ROZDZIAŁ TRZYNASTY: **MS-DOS, PC-BIOS i I/O PLIKÓW**

Typowy system PC składa się z wielu innych składników poza CPU 80x86 i pamięci. MS-DOS i BIOS PC dostarczają połączenia pomiędzy naszą aplikacją a wykorzystywanym sprzętem. Chociaż czasami jest konieczność oprogramować sprzęt bezpośrednio, najczęściej lepiej jest pozostawić to oprogramowaniu systemowemu (MS-DOS i BIOS) wykonującemu to za nas. Co więcej dużo łatwiej jest nam wywołać po prostu podprogram wbudowany w nasz system niż pisać taki podprogram samemu.

Możemy uzyskać dostęp do sprzętu IBM PC na jednym z trzech ogólnych poziomów z języka asemblera. Możemy oprogramować sprzęt bezpośrednio, możemy użyć podprogramów ROM BIOS do uzyskania dostępu do sprzętu lub możemy uczynić wywołanie MS-DOS aby uzyskać dostęp do sprzętu. Każdy z poziomów systemu ma swój własny zbiór zalet i wad.

Oprogramowanie bezpośrednie sprzętu oferuje dwie zalety w stosunku do pozostałych metod.: sterowanie i wydajność. Jeśli będziemy sterowali trybami sprzętu możemy uzyskać poprawę wydajności systemu przez wykorzystanie specjalnych sztuczek sprzętowych lub innych takich, których nie potrafią podprogramy ogólnego przeznaczenia. Dla niektórych programów, takich jak edytory ekranowe (które muszą mieć szybki dostęp do karty video) bezpośredni dostęp sprzętowy jest jedynym sposobem osiągnięcia rozsądnej poprawienia poziomów.

Z drugiej strony, bezpośrednie programowanie sprzętu ma również swoje wady. Edytory ekranowe, mające bezpośredni dostęp do pamięci video, mogą nie działać jeśli pojawi się nowy typ kart video w IBM PC. Dla takich programów mogą być konieczne złożone sterowniki, zwiększające ilość pracy przy tworzeniu i pielęgnacji takiego programu. Co więcej, mając napisanych kilka programów, uzyskujących dostęp do pamięci ekranowej bezpośrednio i zakładając, że IBM wyprodukował nowe, niekompatybilne rozszerzenie, będziemy musieli przepisać wszystkie nasze programy tak, aby działały z nowymi kartami video.

Nasza praca stałaby się znacznie łatwiejsza gdyby IBM dostarczył, w stałej, znanej lokacji jakiś podprogramy, które wykonywałby wszystkie operacje ekranowe I/O za nas. Nasze wszystkie programy mogłyby wywoływać te podprogramy. Kiedy producent wprowadziłby na rynek nowe rozszerzenie do kart, dostarczałby nowego zbioru podprogramów dla wyświetlacza z kartą rozszerzenia. Te nowe podprogramy zastąpią stare lub je uaktualnią, tak aby wywołanie starych podprogramów wywoływało tez podprogramy nowsze. Jeśli interfejs programu jest taki sam pomiędzy dwoma zbiorami podprogramów, nasze programy będą działać z nowszymi podprogramami.

IBM ma zaimplementowany taki mechanizm z oprogramowaniu systemowym. W najwyższym obszarze jedno megabajtowej przestrzeni w PC znajdują się specjalistyczne adresy dla przechowywania danych ROM. Chip pamięci ROM zawiera specjalne oprogramowanie Basic Input Output System (Podstawowy System Wejścia – Wyjścia) lub BIOS. Podprogramy BIOS dostarczają niezależnego od sprzętu interfejsu dla różnych urządzeń w systemie IBM PC. Na przykład jedną z usług BIOS jest sterowanie wyświetlaniem .Poprzez różne wywołania podprogramu video BIOS, nasz program będzie mógł wypisać znaki na ekranie bez względu na rzeczywistą , zainstalowaną kartę graficzną.

Jednym z wyższych poziomów jest MS-DOS. Podczas gdy BIOS pozwala nam manipulować urządzeniami na bardzo niskim poziomie, MS-DOS dostarcza wysokopoziomowego interfejsu dla wielu urządzeń. Na przykład jeden z podprogramów BIOS pozwala nam uzyskać dostęp do dyskietki. Dzięki temu podprogramowi BIOS'a możemy odczytać lub zapisać bloki na dyskietce. Niestety BIOS nie wie nic o takich rzeczach jak pliki czy katalogi. O wie tylko o blokach. Jeśli chcesz uzyskać dostęp do pliku na dyskietce

stosując wywołanie BIOS'a musisz wiedzieć dokładnie gdzie ten plik pojawia się na powierzchni dyskietki. Z drugiej strony wywołanie MS-DOS pozwala nam działać na nazwie pliku zamiast na adresie dyskowym pliku. MS-DOS śledzi gdzie na powierzchni dyskietki są pliki i wywołując ROM BIOS odczytuje właściwy blok dla nas. Ten wysokopoziomowy interfejs znacznie redukuje ilość wysiłku jaki nasz program musi wydatkować na uzyskanie dostępu do danych na dyskietce.

Celem tego rozdziału jest dostarczenie krótkiego wprowadzenia do różnych usług BIOS'a i DOS'a dostępnych dla nas. Rozdział ten nie próbuje opisywać wszystkich podprogramów lub opcji dostępnych przy każdym podprogramie. Jest kilka tekstów większych od tego, które próbują opisać tylko BIOS lub tylko DOS. Zatem każda próba dostarczenia opisu MS-DOS lub BIOS w pojedynczym tekście jest skazana na porażkę już na starcie – oba są ruchomymi celami, zmieniającymi specyfikację przy każdej nowej wersji. Więc zamiast wyjaśniać wszystko, rozdział ten po prostu będzie próbował prezentować smaczki.

13.0 WSTĘP

Rozdział ten przedstawia materiał, który jest specyficzny dla PC. Informacje te o BIOS'ie i DOS'ie nie są konieczne jeśli chcesz nauczyć się programowania w języku assemblera; jednakże informacje te są ważne dla każdego chcącego pisać programy assemblerowe ,które działają pod MS-DOS na kompatybilnych maszynach z PC. W wyniku tego, większość informacji w tym rozdziale jest opcjonalna dla tych , którzy chcą nauczyć się ogólnego programowania w assemblerze. Z drugiej strony, informacje te informacje są przydatne dla tego kto chce pisać aplikacje w assemblerze na PC.

Te części ,które mają prefiks „•" są niezbędne. Te części z „⊗" omawiają zaawansowane tematy, które możemy odłożyć na później.

- BIOS IBM PC
- ⊗ Zrzut ekranu
- Usługi video
- ⊗ Instalowanie sprzętu
- ⊗ Dostępność pamięci
- ⊗ Usługi nisko poziomowe
- Złącze szeregowe We/Wy
- ⊗ Usługi różnorodne
- Usługi klawiatury
- Usługi drukowania
- ⊗ Działanie BASIC
- ⊗ Ponowne uruchomienie komputera
- ⊗ Zegar czasu rzeczywistego
- Sekwencja wywołania MS-DOS
- Funkcje znakowe MS-DOS
- ⊗ Polecenia napędu MS-DOS
- ⊗ Funkcje czasu i daty MS-DOS
- ⊗ Funkcje zarządzania pamięcią MS-DOS
- ⊗ Funkcje sterowania procesem MS-DOS
- „Nowe" segregowanie wywołań MS-DOS
- Otwieranie pliku
- Tworzenie pliku
- Zamykanie pliku
- Czytanie z pliku
- Zapis do pliku
- ⊗ Przeszukiwanie
- ⊗ Ustawienie adresu przekazania dysku
- ⊗ Znajdowanie pierwszego pliku
- ⊗ Znajdowanie kolejnego pliku
- Usuwanie pliku
- Zmiana nazwy pliku
- ⊗ Zmiana / pobranie atrybutu pliku
- ⊗ Pobranie / ustawienie daty i czasu pliku
- ⊗ Inne wywołania DOS

- Przykłady plików I/O
- Zblokowane pliki I/O

⊗ Program Segment Prefix
⊗ Dostęp do parametrów linii poleceń
⊗ ARGC i ARGV

- Pliki podprogramów I/O Standardowej Biblioteki UCR
- FOPEN
- FCREATE
- FCLOSE
- FFLUSH
- FGETC
- FREAD
- FPUTC
- FWRITE

⊗ Przekierowanie na porty I/O przez podprogramy plików I/O STDLIB

13.1 BIOS IBM PC

Zamiast umieścić podprogramy BIOS'u w stałych komórkach pamięci w ROM, IBM użył dużo bardziej elastycznego podejścia przy projektowaniu BIOS'u. Dla wywołania podprogramu BIOS'a, używamy jednej z instrukcji przerwania programowego int 80x86. Instrukcja int używa następującej składni:

int wartość

Wartość jest jakąś liczbą z zakresu 0..255. Wykonanie instrukcji int będzie powodowała w 80x86 przekazanie sterowania do jednej z 256 różnych procedur obsługi przerwań. Tablica wektorów przerwań zaczyna się w fizycznej komórce pamięci pod adresem 0:0, przechowując adresy tych procedur obsługi przerwań. Każdy adres jest pełnym adresem segmentowym, wymagającym czterech bajtów więc jest 400h bajtów w tablicy wektorów przerwań – jeden adres segmentowy dla każdego z 256 możliwych przerwań programowych. Na przykład, int 0 przekazuje sterowane do podprogramu którego adres znajduje się w komórce 0:0, int 1 przekazuje sterowanie do podprogramu, którego adres jest pod 0:4, int 2 pod 0:8, int 3 pod 0:C a int 4 pod 0:10.

Kiedy resetujemy PC jedną z pierwszych czynności jest zainicjalizowanie kilku z tych wektorów przerwań aby wskazały podprogramy usług BIOS. Później, kiedy wykonujemy odpowiednią instrukcje int, sterownie jest przekazane do właściwego kodu BIOS.

Jeśli tylko wywołujemy podprogramy BIOS (w przeciwieństwie do ich napisania) możemy zobaczyć, że instrukcje int są niczym więcej niż specjalnymi instrukcjami call.

13.2 Wprowadzenie do usług BIOS'a

BIOS IBM PC używa przerwań programowych 5 i 10h..1Ah dla realizacji różnych działań. Dlatego też instrukcje int 5 i int 10h.. int 1ah dostarczają interfejsu do BIOS'a. Poniższa tablica streszcza usługi BIOS:

INT	**Funkcja**
5h	Operacja zrzutu ekranowego
10h	Obsługa monitora
11h	Konfiguracja komputera
12h	określenie rozmiaru pamięci
13h	Usługa obsługi dysków
14h	Obsługa złącza szeregowego I/O
15h	Dodatkowe funkcje
16h	Obsługa klawiatury
17h	Obsługa drukarki
18h	BASIC
19h	Gorący restart systemu
1Ah	Usługa zegara czasu rzeczywistego

Większość z tych podprogramów wymaga różnych parametrów w rejestrach 80x86. Niektóre wymagają dodatkowych parametrów w pewnych komórkach pamięci. Poniższe sekcje opisują dokładnie działanie wielu z podprogramów BIOS'a.

13.2.1 INT 5 –WYDRUK ZAWARTOŚCI EKRANU

Instrukcja	int 5h
Działanie BIOS	wydruk bieżącej zawartości ekranu
Parametry	brak

Jeśli wykonujemy instrukcję int 5h, PC wyśle dokładna kopię obrazu monitora na drukarkę jak gdybyś nacisnął klawisz PrtSc na klawiaturze. W rzeczywistości BIOS wywołuje instrukcję int 5h kiedy naciskasz PrtSc, więc te dwie operacje są dokładnie identyczne (jedynie jedna jest sterowana programowo zamiast ręcznie). Zauważmy, że 80286 i późniejsze procesory również używają int 5h dla pułapki BOUNDS.

13.2.2 INT 10h - USŁUGI VIDEO

Instrukcja	int 10h
Działanie BIOS	Usługi video
Parametry	Kilka, przekazywane w rejestrach ax, bx, cx, dx i es:bp

Instrukcja int 10h wykonuje kilka pokrewnych funkcji ekranowych. Możemy zastosować ja do inicjalizacji monitora, ustawienie rozmiaru i pozycji kursora, odczyt pozycji kursora, manipulacje piórem świetlnym, odczyt i zapis aktywnej strony, przesuwanie danych na ekranie w górę i w dół, odczyt i zapis znaków, odczyt i zapis pikseli w trybie graficznym i zapis łańcucha znaków na ekranie. Wybierasz poszczególne funkcje poprze ustawienie wartości w rejestrze ah.

Usługi video przedstawiają jeden z większych zbiorów dostępnych wywołań BIOS. Jest wiele różnych kart graficznych wyprodukowanych dla PC , każda z pomniejszymi zmianami i często mająca swój własny, unikalny zbiór funkcji BIOS.. BIOS odnosi się w tej dodatkowej liście do większości dostępnych funkcji, ale jak powiedziałem wcześniej, lista ta jest niekompletna i przestarzała, nie nadążając za szybkimi zmianami w technologii.

Prawdopodobnie najbardziej powszechnym zastosowaniem wywołania usługi video jest podprogram wypisujący znaki:

Nazwa:	Zapis znaku na ekranie w trybie TTY
Parametry:	ah = 0Eh, al. = kod ASCII (w trybie graficznym, bl = numer strony)

Podprogram ten zapisuje pojedynczy znak na ekranie. MS-DOS wywołuje ten podprogram do wyświetlania znaków na ekranie. Standardowa Biblioteka UCR również dostarcza wywołania , które pozwala nam zapisać znak bezpośrednio na ekranie przy użyciu wywołania BIOS.

Większość podprogramów ekranowych BIOS jest napisanych kiepsko. Są one niezmiernie wolne i nie dostarczają, w pewnym sensie, poprawy funkcjonalności. Z tego powodu większość programistów (którzy potrzebują wysoko wydajnych sterowników ekranowych) kończy na napisaniu swoich własnych kodów ekranowych. Uwzględnia to szybkość przy nakładach na przenośność oprogramowania. Niestety, rzadko mammy inny wybór. Jeśli chcemy funkcjonalności zamiast szybkości, powinniśmy rozważyć zastosowanie sterownika ekranowego ANSI.SYS dostarczanego z MS-DOS. Sterownik ten dostarcza wszystkich rodzajów użytecznych usług, takich jak czyszczenie końca linii, czyszczenie końca ekranu, itp.

