BANKED).

COMTAB+$1F (OSRMFLG): gdy tu jest $FF, system załadowany jest do pamięci „pod ROM-em" (USE OSRAM). Pamięć bankowana, o ile istnieje, może być wykorzystana jako ramdysk oraz przydzielona sterownikom w rodzaju CON64.SYS i CON80.SYS.

Dodatkowo programista może chcieć sprawdzić typ rozszerzenia pamięci: gdy pod **COMTAB+$1B** (_800FLG) jest $FF, mamy do czynienia z komputerem Atari 800 i rozszerzeniem Axlon. W przeciwnym wypadku (gdy pod _800FLG jest zero) jest to komputer XL/XE z rozszerzeniem typu 130XE.

Obliczenie, ile w systemie w ogóle jest pamięci dodatkowej (zajętej, czy nie) jest możliwe na podstawie tak zwanej maski PORTB (PBMASK). Znajduje się ona pod adresem COMTAB+$1C i zawiera jedynki we wszystkich bitach, których ustawianie w PORTB powoduje przełączanie banków pamięci w obszarze $4000-$7FFF. Liczy się tu też bit 4 tego portu, tak więc np. na zwykłym 130XE maska ma wartość $1C, czyli %00011100. Wyjąwszy bit 4 są tu ustawione dwa bity, co oznacza cztery dodatkowe banki pamięci, czyli 64k.

Analogicznie przy rozszerzeniu do 320k RAM typu Compy Shop maska ma wartość $DC, czyli %11011100. Wyjąwszy bit 4 są tu ustawione 4 bity. Oznacza to 16 dodatkowych banków pamięci.

Informacja, który konkretnie bank pamięci dodatkowej jest „systemowy", zawarta jest pod adresem T_+$06 lub COMTAB-$0150.[1)]

1 Jedyną właściwą metodą dostępu do tablicy T_ jest odwołanie za pośrednictwem symbolu T_. Jednak ze względu na programy nieprzeznaczone dla SpartaDOS X, a chcące skorzystać z zawartych tam danych, podajemy – w drodze wyjątku – alternatywną metodę obliczenia jej adresu, mianowicie w oparciu o przesunięcie w stosunku do COMTAB (czyli w stosunku do wartości wektora DOSVEC). Nie należy z tego jednak wnosić, że każdy z symboli jest osiągalny ta drogą, raczej przeciwnie, położenie wszystkich innych obiektów zdefiniowanych symbolicznie

Dla USE NONE jest to $FF, natomiast dla USE BANKED i USE OSRAM znajduje się tam wartość, jaką należy wstawić do rejestru PORTB, żeby podłączyć bank, w którym rezyduje główny sterownik SpartaDOS (czyli SPARTA.SYS).

Lista wolnych banków pamięci (T_)

Uzyskanie odwrotnej informacji, tj. nie, które banki są zajęte, ale raczej, które są wolne, jest nieco bardziej skomplikowane. Niemniej właśnie to jest dużo bardziej użyteczne, gdyż np. bank zajęty przez SPARTA.SYS nie musi być jedynym, w jakim rezydują sterowniki DOS-u. By już pominąć kwestię ramdysków, SpartaDOS X od wersji 4.41 ma dwa sterowniki, CON64.SYS i CON80.SYS, które przydzielają sobie dodatkowe banki pamięci. Nadpisanie ich czymkolwiek w momencie, kiedy są uaktywnione, skutkuje oczywiście zawieszeniem komputera prędzej czy później.

Główną daną wejściową do obliczenia listy wolnych banków pamięci jest ich liczba wykazana przez NBANKS (COMTAB+$1D). Do obliczeń konieczne są też dane zawarte w tablicy T_ oraz w rejestrze PBMASK (COMTAB+$1C). Procedura generująca listę wolnych banków wygląda następująco:

```
;FREELIST

sav   = $80
savy  = $81
temp  = $82
index = $83

        lda COMTAB+$1d
        sta sav
        ldy #$00
        sty savy
```

(zmiennych, tablic, procedur) ma się nijak do adresu COMTAB i może ulegać dowolnym zmianom w kolejnych wersjach rozwojowych SpartaDOS.

```
        sty index
loop    ldy sav
        dey
        sty sav
        bmi exit
        tya
        and #$03
        asl
        asl
        sta temp
        tya
        lsr
        lsr
        tax
        lda T_+$08,x
        ora temp
        eor portb
        and COMTAB+$1C
        eor portb
        pha
        iny
        tya
        ldy index
        sta axlon,y
        pla
        sta list,y
        inc index
        bne loop
exit    rts

axlon .ds 128
list  .ds 128
```

W miejscu oznaczonym etykietą „list” procedura zostawi listę wartości rejestru PORTB odpowiadających wolnym bankom pamięci, natomiast pod etykietą „axlon” znajdzie się korespondująca z nią lista banków rozszerzenia Axlon. Liczba wpisów będzie równa liczbie wolnych banków wykazanych przez NBANKS (COMTAB+$1D). Banki zgodne ze 130XE (czyli pierwsze 64k rozszerzenia) będą na tej liście figurowały jako ostatnie. Jest to zgodne z ogólną logiką alokacji tej pamięci, według której najpierw przydzielane są banki najdalsze, tak by

np. na komputerze wyposażonym w 192k RAM SpartaDOS ulokował się w obszarze ponad podstawowymi 128k, a banki 130XE pozostały wolne dla programów chcących ewentualnie z nich skorzystać.

Alokacja pamięci głównej i dodatkowej (MALLOC)

Istnieje kilka dróg, jakimi program instalujący się rezydentnie może powiadomić SpartaDOS, że zajął dla siebie kawałek pamięci RAM. Zwykłe programy binarne w formacie Atari DOS-u (z nagłówkiem $FFFF) mogą w tym celu podnieść wskaźnik MEMLO. Po zwróceniu sterowania do DOS-u informacja o zajętości pamięci (zawarta w tablicy H_FENCE) zostanie na tej podstawie uaktualniona.

Drugi sposób dostępny jest dla relokowalnych binariów SpartaDOS, które zawsze ładowane są w miejsce wskazane wektorem MEMLO – albo, ściślej, w miejsce wskazane przez pierwsze słowo tablicy H_FENCE. Automatyczną alokację obszaru zajętego przez program uzyskuje się przez wstawienie $FF do zmiennej wskazywanej symbolem INSTALL i zwrócenie sterowania do DOS-u. Przed uruchomieniem programu system zeruje zmienną INSTALL, więc zwykłym sposobem „wstawienia $FF" jest jej zmniejszenie o 1.

Dodatkowe obszary pamięci mogą być przydzielane przez procedurę wskazywaną symbolem MALLOC. „Widzi" ona tylko dwa rodzaje pamięci: pamięć główną oraz ten bank rozszerzenia, w którym rezyduje procedura SPARTA.SYS (bank systemowy). Liczbę bajtów, jakie mają być zarezerwowane ponad wskaźnikiem wolnej pamięci, przekazujemy w FAUX4/5 ($0785/6). Do rejestru X należy załadować kod symbolizujący rodzaj pamięci (tzw. indeks pamięci: 0 główna, 2 bank systemowy), a Y wyzerować. Przy alokacji pamięci dodatkowej, gdy w banku systemowym brakuje na to miejsca, system podejmie próbę przydzielenia pamięci głównej. Gdy procedura wykonała się poprawnie (tj. zakończyła

wynikiem dodatnim), wskaźnik do przydzielonego obszaru znajduje się w FAUX1/2 ($0782/3), a indeks przydzielonej pamięci – w rejestrze X.

Pamięci zajętej przez MALLOC nie można potem zwolnić, ta funkcja jest przeznaczona dla programów rezydujących stale w pamięci RAM.

Alokacja banków pamięci

Oprócz pamięci głównej i banku systemowego, obszarem, jaki może chcieć zająć program rezydentny, są wolne banki pamięci dodatkowej. Procedura alokacji jest bardzo podobna do zademonstrowanego powyżej podprogramu generującego listę wolnych banków.

```
;BANKALLOC

temp  = $80

        ldy COMTAB+$1d
        beq error
        dey
        sty COMTAB+$1d
        tya
        and #$03
        asl
        asl
        sta temp
        tya
        lsr
        lsr
        tax
        lda T_+$08,x
        ora temp
        eor portb
        and COMTAB+$1C
        eor portb
        iny
        clc
        rts
error   sec
        rts
```

Podprogram rezerwuje (na stałe, tj. do najbliższego zimnego startu) jeden bank pamięci dodatkowej. Gdy wykona się poprawnie (z C=0), akumulator będzie zawierał kod PORTB podłączający zarezerwowany bank w komputerze XL/XE, natomiast rejestr Y – odpowiednią wartość dla rejestru Axlon w komputerze Atari 800.

Dostęp do banku systemowego

Dostęp do pamięci dodatkowej, jak napisano powyżej, uzyskuje się różnie w zależności od jej rodzaju. Najpierw omówimy prostszą kwestię dostępu do banku systemowego.

UWAGA: obszar pamięci RAM nazywany tu umownie bankiem systemowym może być ulokowany pod różnymi adresami w zależności od konfiguracji pamięci wybranej przez użytkownika. Zwłaszcza trzeba uważać, żeby kod przełączający znajdował się poza obszarem $4000-$7FFF!

Podłączenie banku systemowego realizuje się przez wywołanie systemowej procedury EXT_ON ($07F1) z odpowiednim argumentem przekazanym w akumulatorze. Odłączenie wraz z przywróceniem poprzedniego układu banków – bo wyjściowo niekoniecznie musi być podłączony bank pamięci głównej – zapewnia procedura EXT_OFF ($07F4). Do tej ostatniej nie przekazujemy żadnych argumentów.

Argument dla EXT_ON to kod pamięci dodatkowej, jaką chcemy podłączyć. Nie może on być stałą, gdyż po pierwsze w chwili wywołania nie jest nigdzie powiedziane, że pamięć dodatkowa w ogóle istnieje, a po drugie, nawet jeśli, to czy którykolwiek jej fragment został nam przydzielony przez DOS. *Wobec tego wartość przekazywaną do EXT_ON trzeba zawsze, w ten czy inny sposób, uzyskać najpierw od systemu operacyjnego.*

Metoda numer jeden dotyczy tylko programów zapisanych w

relokowalnym formacie SpartaDOS, w których co najmniej jeden blok binarny ma zostać automatycznie załadowany do dodatkowej pamięci. Ta pamięć to zawsze bank systemowy, chyba że nie ma w nim miejsca – wtedy blok programu ładowany jest do pamięci głównej.