TABLICA 49: FUNKCJE VIDEO BIOS (Lista częściowa)

AH	Parametry wejściowe	Parametry wyjściowe	Opis
0	al = tryb wyświetlanie		Ustawianie trybu wyświetlania
1	ch - linia początkowa cl - linia końcowa		Ustawienie kształtu i rozmiaru kursora. Wartości linii są w zakresie 0..15. Możemy ukryć kursor poprzez ustawienie ch = 20h
2	bh – numer strony dh - wiersz dl – kolumna		Ustawienie pozycji kursora (x,y) na ekranie. Zazwyczaj określimy stronę zerową. BIOS zachowa oddzielny kursor dla każdego kursora.
3	bh – numer strony	ch – linia początkowa cl – linia końcowa dl – kolumna dh – wiersz	Pobranie pozycji i rozmiaru kursora

4			Przestarzałe (obsługa pióra świetlnego)
5	Al. – numer strony		Ustawienie aktywnej strony. Zmiana aktywnej strony na stronę o wyszczególnionym numerze. Strona zerowa jest standardową stroną tekstu. Większość kart graficznych wspiera do ośmiu stron tekstu (0..7)
6	Al– liczba linii do przewinięcia Bh – atrybut ekranowy dla wyzerowanej przestrzeni cl – kolumna lewego górnego rogu okna ch – wiersz lewego górnego rogu okna dl - kolumna prawego dolnego rogu dh – wiersz prawego dolnego rogu		Czyści lub przesuwa okno. Jeśli al Zawiera zero, funkcja czyści prostokątną część ekranu wyszczególnioną przez cl / ch (lewy górny róg) i dl / dh (dolny prawy róg). Jeśli al. Zawiera jakąś inną wartość, prostokątne okno będzie przewijane w dół o liczbę linii określoną w al.
7	Al.– liczba linii do przewinięcia Bh – atrybut ekranu dla czyszczonej przestrzeni cl – kolumna lewego górnego rogu okna ch – wiersz lewego górnego rogu okna dl – kolumna prawego dolnego rogu okna dh - wiersz prawego dolnego rogu okna		Czyści lub przesuwa okno. Jeśli al zawiera zero, funkcja czyści prostokątną część ekranu wyszczególnioną przez cl / ch (lewy górny róg) i dl / dh (dolny prawy róg). Jeśli al Zawiera jakąś inną wartość, prostokątne okno będzie przewijane w dół o liczbę linii określoną w al
8	bh – numer strony	Al. – kod znaku Ah – atrybut znaku	Odczyt bajtu kodu i atrybuty znaku ASCII spod bieżącej pozycji kursora
9	Al. – znak Bh – numer strony bl – atrybut cx – liczba znaków do zapisania		Zapisuje cx kopii znaku i atrybutu z al./bl zaczynając spod bieżącej pozycji kursora na ekranie. Nie zmienia pozycji kursora.
0Ah	Al. – znak Bh – numer strony		Zapisuje znak z al. w bieżącej pozycji ekranu stosując istniejący atrybut. Nie zmienia pozycji kursora
0Bh	Bh - 0 bl – kolor tła		Ustala paletę kolorów dla wyświetlania tekstu
0Eh	Al. – kod znaku Bh – numer strony		Wypisuje znak na ekranie. Używa istniejącego atrybutu i zmienia pozycję kursora po zapisaniu
0Fh		ah – liczba znaków w wierszu al. – tryb pracy bh – numer strony	Zwraca informacje o trybie video

Zauważmy, że jest wiele innych podfunkcji BIOS 10h. W większości te inne funkcje zajmują się trybem graficznym (BIOS jest zbyt wolny aby zajmować się grafiką, więc nie powinniśmy używać tych wywołań) i rozszerzonym cechami pewnych kart graficznych.

13.2.3 INT 11h – Konfiguracja komputera

Instrukcja: int 11h
Działanie BIOS: zwraca listę wyposażenia komputera
Parametry: na wejściu: żadnych, na wyjściu: AX zawierający listę wyposażenia

Przy zwracaniu z 11h, rejestr AX zawiera kodowaną bitowo listę wyposażenia komputera ,poniższymi wartościami:

Bit 0	zainstalowana dyskietka
Bit 1	zainstalowany koprocesor matematyczny
Bity 2 , 3	zainstalowana płyta RAM (przestarzałe)
Bity 4, 5	tryby video 00 – żaden 01 – 40x25 10 – 80x25 11 – 80x25
Bity 6, 7	Liczba dysków twardych
Bit 8	Obecność DMA
Bity 9, 10, 11	Liczba zainstalowanych złącz szeregowych RS-232
Bit 12	zainstalowany port gier I/O
Bit 13	Przyłączony szeregowy port drukarki
Bity 14, 15	Liczba dołączonych drukarek

Zauważmy, że tą usługa BIOS'a zaprojektowano dla oryginalnych IBM PC, z bardzo ograniczonymi możliwościami rozszerzenia sprzętu. Zwracane bity przez to wywołanie jest prawie zawsze niezrozumiałe dzisiaj.

13.2.4 INT 12h - OKREŚLENIE ROZMAIRU PAMIĘCI

Instrukcja	int 12h
Działanie BIOS:	Określanie rozmiaru pamięci
Parametry:	Rozmiar pamięci jest zwracany w AX

Kiedy wrócimy do dni kiedy pecety IBM miały zainstalowane na płycie głównej 64kb pamięci, to wywołanie miało jakieś znaczenie. Jednakże dzisiejsze pecety mogą działać na pamięci 64 megabajtowej lub większych. Wyraźnie to wywołanie BIOS jest troszkę przestarzałe. Niektóre PS używają tego wywołania dla różnych celów, ale my nie możemy polegać na takim wywołaniu na maszynie.

13.2.5 INT 13h – NISKO POZIOWA OBSŁUGA DYSKÓW

Instrukcja:	int 13h
Działanie BIOS:	Obsługa dysków
Parametry:	ax, es:bx, cx, dx (zobacz poniżej)

Funkcja int 13h dostarcza kilku różnych nisko poziomowych usług dyskowych: Reset systemu dyskowego, pobranie statusu dysku, odczyt sektorów dysku , zapis sektorów dysku, weryfikacja sektorów dysku, format dysku i wiele, wiele innych. Oto inny przykład podprogramu BIOS , który zmieniał się przez lata. Kiedy ten program został stworzony, 10 megabajtowy dysk twardy był uważany za duży Dzisiaj wysoko wydajne gry wymagają 20 do 30 MB pamięci

AH	Parametry wejściowe	Parametry wyjściowe	Opis
0	dl – urządzenie (0..7fh to dyskietka,80h..ffh dysk twardy	ah – stan (0 i flaga przeniesienie wyzerowana jeśli brak błędu, kod błędu jeśli błąd)	Reset określonego napędu dyskowego. Zresetowanie dysku twardego również resetuje dyskietkę
1	dl – urządzenie (jak powyżej)	Ah – 0 Al. – stan poprzedniej operacji dyskowej	To wywołanie zwraca poniższe wartości stanu w al: 0 – żadnego błędu 1 – niepoprawne polecenie 2 – nie znaleziono znacznika adresu 3 – chroniony zapis dysku 4 – nie można znaleźć sektora 5 – błąd resetu 6 – nośnik usunięty 7 – zły parametr tablicy

			8 – przepełnienie DMA 9 – operacja DMA przekroczyła 64k granicę 10 – niedozwolona flaga sektora 11 – niedozwolona flaga ścieżki 12 – niedozwolony nośnik 13 - niepoprawna liczba sektorów 14 – napotkano znacznik adresu danej sterującej 15 – błąd DMA 16 – błąd danej CRC 17 – błąd prawidłowej danej ECC 32 – niedozwolony sterownik dysku 64 – błąd ustawienia głowic dysku 128 – błąd czasu oczekiwania 170 – dysk nie odczytany 187 – błąd niezdefiniowany 204 – błąd zapisu 224 – błąd stanu 225 – niedozwolony odczyt
2	Al. – liczba sektorów do odczytu Es:bx – adres bufora cl – bity 0..5 sektor # cl – bity 6/7 ścieżka bitów 8 i 9 ch – ścieżka bitów 0..7 dl - urządzenie (jak powyżej) dh – bity 0..5 głowica # dh – bity 6 i 7 : ścieżka bitów 10 i 11	ah - zwraca stan al. – błąd sekwencji długości przeniesienia – 0: sukces, 1: błąd	Odczytuje określoną liczbę z 512 bajtowych sektorów z dysku. Dana musi być 64 kilobajtowa lub mniejsza.
3	Tak samo jak (2) powyżej	Tak samo jak (2) powyżej	Zapisuje określoną liczbę z 512 bajtowych sektorów na dysk. Długość danej nie może przekraczać 64 kilobajtów
4	Tak samo jak (2) powyżej z wyjątkiem tego, że nie poturbujemy bufora.	Tak samo jak (2) powyżej	Weryfikacja danych w określonej liczbie 512 bajtowych sektorów na dysku.
0Ch	Tak samo jak (4) powyżej z wyjątkiem tego, że nie potrzebujemy sektor #	Tak samo jak (4) powyżej	Wysłanie głowicy dyskowej do określonej ścieżki na dysku.
0Dh	Dl – urządzenie # (80h lub 81h)	ah – zwraca stan przeniesienia – 0: żadnego błędu 1: błąd	Reset sterownika twardego dysku

Tablica 50: Popularne wywołania dyskowych subsystemów BIOS

Notka: zobacz właściwą dokumentację BIOS'a po dodatkową informację o wsparcie BIOSa dla dyskowych podsystemów

---

13.2.6 INT 14h – OBSŁUGA ZŁĄCZA SZERGOWEGO I/O

Instrukcja: int 14h
Działanie BIOS: dostęp do szeregowych portów komunikacyjnych
Parametry: ax, dx

IBM BIOS utrzymuje cztery różne szeregowe porty komunikacyjne (sprzętowo utrzymuje osiem). Ogólnie, większość PC ma jeden lub dwa zainstalowane porty szeregowe (COM1: i COM2) Int 14h posiada cztery pod funkcje – inicjalizacja, przesłanie znaku, odbiór znaku i status. Dla wszystkich czterech usług, numer

portu szeregowego (wartość w zakresie 0..3) znajduje się w rejestrze dx (0 = COM1, 1 = COM2 itd.) Int 14h oczekuje i zwraca inne dane w rejestrze al. lub ax.

13.2.6.1 AH = 0: INICJALIZACJA PORTU SZEREGOWEGO

podfunkca zero inicjalizuje port szeregowy. Wywołanie to pozwala nam ustawić szybkosć transmisji danych, wybrać tryb parzystości, wybrać liczbę bitów stopu i liczbę bitów przekazywanych poprzez linie szerwgową. Parametry te są wszystkie określone przez wartosci w rejestrze al. uzywając następujacego kodowania bitowego:

Bity	Funkcja
5..7	Wybór szybkości transmisji
	000 – 110 bodów
	001 – 150
	010 – 300
	011 – 600
	100 – 1200
	101 – 2400
	110 – 4800
	111 – 9600
3..4	Wybór parzystości
	00 – żadnej parzystości
	01 – nieparzyste
	10 – żadnej parzystości
	11 – parzyste
2	Bity stopu
	0 – jeden bit stopu
	1 – dwa bity stopu
0..1	Rozmiar znaku
	10 – 7 bitów
	11 – 8 bitów

Chociaż standardowy sprzętowy port szeregowy wspiera 19 600 bodów, niektóre BIOS'y mogą nie wspierać tej szybkości .

Przykład: Inicjalizacja COM1: 2400 bodów, żadnej parzystości, osiem bitów danych i dwa bity stopu –

```
mov     ah, 0               ;opcod inicjalizujący
mov     al., 10100111b      ;parametr danych
mov     dx, 0               ;port COM1:
int     14h
```

po wywołaniu kodu inicjalizującego, status portu szeregowego jest zwracany w ax (zobacz poniżej Status Portu Szeregowego, ah = 3)

13.2.6.2 AH=1: PRZESYŁANIE ZNAKU DO PORTU SZEREGOWEGO

Funkcja ta przesyła znak z rejestru al do portu szeregowego wyszczególnionego w rejestrze dx. Jeśli ah zawiera zero wtedy znak zostanie przesłany właściwie. Jeśli bit 7 ah zawiera 1, wtedy pojawi się jakiś rodzaj błędu. Pozostałe siedem bitów zawiera wszystkie stany błędów zwracanych przez wywołanie GetStatus z wyjątkiem limitu czasu błędu (który jest zwracany w bicie siedem) Jeśli został zakomunikowany błąd powinniśmy użyć podfunkcji trzy do ustawienia rzeczywistej wartości błędu ze sprzętowego portu szeregowego.

Przykład: Przesyłanie znaku do portu COM1:

```
mov     dx, 0           ;wybranie COM1:
mov     al., 'a'        ;Znak do przesłania
mov     ah, 1           ;przesłanie opcodu
int     14h
test    ah, 80h         ;sprawdzenie błędu
jnz     SerialError
```

Funkcja ta będzie czekała dopóki port szeregowy nie skończy przesyłania ostatniego znaku a potem przechowa znak w rejestrze przesyłowym.

13.2.6.3 AH=2 : ODBIÓR ZNAKU Z PORTU SZEREGOWEGO

Podfunkcja dwa jest używana do odczytu znaku z portu szeregowego. Na wejściu dx zawiera numer portu szeregowego. Na wyjściu al. zawiera znak odczytany z portu szeregowego i bit siedem ah zawierający status błędu. Kiedy ten podprogram jest wywołany, nie wraca do kodu wywołującego dopóki znak jest odbierany z portu szeregowego.

Przykład: Odczyt znaku z portu COM1:

```
mov     dx, 0               ;wybór COM1:
mov     ah, 2               ;opcod odbioru
int     14h
test    ah, 80              ;sprawdzenie błędu
jnz     SerialError
```

< Odebrany znak jest teraz w AL.>

13.2.6.4 AH=3: STATUS PORTU SZEREGOWEGO

To wywołanie zwraca informację o statusie portu szeregowego wliczając w to czy błąd wystąpił czy nie, czy znak został odebrany do bufora odbiorczego, czy bufor przesyłowy jest pusty i inne różne użyteczne informacje. Na wejściu tego podprogramu rejestr dx zawiera numer portu szeregowego, na wyjściu rejestr ax zawiera poniższe wartości:

AX:	Znaczenie bitu
15	Błąd limitu czasu
14	Pusty rejestr przesuwu
13	Pusty rejestr bufora
12	Błąd wykrycia przerwy
11	Błąd ramki
10	Błąd parzystości
9	Błąd przepełnienia
8	Dostępne dane
7	Wykryty sygnał na linii
6	Żądanie odbioru
5	Gotowość do odbioru
4	Gotowość do nadawania
3	Zmiana znacznika sygnału na linii
2	Zmiana znacznika żądania odbioru
1	Zmiana znacznika gotowości do odbioru
0	Zmiana znacznika gotowości do nadawania

Jest parę użytecznych bitów , nie odnoszących się do błędów, zwracanych w tej informacji o statusie. Jeśli bit dostępnych danych jest ustawiony (bit 8), wtedy port szeregowy odbiera dane a my powinniśmy je odczytać z portu szeregowego. Pusty rejestr bufora (bit 13) mówi nam, czy operacja przesyłu zostanie opóźniona podczas oczekiwania na bieżący znak do przesłania lub czy następny znak będzie bezpośrednio przesłany. Przez testowanie tych dwóch bitów możemy wykonać inne operacje podczas oczekiwania na to, że rejestr bufora stanie się dostępny lub ,że rejestr nadawczy zawiera znak.

Jeśli interesujesz się komunikacją szeregową powinieneś nabyć kopię *Joe Campbell's C Programmer's Guide to Serial Communications*. Chociaż napisana specjalnie dla programistów C, książka ta zawiera wiele pożytecznych informacji dla programistów pracujących w różnych językach programowania.

13.2.7 INT 15h - DODATKOWE FUNKCJE

Pierwotnie int 15h dostarczał usług do odczytu i zapisu kaset. Prawie bezpośrednio, każdy zdawał sobie sprawę, że kasety to historia, więc IBM zaczął używać int 15h dla wielu innych usług. Dzisiaj, int 15h jest używane dla szerokiego wachlarza usług, wliczając w to dostęp do pamięci rozszerzonej, odczyt karty

rozszerzeń gier / joystik i wielu, wielu innych działań. Z wyjątkiem wywołania joystika, większość z tych usług sięga poza zakres tego tekstu.

13.2.8 INT 16h - OBSŁUGA KLAWIAITURY

Instrukcja:	int 16h
Działanie BIOS:	Odczyt klawisza, test dla klawisza lub pobranie statusu klawiatury
Parametry:	al.

IBM PC BIOS dostarcza kilku funkcji wywołujących działających z klawiaturą. Aczkolwiek jest wiele podprogramów PC BIOS, liczba funkcji zwiększa się co roku. Sekcja ta opisuje trzy wywołania, które były dostępne na oryginalnym IBM PC.

13.2.8.1 AH=0: ODCZUYT KLAWISZA Z KLAWIATURY

Jeśli int 16h jest wywoływane z ah równym zero, BIOS nie zwraca sterowania do programu wywołującego dopóki klawisz jest na początku bufora. klawaitury Przy zwracaniu, al. zawiera kod ASCII dla odczytywanego klawisza z bufora a ah zawiera kod klawisza klawiatury. Kody klawiszy klawiatury są opisane w dodatkach.

Pewne klawisze na klawiaturze PC nie mają odpowiadających kodów ASCII. Klawisze funkcyjne Home, PgUp, End PgDn, klawisze strzałek i klawisze Alt są tego dobrym przykładem. Kiedy jest naciśnięty taki klawisz, int 16h zwraca zero w al. i kod klawisza w ah. Dlatego też kiedy kod ASCII równa się zero musimy sprawdzić rejestr ah dla określenia, który klawisz został naciśnięty.

Zauważmy, że odczytując klawisz z klawiatury stosując BIOS int 16h nie wywołujemy echa naciśniętego klawisza na ekranie. Musimy wywołać putc lub użyć int 10h do wydruku znaku kiedy go odczytamy jeśli chcemy potwierdzić go na monitorze

Przykład: Odczyt sekwencji naciśniętych klawiszy z klawiatury do czasu naciśnięcia klawisza ENTER

```
ReadLoop :      mov     ah, 0           ;Odczyt opcodu klawisza
                int     16h
                cmp     al., 0          ;Funkcja specjalna
                jz      ReadLoop        ;Jeśli tak ,nie potwierdzaj tego naciśnięcia
                putc
                cmp     al., 0dh        ;powrót karetki (ENTER)
                jne     ReadLoop
```

13.2.8.2 AH=1: SPRAWDZENIE CZY KLAWISZ JEST DOSTĘPNY SPOD KLAWIATURY

Ta szczególna podfunkcja int 16h pozwala nam sprawdzić aby stwierdzić czy klawisz jest dostępny w systemowym buforze klawiatury. Nawet jeśli klawisz nie jest dostępny, sterowanie jest zwracane (natychmiast!) do kodu wywołującego. Z tym wywołaniem możemy czasami monitorować klawiaturę aby zobaczyć czy klawisz jest dostępny i kontynuować działanie jeśli klawisz nie był naciśnięty (w przeciwieństwie do zamrożenia komputera do czasu naciśnięcia klawisza)

Nie ma żadnych parametrów wejściowych do tej funkcji. Przy zwracaniu, flaga zera będzie wyzerowana jeśli klawisz jest dostępny, ustawiona jeśli nie ma żadnego klawisza w buforze klawiatury. Jeśli klawisz jest dostępny, wtedy ax będzie zawierał kod i kod ASCII dla tego klawisza. Jednakże, funkcja ta nie usuwa tego klawisza z bufora klawiatury. Podfunkcja #0 musi być zastosowana do usunięcia znaków. Poniższy przykład demonstruje jak zbudować losowy generator liczb używając funkcji testowania klawiatury :

Przykład: generowanie liczb losowych podczas oczekiwania na naciśnięcie klawisza:

;Po pierwsze czyścimy bufor klawiatury ze wszystkich znaków

```
ClrBuffer:      mov     ah, 1           ;czy klawisz jest dostępny?
                int 16h
                jz      BufferIsClear   ;Jeśli nie przerywamy opróżnianie bufora
                mov     ah, 0           ; Opróżniamy ten znak z  bufora i próbujemy dalej
                int     16h
                jmp     ClrBuffer
BufferIsClear:  mov     cx, 0           ;inicjalizacja liczby „ losowej”
```

```
GenRandom:      inc     cx
                mov     ah, 1           ;zobacz czy klawisz jest jeszcze dostępny
                int     16h
                jz      GenRandom
                xor     cl, ch
                mov     ah, 0           ;Odczyt znaku z bufora
                int     16h
;Liczba losowa jest teraz w Cl, klawisz naciśnięty przez użytkownika w AX
```

Podczas oczekiwania na klawisz podprogram ten stale zwiększa rejestr cx. Ponieważ człowiek nie może odpowiedzieć bardzo szybko (przynajmniej pod względem mikrosekund) rejestr cl będzie przepełniany wiele razy, nawet przy najszybszych piszących na klawiaturze. W wyniku tego, cl będzie zawierał losowe wartości ponieważ użytkownik nie będzie mógł sterować tym (przynajmnie około 2 ms) kiedy klawisz jest naciśnięty.

13.2.8.3 AH=2: PYTANIE O STAN KLAWIATURY

Funkcja ta zwraca stan różnych klawiszy klawiatury PC w rejestrze al. Wartości zwracane są następujące:

Bit	Znaczenie
7	Stan INSERT (przełączany po naciśnięciu klawisza INS)
6	CAPS LOCK (1 = Caps lock włączony)
5	NUM LOCK (1 = Num lock włączony)
4	SCROLL LOCK (1 = Scroll lock włączony)
3	ALT (1 = klawisz Alt aktualnie wciśnięty)
2	CTRL (1 = klawisz Ctrl aktualnie wciśnięty)
1	Lewy SHIFT (1 = klawisz lewy Shift wciśnięty)
0	Prawy SHIFT (1 – klawisz prawy Shift wciśnięty)

Z powodu błędu w kodzie BIOS'a , bity te tylko odzwierciedlają bieżący status tych klawiszy, niekoniecznie za to odzwierciedlają status tych klawiszy, kiedy następny klawisz odczytany z bufora będzie naciśnięty. Żeby zapewnić, że te bity statusu odpowiadają stanowi tych klawiszy kiedy kod znaku jest czytany z bufora klawiatury, musimy wyczyścić bufor, poczekać na naciśnięcie klawisza a potem bezpośrednio sprawdzić status klawiatury.

13.2.9 INT 17h – USŁUGI DRUKARKI

Instrukcja:	int 17h
Działanie BIOS:	Drukuje dane i testuje stan drukarki
Parametry:	ax, dx

Int 17h steruje złączem równoległym drukarki na IBM PC w taki sam sposób jak int 14h steruje portem szeregowym. Ponieważ programowanie portu równoległego jest znacznie łatwiejsze niż sterowanie portem szeregowym, używając podprogramu int 17h jest nieco łatwiejsze niż używanie podprogramu int 14h.
Int 17h dostarcza trzech podfunkcji, wyszczególnionych przez wartości w rejestrze ah. Te podfunkcje to:

0 – wydruk znaku z rejestru AL.
1 – inicjalizacja drukarki
2 – zwracanie statusu drukarki

Każda z tych funkcji jest opisana w poniższych sekcjach.
Podobnie jak usługi dla portu szeregowego, usługi dla portu drukarki pozwalają nam wyszczególnić, który z portów drukarki zainstalowanych w systemie życzymy sobie użyć (LPT1:, LPT2: lub LPT3:). Wartość w rejestrze dx (0..2) określa który port drukarki będzie używany.

Jedna końcowa uwaga – pod DOS'em jest możliwe przekierowanie wszystkich danych wyjściowych drukarki do portu szeregowego. Jest to całkiem użyteczne jeśli używamy drukarki szeregowej . Usługi drukarki BIOS tylko przekazują do kontrolera drukarki równoległej. Jeśli potrzebujemy wysłać dane do drukarki szeregowej używając BIOS musimy użyć int 14h do przesłania danych do portu szeregowego.

13.2.9.1 AH=0: WYDRUK ZNAKU

Jeśli ah wynosi zero kiedy wywołujemy int 17h, wtedy BIOS będzie drukował znaki z rejestru al. Dokładnie jak kod znaku w rejestrze al. jest traktowany całkowicie zależy od sterownika drukarki jakiego używamy. Większość drukarek jednak. Uwzględnia drukowalny zbiór znaków ASCII i kilka znaków sterujących. Wiele drukarek również będzie drukowało wszystkie symbole w zbiorze znaków IBM/ASCII (wliczając Europejskie, kreślenie lini i inne specjalne symbole. Wiele drukarek traktuje znaki sterujące (zwłaszcza sekwencję ESC) w kompletnie różny sposób. Dlatego też, jeśli zamierzasz drukować całkiem inne znaki niż standardowe znaki ASCII, bądź przygotowany, że twoje programy mogą nie działać na innych niż te na których rozwijasz swoje programy.

Po powrocie z podprogramu zero podfunkcji int 17h, rejestr ah zawiera bieżący status. Wartości w rzeczywistości zwracane są opisane w sekcji o podfunkcji numer dwa.

13.2.9.2 AH=1: INICJALIZACJA DRUKARKI

Wykonując to wywołanie wysyłamy impuls elektryczny do drukarki mówiący o inicjalizacji. Przy zwracaniu, rejestr ah zawiera status drukarki jaki pokazuje funkcja numer dwa.

13.2.9.3 AH=2: ZWRACANIE STATUSU DRUKARKI

Wywołując tą funkcję sprawdzamy status drukarki i zwracamy go w rejestrze ah. Wartości zwracane to:

AH:	Znaczenie bitów
7	1 = drukarka zajęta, 0 = drukarka nie zajęta
6	1 = potwierdzenie z drukarki
5	1 = brak papieru
4	1 = drukarka włączona
3	1 = błąd I/O
2	Nie używane
1	Nie używane
0	Przekroczenie czasu

Potwierdzenie z drukarki jest , w gruncie rzeczy, zbędnym sygnałem (ponieważ drukarka zajęta / nie zajęta daje nam taką samą informację). Tak długo jak drukarka jest zajęta, nie zaakceptuje dodatkowej danej. Dlatego też, wywołanie funkcji wydruku znaku (ah=0) będzie wykonywane z opóźnieniem.

Sygnał braku papieru wystąpi zawsze kiedy drukarka wykryje brak papieru. Sygnał ten nie jest implementowany na wielu kontrolerach drukarek. W takich kontrolerach jest zawsze zaprogramowane zero logiczne (nawet jeśli drukarce brakuje papieru). Dlatego też pojawiające się zero na tej pozycji bitu, nie zawsze gwarantuje, że w drukarce mamy papier. Jednak patrząc tu, zdecydowanie znaczy, że w drukarce brakuje papieru.

Bit włączenia drukarki tak długo zawiera jeden, jak drukarka jest włączona. Jeśli użytkownik wyłączy drukarkę, bit ten zostanie wyzerowany.

Bit błędu I/O zawiera jeden jeśli wystąpił jakiś ogólny błąd I/O.

Bit przekroczenia czasu zawiera jeden, jeśli podprogram BIOS oczekiwał dłuższy okres czasu na drukarkę aby stała się „nie zajęta", a drukarka wciąż pozostaje „zajęta"

Zauważmy ,że inne urządzenia peryferyjne (inne niż drukarka) również są przyłączane do portu równoległego, często w dodatku do portu drukarki. Niektóre z tych urządzeń używają sygnałów lini błąd/status do zwracania danych do PC. Oprogramowanie sterujące takimi urządzeniami często przejmuje podprogram int 17h (przez technikę której pomówimy później) i zawsze zwraca stan „żadnego błędu" lub :przekroczenie czasu" jeśli wystąpi błąd na urządzeniu drukującym. Dlatego też powinniśmy zadbać aby nie uzależnić się w dużym stopniu od zmian tych sygnałów, kiedy wywołujemy int 17h BIOS'a.

13.2.10 INT 18h – URUCHOMIENIE BASICA

Instrukcja:	int 18h
Działanie BIOS:	aktywacja ROM BASIC
Parametry:	Żadne

Wykonując int 18h, aktywujemy interpretera ROM BASIC w IBM PC. Jednak, nie powinieneś używać tego mechanizmu uruchamiania BASIC'a ponieważ wiele kompatybilnych PC nie ma BASIC'a w ROM i wynik wykonania int 18h jest zdefiniowany.

13.2.11 INT 19h – GORĄCY RESTART SYSTEMU

Instrukcja:	19h
Działanie BIOS:	Restart systemu
Parametry:	żadne

Wykonanie tego przerwania daje taki sam efekt jak naciśniecie klawiszy Ctrl+Alt+Del na klawiaturze. Z oczywistych powodów to przerwanie powinno być stosowane ostrożnie.

13.2.12 INT 1Ah – ZEGAR CZASU RZECZYWISTEGO

Instrukcja:	int 1ah
Działanie BIOS:	usługi czasu rzeczywistego
Parametry:	ax, cx, dx

Są dwie usługi dostarczone przez podprogram BIOS – odczyt zegara i ustawianie zegara. Zegar czasu rzeczywistego PC utrzymuje licznik, który zlicza liczby co 1/18 sekundy, które zdarzyły się od północy. Kiedy odczytujemy zegar, pobieramy liczbę „taktów", które wystąpiły. Kiedy ustawiamy zegar, określamy liczbę „taktów", które wystąpiły od północy. Jak zwykle poszczególne usługi są wybierane przez wartość w rejestrze ah.

13.2.12.1 AH=1: ODCZYT ZEGARA CZASU RZECZYWISTEGO

Jeśli ah = 0, wtedy int 1ah zwraca 33 bitową wartość w al:cx:dx jak następuje:

Rejestr	Wartość zwracana
dx	Mniej znaczące słowo licznika zegara
cx	bardziej znaczące słowo licznika zegara
al.	zero jeśli zegar nie działał dłużej niż 24 godziny W innym wypadku wartość nie zerowa

32 bitowa wartość w cx:dx przedstawia liczbę z okresem 55 milisekund, która upłynęła od północy.

13.2.12.2 AH=1: USTAWIANIE ZEGARA CZASU RZECZYWSITEGO

Wywołanie to pozwala nam ustawić bieżącą wartość czasu systemowego, cx:dx zawiera bieżącą liczbę (z 55 milisekundowym zwiększaniem) od ostatniej północy. Cx zawiera bardziej znaczące słowo, dx zawiera mniej znaczące słowo.

13.3 WPROWADZENIE DO MS-DOS™

MS-DOS dostarcza wszystkich podstawowych funkcji zarządzających plikami i urządzeniami wymaganymi przez większość programów użytkowych uruchamianych na IBM PC. MS-DOS obsługuje pliki I/O, znaki I/O, zarządzanie pamięcią i inne różne funkcje w stosunkowo spójny sposób. Jeśli poważnie myślisz o pisaniu programów na PC, będziesz musiał przyjaźnie nastawić się do MS-DOS'a.

Tytuł tej sekcji to „ Wprowadzenie do MS-DOS". I jest to dokładnie to co znaczy. Nie ma sposobu aby MS-DOS został poznany kompletnie w jednym rozdziale. Danych jest wiele różnych książek, które zajmują się wyłącznie tym problemem. Microsoft napisał 1600 stronicową książkę na ten temat i nawet on nie wyczerpał tematu w pełni. Wszystko to prowadzi do prób wymigania się. Nie ma mowy aby ten temat nie mógł być potraktowany powierzchownie w pojedynczym rozdziale. Jeśli poważnie myślisz o pisaniu programów w języku assemblera na PC, musisz połączyć ten tekst z paroma innymi. Dodatkowe książki na temat MS-DOS to: MS-DOS Programmer's Reference (również zwany MS-DOS Technical Reference Manual). Peter Norton's Programmer's Guide to the IBM PC, The MS-DOS Encyclopedia i MS-DOS Developper's Guide. To czywiście jest tylko część listy książek dostępnych dzisiaj. Bez wątpienia MS-DOS Technical Reference Guide jest najważniejszym tekstem ,który warto przejrzeć jest to oficjalny opis wywołań MS-DOS i parametrów.

MS-DOS ma długa i barwną historię. Przez cały czas swojego życia przechodził kilka zmian, każda po to aby był lepszy niż poprzedni. Początków MS-DOS trzeba szukać w systemie operacyjnym CP/M.-80,

napisanym dla mikroprocesora Intel 8080. Faktycznie, MS-DOS v1.0 był niczym więcej niż klonem CP/M-80 dla mikroprocesora Intel 8088. Niestety, zdolność CP/M. do obsługi plików była straszna, mówiąc krótko. Dlatego też, DOS przewyższył CP/M. Nowe zdolności do obsługi plików , zgodne z Xenix i Unix, dodane do DOS stworzyły MS-DOS v. 2.0. Dodatkowe funkcje zostały dodane , do późniejszych wersji MS-DOS. Nawet wprowadzenie OS/2 i Windows NT ,Microsoft pracuje nad polepszeniem MS-DOS, który może pojawić się nawet w późniejszych wersjach.

Każda nowa cech dodana do DOS wprowadza nowe funkcje DOS, które zachowują całą funkcjonalność poprzednich wersji DOS. Kiedy Microsoft napisał podprogramy obsługi plików w wersji dwa, nie zmienił starych funkcji, po prostu zostały dodane nowe. Zachowanie oprogramowania kompatybilnego z programami, które działały ze starszymi wersjami DOS spowodowało, że DOS posiada dwa zbiory usług plikowych o identycznej funkcjonalności ale zupełnie nie kompatybilnych. W tym rozdziale skoncentrujemy się na małym podzbiorze dostępnych poleceń DOS. Zupełnie zignorujemy przestarzałe polecenia, które zostały powiększone o nowsze, lepsze polecenia w późniejszych wersjach DOS. Co więcej pominiemy opisywanie tych funkcji , które mają małe zastosowanie w codziennym programowaniu. Dla kompletnego szczegółowego opisu poleceń nie opisanych w tym rozdziale powinieneś nabyć jedną z wyżej wymienionych książek.