Kod pamięci, do której załadowano bloki binarne przeznaczone do pamięci dodatkowej, znajduje się w momencie uruchomienia programu w zmiennej wskazywanej symbolem EXTENDED. Program instalujący się rezydentnie w systemie powinien zapamiętać jej ówczesną wartość (po powrocie do DOS-u jest ona zerowana), by następnie, w trakcie swojego działania, przekazywać ją jako argument dla procedury EXT_ON, gdy zajdzie potrzeba odwołania do części rezydującej w banku systemowym.

Metoda numer dwa dotyczy programów rezydentnych, które potrzebują dostępu do pamięci zajętej nie przez siebie, lecz przez inne sterowniki systemowe. Klasycznym przykładem takowego jest procedura RAMDISK.SYS. Dane mogą zostać od niej zażądane przez program użytkownika, powinny wtedy na ogół zostać zapisane do głównej pamięci. Jednak równie dobrze programem wywołującym może być procedura SPARTA.SYS żądająca odczytu do buforów DOS-u, a te *mogą* znajdować się w pamięci dodatkowej. W każdym przypadku RAMDISK.SYS musi więc podjąć decyzję, do jakiej pamięci ma wykonać zapis, albo z jakiej odczyt, czy to sektora, czy bloku statusu, czy bloku PERCOM.

Decyzja taka zapada na podstawie zmiennej wskazywanej symbolem SYSCALL. Każdorazowo zawiera ona indeks pamięci, do której żądany jest odczyt, lub z której żądany jest zapis, a więc, jeśli zachodzi taka potrzeba, to właśnie wartość SYSCALL należy wtedy przekazać jako argument do wywołania EXT_ON mającego podłączyć pamięć docelową.

Podłączenie pamięci przez EXT_ON musi mieć zawsze swój odpowiednik w późniejszym wywołaniu EXT_OFF, gdy dostęp do banku

systemowego przestanie być potrzebny.

Dostęp do pozostałych banków rozszerzenia

Jak nadmieniono powyżej, dostęp do pozostałych banków pamięci realizuje się przez zapisywanie odpowiednich wartości wprost do rejestrów sterujących pamięcią, tj. PORTB ($D301) w komputerach XL/XE oraz Axlon ($CFFF) w komputerach 400/800. Jako wartości do zapisu należy wziąć wyniki działania procedur BANKALLOC lub FREELIST wydrukowanych na poprzednich stronach. Ze względu na oddzielne adresowanie pamięci przez ANTIC i CPU, przy zapisie PORTB dobrą praktyką jest zmiana tylko tych bitów bieżącej wartości, które odpowiadają za przełączenie banku, a pozostawienie reszty bez zmian. Robi się to w następujący sposób:

```
;BANKSWITCH

        lda portb
        sta old_pb
        lda new_pb
        eor portb
        and COMTAB+$1c      ;PBMASK
        eor portb
        sta portb
```

Dotychczasowa wartość PORTB zostaje przechowana w bajcie oznaczonym etykietą „old_pb”, „new_pb” to pobrany z FREELIST kod banku, który chcemy podłączyć.

Program przełączający pamięć na któryś z banków rozszerzenia, gdy zakończy korzystanie z dodatkowej pamięci, powinien przywrócić jej konfigurację początkową. W przypadku XL/XE jest to zupełnie proste, wystarczy, jak pokazano wyżej, przed przełączeniem zapamiętać gdzieś wartość PORTB, a potem ją przywrócić.

Przy rozszerzeniu Axlon sprawa się komplikuje, gdyż rejestr sterujący bankami jest tylko do zapisu – można więc łatwo przełączyć banki w jedną stronę, ale przywrócić układu sprzed przełączenia już tak prosto nie można, bo odczyt spod $CFFF nie zwraca stanu rejestru, lecz jakieś przypadkowe śmieci. Nie da się więc zapamiętać bieżącej konfiguracji pamięci przed przełączeniem.

Trudność tę można rozwiązać na dwa sposoby. Pierwszym – i łatwiejszym – jest zablokowanie działania programu na komputerach innych niż XL/XE. Osiąga się to przez sprawdzenie _800FLG (COMTAB+$1B). Gdy przełączanie pamięci ma krytyczne znaczenie dla szybkości działania – jak to jest np. w przypadku sterowników CON64.SYS i CON80.SYS, gdzie wysłanie na ekran każdego pojedynczego znaku pociąga za sobą bankowanie pamięci – wtedy po prostu nie ma innego wyjścia, jak pogodzić się z tym, że nasz kod na 400/800 z rozszerzeniem Axlon działać nie będzie.

Drugi sposób wiąże się z zastosowaniem pewnego triku. Otóż bieżący układ banków rozszerzenia Axlon zna DOS, bo to (w zasadzie tylko) on je przełącza. Zapamiętanie tego stanu osiągamy przez wywołanie EXT_ON z wartością zapewniającą udostępnienie banku systemowego (czyli z EXTENDED lub kodem zwróconym przez MALLOC w rejestrze X, patrz wyżej). DOS przełączy przy tym banki pamięci, ale to nas nie interesuje. Gdy funkcja EXT_ON odda sterowanie, można wpisać żądaną wartość do rejestru sterującego Axlon, co udostępni programowi bank, o który chodzi. Natomiast przywrócenie poprzedniego stanu uzyskuje się po prostu przez wywołanie EXT_OFF.

Rozdział 4: obsługa błędów

Standardowa procedura obsługi błędu (U_FAIL)

Obsługa błędu w SpartaDOS polega na wywołaniu procedury bibliotecznej U_FAIL z kodem błędu w akumulatorze. U_FAIL może być wywołana jawnie z programu użytkownika, na ogół jednak dochodzi do tego w sposób niejawny, gdy procedurę tę, stwierdziwszy jakieś nieprawidłowości, automatycznie wywołuje biblioteka systemowa.

U_FAIL ma tę nieprzyjemną cechę, że nigdy nie wraca. Innymi słowy, program wywołujący jest przerywany i usuwany z pamięci, otwarte pliki są zamykane, a system wypisawszy na konsoli stosowny komunikat błędu oddaje sterowanie do interpretera poleceń. Jest to wygodne w przypadku błędów krytycznych, po wystąpieniu których program i tak nie może się wykonywać dalej, jednakże stosunkowo często zachodzi też potrzeba obsłużenia błędu wewnątrz programu.

Zakładanie pułapki (U_SFAIL)

Program chcący przejąć obsługę konkretnego błędu może zastawić pułapkę. Służy do tego procedura biblioteczna U_SFAIL. Przed jej wywołaniem do rejestrów AX trzeba załadować odpowiednio młodszy i starszy bajt adresu miejsca w programie, do którego zostanie przekazane sterowanie w chwili wystąpienia błędu.

Można założyć więcej niż jedną pułapkę – reaguje zawsze ta, która była założona ostatnio. Jednak z liczbą jednocześnie założonych pułapek nie należy przesadzać, program nie może mieć ich więcej niż 10 na raz.

Gdy błąd wystąpi przy założonej pułapce, zostaje ona przede wszystkim usunięta (jest jednorazowa). Następnie biblioteka ładuje wskaźnik stosu wartością zapamiętaną przez U_SFAIL, przełącza banki

pamięci w stan zapamiętany tamże i wykonuje skok JMP pod adres wskazany przez użytkownika przy zakładaniu ostatniej pułapki. Kod błędu, jaki wystąpił, jest przy tym ładowany do akumulatora, a znacznik N rejestru znaczników procesora – ustawiany na jeden. Reszta pozostaje bez zmian, tj. zwłaszcza nie są zamykane pliki, które program poprzednio otworzył.

Zdejmowanie pułapki (U_XFAIL)

Jako się rzekło, pułapka jest usuwana automatycznie, gdy wystąpi błąd. Jednak gdy błąd nie wystąpi, pułapka na ogół staje się niepotrzebna i należy usunąć ją „ręcznie”. Służy do tego procedura U_XFAIL. Nie przekazujemy jej żadnych parametrów, usuwa ona po prostu pułapkę założoną ostatnio. U_XFAIL nie zmienia zawartości rejestrów CPU.

Wszystkie pułapki założone przez program usuwane są w momencie jego zakończenia i oddania sterowania do DOS-u.

Komunikat błędu (U_ERROR)

U_ERROR wyświetla na ekranie systemowy komunikat błędu, którego kod przekazany został w akumulatorze. Przedtem wysyłany jest do edytora znak EOL ($9B). Jest to część standardowej procedury obsługi błędu uruchamianej przez U_FAIL.

Przykład użycia pułapki

```
;ustawienie pulapki na adres wskazany
;przez wektor trapptr, zawiera on adres
;oznaczony tu etykieta trap

        lda trapptr
        ldx trapptr+1
```

```
        jsr U_SFAIL

;tu wywolujemy procedure systemowa
;w ktorej oczekujemy wystapienia bledu

        ...

;gdy bledu nie bylo, usuwamy pulapke

        jsr U_XFAIL

;gdy blad wystapi, system sam zdejmie
;pulapke i odda sterowanie w to miejsce
;z N=1 i kodem bledu w akumulatorze

trap  bmi error

;tu dalsze postepowanie w przypadku
;bezblednego przebiegu procedury

        ...
        rts

;tu obsluga bledu

error jmp U_ERROR
```

Rozdział 5: obróbka wiersza poleceń

Bufor LBUF i indeks BUFOFF

Komenda wydana interpreterowi poleceń DOS-u, czy to bezpośrednio przez użytkownika z klawiatury, czy odczytana z pliku wsadowego, zapisywana jest w buforze LBUF (*line buffer*). Ma on 64 bajty i znajduje się pod adresem COMTAB+$3F. Bieżącą pozycję w tym buforze, tj. na ogół następny parametr, wskazuje indeks BUFOFF (*buffer offset*, COMTAB+$0A).

Bufor LBUF i indeks BUFOFF dostarczają danych *wejściowych* dla procedur biblioteki zajmujących się obróbką wiersza poleceń, i programy normalnie nie mają potrzeby interesować się ich zawartością. Może jednak czasem zajść potrzeba więcej niż jednokrotnego odczytu danego elementu wiersza poleceń. Należy wtedy zapamiętać stan BUFOFF i przywrócić go przed ponownym wywołaniem procedury bibliotecznej (z których wszystkie automatycznie zwiększają BUFOFF ustawiając go na następną pozycję do obróbki).

Trzeba zwrócić uwagę, że pierwszym parametrem w LBUF jest na ogół nazwa wywoływanego programu. W momencie jego uruchomienia BUFOFF wskazuje jednak na następny parametr – gdyż nazwa programu została już odczytana przez interpreter poleceń DOS-u. Żeby pobrać tę nazwę, trzeba wyzerować BUFOFF.