13.3.1 SEKWENCJA WYWOŁANIA MS-DOS

MS-DOS jest wywoływany poprzez instrukcję int 21h. Po wybraniu właściwej funkcji DOS, ładujesz do rejestru ah numer funkcji przed instrukcją int 21h. Większość funkcji DOS wymaga również innych parametrów. Ogólnie te inne parametry są przekazywane w zbiorze rejestrów CPU. Określone parametry będą omawiane wraz z każdą funkcją. Chyba, że MS-DOS zwraca jakąś określoną wartość w rejestrze, wszystkie rejestry CPU są zachowywane po wywołaniu DOS.

13.3.2 FUNKCJKE MS-DOS ZORIENTOWANE ZNAKOWO

DOS dostarcza 12 funkcji zorientowanych znakowo I/O. Większość z nich zajmuje się zapisem i odczytem danych do / z klawiatury, monitora, portu szeregowego i portu drukarki. Wszystkie te funkcje mają swoje odpowiedniki w usługach BIOS. Faktycznie, DOS zazwyczaj wywołuje stosowną funkcję BIOS wykonującą operację I/O. Jednak wskutek tego, że DOS przekierowywuje I/O i sterowniki urządzeń, funkcje te nie zawsze wywołują podprogramy BIOS. Dlatego nie powinniśmy wywoływać podprogramów BIOS (zamiast DOS) po prostu dlatego, że DOS kończy wywoływanie BIOS. Robiąc tak możemy zapobiec temu aby nasz program pracował z pewnymi wspieranymi przez DOS urządzeniami.

Z wyjątkiem funkcji siedem, wszystkie funkcje zorientowane znakowo sprawdzają urządzenia wejściowe konsoli (klawiatura) przez control-C Jeśli użytkownik naciśnie control-C, DOS wykona instrukcję int 23h. Zazwyczaj instrukcja ta powoduje, że program jest przerywany a sterowanie zwracane do DOS. Zapamiętajmy to, kiedy będziemy korzystać z tych funkcji.

Microsoft uznał te funkcje za przestarzałe i nie gwarantował, że będą dostarczone w przyszłych wersjach DOS. Więc ujmiemy te 12 funkcji w pigułce. Zauważ ,że Standardowa Biblioteka UCR dostarcza funkcjonalności wielu z tych funkcji i posiada właściwe funkcje DOS, które nie są przestarzałe.

Numer funkcji (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
1		al. – znak odczytany	Odczyt pojedynczego znaku z klawiatury i wyświetlenie wypisanego znaku na ekranie
2	dl - znak do wypisania		Wypisuje pojedynczy znak na monitorze
3		al. – znak do odczytu	Odczyt pojedynczego znaku z portu szeregowego
4	dl – znak do wypisania		Zapis pojedynczego znaku do portu wyjściowego
5	dl – znak do wypisania		Zapis pojedynczego znaku do drukarki
6	dl – znak wyjściowy 9jeśli nie 0FFh	al. – znak do odczytu (jeśli dl = 0FFh)	Na wejściu, jeśli dl zawiera 0FFh, funkcja ta próbuje odczytać znak z klawiatury. Jeśli znak jest

			dostępny, zwracane jest wyzerowana flaga zera i znak w al. Jeśli żaden znak nie jest dostępny, zwracana jest ustawiona flaga zera. Jeśli dl zawiera wartość inną niż 0FFh, podprogram wysyła znak na ekran. Ten podprogram nie sprawdza ctrl-C
7		al. – znak odczytany	Odczytuje znak z klawiatury. Nie daje echa znaku na wyświetlacz. Ta funkcja nie robi sprawdzenia dla ctrl-C
8		al. –znak odczytany	Podobnie jak funkcja 7, z wyjątkiem sprawdzania dla ctrl-C
9	ds.:dx – wskaźnik do łańcucha zakończonego „$”		Funkcja ta wyświetla znaki spod lokacji ds.:dx w górę (ale nie zawiera) kończącego znaku „$”
0Ah	ds.:dx – wskaźnik do bufora wejściowego		Funkcja odczytuje linię tekstu z klawiatury i przechowuje ją w buforze wejściowym pod ds.:dx. Pierwszy bajt buforu musi zawierać liczbę pomiędzy 0 a 255, która zawiera maksymalną liczbę dozwolonych znaków w buforze wejściowym. Podprogram przechowuje rzeczywista liczbę odczytanych znaków w drugim bajcie. Rzeczywiste znaki wejściowe zaczynają się w trzecim bajcie
0Bh		al. – status (0 = nie gotowe, 0FFh = gotowe)	Określa czy znak jest dostępny z klawiatury
0Ch	al. – Dosowy opcod 0,1, 6, 7 lub 8	al. - znak wejściowy jeśli opcod 1,6,7 lub 8	Ta funkcja opróżnia systemowy bufor klawiatury a potem wykonuje polecenia DOS określone w rejestrze al. (jeśli al. = 0, brak akcji)

Tablica 51: Funkcje DOS zorientowane znakowo

Funkcje 1,2,3,4,5,9 i 0Ah są przestarzałe i nie powinniśmy ich używać. Zamiast nich używajmy DOS owych funkcji plików I/O 9opcody 3Fh i 40h).

13.3.3 POLECENIA NAPĘDÓW MS-DOS

MS-DOS dostarcza kilka poleceń, które pozwalają nam na ustawienie domyślnego napędu, określenie który napęd jest domyślny i wykonanie innych działań. Poniższa tabela pokazuje te funkcje.

Numer funkcji (AH	Parametry wejściowe	Parametry wyjściowe	Opis
0Dh			Opróżnianie wszystkich bufor ów plikowych na dysku. Ogólnie

			wywoływana przez ctrl-C lub sekcję kodu, który musi zagwarantować zgodność plików, ponieważ może wystąpić błąd
0Eh	dl – numer napędu	Al. – numer napędu logicznego	Ustawia domyślny napęd według określonych wartości (0=A,1=B,2=C itd.). Zwraca liczbę logicznych napędów, chociaż może nie być ciągłości od 0 –al.
19h		al. – numer domyślnego napędu	Zwraca numer bieżącego domyślnego napędu (0=A,1=B,2=C, itp.)
1Ah	ds.:dx - adres DTA		Ustawia adres, którego MS-DOS używa dla przestarzałych plików I/O i poleceń Find First / Find Next
1Bh		al. – sektory /klastry cx – bajty / sektor dx – # klastrów ds.:bx – wskazuje bajt deskryptora nośnika	Zwraca informację o dysku w domyślnym nośniku. Zobacz również funkcję 36h. Typowe wartości dla deskryptora nośnika: 0F0h - 3.5” 0F8h – dysk twardy 0F9h – 720k 3.5” lub 1.2m. 5.25” 0FAh – 320k 5.25” 0FBh - 640k 3.5” 0FCh – 180k 5.25” 0FDh – 360k 5.25” 0FEh – 160k 5.25” 0FFh – 320k 5.25”
1Ch	Dl – numer nośnika	Zobacz powyżej	To samo co powyżej z wyjątkiem tego, ze możemy określić numer nośnika w rejestrze dl (0=domyślnie, 1=A,2=B,3=C, itd.).
1Fh		al. –zawiera 0FFH jeśli błąd, 0 jeśli nie ma błędu ds.:bx – wskaźnik do DPB	Ustawia domyślny DPB: jeśli z powodzeniem, funkcja ta zawraca wskaźnik do poniższych struktur; Dysk (bajt) – Numer nośnika (0-A, 1-B itd. Jednostka (bajt0 –liczba jednostkowa dla napędu Rozmiar Sektora (słowo) - # bajt / sektor ClusterMask (bajt) – sektor/ klaster minus jeden Cluster2 (bajt) – $2^{\text{klaster /sektor}}$. PierwszyFAT(słwo) – adres sektora gdzie zaczyna się FAT LicznikFAT (bajt) – liczba FAT’ów RootEntries(słowo) – liczba wejść w katalogu głównym PierwszySektor (słowo) – pierwszy sektor pierwszego klastra MaxCluster(słwo) – liczba klastrów na dysku plus jeden RozmiarFAT (słowo) – rozmiar FAT w sektorach DirSector (dword) – adres nagłówka urządzenia Nośnik(bajt) – bajt deskryptora nośnika PierwszyDostęp (bajt) – ustawia dostęp do urządzenia NextDPB (dword) link do następnego

			DPB na liście NextFree (słowo) ostatni zaalokowany klaster FreeCnt (słowo) – liczba wolnych klastrów
2EH	al. – sygnalizator weryfikacji (0 = nie zweryfikowane, 1= zweryfikowane		Włącza lub wyłącza weryfikację po zapisie. Zazwyczaj wyłącza ponieważ jest wolną operacją, ale może ją włączyć ,kiedy wykonuje się krytyczne I/.O
2Fh		es:bx wskaźnik do DTA	Zwraca wskaźnik do bieżącego DTA w es:bx
32h	dl – numer napędu	To samo co 1Fh	To samo co funkcja 1Fh z wyjątkiem tego, ze możesz pobrać określony numer napędu (0 = domyślny, 1=A,2=B, 3=C itd.).
33h	al. –05 (kod podfunkcji)		Zwraca numer urządzenia używanego do inicjowania DOS (1=A, 2=B itd.)
36h	dl – numer urządzenia	ax – sektory / klastry bx – dostępne klastry cx – bajty / sektor dx – klastry całkowite	Raportuje o ilości wolnej przestrzeni. Funkcja ta wypiera funkcje 1Bh i 1Ch, które wspierają dyski do 32MB. Ta funkcja działa z dużymi dyskami,. Możemy obliczyć wielkość wolnego miejsca (w bajtach) poprzez bx*cx*ax. Jeśli wystąpi błąd, funkcja zwróci 0FFFFh w ax.
54h		al. – weryfikacja stanu	Zwraca bieżący stan sygnalizatora weryfikacji (al. = 0 jeśli wyłączony, al. =1 jeśli włączony)

Tablica 52: Funkcje DOS dla napędów dyskowych

13.3.4 „PRZESTARZAŁE" FUNKCJE KATALOGUJĄCE MS-DOS

Funkcje DOS 0Fh – 18h, 1Eh, 20-24h i 26h – 29h są zapomnianymi funkcjami od czasu CP/M.-80. Generalnie nie powinniśmy sobie zawracać głowy tymi funkcjami ponieważ MS-DOS v.2.0 i późniejsze dostarczają dużo lepszych sposobów do wykonania tych operacji niż te funkcje.

13.3.5 FUNKCJE DOS DOTYCZĄCE DATY I CZASU

Funkcje daty i czasu MS-DOS zwracają bieżącą datę i czas w oparciu o wewnętrzne wartości podtrzymywane przez zegar czasu rzeczywistego (RTC). Funkcje dostarczone przez DOS zawierają odczyt i ustawianie daty i czasu. Te wartości daty i czasu są używane do elektronicznego oznaczania daty i czasu plików , które są tworzone na dysku. Dlatego też jeśli zmieniamy datę lub czas ,zapamiętajmy, że ma to wpływ na pliki, stworzone od tego czasu. Zauważmy, że Standardowa Biblioteka UCR również dostarcza zbioru funkcji daty i czasu, które w wielu przypadkach, są dużo łatwiejsze do zastosowania niż funkcje DOS.

Numer funkcji (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
2Ah		al. – dzień (0= Niedz., 1 = Poniedx. itd.) cx – rok dh – miesiąc (1 = Stycz, 2 = luty itd.) dl – dzień miesiąca (1 – 31)	Zwraca aktualną datę w zegarze systemowym
2Bh	cx – rok (1980 – 2099) dh – miesiąc (1 –12)		Ustawia aktualną datę w zegarze systemowym

	dl – dzień (1-31)		
2Ch		ch – godziny (24) cl – minuty dh – sekundy dl – setne sekundy	Odczytuje aktualny czas zegara systemowego. Zauważ, że pole setnej sekundy ma rozdzielczość 1/18 sekundy
2Dh	ch – godziny cl – minuty dh – sekundy dl – setne sekundy		Ustawia bieżący czas w zegarze systemowym.

Tablica 53; Funkcje daty i czasu

13.3.6 FUNKCJE ZARZĄDZANIA PAMIĘCIĄ

MS-DOS dostarcza trzech funkcji zarządzania pamięcią – przydzielanie , zwalnianie i zmiana rozmiaru (weryfikacja). Dla większości programów te trzy funkcje przydzielania pamięci nie są używane. Kiedy DOS wykonuje program, dostarcza całej dostępnej pamięci od startu tego programu do końca RAM w procesie wykonywania. Każda próba alokowania pamięci bez zwrócenia nie używanej pamięci do systemu wywoła błąd „niewystarczającej pamięci".
Skomplikowane programy, kończące się i pozostające w pamięci, uruchamiające inne programy lub wykonujące złożone zadania zarządzania pamięcią mogą wymagać użycia tych trzech funkcji zarządzania pamięcią . Ogólnie tego typu programy bezpośrednio zwalniają całą nie wykorzystaną pamięć a potem zaczynają przydzielać i zwalniać pamięć taką jak jest potrzebna. Ponieważ są to złożone funkcje, nie powinny być stosowane, chyba ,ż mamy dl nich specjalne zastosowanie. Niewłaściwe stosowanie tych poleceń może spowodować, że zagubimy pamięć w systemie, którą można będzie odzyskać tylko przez restart systemu. Każda z poniższych funkcji zwraca stan błędu we fladze przeniesienia. Jeśli przeniesienie jest wyzerowane przy zwrocie, wtedy operacja zakończyła się sukcesem, Jeśli flaga ta jest ustawiona, kiedy DOS powraca, wtedy rejestr ax zawiera jeden z poniższych kodów błędu:
7 – Zniszczony blok sterujący pamięci
8 – Niewystarczająca ilość pamięci
9 – Nieprawidłowy blok adresowy pamięci
Dodatkowo odnotujmy, że o tych błędach będziemy mówili w stosownym momencie

13.3.6.1 PRZYDZIELANIE PAMIĘCI

Funkcja (ah):	48h
Parametry wejściowe:	bx – żądany rozmiar bloku (w paragrafach)
Parametry wyjściowe:	Jeśli nie ma błędu (przeniesienie wyzerowane) Ax:0 wskazuje na przydzielony blok pamięci Jeśli wystąpił błąd (przeniesienie ustawione) bx – maksymalny możliwy przydzielony rozmiar ax – kod błędu (7 lub 8)

Funkcja ta jest stosowana do przydzielania bloku pamięci. Na wejściu do DOS, bx zawiera rozmiar żądanego bloku w paragrafach (grupa 16 bitowa). Na wyjściu, zakładając brak błędu, rejestr ax zawiera adres segmentowy początku przydzielonego bloku. Jeśli błąd wystąpi ,blok nie jest przydzielany a rejestr ax zawraca zawarty kod błędu. Jeśli wystąpi brak dostatecznej ilości pamięci, rejestr bx zwróci maksymalną liczbę rzeczywiście dostępnych paragrafów.

13.3.6.2 ZWALNIANIE PAMIĘCI

Funkcja (ah):	49h
Parametry wejściowe:	es:0 - Adres segmentowy zwalnianego bloku
Parametry wyjściowe	Jeśli przeniesienie jest ustawione, ax zawiera kod błędu (7,9)

Funkcja ta używa zwalnianej pamięci do przydzielenia jej przez powyższą funkcję 48h. Rejestr es nie może zawierać przypadkowego adresu pamięci. Musi zawierać wartość zwracaną przez funkcję przydzielania

pamięci. Nie możemy użyć tej funkcji do zwalniania części bloku. Do tej operacji służy zmodyfikowana funkcja przydzielania.

13.3.6.3 ZMODYFIKOWANA FUNKCJA PRZYDZIELANIA PAMIĘCI

Funkcja (ah):	4Ah
Parametry wejściowe:	es:0 – adres bloku do modyfikacji bx – rozmiar nowego bloku
Parametry wyjściowe:	jeśli przeniesienie jest ustawione, wtedy ax zawiera kod błędu 7,8 lub 9 bx zawiera maksymalny możliwy rozmiar (jeśli błąd 8)

Ta funkcja jest używana do zmiany rozmiaru przydzielanego bloku. Na wejściu es musi zawierać adres segmentowy przydzielanego bloku zwracany przez funkcję przydzielania pamięci. Bx musi zawierać nowy rozmiar tego bloku w paragrafach. Możemy prawie zawsze zredukować rozmiar, nie możemy normalnie zwiększać rozmiaru tego bloku jeśli inne bloki zostały przydzielone po bloku , na którym dokonujemy modyfikacji. Pamiętajmy o tym, kiedy stosujemy tą funkcję.

13.3.6.4 ZAAWANSOWANE FUNKCJE ZARZĄDZANIA PAMIĘCIĄ

Opcod 58h MS-DOS pozwala programiście modyfikować strategię alokowania pamięci MS-DOS i sterować blokiem wyższej pamięci (UMB) Są cztery podfunkcje do wykonania tego, wartości tych podfunkcji pojawiają się w rejestrze al. Poniższa tablica opisuje te funkcje:

Numer funkcji (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
58h	al. - 0	ax – strategia	Zwraca bieżącą strategię alokacji w ax. (zobacz tablicę poniżej po szczegóły)
58h	al. – 1 bx - strategia		Ustawia strategię alokacji MS-DOS na wartość określoną w bx (zobacz poniższą tabele po szczegóły)
58h	al. - 2	al.- flaga łączenia	Zwraca prawda/fałsz (1/0) w al. do określenia czy program może alokować pamięć w wyższym bloku pamięci
58h	al. – 3 bx –flaga łączenia (0 = żadnego łączenia, 1 =łączenie OK)		Łączy lub rozłącza obszar wyższej pamięci. Kiedy jest połączona, aplikacja może alokować pamięć w UMB (używając zwykłej funkcji alokującej DOS

Tablica 54: Funkcje zaawansowanego zarządzania pamięcią

Wartość	Nazwa	Opis
0	Pierwsze Niższe Dopasowanie	Przeszukuje pamięć konwencjonalna po pierwszy wolny blok pamięci dostatecznie duży aby spełnić żądanie alokacji
1	Najlepsze Niższe Dopasowanie	Przeszukuje pamięć konwencjonalną po najmniejszy blok spełniający żądanie alokacji.
2	Ostatnie Niższe Dopasowanie	Przeszukuje pamięć konwencjonalna po najwyższy adres zstępujący dla pierwszego dość dużego bloku spełniającego żądanie.
80h	Pierwsze Wyższe Dopasowanie	Przeszukuje wyższą pamięć, potem konwencjonalną, po pierwszy dostępny blok, który może spełnić żądanie alokacji
81h	Najlepsze Wyższe Dopasowanie	Przeszukuje wyższą pamięć, potem konwencjonalną, po najmniejszy blok, dosyć duży który może spełnić żądanie alokacji.
82h	Ostatnie Wyższe Dopasowanie	Przeszukuje wyższą pamięć od adresów wyższych do niższych, potem pamięć konwencjonalną od adresów wyższych do niższych szukając pierwszego bloku dość

		aby spełnić żądanie alokacji
40h	Pierwsze Wyższe (tylko) Dopasowanie	Przeszukuje tylko wyższą pamięć po pierwszy blok, dosyć duży aby spełnić żądanie alokacji
41h	Najlepsze Wyższe (tylko) Dopasowanie	Przeszukuje tylko wyższą pamięć po najmniejszy blok, dosyć duży aby spełnić żądanie alokacji
42h	Ostatnie Wyższe (tylko) Dopasowanie	Przeszukuje tylko wyższą pamięć od końca pamięci w dół, po pierwszy blok dosyć duży aby spełnić żądanie alokacji

Tablica 55: Strategie Alokacji Pamięci

Te różne strategie alokacji pamięci mogą mieć wpływ na wydajność systemu. Po analizę różnych strategii zarządzania pamięcią sięgnij po dobry teoretyczny tekst o systemach operacyjnych.

13.3.7 FUNKCJE MS-DOS STERUJĄCE PROCESEM

DOS dostarcza kilku usług zajmujących się ładowaniem, wykonywaniem i kończeniem programów. Wiele z tych funkcji zostało uznanych za przestarzałe w późniejszych wersjach DOS'a Tu mamy trzy funkcje ogólnie zajmujące się – zakończeniem programu, wstrzymaniem i pozostaniem w pamięci i wykonaniem programu. Te trzy funkcje będą omówione w poniższych sekcjach.

13.3.7.1 ZAKOŃCZENIE WYKONYWANIA PROGRAMU

Funkcja : 4Ch
Parametry wejściowe: al. – kod powrotu
Parametry wyjściowe: Nie zwraca nic do programu

Jest to wywołanie funkcji, która zazwyczaj jest używana do kończenia programu. Zwraca sterowanie do procesu wywołującego (normalnie, ale nie koniecznie, DOS) Zwracany kod może być przekazany do procesu wywołującego w rejestrze al. Ten kod powrotu może być sprawdzony przez DOS'owskie polecenie „kod powrotu IF ERRORLEVEL" w pliku wsadowym. Wszystkie pliki otwarte przez bieżący proces będą automatycznie zamknięte przy zakończeniu programu.

Zauważmy, że funkcja „ExitPgm" Standardowej Biblioteki UCR jest po prostu makrem, które wykonuje to szczególne wywołanie DOS. Jest to zwykły sposób zwracania sterowania do MS-DOS lub innego programu, który działa jako aktywna aplikacja.

13.3.7.2 ZAKOŃCZENIE, ALE POZOSTAWANIE W PAMIĘCIA

Funkcja(9ah): 31h
Parametry wejściowe: al. – kod powrotu
dx - rozmiar pamięci , w paragrafach
Parametry wyjściowe: nie zwracane do programu

Ta funkcja również powoduje zakończenie wykonywania programu, ale po powrocie do DOS , pamięć używana w procesie nie jest zwracana do DOS, do jego wolnego obszaru pamięci. Zasadniczo program pozostaje w pamięci. Programy, które pozostają rezydentne z pamięci po zwróceniu do DOS, często są nazywane TSR'ami (terminate and stay resident - oprogramowanie rezydentne). Kiedy jest wykonywane to polecenie, dx zawiera liczbę paragrafów pamięci pozostawionych w pamięci. Wartość ta jest odmierzana od początku „program segment prefix" ,segment oznacza początek pliku w pamięci. Adres PSP jest przekazywany do programu w rejestrze ds., kiedy program jest wykonywany po raz pierwszy. Będziemy musieli zachować tą wartość jeśli jest to program TSR.

Programy które są TSR, muszą dostarczyć kilka mechanizmów do restartowania. Ponieważ wracają do DOS, nie mogą być restartowane normalnie. Większość TSR'ów wstawiana jest do jednego z wektorów przerwań (takich jak klawiatura, drukarka lub szeregowy wektor przerwań), żeby zostać restartowanymi gdy tylko wystąpi zdarzenie powiązane ze sprzętem (takim jak naciśnięcie klawisza) .Jest to jak programy „pop – up" taki jak SmartKey.

Ogólnie programy TSR są pop – up lub specjalnymi sterownikami urządzeń. Mechanizm TSR dostarcza dogodnego sposobu do ładowania własnych podprogramów dla zamiany lub zwiększenia podprogramów BIOS. Nasz program załadowany do pamięci, wstawiany jest do właściwego wektora przerwań, żeby wskazywał wewnętrzny program obsługi przerwań naszego programu a potem zakończył i pozostał w

pamięci. Teraz , kiedy jest wykonywana właściwa instrukcja przerwania, będzie wywoływany nasz kod zamiast podprogram BIOS.

Jest daleko więcej szczegółów dotyczących TSR'ów zawierających różne kwestie. DOS poruszy te kwestie, i jak działają przerwania tutaj. Dodatkowe szczegóły pojawia się w późniejszych rozdziałach.

13.3.7.3 WYKONANIE PROGRAMU

Funkcja (ah):	40h
Parametry wejściowe:	ds.:dx – wskaźnik do ścieżki dostępu programu do wykonania es:bx – wskaźnik do bloku parametrów al. –0 = załadowanie i wykonanie, 1 = tylko załadowanie, 3 =załadowanie nakładki
Parametry wyjściowe:	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów: 1 – niewłaściwa funkcja 2 – plik nie znaleziony 5 – dostęp zastrzeżony 8 – nie wystarczająca pamięć 10 – złe otoczenie 11 – zły format

Funkcja wykonania (exec) jest niezmiernie złożoną, ale jednocześnie, bardzo użyteczną operacją. Polecenie to pozwala nam załadować i wykonać program z dysku. Na wejściu do funkcji exec , rejestry ds.:dx zawierają wskaźnik do łańcucha zakończonego zerem z nazwą pliku będącego ładowanym lub wykonywanym, es:bx wskazuje na blok parametrów a al. zawiera zero lub jeden w zależności czy chcemy załadować i wykonać program, czy tylko załadować go do pamięci. Przy powrocie, jeśli przeniesienie jest wyzerowane, wtedy DOS właściwie wykona polecenie. Jeśli flaga przeniesienia jest ustawiona, wtedy DOS napotka błąd podczas wykonania polecenia.

Parametr nazwy pliku może być pełną ścieżką dostępu wliczając w to informacje o napędzie i podkatalogach. „B:\DIR1\DIR2\MYPGM.EXE" jest doskonałą ,poprawną nazwą pliku (pamiętajmy jednak, że musi być zakończona zerem) Adres segmentowy tej ścieżki dostępu jest przekazany w rejestrach ds.:dx

Rejestry es:bx wskazują blok parametrów dla funkcji exec. Ten blok parametrów przybiera trzy różne formy w zależności od tego czy .program jest załadowany i wykonywany (al. = 0),tylko załadowany do pamięci (al. =1) lub załadowany jako nakładka (al. = 3)

Jeśli al. = 0, wtedy funkcja exec ładuje i wykonuje program. W tym przypadku rejestry es:bx wskazuje blok parametrów zawierający poniższe wartości:

Offset	Opis
0	Wartość słowa zawierająca adres segmentowy domyślnego otoczenia (zazwyczaj jest ustawione na zero jeśli standardowym środowiskiem jest DOS)
2	Wskaźnik podwójnego słowa zawierający adres segmentowy łańcucha lini poleceń.
6	Wskaźnik podwójnego słowa do domyślnego FCB pod adresem 5Ch
0Ah	Wskaźnik podwójnego słowa do domyślnego FCB pod adresem 6Ch

Obszar środowiska jest zbiorem łańcuchów zawierających domyślne ścieżki dostępu i inne informacje (informacje te są dostarczane przez DOS przy użyciu poleceń PATH, SET i innych). Jeśli te parametry wejściowe zawierają zero, wtedy exec przekaże standardowe środowisko DOS do nowej procedury. Jeśli nie – zero, wtedy ten parametr zawiera adres segmentowy bloku środowiska, do którego twoje działanie przekazuje program do wykonania. Generalnie powinniśmy przechowywać zero pod tym adresem.

Wskaźnik do łańcucha poleceń powinien zawierać adres segmentowy długości prefiksu łańcucha, który również jest zakończony przez znak powrotu karetki (znak powrotu karetki nie pojawia się w długości łańcucha). Łańcuch ten odpowiada danej, która jest zazwyczaj wypisywana po nazwie programu w lini poleceń DOS. Na przykład, jeśli wykonujemy automatyczne linkowanie, możemy przekazać polecenie łańcuchowe w poniższej formie:

```
CmdStr    byte    16, „MyPgm+Podprogram /m.", 0dh
```

Druga pozycja w bloku parametrów musi zawierać adres segmentowy tego łańcucha

Trzecia i czwarta pozycja w bloku parametrów wskazują domyślne FCB'y. FCB'y są używane przez przestarzałe polecenia zapisu danych do pliku, wiec są rzadko używane w nowszych aplikacjach. Ponieważ struktury danych tych dwóch wskaźników wskazują na rzadkie stosowanie, możemy zaznaczyć je jako grupę 20 zer.

Przykład: Formatowanie dyskietki w napędzie A: używamy polecenia FORMAT.EXE

```
                    mov     ah, 4Bh
                    mov     al., 0
                    mov     dx, seg   PathName
                    mov     ds., dx
                    lea     dx, PathName
                    mov     bx, seg   ParmBlock
                    mov     es, bx
                    lea     bx, ParmBlock
                    int     21h
                    -
                    -
                    -
PathName            byte    'C:\DOS\FORMAT.EXE', 0
ParmBlock           word    0                       ;domyślne środowisko
                    dword   CmdLine                 ;łańcuch lini poleceń
                    dword   Dummy, Dummy            ;fikcyjny FCB

CmdLine             byte    3, ' A;', 0dh
Dummy               byte    20 dup (?)
```

Wersje wcześniejsze MS-DOS niż 3.0 nie zabezpieczały żadnych rejestrów z wyjątkiem cs:ip kiedy wykonywana byłą funkcja exec. W szczególności, ss:sp nie był zabezpieczony. Jeśli używasz DOS'a v 2.x lub wcześniejszego, musisz użyć poniższego kodu:

```
;Przykład: Formatowanie dyskietki w napędzie A: przy użyciu polecenia FORMAT.EXE
            < odłożenie wszystkich rejestrów jakie musisz zachować>
            mov     cs :SS_save, ss             ;zachowanie SS:SP w komórce
            mov     cs: SP_save, sp             ;do której mamy dostęp później
            mov     ah, 4Bh                     ;opcod DOS'owy EXEC
            mov     al., 0                      ;załadowanie i wykonanie
            mov     dx, seg PathName            ;pobranie nazwy pliku do DS.:DX
            mov     ds., dx
            lea     dx, PathName
            mov     bx, seg ParamBlock          ;wskazuje ES:BX jako blok parametrów
            mov     es, bx
            lea     bx, ParamBlock
            int     21h
            mov     ss, cs: SS_save             ;przywrócenie SS:SP z lokacji
            mov     sp, cs: SP_save
            <przywrócenie rejestrów odłożonych na stos>
            -
            -
            -
SS_Save     word    ?
SP_Save     word    ?
            -
            -
            -
PathName    byte    'C:\DOS\FORMAT.EXE', 0
ParmBlock   word    0                           ;domyślne środowisko
            dword   CmdLine                     ;łańcuch lini poleceń
            dword Dummy, Dummy; Dummy           ;FCB'y
CmdLine     byte    3, 'A:', 0dh
Dummy       byte    20   dup  (?)
```

SS_Save i SP_Save muszą być zadeklarowane wewnątrz naszego segmentu kodu. Jedynie zmienne mogą być zadeklarowane gdziekolwiek.

Polecenie exec automatycznie alokuje pamięć dla programu będącego wykonywanym .Jeśli nie będziemy mieli wolnej nieużywanej pamięci przed wykonaniem tego polecenia, możemy otrzymać błąd niewystarczającej pamięci/ Dlatego też powinniśmy używać poleceń dealokacji pamięci DOS dla uwolnienia nieużywanej pamięci przed przystąpieniem do używania polecenia exec.

Jeśli al. = 1 kiedy wykonuje się funkcja exec, DOS załaduje określony plik, ale go nie wykona. Funkcja ta jest ogólnie używana do ładowania programu do wykonania w pamięci ale przekazuje kodowi wywołującemu sterownie i pozwala mu zacząć ten kod. Kiedy jest dokonywane wywołanie tej funkcji, es:bx wskazuje na jeden z poniższych bloków parametrów:

Offset	Opis
0	Wartość słowa zawierającego adres segmentowy bloku środowiska dla nowego procesu. Jeśli chcemy używać macierzystego procesu bloku środowiska, ustawiamy to słowo na zero.
2	Wskaźnik dword polecenia stopu do dla tej operacji. Polecenie stopu jest łańcuchem lini poleceń, który będzie się pojawiał pod lokacją PSP:80
6	Adres domyślnego FCB #1. Dla większości programów, powinien wskazywać blok 20 zer (chyba , że uruchamiamy program przy użyciu FCB)
0Ah	Adres domyślnego FCB #2. Również powinien wskazywać blok 20 zer
0Eh	Wartość SS:SP . Musimy załadować te cztery bajty do SS i SP przed zastartowaniem aplikacji
12h	Wartość CS:IP. Te cztery bajty zawierają adres startowy programu

Pola SSSP i CSIP są wartościami wyjściowymi. DOS wypełnia te pola i zwraca je w ładowanej strukturze. Wszystkie inne pola są wejściowe i musimy je wypełnić przed wywołaniem funkcji exec z al. =1.