Bufor COMFNAM

Wyjście procedur obróbki wiersza poleceń, tych przynajmniej, których wyniki mają postać tekstową, znajduje się pod adresem COMTAB+$21. Jest to bufor COMFNAM (*complete file name*) o długości 30 bajtów. Związaną z nim zmienną jest TRAILS (*trailing*

space, COMTAB+$1A), zawiera ona długość znajdującego się w COMFNAM parametru.

O ile LBUF zawiera cały wiersz polecenia, o tyle COMFNAM jest przeznaczony na pojedynczy parametr. LBUF znajduje się zaraz za COMFNAM, w zasadzie te dwa bufory łączą się ze sobą. Wynika z tego, że bardzo długie parametry (ponad 30 znaków) pobrane przez bibliotekę z wiersza poleceń i wkopiowane do COMFNAM mogą nadpisywać początkową część LBUF.

Odczytywanie elementów wiersza polecenia

Wiersz polecenia składa się z grupy elementów, tj. oddzielonych od siebie spacjami nazw plików, przełączników, opcji itp. zapisanych na ogół w kolejności, która jest z góry znana i zdefiniowana jako składnia danego polecenia. Wynika z tego, że program realizujący polecenie odczytując linię komend od początku do końca element za elementem może w większości wypadków z góry przewidzieć, jakiego rodzaju parametr wystąpi jako następny. Program przy tym nie musi parsować wiersza poleceń na piechotę – aczkolwiek, jeśli zachodzi taka potrzeba, oczywiście może – gdyż biblioteka oferuje procedury przetwarzania najczęściej spotykanych typów parametrów. Omówimy teraz takie proste przypadki, gdy typ parametru jest znany z góry, kwestię analizy bardziej skomplikowanych linii komend zostawiając na koniec.

Parametry numeryczne (U_GETNUM)

Stosunkowo najprostsze jest pobranie i interpretacja parametru numerycznego. Może to być liczba z zakresu od 0 do 65535 podana w notacji dziesiętnej lub szesnastkowej. Wywołanie U_GETNUM zwraca zero (Z=1), gdy parametr nie jest liczbą. W przeciwnym wpadku (Z=0) w

rejestrach AX znajduje się odpowiednio młodszy i starszy bajt wartości binarnej odpowiadającej podanej liczbie.

Gdy parametrów jest więcej, mogą być rozdzielone przecinkami albo spacjami. Z U_GETNUM korzystają np. komendy PEEK i POKE interpretera poleceń SpartaDOS.

Przełącznik dwustanowy: ON i OFF (U_GONOFF)

Niektórymi komendami użytkownik tylko coś włącza lub wyłącza podając odpowiednio ON lub OFF jako parametr. Do obróbki tego typu parametru służy procedura biblioteczna U_GONOFF. Gdy parametrem jest ON, znacznik C rejestru znaczników procesora ustawiany jest na 1, a gdy OFF – to na zero. Kiedy podanym parametrem nie jest ani ON ani OFF, procedura zgłasza błąd nr 156 (Bad parameter) oddając sterowanie do procedury U_FAIL i tym samym przerywając program. Jak sobie z tym poradzić, opisaliśmy w rozdziale pod tytułem „Obsługa błędów”.

W opisany sposób z linią komend postępuje nakładka KEY.COM.

Opcje (U_SLASH)

Parametry do niektórych programów wygodnie jest przekazać w postaci jednoznakowych „opcji” poprzedzonych znakiem ukośnika „/”. Do odczytu takowych z wiersza poleceń służy procedura U_SLASH. Wszystkie rozpoznawane opcje trzeba umieścić w tablicy. Jeden wpis tej tablicy, dotyczący jednej opcji, to dwa bajty, kolejno: znacznik wystąpienia opcji oraz odpowiadająca jej litera. Przykładowo, jeśli program chce reagować na opcje /A i /X, tablica powinna wyglądać następująco:

```
switch
```

```
?a      .byte 0
        .byte 'A'
?x      .byte 0
        .byte 'X'
```

Adres tablicy trzeba przekazać procedurze U_SLASH w rejestrach AX, a całkowitą wielkość w bajtach – w rejestrze Y. Gdy na analizowanej pozycji linii komend podana jest któraś z oczekiwanych opcji, wtedy odpowiedni znacznik, w powyższym przykładzie są one oznaczone jako „switch?a” i „switch?x”, przybierze wartość $FF.

Gdy podanej opcji brakuje w tablicy, U_SLASH przekazuje sterowanie do U_FAIL przerywając program błędem nr 156 (Bad parameter). Natomiast gdy bieżąco obrabiany parametr w ogóle nie jest opcją (tj. nie zaczyna się od znaku „/”), U_SLASH wraca do miejsca wywołania nic nie robiąc.

W opisany sposób wiersz polecenia interpretuje komenda MEM.

Słowo kluczowe (U_TOKEN)

Analizę wiersza poleceń pod kątem występowania określonych słów kluczowych zapewniają dwie procedury: U_GETPAR oraz U_TOKEN.

U_GETPAR kopiuje kolejny (tj. ten wskazywany przez BUFOFF) parametr z LBUF do COMFNAM, uaktualnia BUFOFF, wpisuje długość parametru do TRAILS, a do wektora FILE_P – adres COMFNAM.

U_TOKEN pobiera parametr tekstowy znajdujący się w COMFNAM i próbuje znaleźć go w tablicy słów kluczowych, której adres program przekazał w rejestrach AX. Słowa kluczowe muszą być umieszczone w tablicy jedno za drugim, przy czym ostatni znak każdego musi być w negatywie (tj. z ustawionym bitem 7). Koniec tablicy oznaczamy przez zero.

Gdy U_TOKEN nie odnajdzie słowa kluczowego w tablicy, wraca ze

skasowanym znacznikiem C. W przeciwnym wypadku, gdy odnajdzie, znacznik C jest ustawiony, a akumulator zawiera numer kolejny (czyli *token*) słowa kluczowego w tablicy, licząc od zera.

W ten sposób postępuje instrukcja pliku wsadowego IF oraz interpreter poleceń SpartaDOS.

Nazwa urządzenia (U_GETPAR/U_GEFINA)

Gdy parametrem ma być sama nazwa urządzenia, na przykład identyfikator dysku, do jej pobrania trzeba użyć pary procedur U_GETPAR i U_GEFINA. U_GETPAR, jak już napisano powyżej, pobiera parametr z LBUF i wstawia go do COMFNAM, natomiast adres COMFNAM zapisuje do wektora FILE_P.

U_GEFINA natomiast pobiera parametr z miejsca wskazanego przez FILE_P, interpretuje go jako nazwę urządzenia i według tego odpowiednio ustawia rejestr DEVICE ($0761). Jego młodsze cztery bity oznaczają numer urządzenia (np. numer stacji dysków), starsze natomiast kodują rodzaj urządzenia jak następuje:

$0x – dysk
$1x – zegar
$2x – urządzenie CAR:
$3x – urządzenie CON:
$4x – urządzenie PRN:
$5x – urządzenie COM:
$6x – urządzenie NUL:
$7x – zarezerwowane

Gdy żadnego identyfikatora nie podano, przyjmowany jest kod bieżąco ustawionego urządzenia (na ogół, bieżącego dysku) pobrany ze

zmiennej wskazywanej symbolem CURDEV. Natomiast gdy podany identyfikator jest błędny i nie daje się zdekodować, sterowanie przekazywane jest do U_FAIL z kodem błędu nr 130 (Nonexistent device).

W podany sposób z U_GETPAR i U_GEFINA korzysta np. polecenie CHKDSK.

Specyfikacja pliku (U_GETPAR/U_GEFINA)

Opisana powyżej sekwencja U_GETPAR/U_GEFINA może też być użyta do pobrania nazwy pliku. Identyfikator urządzenia, jak wyżej, tłumaczony jest wtedy na wartość DEVICE ($0761), a reszta specyfikacji rozdzielana jest na ścieżkę dostępu kopiowaną do PATH ($07A0, 64 bajty), oraz nazwę pliku wstawioną do NAME ($0762, 11 bajtów). Nazwa zostaje przy tym przetłumaczona na format wewnętrzny DOS-u, tj. do postaci NNNNNNNNXXX. Gdy któraś część nazwy, tj. główna lub rozszerzenie, ma mniej znaków niż odpowiednie 8 lub 3, zostaje uzupełniona spacjami, ewentualne jokery (*wildcards*) zostają rozwinięte w ciągi znaków zapytania itp.

Taka postać specyfikacji pliku jest strawna dla kernela SpartaDOS, przede wszystkim dla sterownika DSK: zawartego w SPARTA.SYS. Wywołujące go funkcje biblioteki systemowej na ogół same wykonują skok do U_GEFINA, program użytkownika potrzebuje więc tylko użyć U_GETPAR przedtem.

Specyfikacja katalogu (U_GETPAR/U_GEPATH)

Gdy oczekiwanym parametrem jest specyfikacja katalogu, postępujemy podobnie jak wyżej, z tą tylko zmianą, że jako drugą z procedur, zamiast U_GEFINA, trzeba wywołać U_GEPATH. Dokonuje

ona tłumaczenia nazwy urządzenia na kod DEVICE ($0761) tak samo, jak U_GEFINA, a kompletna ścieżka dostępu jest kopiowana do PATH ($07A0, 64 bajty). Do NAME nic nie jest wstawiane.

Taka forma specyfikacji pliku jest użyteczna przy wywoływaniu procedur kernela nr 16 (kchdir) i 17 (kgetcwd). Odpowiednie funkcje biblioteki (CHDIR i GETCWD) same wywołują U_GEPATH, program użytkownika potrzebuje więc tylko wywołać U_GETPAR przedtem.

Specyfikacja pliku i atrybutów (U_GETATR)

Niektóre polecenia, jak COPY czy DUMP, przyjmują jako parametr specyfikację pliku z opcjonalnym wyszczególnieniem atrybutów. Np. *DUMP +H FOO.BAR* wyświetli zawartość pliku ukrytego (z atrybutem +H) FOO.BAR. Normalnie natomiast plik ukryty zostanie zignorowany.