Kiedy wykonujemy polecenie exec z al. =3, DOS po prostu .ładuje nakładkę do pamięci. Nakładki ogólnie składają się z pojedynczego segmentu kodu, który zawiera jakieś funkcje które chcemy wykonać. Ponieważ nie tworzymy nowego procesu, blok parametrów dla tego ładowanego typu jest dużo prostsze niż dla pozostałych dwóch typów operacji ładowania, Na wejściu es:bx musi wskazywać poniższy blok parametrów w pamięci:

Offset	Opis
0	Wartość słowa zawierającą adres segmentu, pod który zostanie załadowany plik. Plik będzie załadowany pod offsetem zero wewnątrz tego segmentu
2	Wartość słowa zawierająca stałą przemieszczenia dla tego pliku

W odróżnieniu od funkcji ładowania i wykonania, funkcja nakładki automatycznie nie alokuje pamięci dla pliku będącego ładowanym .Nasz program musi alokować wystarczająco pamięci a potem przekazać adres tego bloku pamięci do polecenia exec (przez powyższy blok parametrów). Do polecenia exec jest przekazywany tylko adres segmentu, offset jest zawsze zakładany jako zero. Stała przemieszczenia również powinna zawierać adres segmentu dla plików „.EXE”. Dla plików „.COM” parametr stałej przemieszczenia powinien być zero.

Polecenie nakładki jest bardziej użyteczne przy ładowaniu nakładek z dysku do pamięci. Nakładka jest segmentem kodu, który rezyduje na dysku dopóki program w rzeczywistości będzie musiał wykonać ten kod. Wtedy kod jest ładowany do pamięci i wykonywany. Nakładki mogą zredukować ilość pamięci programu poprzez przyjęcie założenia o ponownym użyciu tej samej części pamięci dla różnych procedur nakładki(rzecz jasna , tylko jedna taka procedura może być aktywna w tym samym czasie. Poprzez umieszczenie rzadko używanego kodu i kodu inicjalizującego w plikach nakładkowych, możemy zredukować ilość pamięci używanej przez pliki naszego programu. Jednak, słowo uwagi, obsługa nakładek jest bardzo skomplikowanym zadaniem. Nie jest to coś do czego początkujący programiści asemblerowi zabierają się w pierwszej kolejności. Kiedy ładujemy plik do pamięci 9jako przeciwieństwo ładowania i wykonania pliku), DOS nie zmienia wszystkich rejestrów, więc nie musimy specjalnie dbać o przechowanie ss:sp i innych rejestrów.

„Encyklopedia MS-DOS’ zawiera doskonały opis użycia funkcji exec.

13.3.8 „NOWE” FUNKCJE ZAPISUJĄCE DANE MS-DOS

Poczynając od DOS v .2.0 Microsoft wprowadził zbiór plików działających procedur, które (ostatecznie) uzyskują dostęp do plików dyskowych znośnych pod MS-DOS. Nie tylko znośnych, ale i łatwych do użycia! Poniższe sekcje opisują użycie tych poleceń dostępu do plików na dysku.

Polecenia plikowe, które zajmują się nazwami plików (Create, Open, Delete, Rename i inne) przekazują adres ścieżki odstępu zakończonej zerem. Te , które w rzeczywistości otwierają plik (Create i Open) zwracają jako wynik logiczny numer pliku (zakładając, oczywiście, że nie wystąpił błąd) Ten logiczny numer pliku jest używany z innymi funkcjami (read, write, seek, close, itp.) aby zwiększyć dostęp do otwartego pliku. Pod tym względem, logiczny numer pliku jest podobny do zmiennej plikowej w Pascalu. Rozważmy poniższy kod Microsoft/Turbo Pascal:

```
program DemoFile; var  F:TEXT;
begin
        assign (f, 'FileName.TXT');
        rewrite(f);
        writeln (f, 'Hello there');
        close (f);
end.
```

Zmienna plikowa f jest używana w tym przykładzie Pascalowskim w taki sam sposób w jaki logiczny numer pliku jest używany w programie asemblerowym – polepszenie dostępu do pliku, który został stworzony w programie.

Wszystkie poniższe polecenia DOS'a zwracają stan błędu we fladze przeniesienia. Jeśli flaga przeniesienia jest wyzerowana kiedy DOS wraca do naszego programu, wtedy operacja kończy się powodzeniem. Jeśli flaga przeniesienia jest ustawiona przed powrotem, wtedy wystąpi jakiś rodzaj błędu a jego numer zawiera rejestr AX. Rzeczywiste błędy wartości zwracanych będziemy omawiali wraz z każdą funkcją w poniższych sekcjach.

13.3.8.1 OTWIERANIE PLIKU

Funkcja (ah):	3Dh
Parametry wejściowe:	
	al. – wartość dostępu do pliku 0 – plik otwarty do odczytu 1 – plik otwarty do zapisu 2 – plik otwarty do odczytu i zapisu
	ds.:dx - wskazuje łańcuch zakończony zerem zawierający nazwę pliku
Parametry wyjściowe:	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów: 2 – plik nie znaleziony 4 – zbyt wiele otwartych plików 5 – dostęp zastrzeżony 12 – dostęp nieprawidłowy Jeśli przeniesienie jest wyzerowane, ax zawiera wartość logicznego numeru pliku powiązanego przez DOS

Plik musi być otwarty przed tym zanim uzyskamy do niego dostęp. Polecenie open, otwiera plik, który już istnieje. Powoduje to, że jest to trochę podobne do procedury pascalowskiej Reset. Próbując otworzyć plik , który nie istnieje, wywołamy błąd. Przykład:

```
lea     dx, FileName            ;zakładamy, że DS. wskazuje segment
mov     ah, 3dh                 ; z nazwą pliku
mov     al., 0                  ;otwieramy do czytania
int     21h
jc      OpenErroe
mov     FileHandle, ax
```

jeśli wystąpi błąd podczas otwierania pliku, plik nie zostanie otwarty. Powinniśmy zawsze sprawdzać błędy przed wykonaniem DOS'owskiego polecenia open, ponieważ kontynuowanie działania na pliku, który nie został poprawnie otwarty doprowadzi do katastrofalnych konsekwencji. Dokładnie jak operowanie błędem otwarcia jest naszym zadaniem, przynajmniej powinniśmy wydrukować informację o błędzie aby dąć sposobność użytkownikowi do określenia inne nazwy pliku.

Jeśli polecenie open zakończy się bez wygenerowania błędu, DOS zwróci logiczny numer pliku dla tego pliku w rejestrze ax. Zazwyczaj powinniśmy zachować tą wartość gdzieś, aby można było jej użyć, kiedy będziemy chcieli uzyskać dostęp do pliku później.

13.3.8.2 TWORZENIE PLIKU

Funkcja (ah)	3Ch
Parametry wejściowe:	ds.:dx – adres ścieżki dostępu zakończonej zerem cx – Atrybut pliku
Parametry wyjściowe:	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów: 3 – Ścieżka nie znaleziona 4 – zbyt wiele otwartych plików 5 – dostęp zastrzeżony Jeśli przeniesienie jest wyzerowane, ax zwraca zawartość logicznego numeru pliku

Tworzy nowy plik. Tak jak przy poleceniu OPEN, ds.:dx wskazuje łańcuch zakończony zerem zawierający nazwę pliku. Ponieważ funkcja ta tworzy nowy plik, DOS zakłada, że otworzyliśmy plik tylko do zapisu. Innym parametrem, przekazany w cx, jest początkowe ustawienie atrybutu pliku. Najmniej znaczące sześć bitów cx zawiera poniższe wartości:

Bit	Znaczenie
0	Plik jest plikiem tylko do odczytu
1	Plik jest plikiem ukrytym
2	Plik jest plikiem systemowym
3	Plik jest identyfikatorem dysku
4	Plik jest podkatalogiem
5	Plik był archiwizowany

Generalnie nie powinniśmy ustawiać żadnego z tych bitów. Większość zwykłych plików powinno być stworzonych z atrybutem zero. Dlatego też rejestr cx powinien być załadowany wartością zero przed wywołaniem funkcji Create.

Na wyjściu flaga przeniesienia jest ustawiona, jeśli wystąpi błąd. Błąd „Ścieżka nie znaleziona" wymaga dodatkowych wyjaśnień. Błąd ten jest generowany nie wtedy kiedy plik nie został znaleziony (co może występować cały czas ponieważ polecenie to jest zazwyczaj używane do tworzenia nowych plików), ale wtedy kiedy podkatalog w ścieżce dostępu nie może być znaleziony.

Jeśli flaga przeniesienia jest wyzerowana, kiedy DOS wraca do naszego programu, wtedy plik będzie poprawnie otwarty a rejestr ax zawiera logiczny numer pliku dla tego pliku.

13.3.8.3 ZAMYKANIE PLIKU

Funkcja:	3Eh
Parametry wejściowe:	bx – logiczny numer pliku
Parametry wyjściowe:	Jeśli flaga przeniesienia jest ustawiona, ax zawiera 6, jedyny możliwy błąd, który jest niewłaściwym błędem obsługi.

Funkcja ta jest używana do zamykania otwartego pliku powyższymi poleceniami Open lub Create. W rejestrze bx jest przekazywany logiczny numer pliku, i zakładając, że logiczny numer pliku jest poprawny, zamyka określony plik. Powinniśmy zamykać wszystkie pliku w używanym programie jeśli tylko mamy z nimi połączenie, aby uniknąć uszkodzenia plików dyskowych przy wyłączeniu zasilania systemu lub resetu maszyny, podczas gdy plik są jeszcze otwarte.

Zauważmy, ze wyjście do DOS (lub przerwanie DOS przez naciśnięcie control – C lub control – break) automatycznie zamyka wszystkie otwarte pliki. Jednakże, nigdy nie powinniśmy polegać na tej cesze ponieważ pokazuje to złą praktykę programowania.

13.3.8.4 ODCZYT Z PLIKU

Funkcja (ah):	3Fh
Parametry wejściowe:	bx - logiczny numer pliku

cx – liczba bajtów do odczytu
ds.:dx - adres tablicy do przetrzymania odczytanych bajtów

Parametry wyjściowe: Jeśli flaga przeniesienia jest ustawiona, ax zawiera jeden z poniższych kodów błędów
5 – dostęp zastrzeżony
6 – nieprawidłowy uchwyt
Jeśli flaga przeniesienia jest wyzerowana, ax zawiera liczbę bajtów rzeczywiście odczytanych z pliku

Funkcja read jest używana do odczytu pewnej liczby bajtów z pliku. Rzeczywista liczba bajtów jest określona przez rejestr cx na wejściu do DOS'a. Logiczny numer pliku, który określa z którego pliku będą czytane bajty, jest przekazywany w bx. Rejestr ds.:dx. Zawiera adres bufora w którym odczytane bajty będą przechowywane.

Przy zwrocie, jeśli nie wystąpił błąd, rejestr ax zawiera liczbę bajtów rzeczywiście odczytanych.. Chyba, że osiągniemy koniec pliku (EOF), wtedy liczba ta będzie się zgadzała z wartością przekazaną do DOS'a w rejestrze cx. Jeśli zostanie osiągnięty koniec pliku, wartość zwracana w ax będzie gdzieś pomiędzy zero a wartością przekazaną do DOS w rejestrze cx. Jest to jedynie test dla warunku EOF.

Przykład: Przykład otwarcia pliku i odczytu do końca pliku

```
        mov   ah, 3dh              ;otwarcie pliku
        mov   al., 0               ;otwarcie do odczytu
        lea   dx, Filename         ;zakładamy, że DS. wskazuje nazwę pliku
        int   21h
        jc    BadOpen
        mov   FHndl, ax            ,zachowanie numeru logicznego pliku
LP:     mov   ax, 3fh              ;odczyt danej z pliku
        lea   dx, Buffer           ;adres danej bufora
        mov   cx,1                 ;odczyt jednego bajtu
        mov   bx, FHndl            ;pobranie wartości logicznego numeru pliku
        int   21h
        jc    ReadError
        cmp   ax, cx               ;czy osiągnięty EOF?
        jne   EOF
        mov   al., Buffer          ;pobranie odczytanego znaku
        putc                       ;wydrukowanie go
        jmp   LP                   ;odczytaj następny bajt
EOF:    mov   bx, FHndl
        mov   ah, 3eh              ;zamykanie pliku
        int   21h
        jc    CloseError
```

Ta część kodu będzie odczytywała cały plik, którego (zakończona zerem) nazwa pliku została znaleziona pod adresem „Filename" w bieżącym segmencie danych i zapisuje każdy znak do pliku standardowego urządzenia wyjściowego, stosując podprogram putc z UCR StdLib. Przestrzegam, że takie jednoznakowe działanie jest zdecydowanie wolne. Będziemy omawiali lepsze sposoby szybkiego odczytu pliku trochę później w tym rozdziale

---

13.3.8.5 ZAPIS DO PLIKU

Funkcja (ah): 40h

Parametry wejściowe: bx – numer logiczny pliku
cx – liczba bajtów do zapisu
ds.:dx – adres bufora zawierającego dane do zapisu

Parametry wyjściowe: Jeśli flaga przeniesienia jest ustawiona, ax zawiera poniższy kod błędu
5 – Dostęp zastrzeżony
6 – nieprawidłowy uchwyt
Jeśli flaga przeniesienia jest wyzerowana, ax zawiera liczbę bajtów rzeczywiście zapisanych do pliku.

Funkcja ta jest prawie odwrotnością polecenia read przedstawianego wcześniej. Zapisuje określoną liczbę bajtów pod ds.:dx do pliku zamiast go odczytać. Przy zwracaniu, jeśli liczba bajtów zapisanych do pliku nie jest równa pierwotnej liczbie w rejestrze cx, dysk jest pełny i powinno to być traktowane jako błąd.

Jeśli cx zawiera zero, kiedy ta funkcja jest wywoływana, DOS skraca plik do bieżącej pozycji pliku *tj, wszystkie dane z bieżącej pozycji w pliku będą skasowane)

13.3.8.6 POZYCJONOWANIE (PRZESUWANIE WSKAŹNIKA PLIKU)

Funkcja:	42h
Parametry wejściowe:	al. – sposób przesuwania 0 – offset określony od początku pliku 1 – offset określony od bieżącego wskaźnika pliku 2 – Wskaźnik jest przesuwany od końca pliku minus określony offset bx – logiczny numer pliku cx:dx – odległość na jaką trzeba przesunąć, w bajtach
Parametry wyjściowe:	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów 1 – Nieprawidłowa funkcja 6 – niepoprawny uchwyt Jeśli przeniesienie jest wyzerowane, dx:ax zawiera nową pozycję pliku

Polecenie to jest używane do przesuwanie wskaźnika pliku w pliku o dostępie swobodnym. Są trzy metody przesuwania wskaźnika pliku na odległość absolutną wewnątrz pliku (jeśli al. =0), jakąś dodatnią odległość od bieżącej pozycji wskaźnika pliku (jeśli al. = 1), lub jakąś odległość od końca pliku (jeśli al. =2). Jeśli AL. nie zawiera 0,1 lub 2 DOS zwraca błąd niepoprawnej funkcji. Jeśli funkcja zakończy się powodzeniem. Następny bajt do zapisania lub odczytu będzie występował pod określoną lokacją.