Zadanie analizy takiego parametru wykonuje procedura U_GETATR. Robi to samodzielnie. tj. nie ma potrzeby uprzedniego wywoływania U_GETPAR. W akumulatorze należy przedtem przekazać domyślne atrybuty dla pliku, w razie gdyby użytkownik nie podał żadnych. Maskę bitową atrybutów domyślnych zestawiamy według następującego schematu:

+$01: zabezpieczony (+P)
+$02: ukryty (+H)
+$04: archiwalny (+A)
+$08: katalog (+S)
+$10: nie zabezpieczony (-P)
+$20: nie ukryty (-H)
+$40: nie archiwalny (-A)
+$80: nie katalog (-S)

Domyślnymi maskami stosowanymi najczęściej są: $20 (nie ukryty) i $A0 (nie ukryty i nie katalog).

U_GETATR zwraca zero (Z=1), gdy w LBUF jest za mało parametrów do pobrania – tj. np. podano atrybut, ale nie podano nazwy pliku. Maska bitowa atrybutów wstawiana jest do FATR1 ($0779), jest to jedno z kryteriów poszukiwania plików w katalogu przez funkcje biblioteczne FFIRST i FNEXT (a tym samym również przez wszystkie inne funkcje z nich korzystające, przede wszystkim FOPEN). Poza tym szczegółem U_GETATR działa tak samo, jak U_GETPAR, to jest, jak już napisano powyżej, pobiera parametr z LBUF i wstawia go do COMFNAM, długość parametru zapisuje w TRAILS, natomiast adres COMFNAM wpisuje do wektora FILE_P.

Specyfikacja pliku z domyślną maską (U_FSPEC)

Niektóre polecenia, np. COPY, przyjmują jako parametr ścieżkę dostępu wraz ze specyfikacją pliku lub maską wybierającą grupę plików. Maska przy tym może być pominięta, gdy ma nią być '*.*' – COPY FOO> ma skopiować wszystkie pliki (*.*) z katalogu FOO. Program realizujący polecenie musi oczywiście, w ramach obróbki zadanych parametrów, rozpoznać, czy nazwa pliku lub maska została podana, a gdy stwierdzi, że nie, dodać ją.

To nieskomplikowane wprawdzie, ale uciążliwe zadanie – trzeba sprawdzić kilka warunków – realizuje funkcja biblioteczna U_FSPEC. Wywołana bezpośrednio po U_GETPAR lub U_GETATR sprawdza, czy znajdujący się w COMFNAM parametr spełnia warunki konieczne do tego, by dopisać na końcu maskę *.* i, gdy tak jest, dopisuje ją poprawiając zarazem wartość TRAILS.

Kombinacje różnych typów parametrów

Najczęściej spotyka się sytuację, kiedy program pobiera najpierw nazwę pliku, a następnie opcje zaczynające się od ukośnika. Analiza takiej komendy nie nastręcza żadnych trudności, program najpierw powinien postępować tak, jak przy pobieraniu specyfikacji pliku (konkretne przypadki tego są opisane powyżej), a następnie wywołać U_SLASH w celu odczytania opcji.

Trudniejszy przypadek to komenda w rodzaju ECHO, która ma dwie postaci: ECHO ON/OFF lub ECHO TEKST. W pierwszej postaci włączane lub wyłączane jest echo komend wykonywanych przez interpreter poleceń. Druga postać po prostu wyświetla podany tekst na ekranie. Trudność polega tu na tym, że do rozpoznania przełącznika ON/OFF trzeba wywołać procedurę U_GONOFF, a ta, gdy parametrem nie jest ani ON ani OFF (lecz TEKST), bezpowrotnie przerywa program skokiem do U_FAIL.

Jedynym rozwiązaniem jest zastawienie pułapki na błąd, jaki może wygenerować U_GONOFF. Zostało to opisane w rozdziale pod tytułem „Obsługa błędów”.

Rozdział 6: obróbka nazwy pliku

Konwersja z 8+3 na format wewnętrzny (PRO_NAME)

Funkcja biblioteczna PRO_NAME przekształca nazwę pliku wskazywaną przez wektor *bufadr* ($15) z przesunięciem Y-1 na format wewnętrzny i zapisuje go w buforze NAME ($0762-$076C). W przypadku, gdy nazwa jest nazwą pliku, funkcja wraca z wynikiem niezerowym (Z=0) i znakiem EOL ($9B) w akumulatorze. Gdy jest to nazwa katalogu, wynik jest zerowy (Z=1), a w akumulatorze znajduje się separator ‘>’ lub ‘<’.

Przekształcenie do formatu wewnętrznego polega na rozdzieleniu nazwy pliku na część główną i rozszerzenie, ewentualnym uzupełnieniu obydwu części spacjami oraz równie ewentualnym rozwinięciu jokerów ‘*’ w ciągi znaków zapytania.

PRO_NAME jest procedurą usługową, z której korzystają opisane w poprzednim rozdziale funkcje U_GEFINA i U_GEPATH. W związku z tym programy użytkownika nigdy normalnie nie mają potrzeby jej bezpośrednio wywoływać.

Konwersja z formatu wewnętrznego na 8+3 (U_EXPAND)

Odwrotnego przekształcenia, tj. nazwy zapisanej w formacie wewnętrznym i znajdującej się w NAME ($0762-$076C) na format 8+3 dokonuje funkcja biblioteczna U_EXPAND. Wynik zostanie wpisany do bufora o adresie AX, od pozycji Y. Powstały ciąg znaków zakończony jest znakiem EOL ($9B).

Rozdział 7: zmienne środowiskowe

Co to jest zmienna środowiskowa

SpartaDOS X jest jedynym DOS-em na ośmiobitowe Atari, który implementuje znane z większych komputerów zmienne środowiskowe. Zmienna środowiskowa jest ciągiem znaków ASCII z przypisaną unikalną nazwą. Zmienne te przechowują niektóre globalne ustawienia dla systemu, interpretera poleceń lub programów aplikacyjnych. Przykładem pierwszego rodzaju są zmienne $PATH i $DAYTIME, drugiego $COPY, trzeciego $MANPATH.

Zmienne środowiskowe przechowywane są w dodatkowej pamięci w buforze o wielkości 256 bajtów. Biblioteka oferuje trzy funkcje pozwalające na łatwy dostęp do danych tam zawartych.

Odczyt zmiennej wg jej nazwy (GETENV)

Do odczytania wartości zmiennej o znanej nazwie służy funkcja GETENV. Adres nazwy zmiennej, jaką funkcja ma znaleźć, trzeba podać w AX (nazwa ta powinna być zakończona znakiem EOL albo znakiem równości „=”). Zakończenie z wynikiem ujemnym (N=1) oznacza, że w buforze nie ma takiej zmiennej. W przeciwnym wypadku wynik jest dodatni (N=0), a wartość zmiennej zapisana jest w postaci ciągu znaków ASCII zakończonego znakiem EOL w buforze wyjściowym pakietu matematycznego LBUFF ($0580).

Odczyt zmiennej wg jej numeru (NUMENV)

Alternatywnie można wyszukiwać zmienne nie według nazw, lecz według ich numerów kolejnych w buforze. Służy do tego procedura

NUMENV. Numer zmiennej trzeba jej podać w akumulatorze, gdy wynik jest dodatni (N=0), wartość zmiennej jest zapisywana w takim samym formacie i w to samo miejsce, co w przypadku GETENV (patrz wyżej).

Funkcja ta na ogół wykorzystywana jest do odczytu wszystkich zmiennych środowiskowych po kolei w celu np. wyświetlenia ich na ekranie (tak jak to robi polecenie SET interpretera poleceń), przeszukania całości itp.

Zapis i kasowanie zmiennych (PUTENV)

Do zapisu zmiennej do bufora służy funkcja PUTENV. Adres nowej treści zmiennej należy podać w AX. Pod tym adresem powinien się znajdować ciąg znaków ASCII zakończony znakiem EOL ($9B) w postaci: NAZWA=TEKST

Spowoduje to utworzenie zmiennej NAZWA i przypisanie jej tekstu „TEKST" jako wartości. Gdy w buforze środowiskowym zmienna o tej nazwie już istnieje, jest przedtem kasowana.

Podanie do PUTENV samej nazwy zmiennej, to jest ciągu znaków ASCII zakończonego przez EOL i niezawierającego znaku równości spowoduje skasowanie z bufora zmiennej o takiej nazwie.

UWAGA: przy kombinowanym użyciu NUMENV i PUTENV w jednej pętli do wyszukiwania określonych zmiennych i kasowania ich, należy pamiętać, że skasowanie zmiennej powoduje zmniejszenie o 1 numerów wszystkich zmiennych znajdujących się po niej w buforze. Dlatego w takiej pętli po wywołaniu PUTENV kasującym zmienną NIE należy zwiększać licznika dla NUMENV.

Rozdział 8: odczyt i zapis plików

Otwieranie plików (FOPEN)

Otwarcie pliku realizuje funkcja FOPEN. Wymagane parametry otwarcia przekazywane są jak następuje:

1) specyfikację pliku powinien wskazywać wektor FILE_P
2) tryb otwarcia wpisujemy do OPMODE ($0778)
3) maskę atrybutów poszukiwanych do FATR1 ($0779)

Specyfikacja pliku wskazywana przez FILE_P powinna mieć postać tekstową (ciąg ASCII zakończony znakiem EOL). Specyfikację urządzenia trzeba podać w konwencji SpartaDOS X, a nie Atari OS – tj. np. A:>KATALOG>PLIK.TXT zamiast D1:>KATALOG>PLIK.TXT.

OPMODE ma taką samą funkcję, jak bajt ICAX1 kanału I/O XL OS przy otwieraniu pliku (przy wywołaniu funkcji OPEN urządzenia D: za pośrednictwem CIO wartość OPMODE jest pobierana właśnie z ICAX1). Wartość tej zmiennej składa się z dwóch półbajtów. Młodszy sygnalizuje tryb dostępu do danych:

$x4 – odczyt
$x8 – zapis
$x9 – dopisywanie
$xC – wymiana danych (odczyt i zapis)

Starszy to maska bitowa, w której ustawienie kolejnych bitów ma następujące znaczenie:

7 – długi format katalogu

6 – tryb śledzenia atrybutów

5 – otwarcie z przeszukiwaniem szlaku ($PATH)

4 – bezpośredni dostęp do katalogu

Maska atrybutów poszukiwanych FATR1 ($0779) używana jest przy otwieraniu pliku do odczytu. Plik o podanej nazwie zostanie wyszukany w katalogu i otwarty tylko wtedy, kiedy spełni warunek zakodowany bitami FATR1:

+$01: zabezpieczony (+P)

+$02: ukryty (+H)

+$04: archiwalny (+A)

+$08: katalog (+S)

+$10: nie zabezpieczony (-P)

+$20: nie ukryty (-H)

+$40: nie archiwalny (-A)

+$80: nie katalog (-S)

Normalnie przy otwieraniu zwykłych plików stosuje się maskę $A0 (*nie ukryty* i *nie katalog*).

W wypadku błędu sterowanie przekazywane jest do U_FAIL (patrz rozdział „Obsługa błędów”). Gdy otwarcie się powiodło, uchwyt pliku (ang. *handle*) wpisywany jest do FHANDLE ($0760).

Zamykanie plików (FCLOSE/FCLOSEAL)

Biblioteka oferuje tu dwie funkcje: FCLOSE zamyka konkretny plik, ten mianowicie, którego uchwyt w chwili jej wywołania znajduje się w FHANDLE ($0760). FCLOSEAL natomiast zamyka wszystkie pliki, jakie są otwarte w danej chwili.

Istnieje możliwość zabezpieczenia pliku przed zamknięciem przez FCLOSEAL. Należy w tym celu wpisać jego uchwyt do FHANDLE ($0760) i wywołać funkcję FCLEVEL z wartością $FF w akumulatorze. Identyczne postępowanie przy wartości akumulatora ustawionej na aktualną wartość zmiennej SYSLEVEL znosi ochronę.

Odczyt i zapis pojedynczych bajtów (FGETC/FPUTC)

Odczyt i zapis pojedynczych bajtów pliku, którego uchwyt znajduje się w FHANDLE ($0760), realizują funkcje FGETC i FPUTC.

FGETC zwraca odczytany bajt w akumulatorze. Gdy nastąpił koniec pliku, w akumulatorze będzie znak EOL ($9B), w rejestrze X wartość $FF, a rejestr znaczników będzie wskazywał wynik ujemny (N=1). Każdy inny błąd powoduje przekazanie sterowania do U_FAIL. FGETC nie zmienia zawartości rejestru Y.

FPUTC wysyła do pliku bajt przekazany w akumulatorze. Wywołanie nie zmienia zawartości rejestrów A, X i Y. Wystąpienie błędu powoduje przekazanie sterowania do U_FAIL.

Operacje we/wy wykonywane przez FGETC i FPUTC dla urządzeń DSK: i CAR: są przez bibliotekę mikrobuforowane, dzięki czemu przebiegają znacznie szybciej niż odczyty i zapisy pojedynczych bajtów funkcjami FREAD i FWRITE.

Odczyt i zapis rekordów (FGETS/FPUTS)

Odczyt i zapis rekordów pliku, którego uchwyt jest w FHANDLE ($0760) realizują funkcje FGETS i FPUTS. Parametry dla obydwu przekazuje się w rejestrach: w AX adres bufora, w Y jego długość. Rekord jest to ciąg znaków ASCII zakończony znakiem EOL i nie dłuższy niż 255 bajtów.

Gdy odczyt przez FGETS przebiegnie poprawnie, funkcja wraca z wynikiem dodatnim (N=0), zerem w akumulatorze i liczbą odczytanych bajtów w Y. Gdy wystąpi nadmiar danych, wynik jest ujemny (N=1), a w akumulatorze jest $FF. Sytuacja ta odpowiada wystąpieniu błędu nr 137 (Truncated record) w XL OS. Każdy inny błąd powoduje automatyczne przekazanie sterowania do U_FAIL.

Przy zapisie przez FPUTS, jeśli ciąg jest krótszy niż wartość Y w chwili wywołania, musi być zakończony znakiem EOL ($9B). Wystąpienie błędu powoduje automatyczne wywołanie U_FAIL.

FGETS i FPUTS korzystają z funkcji bibliotecznych FGETC i FPUTC, dzięki czemu dla urządzenia CAR: odczyty, a dla DSK: zapisy i odczyty rekordów są mikrobuforowane.

Zapis formatowanego tekstu do pliku (FPRINTF)

Zapis formatowanego tekstu do pliku realizuje funkcja biblioteczna FPRINTF. Działa ona identycznie do opisanej w następnym rozdziale funkcji PRINTF (są to dwa różne wejścia do tej samej funkcji), z tym tylko, że zapis jest wykonywany do pliku, którego uchwyt znajduje się w FHANDLE ($0760), a nie na konsolę.

Podobnie jak w przypadku zapisu rekordów, FPRINTF wysyła dane za pośrednictwem FPUTC, dzięki czemu zapis na dysk jest mikrobuforowany.

Odczyt i zapis bloków binarnych (FREAD/FWRITE)

Odczyt i zapis bloków binarnych realizują funkcje FREAD i FWRITE. Parametry, oprócz uchwytu pliku w FHANDLE ($0760) przekazujemy w:

- FAUX1/2 ($0782-$0783): adres bufora

- FAUX4/5 ($0785-$0786): wielkość bufora

Wielkość bufora musi być większa od zera. W przypadku powodzenia funkcje wracają z wynikiem dodatnim (N=0). Gdy w FREAD nastąpi koniec pliku, wynik jest ujemny (N=1). Każdy inny błąd powoduje skok do U_FAIL.

FREAD i FWRITE nie są mikrobuforowane, dlatego używanie tych funkcji do odczytu lub zapisu bardzo małych porcji danych (po kilka albo kilkanaście bajtów) nie opłaca się. Lepiej w tym celu użyć FGETC i FPUTC.

Odczyt długości pliku (FILELENG)

Funkcja FILELENG odczytuje długość otwartego pliku, którego uchwyt jest w FHANDLE ($0760) i umieszcza wynik w FAUX1-3 ($0782-$0784).

Zmiana pozycji w pliku (FTELL/FSEEK)

Do odczytania bieżącej pozycji odczytu lub zapisu do pliku służy funkcja FTELL. Zapisuje ona do rejestrów FAUX1-3 ($0782-$0784), licząc od początku pliku, numer bajtu, jaki zostanie odczytany lub zapisany w następnej operacji I/O.

Operację odwrotną, tj. zmianę tej pozycji przeprowadza funkcja FSEEK. Nową pozycję trzeba jej podać w FAUX1-3 ($0782-$0784).

W obu przypadkach uchwyt pliku musi być zapisany w FHANDLE ($0760).

Przy próbie ustawienia pozycji poza końcem pliku otwartego do odczytu sterowanie zostanie przekazane do U_FAIL z błędem nr 166 (Range error). Natomiast podobna operacja na pliku otwartym do zapisu nie wywołuje błędu, następujący po tym zapis danych i zamknięcie

powoduje na ogół powstanie pliku nieciągłego (z „dziurą”, której nie są przypisane żadne sektory danych).

Rozdział 9: konsola, wejście i wyjście

Zapis pojedynczych znaków na ekran (PUTC)

Zapis pojedynczych znaków na ekran (tj. do urządzenia CON: - przypominamy tu, że wyjście na CON: może zostać przez użytkownika przekierowane do dowolnego innego pliku) realizuje funkcja biblioteczna PUTC. Znak do zapisania należy jej przekazać w akumulatorze. Wywołanie nie zmienia zawartości rejestrów A, X i Y ani zmiennych FHANDLE ($0760) i DEVICE ($0761).

Zapis rekordu na ekran (PUTS)

Zapis rekordu tekstowego do urządzenia CON: realizuje funkcja biblioteczna PUTS. Adres ciągu znaków przekazujemy bibliotece w rejestrach AX, a w Y podajemy jego długość. Jeśli ciąg jest krótszy niż wartość Y w chwili wywołania, musi być zakończony znakiem EOL ($9B).

PUTS nie zmienia zawartości rejestrów A, X i Y ani zmiennych FHANDLE ($0760) i DEVICE ($0761).

Zapis formatowanego tekstu na ekran (PRINTF)

Funkcja PRINTF wysyła na konsolę teksty i ciągi danych przetworzone na postać tekstową i sformatowane według podanego wzorca. Osobliwością tej funkcji jest to, że wszystkie dane przekazuje się jej przez umieszczenie ich bezpośrednio za skokiem JSR PRINTF. Schemat wywołania jest następujący:

```
JSR PRINTF
```

.byte *wzorzec formatujący*,0

ewentualne dane

...

„Wzorzec formatujący” jest to ciąg tekstowy o długości do 253 znaków, *zakończony zerem*. W najprostszym przypadku to zwykły tekst do wyświetlenia, np.:

```
JSR PRINTF
.byte "Cukier krzepi",0
```

Wyświetli to po prostu podany tekst na ekranie zatrzymując kursor na jego końcu. Żeby to połączyć z przejściem do nowej linii, trzeba na końcu ciągu (przed zerem) dorzucić znak EOL ($9B).

Jednak cała siła funkcji PRINTF polega na tym, że w podanym ciągu znaków, który jest, przypominamy, tylko *wzorcem formatującym* tekst wyjściowy, można umieścić polecenia formatujące. Oto ich lista:

%% – wypisz pojedynczy znak %

%c – wypisz pojedynczy znak znajdujący się pod podanym adresem

%s – wypisz ciąg znaków o podanym adresie

%p – to samo, tylko zamiast adresu ciągu jest adres jego wskaźnika

%x – wartość spod podanego adresu wypisz jako 16-bit liczbę hex

%b – wartość spod podanego adresu wypisz jako 8-bitową liczbę dec

%d – to samo, tylko jako 16-bitową liczbę dec.

%e – to samo, wartość 24-bitowa

%l – to samo, wartość 32-bitowa (od SpartaDOS X 4.40)

Jednemu poleceniu formatującemu musi odpowiadać oddzielny wskaźnik umieszczony za wzorcem formatującym (na powyższym schemacie wskaźniki te zostały nazwane *ewentualnymi danymi*).

Użycie tego jest nader proste. Załóżmy, że etykieta *value* symbolizuje adres 16-bitowego słowa zapisanego w zwykłej konwencji młodszy/starszy. Chcemy przekształcić tę wartość na ciągi znaków reprezentujące tę liczbę w systemie szesnastkowym i dziesiętnym:

```
JSR PRINTF
.byte "VALUE = $%x = %d",$9B,0
.word value,value
```

Gdy pod adresem *value* mamy wartość $98AB, na ekranie pojawi nam się:

```
VALUE = $98AB = 39083
```

Napisaliśmy powyżej, że %x to polecenie sformatowania „16-bitowej" liczby szesnastkowej. Tak jest rzeczywiście, ale tylko o ile programista nie zażyczy sobie inaczej (czyli, %x *defaultowo* traktuje podane wartości jako 16-bitowe). W rzeczywistości biblioteka odczytuje wszystkie wartości numeryczne jako 32-bitowe (a w wersjach SpartaDOS starszych niż 4.40 – 24-bitowe). Gdy wartość jest mniejsza lub większa niż defaultowe 16 bitów, musimy ją podać, np.

```
JSR PRINTF
.byte "VALUE = $%2x",$9B,0
.word value
```

uwzględni tylko najmłodszy bajt wartości znajdującej się pod adresem *value* i wyprowadzi ją jako dwuznakową liczbę szesnastkową. Natomiast:

```
JSR PRINTF
.byte "VALUE = $%8x",$9B,0
.word value
```

spowoduje, że wypisana na ekranie liczba będzie miała wszystkie osiem cyfr. Liczba podana tu za znakiem % musi być liczbą dziesiętną z zakresu od 1 do 255 (w rzeczywistości może to być maksymalnie 65535, ale starszy bajt zostanie zignorowany).