Zauważ, że DOS traktuje cx:dx jako liczbę całkowitą bez znaku. Dlatego też pojedyncze polecenie seek nie może być używane do przesuwania wstecz w pliku. Zamiast tego musi być użyta metoda #0 do pozycjonowania wskaźnika pliku pod jakąś absolutną pozycją w pliku. Jeśli nie wiemy gdzie jesteśmy obecnie i chcemy cofnąć 256 bajtów możemy użyć poniższego kodu:

```
mov     ah, 42h                 ;polecenie seek
mov     al., 1                  ;przesunięcie z bieżącej lokacji
xor     cx, cx                  ;zerowanie cx i dx
xor     dx, dx
mov     bx, Filename
int     21h
jc      SeekError
sub     ax, 256                 ;DX:AX zawiera teraz bieżącą pozycję
sbb     dx, 0                   ;pliku, więc obliczamy pozycję 256 bajtów
mov     cx, dx
mov     dx, ax
mov     ah, 42h
mov     al., 0                  ;absolutna pozycja pliku
int     21h                     ;BX zawiera jeszcze uchwyt
```

13.3.8.7 USTAWIENIE BUFORA ROBOCZEGGO OPERACJI DYSKWYCH (DTA)

Funkcja:	1Ah
Parametry wejściowe:	ds.:dx – wskaźnik do DTA
Parametry wyjściowe;	żadnych

Polecenie to jest nazywane „ustawieniem bufora roboczego operacji dyskowych „ ponieważ było (jest) używane z oryginalną funkcją pliku DOS v.1.0. Zazwyczaj nie rozpatrujemy tej funkcji, z wyjątkiem faktu, że jest również używane przez funkcję 4Eh i 4Fh 9opisane poniżej) do ustawienia wskaźnika do 43 –bajtowego obszaru bufora. Jeśli ta funkcja nie jest wykonywana przed wykonaniem funkcji 4Eh lub 4Fh, DOS będzie używał domyślnej przestrzeni bufora spod PSP:80h

---

13.3.8.8 ZNAJDOWANIE PIERWSZEGO PLIKU W KATALOGU

Funkcja (ah):	4EH
Parametry wejściowe:	cx – atrybuty ds.:dx – wskaźnik do nazwy pliku

Parametry wyjściowe: Jeśli przeniesienie jest ustawione, ax zawiera jeden z następujących kodów błędów:

2 – plik nie znaleziony
18 – nie ma więcej plików

Funkcje Find First File i Find Next File (opisana poniżej) są używane do wyszukiwania plików, określonych przez zastosowanie niejednoznacznego odniesienia do pliku. Niejednoznaczne odniesienie do pliku jest nazwa pliku zawierającą znaki wieloznaczności „*" i „?". Funkcja Find First File jest używana do znalezienie pierwszej takiej nazwy pliku wewnątrz określonego katalogu. Funkcja Find Next File jest używana do znajdowania następujących po sobie wejść w katalogu

Ogólnie, kiedy używamy niejednoznacznego odniesienia do pliku, polecenie Finf First File jest stosowane do zlokalizowania pierwszego wystąpienia pliku ,potem jest wywoływana pętla, Find Next File dla zlokalizowania wszystkich innych wystąpień pliku, dopóki nie będzie więcej plików (błąd numer 18). Gdziekolwiek Find First File jest wywoływana, ustawia określone informacje o DTA:

Offset	Opis
0	Zarezerwowane do użycia przez Find Next File
21	Atrybut znalezionego pliku
22	Czas ostatniej modyfikacji
24	Data ostatniej modyfikacji
26	Rozmiar pliku w bajtach
30	Nazwa pliku i rozszerzenie w kodzie ASCIIZ

(Offsety są pisane dziesiętnie)

Zakładając że Find First File nie zwraca żadnego rodzaju błędu, dopasowana nazwa pierwszego pliku do opisu pliku niejednoznacznego pojawi się pod offsetem 30 w DTA.

Notka: jeśli określona ścieżka dostępu nie zawiera żadnego znaku wieloznaczności, wtedy Find First File będzie zwracał dokładnie określoną nazwę pliku, jeśli taka istnieje. Każde późniejsze wywołanie Find First File będzie zwracało błąd.

Rejestr cx zawiera atrybut dla tego pliku. Zazwyczaj, cx powinien zawierać zero. Jeśli nie – zero, Find First File (i Find Next File) będzie obejmował nazwy plików mających określony atrybut, również wszystkie zwykłe nazwy plików.

13.3.8.9 ZNAJDOWANIE NASTĘPNEGO PLIKU W KATALOGU

Funkcja (ah): 4Fh
Parametry wejściowe: żadne
Parametry wyjściowe; Jeśli przeniesienie jest ustawione, wtedy nie ma więcej innych plików, a ax będzie zwracał 18.

Funkcja Find Next File jest używana do wyszukiwania dodatkowych nazw plików dopasowanych do niejednoznacznego odwołania do pliku po wywołaniu Find First File. DTA musi wskazywać rekord danych ustawiony przez funkcję Find First File.

Przykład: Poniższy kod listuje nazwy wszystkich plików w bieżącym katalogu, które kończą się „.EXE". Przypuszczalnie zmienna „DTA" znajduje się w bieżącym segmencie danych:

```
                mov     ah, 1Ah         ;ustawienie DTA
                lea     dx, DTA
                int     21h
                xor     cx, cx          ;Żadnych atrybutów
                lea     dx, FileName
                mov     ah, 4Eh         ;Find First File
                int     21h
                jc      NoMoreFiles     ;robimy jeśli błąd
DirLoop:        lea     si, DTA+30      ;Adres nazwy pliku
                cld
PrtName:        lodsb
                test    al., al.        ;zero bajtów?
                jz      NextEntry
                putc
                jmp     PrtName
```

```
NextEntry:      mov     ah, 4Fh         ;Find Next File
                int     21h
                jnc     DirLoop         ;drukuj następną nazwę
```

13.3.8.10 USUWANIE PLIKU

Funkcja (ah0;	41h
Parametry wejściowe:	ds.:dx – adres ścieżki do usunięcia
Parametry wyjściowe:	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów 2 – Plik nie znaleziony 5 – Dostęp zastrzeżony

Funkcja ta usuwa określony plik z katalogu. Nazwa pliku musi być nazwą nie niejednoznaczną (tj. nie może zawierać znaków wieloznacznych)

13.3.8.11 ZMAINA NAZWY PLIKU

Funkcja (ah0:	56h
Parametry wejściowe:	ds.:dx – wskaźnik do ścieżki istniejącego pliku es:di – wskaźnik do nowej ścieżki dostępu
Parametry wyjściowe :	Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów 2 – plik nie znaleziony 5 – dostęp zastrzeżony 17 – nie takie samo urządzenie

Polecenie to służy dwóm celom: pozwala ono zmieniać nazwę pliku i pozwala na przenoszenie pliku z jednego katalogu do innego (tak długo jak te dwa podkatalogi są na tym samym dysku)

Przykład: Zmaian nazwy z „MYPGM.EXE” na „YPURPGM.EXE”

```
; Zakładamy, że ES i DS. wskazują na bieżący segment danych
; zawierający  nazwy plików.
                lea     dx, OldName
                lea     di, NewName
                mov     ah, 56h
                int     21h
                jc      BadRename
                -
                -
                -
OldName         byte    „MYPGM.EXXE”, 0
NewName         byte    „YOURPGM.EXE”,0
```

Przykład numer 2: Przenoszenie nazwy pliku z jednego katalogu do innego:

```
;Zakładamy, że ES i DS. wskazują na bieżący segment danych
; zawiera nazwę pliku
                lea     dx, OldName
                lea     di, NewName
                mov     ah, 56h
                int     21h
                jc      Badrename
                -
                -
                -
OldName         byte    „\DIR1\MYPGM.EXE”, 0
NewName         byte    „\DIR2\MYPGM.EXE”,0
```

13.3.8.12 ZMIANA / POBRANIE ATRYBUTU PLIKU

Funkcja (ah): 43h

Parametry wejściowe: al. – kod podfunkcji
0 – zwracany atrybut pliku w cx
1 – ustawienie atrybutu w cx
cx – nowy atrybut jeśli AL. = 01
ds.:dx – adres ścieżki dostępu

Parametry wyjściowe: Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędu
1 – niepoprawna funkcja
3 – Ścieżka nie znaleziona
5 – Dostęp zastrzeżony
Jeśli przeniesienie jest wyzerowane, a podfunkcja byłą zerem, cx będzie zawierał atrybut pliku.

Funkcja ta jest użyteczna do ustawiania / przestawiania i odczytywania bitów atrybutu pliku. Może być użyta do ustawienia pliku jako tylko do odczytu, ustawienia wyzerowania bitu archiwizacji.

13.3.8.13 POBRANIE/ USTAWIENIE DATY I CZASU MODYFIKACJI PLIKU

Funkcja (ah): 57h

Parametry wejściowe: al. – kod podfunkcji
0 – pobranie daty i czasu
1 – ustawienie daty i czasu
bx – logiczny numer pliku
cx – czas do ustawienia (jeśli AL. = 01)
dx – data do ustawienia (jeśli AL. = 01)

Parametry wyjściowe: Jeśli przeniesienie jest ustawione, ax zawiera jeden z poniższych kodów błędów
1 – Nieprawidłowa funkcja
6 – nieprawidłowy uchwyt
Jeśli przeniesienie jest wyzerowane, cx/dx ustawiają czas/ datę jeśli al. = 00

Funkcja ta ustawia „ostatni zapis" daty / czasu dkla określonego pliku. Plik musi być otwarty (używając open lub create) przed zastosowaniem tej funkcji. Data nie będzie zapisana dopóki plik jest zamknięty.

13.3.8.14 INNE FUNKCJE DOS

Poniższe tablice pokrótce przedstawiają wiele innych funkcji DOS. Po więcej informacji na temat zastosowania tych funkcji zajrzyj do Microsoft MS-DOS Programmer's Reference lub MS-DOS Technical Reference

Funkcja (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
39h	ds.:dx – wskaźnik do ścieżki dostępu zakończonej znakiem zero		Tworzy nowy katalog o określonej nazwie
3Ah	ds.:ds. wskaźnik do ścieżki dostępu zakończonej znakiem zera		Usuwa katalog z określonej ścieżki dostępu. Wystąpi błąd jeśli katalog nie jest pusty lub jest katalogiem bieżącym
3Bh	ds.:dx – wskaźnik do ścieżki dostępu zakończonej znakiem zera		Zmienia domyślny katalog na określony w ścieżce dostępu
45h	bx – logiczny numer pliku	ax – nowy uchwyt	Tworzy kopię logicznego numeru pliku, tak aby program mógł mieć dostęp używając dwóch oddzielnych zmiennych plikowych. Pozwala programowi zamykać plik

			z jednym uchwytem a kontynuować z innym
46h	bx –logiczny numer pliku cx – powtórny uchwyt		Podobnie jak funkcja 45h, z wyjątkiem określenia, który uchwyt (w cx) chcemy odnieść do istniejącego pliku (określony przez bx)
47h	ds.:si – wskaźnik do bufora dl – napęd		Przechowuje łańcuch zawierający bieżącą ścieżkę dostępu (zakończoną zerem) zaczynającą się pod ds.:si. Rejestry te muszą wskazywać na bufor zawierający przynajmniej 64 bajty. Rejestr dl zawiera określony numer napędu (0 = domyślny, 1 =A, 2 =B, 3=C)
5Ah	cx – atrybuty ds.:dx – wskaźnik do ścieżki tymczasowej	ax - uchwyt	Tworzy plik o unikalnej nazwie, w katalogu określonym przez łańcuch zakończony zerem, który jest wskazany przez ds.:dx. Musi być przynajmniej 13 zero bajtowy poza końcem ścieżki dostępu ponieważ funkcja ta będzie przechowywała wygenerowaną nazwę pliku na końcu ścieżki dostępu. Atrybuty są takie same jak dla funkcji Create.
5Bh	cx – atrybuty ds.:dx – wskaźnik do ścieżki zakończonej zerem	ax – uchwyt	Podobna jak funkcja call, ale ta funkcja upiera się, że plik nie istnieje. Zwraca błąd jeśli plik istnieje 9zamaist usunąć stary plik)
67h	bx - uchwyt		Funkcja ustawia maksymalną liczbę uchwytów, których program może użyć w danym czasie
68h	bx - uchwyt		Opróżnia wszystkie dane do pliku bez jego zamykania, zapewniając, że dane pliku są bieżące i spójne

Tablica 56: Różne funkcje plikowe DOS

Funkcja (AH)	Parametry wejściowe	Parametry wyjściowe	Opis
25h	Al. – numer przerwania Ds.:dx – wskaźnik do podprogramu obsługi programu		Przechowuje określony adres w ds.:dx tablicy wektora przerwań, przy wejściu określonym przez rejestr al.
30h		al. – wersja główna ah – wersja pomocnicza bh – znacznik wersji bl:cx – 24 bitowy numer seryjny	Zwraca bieżący numer wersji DOS'a (lub wartość ustawioną przez SETVER)
33h	Ah – 0	dl – znacznik break (0 = off, 1= on)	Zwraca stan znacznika break MS-DOS. Jeśli on, MS-DOS sprawdza Ctrl-C kiedy wykonujemy jakieś polecenie; jeśli off, MS-DOS sprawdza tylko funkcje 1 –0Ch
33h	al. –1 dl – znacznik break		Ustawia znacznik break MS-DOS wedle wartości w dl
33h	al. - 6	bl – wersja główna bh – wersja pomocnicza dl – powtórka dh – znacznik wersji	Zwraca „rzeczywisty" numer wersji, nie tylko ustawioną przez polecenie SETVER. Bity trzy i cztery znacznika wersji są jedynkami jeśli, odpowiednio, DOS jest w ROM lub DOS jest w wysokiej pamięci.
34h		es:bx – wskaźnik do znacznika InDOS	Zwraca adres znacznika InDOS. Znacznik ten pomaga zapobiegać
35h	al. – numer przerwania	es:bx – wskaźnik do podprogramu obsługi przerwań	Zwraca wskaźnik do podprogramu obsługi przerwań dla określonego numeru przerwania.
44h	al. – podkod Inne parametry!		Jest to cała rodzina dodatkowych funkcji DOS dla sterowania różnymi urządzeniami.
4Dh		al. – wartość zwracana ah – metoda zakończenia	Zwraca ostatni kod wynikowy z podprogramu potomnego w al. rejestr ah zawiera metodę zakończenia, która jest jedną z następujących wartości: 0 –normalna, 1 – ctrl-C, 2 – krytyczny błąd urządzenia, 3 – zakończenie i pozostanie w pamięci
50h	bx – adres PSP		Ustawia bieżący DOS'owski adres PSP wartością określoną w rejestrze bx
51h		bx – adres PSP	Zwraca wskaźnik do bieżącego PSP w rejestrze bx

59h		ax- rozszerzony kod błędu bh –klasa błędu bl – reakcja na błąd ch – miejsce błędu	Zwraca dodatkowe informacje kiedy wystąpi błąd w wywołaniu DOS
5Dh	al. –0Ah ds.:si –wskaźnik do rozszerzonej struktury błędu		Kopiuje dane z rozszerzonej struktury błędu do wewnętrznego DOS'owego rekordu

Tablica 57: Różne funkcje DOS

W dodatku do powyższych poleceń, jest kilka dodatkowych poleceń DOS, które działają z Siecią i międzynarodowym zbiorem znaków..

13.3.9 PRZYKŁADY PLIKÓW I/O

Oczywiście , jednym z głównych powodów dla funkcji DOS jest manipulowanie plikami na urządzeniach pamięci masowej. Poniższe przykłady demonstrują kilka zastosowań znaków I/O używających DOS.