We wszystkich podanych przykładach, gdy ciąg cyfr do wypisania zaczyna się zerami, zostaną one obcięte. Można jednak wymusić ich wyprowadzenie przez poprzedzenie zerem wartości oznaczającej oczekiwaną długość ciągu, np.

```
JSR PRINTF
.byte "VALUE = $%04x",$9B,0
.word value
```

W końcu, liczba ta (tj. długość ciągu, w ostatnim przykładzie „4”) nie musi być stałą, można skłonić bibliotekę do pobrania jej ze zmiennej, w ten sposób:

```
JSR PRINTF
.byte "VALUE = $%0*x",$9B,0
.word numlen,value
```

Etykieta *numlen* wskazuje miejsce w pamięci, skąd PRINTF powinna pobrać żądaną długość ciągu cyfr.

Podobnie działa formatowanie ciągów cyfr dziesiętnych: podając docelową liczbę cyfr w poleceniach %b, %d, %e i %l można obciąć wyprowadzany ciąg znaków do ich żądanej liczby.

W przypadku %c zawsze wyprowadzany jest jeden znak, wszystkie dodatkowe atrybuty komendy formatującej są ignorowane.

%s powoduje wyprowadzenie na konsolę ciągu znaków ASCII znajdującego się pod podanym adresem. Ciąg ten powinien być zakończony zerem. Po znaku % można podać liczbę znaków. Wtedy, gdy

ciąg jest dłuższy, zostanie obcięty, a gdy jest krótszy, zostanie dopełniony spacjami. %p działa identycznie jak %s z tym tylko, że zamiast adresu ciągu podajemy adres dwubajtowego wskaźnika zawierającego ten adres.

PRINTF nie zmienia zawartości rejestrów A, X i Y ani zmiennych FHANDLE ($0760) i DEVICE ($0761).

Odczyt pojedynczego znaku z konsoli (GETC)

Zadanie odczytu pojedynczego znaku z konsoli spełnia funkcja GETC. Odczytany bajt zwracany jest w akumulatorze. Gdy odczyt przebiegł pomyślnie, wynik jest dodatni (N=0), a do X załadowane jest $00. Gdy nastąpił koniec pliku, funkcja wraca z wynikiem ujemnym (N=1) i wartością $FF w rejestrze X. Każdy inny błąd powoduje automatyczne przekazanie sterowania do U_FAIL (patrz rozdział „Obsługa błędów”).

Gdy potrzebny jest odczyt bajtu nie z konsoli, lecz z klawiatury, należy się posłużyć funkcją U_GETKEY. Czeka ona na naciśnięcie klawisza, po czym zwraca jego kod ASCII w akumulatorze. Pamiętać jednak przy tym trzeba, że istnieje ona dopiero w SpartaDOS X od wersji 4.40 wzwyż.

Odczyt rekordu z konsoli (GETS)

Odczyt rekordu z konsoli przeprowadza funkcja GETS. Rekord jest to ciąg znaków ASCII zakończony znakiem EOL ($9B). Podobnie jak dla PUTS, parametry przekazuje się w rejestrach: w AX adres bufora, w Y długość. Rekordy nie mogą być dłuższe niż 255 bajtów.

Gdy odczyt przebiegnie poprawnie, funkcja wraca z wynikiem dodatnim (N=0), zerem w akumulatorze i liczbą odczytanych bajtów w Y. Gdy wystąpi nadmiar danych, wynik jest ujemny (N=1), a w

akumulatorze jest $FF. Sytuacja ta odpowiada wystąpieniu błędu nr 137 (Truncated record) w XL OS. Każdy inny błąd powoduje automatyczne przekazanie sterowania do U_FAIL (patrz rozdział „Obsługa błędów”).

Funkcje PUTC, PUTS, PRINTF, GETC i GETS realizujące odczyt i zapis danych dla konsoli to tylko oddzielne wejścia (tzw. wrappery) do opisanych w poprzednim rozdziale, ogólniejszych funkcji zapisu i odczytu plików FPUTC, FPUTS, FPRINTF, FGETC i FGETS. Użycie tych pierwszych zamiast tych drugich pozwala bezboleśnie przerzucić na bibliotekę systemową zadanie obsługi przekierowań we/wy.

Przekierowania we/wy (DIVIO/XDIVIO)

Wejście i wyjście z konsoli może zostać przekierowane do dowolnego innego pliku przy użyciu funkcji DIVIO. Posługuje się nią interpreter poleceń SpartaDOS w sytuacji, gdy użytkownik wyda komendę np. `DIR >>plik`.

DIVIO wymaga ustawienia wektora FILE_P na specyfikację pliku, do którego (lub z którego) ma zostać ustanowione przekierowanie z konsoli (lub na nią). W rejestrze Y podajemy $00, gdy przekierowywane jest wyjście (tj. funkcje PUTC itp.), a $01, gdy przekierowane ma być wejście (GETC). Ewentualne błędy przy otwarciu przekierowania zgłaszane są do U_FAIL. W przypadku powodzenia uchwyt pliku, gdzie są przekierowywane dane, znajdzie się w FHANDLE ($0760).

Zadanie odwrotne, to jest zakończenie przekierowania, realizuje funkcja XDIVIO. Jako parametr podajemy $00 lub $01 w rejestrze Y, tak samo, jak napisano powyżej. Uchwyt pliku z przekierowaniem powinien być w FHANDLE ($0760). Wywołanie XDIVIO powoduje zamknięcie tego pliku.

Gdy wejście konsoli zostało przekierowane i nastąpi koniec pliku, biblioteka wywołuje XDIVIO automatycznie. Podobnie dzieje się przy

zamykaniu wszystkich plików przez FCLOSEAL oraz po wystąpieniu błędu.

Wektorowane wyjście (PUT_V/VPRINTF)

Inną metodą przejęcia, tym razem tylko wyjścia na konsolę, jest skorzystanie z wektora PUT_V. Normalnie nie ma on żadnej sensownej wartości, program musi ustawić w nim adres procedury obsługującej zapis. Może to być np. procedura dokonująca bezpośrednich zapisów do pamięci ekranu, z pominięciem urządzenia CON: czy E:. Powinna się ona kończyć przez RTS, wskazane jest przechowywanie wartości rejestrów.

Gdy uchwyt pliku w FHANDLE ($0760) ma wartość 100 ($64), wywołanie FPUTC powoduje skok pośredni (JMP) przez PUT_V z daną do wyświetlenia znajdującą się w akumulatorze.

Ponieważ FPUTC jest niejawnie wywoływana przez FPUTS i FPRINTF, więc wyjście z tych funkcji zostanie w ten sposób również przekierowane.

Zamiast FPRINTF wygodniej jest w tym celu użyć VPRINTF – jest to kolejne (trzecie już) wejście do tej samej procedury, i jej działanie jest identyczne, jak PRINTF, z tą tylko różnicą, że aktywny kanał I/O zostaje automatycznie przełączony na uchwyt 100 przed wyprowadzeniem tekstu, i równie automatycznie przełączony z powrotem po nim. Programista nie musi się więc troszczyć o zawartość FHANDLE ($0760) w tym przypadku.

Rozdział 10: katalogi

Wyszukiwanie plików (FFIRST/FNEXT)

Do wyszukiwania plików we wskazanych katalogach służą dwie funkcje: FFIRST i FNEXT. Parametrem wejściowym tej pierwszej jest kompletna specyfikacja katalogu razem z podaną na końcu maską plików wskazywana przez wektor FILE_P, oraz poszukiwane atrybuty w FATR1 ($0779). Znaleziony wpis jest umieszczany w DIRBUF ($0789-$79F) w postaci „surowego" wpisu katalogowego w formacie SpartaDOS (tj. w takiej, jaka jest zapisana na dysku, patrz „SpartaDOS X. Podręcznik użytkownika", rozdział 7).

FNEXT nie wymaga żadnych dodatkowych parametrów, wyszukuje po prostu następny wpis według kryteriów zadanych poprzednio funkcji FFIRST.

Koniec katalogu sygnalizowany jest w obu funkcjach przez powrót z wynikiem ujemnym (N=1). Każdy inny błąd powoduje skok do U_FAIL.

UWAGA: FFIRST w rzeczywistości otwiera wskazany katalog do odczytu, a uchwyt pliku umieszcza w FHANDLE ($0760). Dlatego po zakończeniu przeszukiwania należy ***koniecznie*** *wywołać FCLOSE dla tego uchwytu w celu zamknięcia pliku katalogu.*

Odczyt katalogu (FDOPEN/FDGETC/FDCLOSE)

Biblioteka zawiera trzy funkcje udostępniające katalog w postaci sformatowanej, czyli czytelnej dla człowieka (np. takiej, jaka pojawia się na ekranie po podaniu interpreterowi poleceń komendy DIR). Są to kolejno: FDOPEN, FDGETC i FDCLOSE. FDOPEN otwiera strumień danych katalogu, FDGETC pobiera z niego bajt zwracając go w akumulatorze (i wykazując wynik dodatni w rejestrze znaczników

procesora), a FDCLOSE zamyka katalog otwarty uprzednio przez FDOPEN. Błędy FDGETC zgłasza do U_FAIL, za wyjątkiem statusu końca pliku – gdy takowy wystąpi, funkcja wraca do programu wywołującego z wynikiem ujemnym (N=1).

Każda z nich robi w zasadzie tylko jedno: wywołuje odpowiednią funkcję wejścia misc_, tj. kolejno nr 6 (misc_fdopen), 7 (misc_fdgetc) i 8 (misc_fdclose). Odczyt formatowanego katalogu można więc realizować tą drogą (tj. przez wektor misc_), gdy biblioteka systemowa jest niedostępna na skutek uruchomienia programu komendą X. Trzeba tylko pamiętać, że misc_ nigdy nie oddaje sterowania do U_FAIL, zwracając po prostu kod błędu w akumulatorze zamiast tego, a dodatkowo bajt odczytany przez misc_fdgetc nie jest przekazywany w rejestrach, lecz na stronie zerowej w ICAX6Z (adres $2F).

Parametry otwarcia dla FDOPEN to specyfikacja katalogu wskazywana przez FILE_P oraz żądany sposób formatowania w OPMODE ($0778). Wersje SpartaDOS X starsze od 4.41 znają tylko dwa sposoby formatowania: format „długi” SpartaDOS (OPMODE=$80) oraz format „krótki” AtariDOS (OPMODE=$00).

W SpartaDOS od wersji 4.41 wzwyż OPMODE jest traktowane przez FDOPEN jako maska bitów wybierających (oddzielnie) sposoby formatowania poszczególnych elementów katalogu. Znaczenie bitów:

+$80 – długi format katalogu

+$40 – wyświetlanie atrybutów

+$20 – wstawianie odstępu między nazwą a rozszerzeniem

+$10 – kropka po nazwie (gdy bit 5=1)

+$08 – dwie spacje przed nazwą, ‘*’ dla pliku zabezpieczonego

+$04 – czas w formacie 24-godzinnym

+$02 – gdy długi format katalogu (bit 7=1), czas bez sekund; w krótkim formacie będą wyświetlane rozszerzenia nazw katalogów

(zamiast standardowego DIR), katalog będzie oznaczany dwukropkiem przed nazwą.

+$01 – wyświetlanie wielkości pliku w sektorach (gdy bit 7=0).

Dla utrzymania zgodności ze starszymi wersjami SpartaDOS X, podane przez użytkownika $00 przekładane jest na $0B, a $80 na $A2.

UWAGA: misc_fdopen, a co za tym idzie również funkcja biblioteki FDOPEN, dokonuje niejawnego otwarcia pliku katalogu do odczytu. W związku z tym (a) nie należy zaniedbywać wywołania FDCLOSE (lub misc_fdclose), gdy odczyt się zakończy, a ponadto (b) ***można otworzyć tylko jeden taki plik na raz****, bo misc_ nie jest w stanie zapamiętać większej liczby uchwytów.*

Rozdział 11: ładowanie programów binarnych

Załadowanie programu do pamięci (U_LOAD)

Symbol U_LOAD wskazuje loader binarny SpartaDOS. Wypełnia on trzy zadania: (a) wczytywanie programów binarnych do pamięci wraz z uruchomieniem, (b) wczytywanie bez uruchomienia oraz (c) uruchamianie ich.

Parametry wejściowe U_LOAD to specyfikacja pliku wskazywana wektorem FILE_P oraz wartość sterująca w rejestrze FLAG. Wskazany plik jest otwierany do odczytu w trybie z przeszukiwaniem $PATH (OPMODE=$24) i, gdy jest to poprawny plik binarny, ładowany jest do pamięci. Ewentualne błędy powodują przerwanie tej czynności i skok do U_FAIL.

Rejestr FLAG decyduje, co z załadowanym programem zrobić. Gdy bit 7 FLAG jest równy zero, program jest natychmiast uruchamiany. W przeciwnym wypadku aktualizacji ulegają tylko wskaźniki wolnej pamięci (dla programów w formacie relokowalnym SpartaDOS), a sterowanie wraca do programu wywołującego. Uruchomienie (już bez ponownego ładowania) następuje po ponownym wywołaniu U_LOAD z tą samą specyfikacją pliku i skasowanym bitem 7 rejestru FLAG.

Usunięcie programu z pamięci (U_UNLOAD)

Gdy program był normalnie uruchomiony, jego usunięcie z pamięci następuje po tym, jak odda sterowanie do interpretera poleceń SpartaDOS. W przeciwnym wypadku (załadowanie z FLAG > $7F) w celu usunięcia kodu z pamięci trzeba wywołać funkcję U_UNLOAD. Usuwa ona wszystko, co program użytkownika poprzednio załadował przez U_LOAD.

Zmiana nazwy pliku (RENAME)

Zmianę nazwy wskazanego pliku przeprowadza funkcja RENAME. Parametrem wejściowym jest kompletna specyfikacja nazwy pliku, która ma zostać zmieniona. Specyfikację tę powinien wskazywać wektor FILE_P. Po specyfikacji źródłowej powinna następować, oddzielona spacją lub przecinkiem, nazwa, jaka ma być nadana plikowi. Ewentualne błędy zgłaszane są do U_FAIL.

W starszych wersjach SpartaDOS X można było, przy użyciu RENAME, kilku plikom znajdującym się w jednym katalogu nadać tę samą nazwę. Od wersji 4.40 jest to niemożliwe.

Usunięcie pliku (REMOVE)

Skasowanie pliku wykonuje funkcja REMOVE. Parametrem wejściowym jest specyfikacja pliku do usunięcia wskazywana przez FILE_P. Wystąpienie błędu powoduje skok do U_FAIL.

Usunięcie katalogu (RMDIR)

Skasowanie katalogu realizuje funkcja RMDIR. Parametrem wejściowym jest specyfikacja katalogu (bez końcowego separatora) wskazywana przez FILE_P. Wystąpienie błędu powoduje skok do U_FAIL.

Katalog musi być pusty. Usunięcie katalogu razem z zawartymi w nim plikami i innymi katalogami leży poza możliwościami biblioteki. Gdy to konieczne, należy wywołać zawarty w module ROM program DELTREE.COM z nazwą katalogu do skasowania jako parametrem.

Utworzenie katalogu (MKDIR)

Nowy katalog tworzy funkcja MKDIR. Parametrem wejściowym jest specyfikacja katalogu (bez końcowego separatora) wskazywana przez FILE_P. Wystąpienie błędu powoduje skok do U_FAIL.

Zmiana atrybutów (CHMOD)

Zmianę atrybutów pliku wykonuje funkcja CHMOD. Wymaganymi parametrami są: specyfikacja pliku wskazywana przez FILE_P, maska poszukiwanych atrybutów w FATR1 ($0779) oraz maska nowych atrybutów w FATR2 ($077A).

Maska poszukiwanych atrybutów wybiera pliki, w których atrybuty mają zostać zmienione. Działa ona tak samo, jak w pozostałych funkcjach ją uwzględniających (np. FOPEN):

+$01: zabezpieczony (+P)
+$02: ukryty (+H)
+$04: archiwalny (+A)
+$08: katalog (+S)
+$10: nie zabezpieczony (-P)
+$20: nie ukryty (-H)
+$40: nie archiwalny (-A)
+$80: nie katalog (-S)

W masce atrybutów ustawianych poszczególne bity mają następujące znaczenie (gdy ustawione na 1):

+$01: zabezpieczenie (+P)

+$10: odbezpieczenie (-P)

+$02: ukrycie (+H)

+$20: ujawnienie (-H)

+$04: archiwalny (+A)

+$40: niearchiwalny (-A)

Atrybutu S (katalog), naturalnie, nie można w ten sposób zmienić. Wystąpienie błędu oddaje sterowanie do U_FAIL.

Wybranie pliku BOOT (SETBOOT)

SETBOOT ustawia wskazany (przez FILE_P) plik binarny jako ten, który przy starcie systemu ma zostać automatycznie załadowany przez loader znajdujący się na początku dysku. Funkcja ta działa oczywiście tylko na dyskach w formacie SpartaDOS. Ewentualny błąd powoduje przekazanie sterowania do U_FAIL.

Rozdział 13: inne funkcje dyskowe

Formatowanie dysku (FORMAT)

Funkcja FORMAT nie robi nic innego poza wywołaniem na ekran menu *SpartaDOS Formatter*. Program wywołujący nie przekazuje jej żadnych parametrów.

W starszych wersjach SpartaDOS X wywołanie formattera tą drogą wymagało uprzedniego „ręcznego" (przez ingerencję w rejestry przełączające banki modułu) przełączenia cartridge'a na bank 0. Od wersji 4.40 DOS-u ta niedogodność została usunięta, system sam wybiera odpowiedni bank, a i formatter nie rezyduje już w banku 0. Niemniej, jeśli zależy nam na kompatybilności, lepiej jest użyć XIO 254.

Zapis świeżego katalogu (BUILDDIR)

BUILDDIR wykonuje „miękkie" formatowanie, to jest po prostu zapisuje od nowa pusty katalog główny, mapę bitową dysku i bootsektor w formacie SpartaDOS. Jest to jedyna metoda formatowania ramdysków i partycji twardego dysku. Parametrami wejściowymi są:

1) kod urządzenia w DDEVIC ($0300). Dla dysku $31.

2) numer urządzenia w DUNIT ($0301).

3) wskaźnik do nowej etykiety dysku w AX.

Etykieta to osiem znaków ASCII. Jeśli ma być krótsza, trzeba uzupełnić ją spacjami do tej długości. Etykieta *nie może* zaczynać się od znaku spacji.

Funkcja automatycznie odczytuje konfigurację napędu i na tej podstawie ustala resztę parametrów (liczbę sektorów itp.). W SpartaDOS X 4.40 samoczynnie włączana jest optymalizacja mapy bitowej (funkcja *Optimize* formattera). Status operacji zwracany jest w rejestrze Y.

Odczytanie parametrów dysku (GETDFREE)

GETDFREE odczytuje różne informacje o dysku i umieszcza je w PATH ($07A0-$07DF). Przed wywołaniem trzeba podać specyfikację urządzenia wskazawszy ją wektorem FILE_P. Ewentualne błędy są zgłaszane do U_FAIL.

Dane (17 bajtów) są umieszczane w PATH jak następuje:

+$00: kod identyfikacyjny filesystemu

+$01: zakodowana liczba bajtów w sektorze

+$02: całkowita liczba sektorów (2 bajty)

+$04: aktualna liczba wolnych sektorów (2 bajty)

+$06: nazwa dysku (8 bajtów)

+$0E: numer sekwencyjny

+$0F: numer losowy

+$10: bajt bez znaczenia

Kod identyfikacyjny filesystemu pozwala sprawdzić, jaki system plików znajduje się na danym dysku, według następującego schematu:

$0x – filesystem AtariDOS v. x.0 (np. $02 = DOS 2.0)

$11 – filesystem dyskowy SpartaDOS v. 1.1

$2x – filesystem dyskowy SpartaDOS v. 2.x

$3C – PC-MIRROR

$40 – filesystem modułu SpartaDOS (CAR:)

$FF – filesystem dyskowy MyDOS

UWAGA: GETDFREE wywołuje funkcję kernela nr $13. Od wersji 4.41 SpartaDOS format danych zwracanych przez kernel na to wywołanie różni się kompletnie od tego, co zwraca GETDFREE. W SpartaDOS 4.2x tak nie było – funkcja kernela $13 w 4.4x jest niezgodna wstecz z 4.2x. Dla zachowania kompatybilności programy powinny się

do niej odwoływać ***wyłącznie*** *przez bibliotekę (tj. przez GETDFREE), a przy braku takiej możliwości, przez funkcję XIO 47 systemu operacyjnego. Obie dokonują konwersji danych zwróconych przez kernel do „starego" formatu, zgodnego z 4.2x*

Zmiana katalogu bieżącego (CHDIR)

CHDIR zmienia katalog bieżący dysku na ten, którego specyfikację wskazuje wektor FILE_P. Błąd powoduje przeskok do U_FAIL.

Odczyt katalogu bieżącego (GETCWD)

GETCWD odczytuje ścieżkę od katalogu głównego dysku do katalogu bieżącego i umieszcza ją w PATH ($07A0-$07DF). Specyfikację dysku (w postaci tekstowej) należy wskazać wektorem FILE_P. Błąd skutkuje wywołaniem U_FAIL.

W starszych wersjach SpartaDOS X funkcja GETCWD nie zwracała rozszerzeń nazw katalogów. Od wersji 4.40 zostało to poprawione.

Rozdział 14: funkcje pomocnicze

32-bitowe mnożenie i dzielenie (MUL_32/DIV_32)

Biblioteka zawiera dwie procedury obliczeń na liczbach całkowitych bez znaku: MUL_32 wykonuje mnożenie dwóch liczb 32-bitowych, a DIV_32, odpowiednio, dzielenie takowych.

Składniki operacji muszą zostać wpisane *w kolejności bajtów odwrotnej od normalnej (najstarszy bajt najpierw!)* jak następuje:

1) mnożna (lub dzielna) pod COMTAB+$FF (4 bajty)

2) mnożnik (lub dzielnik) pod COMTAB+$0103 (4 bajty)

Wynik obliczenia, zapisany tak samo w odwrotnej kolejności bajtów, odbieramy spod COMTAB+$0107. Obie procedury sygnalizują przepełnienie przez ustawienie bitu C rejestru znaczników (C=1).

Zamiana małych liter na duże (TOUPPER)

Wywołanie TOUPPER z kodem ASCII małej litery w akumulatorze zamienia go na kod litery dużej. Jeśli przekazany kod nie oznacza małej litery, nie jest robione nic zgoła.

Sprawdzenie separatora katalogu (CKSPEC)

Procedura CKSPEC sprawdza, czy znak przekazany w akumulatorze jest jednym z następujących: ‘:’, ‘>’, ‘\’, ‘<’ (są to separatory mogące wystąpić w specyfikacji pliku), a gdy tak jest, wraca z wynikiem zerowym (Z=1). W przeciwnym wypadku wynik jest niezerowy (Z=0).

Rozdział 15: procedury inicjowania

Inicjowanie nakładek po RESET (S_ADDIZ)

Część nakładek musi zostać zainicjowana po każdorazowym naciśnięciu klawisza RESET. Będą to np. wszelkiego rodzaju sterowniki instalujące się w CIO, tak jak zawarte w SpartaDOS X 4.41 CON64.SYS i CON80.SYS. Po zresetowaniu systemu potrzebują one choćby ponownie zainstalować się w tablicy handlerów OS-u.

Do rejestrowania takich nakładek służy funkcja S_ADDIZ biblioteki. Należy jej przekazać w rejestrach AX adres procedury inicjowania nakładki. DOS wpisuje ten adres do specjalnej kolejki, a po RESET wykonuje po kolei zarejestrowane tam podprogramy.

Kolejka ma tylko pięć miejsc, brak możliwości rejestracji sygnalizowany jest przez powrót z S_ADDIZ z ustawionym znacznikiem C rejestru znaczników (C=1).

Wyjście do DOS-u (_DOS)

Wywołanie JMP _DOS ma takie samo działanie, jak JMP (DOSVEC) – program wywołujący zostaje zakończony i usunięty z pamięci, a sterowanie wraca do interpretera poleceń. Lepszą metoda zakończenia programu jest wykonanie rozkazu RTS.

Ciepły restart SpartaDOS (_INITZ)

Wywołanie _INITZ powoduje ciepły restart SpartaDOS. Wszystkie pliki zostają zamknięte, ustawienia zresetowane, rezydujące sterowniki kernela zainicjowane od nowa.

_INITZ jest jedną z procedur wywoływanych po wciśnięciu klawisza

RESET przez użytkownika. W starszych wersjach SpartaDOS X wykonanie _INITZ powodowało ustawienie bieżących ścieżek wszystkich dysków na katalogi główne. Od wersji 4.40 sterownik SPARTA.SYS odróżnia wywołanie inicjowania ze środka _INITZ od każdego innego i ścieżki nie są resetowane.

Rozdział 16: manipulowanie listą symboli

Przeszukiwanie listy symboli (S_LOOKUP)

System zawiera procedury przeszukiwania listy symboli, oraz dodawania i usuwania symboli. Normalnie jednak programy nie korzystają z nich bezpośrednio, bo wszystkimi sprawami związanymi z symbolami zajmuje się loader binarny SpartaDOS. Wykaz symboli dołączony do bloków binarnych programu jest, po załadowaniu kodu do pamięci, automatycznie tłumaczony na adresy, a te są wstawiane w odpowiednie miejsca programu. Podobnie wygląda sprawa definicji nowego symbolu przez załadowany program: definicja ta zakodowana jest w strukturze samego pliku binarnego, a stworzeniem nowego symbolu i dołączeniem go do globalnej listy zajmuje się loader.

Czasami jednak zachodzi konieczność „ręcznego" przeszukania listy symboli przez program. Na przykład znajdujący się w ROM-dysku CAR: sterownik Z.SYS potrzebuje dostępu do symbolu I_FMTTD definiowanego przez TD.COM, ale nie może być uzależniony od jego obecności, bo nie byłoby wtedy możliwe załadowanie Z.SYS bez uprzedniego wczytania TD.COM (niemożność „zresolwowania" symbolu ujętego na dołączonym do programu wykazie powoduje przerwanie ładowania i komunikat „Loader: Symbol not defined").

W takim układzie trzeba się posłużyć procedurą S_LOOKUP. Wymaga ona wpisania nazwy symbolu, uzupełnionej do ośmiu znaków spacjami, jeśli trzeba, pod SYMBOL+$02. Wywołanie JSR S_LOOKUP zwraca zero (Z=1), gdy poszukiwany symbol nie istnieje. W przeciwnym wypadku wynik jest różny od zera (Z=0), a pod SYMBOL+$0B i SYMBOL+$0C znajduje się kolejno młodszy i starszy bajt adresu wskazywanego przez symbol, natomiast w EXTENDED mamy indeks pamięci. Bajt EXTENDED dobrze jest przedtem wyzerować.

Jako że sama procedura S_LOOKUP, struktura SYMBOL oraz zmienna EXTENDED wskazywane są symbolami, jasne jest, że ta droga dostępu do listy symboli stoi otworem tylko dla programów zapisanych w formacie binarnym SpartaDOS. Zresztą zwykłe binaria Atari DOS-u zwykle nie potrzebują dostępu do listy symboli. Jednak gdyby zaszła taka potrzeba, istnieje drugie wejście do S_LOOKUP, nie dość, że znajdujące się pod stałym adresem, to jeszcze dużo prostsze w użyciu.

Procedurę tę, nazywającą się FSYMBOL, wprowadzono w SpartaDOS v. 4.40. Wejście do niej znajduje się pod adresem $07EB. Daną wejściową jest adres uzupełnionej spacjami nazwy symbolu przekazany w rejestrach AX (mł./st.). W razie nieznalezienia symbolu procedura zwraca zero (Z=1), w przeciwnym wypadku rejestry AX będą zawierać wskazywany adres, a rejestr Y – indeks pamięci.

UWAGA: w chwili uruchomienia programu komendą X lista symboli oraz opisywane tu procedury stają się NIEDOSTĘPNE.

Dodanie symbolu (S_ADD)

Po wywołaniu S_ADD symbol o wskazanych parametrach zostanie dopisany do globalnej listy symboli. Dane wejściowe przekazujemy w strukturze SYMBOL, wygląda ona, dla przypomnienia, następująco:

+$00-$01: wskaźnik do następnego symbolu (2 bajty)
+$02-$09: nazwa symbolu (8 znaków ASCII)
+$0A: bajt kontrolny
+$0B-$0C: adres wskazywany przez symbol (2 bajty)

S_ADD wymaga od programu wywołującego wypełnienia nazwy (z uzupełnieniem spacjami do ośmiu znaków, jeśli jest krótsza) oraz ustawienia adresu w bajtach SYMBOL+$0B i SYMBOL+$0C. Indeks

pamięci należy zapisać w EXTENDED, o ile się tam już nie znajduje. Bajt kontrolny wypełniany jest automatycznie, tak samo wskaźnik do następnego symbolu na liście.

Symbol dodawany jest w miejsce wskazywane przez MEMLO, wskaźnik ten jest potem zwiększany o 13 bajtów.

Usuwanie symboli (S_CLEAR)

Procedura S_CLEAR usuwa z globalnej listy symboli symbole, których PID jest większy niż bieżąco ustawiony. Jest to w zasadzie część procedury U_UNLOAD, a „bieżąco ustawiony” numer aplikacji nie może zostać jawnie zmieniony przez program użytkownika. Wobec tego cel wyeksportowania S_CLEAR do globalnej listy symboli nie jest całkiem jasny.

Rozdział 17: pozostałe symbole biblioteki

Poza opisanymi powyżej na liście symboli istnieje jeszcze garść dotąd nieopisanych, a niektóre nie zostały nawet wspomniane. Są to zmienne: STATUS, SYSLEVEL, H_FENCE, tablice CARVARS, DEVSPEC, DEVNAME, COMTAB2, procedury _CIO, _EDIT, _CRUNCH i XDFREE. Większość z nich ma zastosowanie tylko dla specjalnych sterowników systemowych, opis ich funkcji i użycia znajdzie się w przyszłości w suplemencie do niniejszego opracowania poświęconym pisaniu sterowników.

Symbol XDFREE wskazuje rozbudowaną procedurę zastępującą GETDFREE, jednak ponieważ została ona wprowadzona do SpartaDOS X 4.41 tak świeżo, że w zasadzie można ją nazwać eksperymentalną, jej opis zostanie na razie pominięty.

Indeks procedur i zmiennych systemowych