13.3.9.1 PRZYKŁAD 1: NARZĘDZIE DO ZRZUTU HEKSADECYMALNEGO

Program ten zrzuca plik w postaci heksadecymalnej. Nazwa pliku musi być ustalona w pliku.

```
                include  stdlib.a
                includelib stdlib.lib
cseg            segment  byte public 'CODE'
                assume cs :cseg, ds.: dseg, es: dseg,  ss: sseg
MainPgm         proc     far
;właściwe ustawienie rejestrów segmentowych:
                mov      ax, seg  dseg
                mov      ds., ax
                mov      es, ax
                mov      ah, 3dh
                mov      al., 0                    ;otwarcie pliku do odczytu
                lea      dx, FileName              ;plik do otwarcia
                int      21h
                jnc      GoodOpen
                print
                byte     'Nie można otworzyć pliku., program przerwany...', cr, 0
                jmp      PgmExit

GoodOpen:       mov      FileHandle, ax            ;zachowanie uchwytu plik u
                mov      Position, 0               ;inicjalizacja pozycji licznika pliku
ReadFileLp:     mov      al., byte  ptr  Position
                and      al., 0Fh                  ;obliczanie (Position MOD 16)
                jnz      NotNewLine                ;początek nowej linie każdych 16 bajtów
                putcr
                mov      ax, Position
                xchg     al., ah
                puth
                xchg     al., ah
                puth
                print
                byte     ': ', 0

NotNewLine:     inc      Position                  ;zwiększenie licznika znaków
                mov      bx, FileHandle
                mov      cx, 1                     ;odczyt 1 bajtu
                lea      dx, buffer                ;miejsce dla przechowania tego bajtu
```

```
                    mov     ah, 3Fh                 ;operacja odczytu
                    int     21h
                    jc      BadRead
                    cmp     ax, 1                   ;osiągniecie EOF?
                    jnz     AtEOF
                    mov     al., Buffer             ;pobranie odczytanego znaku
                    puth                            ;i wydruk w hex
                    mov     al., ' '                ;wydruk spacji między wartościami
                    putc
                    jmp     ReadFileLp

BadRead:            print
                    byte    cr, lf
                    byte    'Błąd odczytu danych z pliku, przerwanie'
                    byte    cr, lf, 0
AtEOF:              mov     bx, FileHandle          ;zamknięcie pliku
                    mov     ah, 3Eh
                    int     21h

PgmExit:            ExitPgm
MainPgm:            endp

cseg                ends
dseg                segment  byte  public 'data'

Filename            byte    'hexdump.asm', 0        ;nazwa pliku do zrzutu
FileHandle          word    ?
Buffer              byte    ?
Position            word    0

dseg                ends

sseg                segment  byte stack 'stack'
stk                 word    0ffh  dup (?)
sseg                ends
zzzzzzseg           segment  para public 'zzzzzz'
LastBytes           byte    16  dup (?)
Zzzzzzseg           ends
                    End     MainPgm
```

13.3.9.2 PRZYKŁAD 2: KONWERSJA NA DUŻE LITERY

Poniższy program odczytuje jeden plik, konwertuje wszystkie małe litery na duże , i zapisuje dane do innego pliku wyjściowego.

```
                    include  stdlib.a
                    includelib stdlib.lib
cseg        segment  byte public 'CODE'
            assume   cs: cseg, ds : dseg, es: dseg, ss: sseg

MainPgm     proc     far
;właściwe ustawienie rejestrów segmentowych      :
            mov      ax, seg  dseg
            mov      ds., ax
            mov      es, ax
;-------------------------------------------------------------------------------------------------------------------------
;
;       Konwersja  UCCONVERT.ASM na duże litery
;
;       Otwarcie pliku wejściowego:
            mov      ah, 3dh
```

```
                mov     al., 0                          ;otwarcie pliku do odczytu
                lea     dx, Filename                    ;Plik do otwarcia
                int     21h
                jnc     GoodOpen
                print
                byte    „Nie można otworzyć pliku, przerwanie programu...', cr, lf, 0
                jmp     PgmExit

GoodOpen:       mov     FileHandle1, ax                 ;zachowanie uchwytu pliku wejściowego

;Otwarcie pliku wyjściowego:
                mov     ah, 3Ch                         ;funkcja tworzenia pliku
                mov     cx, 0                           ;normalny atrybut
                lea     dx, OutFileName                 ;plik do otwarcia
                int     21h
                jnc     GoodOpen2
                print
                byte    'Nie można otworzyć pliku wyjściowego , przerwanie programu...'
                byte    cr, lf, 0
                mov     ah, 3Eh                         ;zamknięcie pliku wejściowego
                mov     bx,  FileHandle1
                int     21h
                jmp     PgmExit                         ;ignoruj błąd

GoodOpen2:      mov     FileHandle2, ax                 ;zachowanie uchwytu pliku wyjściowego

ReadFileLp:     mov     bx, FileHandle1
                mov     cx, 1                           ;odczyt jednego bajtu
                lea     dx, buffer                      ;miejsce do przechowania tego bajtu
                mov     ah, 3Fh                         ;operacja odczytu
                int     21h
                jc      BadRead
                cmp     ax, 1                           ;osiągnięcie EOF?
                jz      ReadOK
                jmp     AtEOF

ReadOK:         mov     al., Buffer                     ;pobranie odczytanego znaku i
                cmp     al., 'a'                        ;konwersja do znaku dużego
                jb      NotLower
                cmp     al., 'z'
                ja      NotLower
                and     al., 5Fh                        ;ustawienie bitu 5 na zero
NotLower:       mov     Buffer, al.
;teraz zapisujemy dane do pliku wyjściowego
                mov     bx, FileHandle2
                mov     cx, 1                           ;odczyt jednego bajtu
                lea     dx, buffer                      ;miejsce do przechowania tego bajtu
                mov     ah, 40h                         ;operacja zapisu
                int     21h
                jc      BadWrite
                cmp     ax, 1                           ;upewnienie czy dysk nie jest pełny
                jz      ReadFileLp

BadWrite        print
                byte    cr, lf
                byte    'Błąd zapisu danych do pliku, operacja przerwana'
                byte    cr, lf, 0
                jmp     short  AtEOF
BadRead:        print
                byte    cr, lf
```

```
                byte    'Błąd odczytu danych z pliku, przerwanie '
                byte    'operacji', cr, lf, 0
AtEOF:          mov     bx, FileHAndle1         ;zamknięcie pliku
                mov     ah, 3Eh
                int     21h
                mov     bx, FileHandle2
                mov     ah, 3Eh
                int     21h
;-------------------------------------------------------------------------------------------------------------------------------

PgmExit:        ExitPgm
MainPgm         endp
cseg            ends

dseg            segment  byte  public 'data'

Filename        byte    'unconvrt.asm', 0       ;nazwa pliku do konwersji
OutFileName     byte    'output.txt', 0         ;nazwa pliku wyjściowego
FileHandle1     word    ?
FielHandle2     word    ?
Buffer          byte    ?
Position        word    0

dseg            ends

sseg            segment  byte  stack 'stack'
stk             word    offh  dup (?)
sseg            ends

zzzzzzseg       segment  para public 'zzzzzz'
LastBytes       byte    16 dup (?)
Zzzzzzseg       ends
                end     MainPgm
```

13.3.10 ZBLOKOWANE PLIKI I/O

Przykłady w poprzedniej sekcji cierpiały z powodu jednej wady, były zbyt wolne. Problem wydajności w powyższym kodzie spowodowane są wyłącznie DOS'em. Czyniąc wywołania DOS powinniśmy wiedzieć, że nie są one najszybszymi operacjami na świecie. Wywołując DOS za każdym razem kiedy chcemy odczytać lub zapisać pojedynczy znak z / do pliku, będziemy rzucać system na kolana. Jak się okazuje, nie robi zabiera (praktycznie) więcej czasu DOS'owi odczyt lub zapis dwóch znaków niż odczyt lub zapis jednego znaku. Ponieważ ilość czasu jaką (zazwyczaj) spędzamy przetwarzając dane jest nieistotna w porównaniu z ilością czasu jaki DOS pobiera do zwrotu lub zapisu danych, odczyt dwóch znaków po kolei, w gruncie rzeczy podwoi prędkość programu. Jeśli odczyt dwóch znaków podwaja szybkość przetwarzania, jak odczytać cztery znaki? Z pewnością czterokrotnie zwiększy prędkość przetwarzania. Podobnie przetwarzanie dziesięciu znaków po kolei prawie zwiększa prędkość przetwarzania według kolejności wag. Niestety, ten postęp nie będzie trwał wiecznie. Nadchodzi kiedyś punkt zmniejszania, kiedy bierzemy stanowczo za dużo pamięci dla uzasadnienia (bardzo) małej poprawy wydajności (zapamiętajmy, że odczyt 64K w pojedynczej operacji wymaga 64K bufora pamięci dla przechowania danych). Dobrym kompromisem jest 256 lub 512 bajtów. Odczyt większej ilości danych w rzeczywistości nie poprawia wiele wydajności, ale mimo to bufor 256 bajtowy lub 52\12 bajtowy jest łatwiejszy w postępowaniu niż bufory większe.

Odczyt danych w grupach lub blokach jest nazywany zblokowanym I/O . Zblokowany I/O jest często rzędu jeden do dwóch razy szybsze niż pojedynczy znak I/O, więc oczywiście powinniśmy używać zblokowanych I/O gdziekolwiek to możliwe.

Jest jedna pomniejsza wada zblokowanych I/O – jest trochę bardziej złożone do oprogramowania niż pojedynczy znak I/O. Rozważmy przykład przedstawiony w sekcji o poleceniu DOS'a Read:

Przykład: Ten przykład otwiera plik i odczytuje go do EOF

```
                mov     ah,3dh                  ;otwarcie pliku
                mov     al., 0                  ;otwarcie do odczytu
```

```
                lea     dx, Filename            ;zakładamy, że wskazuje DS.
Filename:
                int     21h                     ;segment
                jc      BadOpen
                mov     FHndl, ax               ;zachowanie uchwytu pliku
LP:             mov     ah, 3Fh                 ;odczyt danych z pliku
                lea     dx, Buffer              ;adres bufora danych
                mov     cx, 1                   ;odczyt jednego bajtu
                mov     bx, FHndl               ;pobranie wartości uchwytu pliku
                int     21h
                jc      ReadError
                cmp     ax, cx                  ;osiągnięte EOF?
                jne     EOF
                mov     al., Buffer             ;pobranie odczytanego znaku
                putc                            ;wydruk go (wywołanie IOSHELL)
                jmp     LP                      ;odczyt następnego bajtu
EOF:            mov     bx, FHndl
                mov     ah, 3Eh                 ;zamknięcie pliku
                int     21h
                jc      CloseError
```

Teraz przepiszmy ten program przy użyciu zblokowanych I/O:

Przykład: Ten przykład otwiera plik i odczytuje go do EOF używając zblokowanych I/O

```
                mov     ah, 3dh                 ;otwórz plik
                mov     al., 0                  ;otwarcie do odczytu
                lea     dx, Filename            ;zakładamy, że wskazuje DS.
Filename
                int     21h                     ;segment
                jc      BadOpen
                mov     FHndl, ax               ;zachowanie uchwytu pliku

LP:             mov     ah, 3fh                 ;odczyt danych z pliku
                lea     dx, Buffer              ;adres bufora danych
                mov     cx, 256                 ;odczyt 256 bajtów
                mov     bx, FHndl               ;pobranie wartości uchwytu pliku
                int     21h
                jc      ReadError
                cmp     ax, cx                  ;osiągnięte EOF?
                jne     EOF
                mov     si, 0                   ;CX =256 w tym punkcie
PrtLP:          mov     al., Buffer[si]         ;pobranie odczytanego znaku
                putc                            ;jego wydruk
                inc     si
                loop    PrtLp
                jmp     LP                      ;odczyt następnego bloku
EOF:            mov     cx, ax
                jcxz    EOF2                    ;jeśli CX = 0 , rzeczywiście robimy
                mov     si, 0                   ;przetwarzanie ostatniego bloku danych
Finis:          mov     al., Buffer [si]        ;odczytanych z pliku, który zawiera
                putc                            ;1..255 poprawnych danych
                inc     si
                loop    Finis
EOF2:           mov     bx, FHndl
                mov     ah, 3eh                 ;zamknięcie pliku
                int     21h
                jc      CloseError
```

Przykład ten demonstruje jeden główny kłopot ze zblokowanymi I/O – kiedy osiągamy koniec pliku, nie musimy koniecznie przetworzyć wszystkie dane w pliku. Jeśli rozmiar bloku ma 256 a jest 255 bajtów opuszczonych w pliku, DOS będzie zwracała warunek EOF (liczba bajtów odczytanych nie pasuje do

zgłoszenia). W tym przypadku, musimy jeszcze przetwarzać znaki, które zostały odczytane. Powyższy kod robi to raczej w prosty sposób, używając drugiej pętli, kończącą kiedy osiągnięty zostanie EOF. Zauważyliśmy, że dwie pętle print są praktycznie identyczne. Program ten może być zredukowany rozmiarowo, przez zastosowanie kodu, który jest trochę bardziej złożony:

Przykład: ten przykład otwiera plik i odczytuje go do EOF używając zblokowanych I/O

```
                mov     ah, 3dh                 ;otwarcie pliku
                mov     al., 0                  ;otwarcie do odczytu
                lea     dx, FileName            ;zakładamy, że DS. wskazuje
FileName
                int     21h                     ;segment
                jc      BadOpen
                mov     FHndl, ax               ;zachowanie uchwytu pliku

LP:             mov     ah, 3fh                 ;odczyt danej z pliku
                lea     dx, Buffer              ;adres bufora danych
                mov     cx,  256                ;odczyt 256 bajtów
                mov     bx, FHndl               ;pobranie wartości uchwytu pliku
                int     21h
                jc      ReadError
                mov     bx, ax                  ;zachowanie na później
                mov     cx, ax
                jcxz    EOF
                mov     si, 0                   ;CX=256 w tym punkcie
PrtLp;          mov     al., Buffer [si]        ;pobranie odczytanego znaku
                putc                            ;jego wydruk
                inc     si
                loop    PrtLp
                cmp     bx, 256                 ;osiągnięte EOF?
                je      LP

EOF:            mov     bx, FHndl
                mov     ah, 3eh                 ;zamknięcie pliku
                int     21h
                jc      CloseError
```

Zblokowane I/O pracują najlepiej na plikach sekwencyjnych. To znaczy, plik te są otwierane tylko do odczytu lub zapisu (żadnego pozycjonowania). Kiedy działamy na plikach o dostępie swobodnym, powinniśmy odczytywać lub zapisywać cały rekord używając polecenie odczyt / zapis DOS'a do działania na całym rekordzie. Jest to znacznie szybsze niż manipulowanie danymi jedno bajtowymi.

13.3.11 PRZEDROSTEK SEGMENTU PROGRAMU (PSP)

Kiedy program jest ładowany do pamięci dla wykonania, DOS buduje najpierw przedrostek segmentu programu bezpośrednio przed programem, który jest ładowany do pamięci. PSP zawiera wiele informacji, jedne są użyteczne, inne przestarzałe. Zrozumienie rozkładu PSP jest niezbędne dla programistów piszących programy w assemblerze

PSP ma długość 256 bajtów i zawiera następujące informacje:

Offset	Długość	Opis
0	2	Tu jest przechowywana instrukcja 20h
2	2	Pułap pamięci programu
4	1	Nieużywane, zarezerwowane przez DOS
5	5	Wywołanie funkcji DOS obsługującej funkcje systemowe
0Ah	4	Adres procedury obsługi przerwania 22h
0Eh	4	Adres procedury obsługi przerwania 23h
12h	4	Adres procedury obsługi przerwania 24h
16h	22	Zarezerwowane przez DOS
2Ch	2	Adres segmentowy środowiska systemowego
2Eh	34	Zarezerwowane przez DOS
50h	3	INT 21h, instrukcja RETF

53h	9	Zarezerwowane przez DOS
5Ch	16	Domyślny FCB 1
6Ch	20	Domyślny FCB 2
80h	1	Długość łańcucha lini poleceń
81h	127	Łańcuch lini poleceń

Notka: lokacje 80h..FFh są używane przez domyślny DTA.

Większość informacji w PSP jest używana w nowoczesnych programach asemblerowych MS-DOS. Ukryte w PSP są jednak warte wiedzy. My jednak będziemy się przyglądać wszystkim polom w PSP.

Pierwsze pole w PSP zawiera instrukcję int 20h. Int 20h jest przestarzałym mechanizmem używanym do zakańczania wykonywania programu. We wcześniejszych dniach DOS v. 1.0, program wykonywał jmp do tej lokacji żeby zakończyć. Obecnie oczywiście, mamy funkcję DOS 4Ch, która jest dużo łatwiejsza (i bezpieczniejsza) niż skakanie do lokacji zero w PSP. Dlatego to pole jest przestarzałe.

Pole numer dwa zawiera wartość, która wskazuje ostatni paragraf alokowany w naszym programie. Przez odjęcie adresu PSP od tej wartości, możemy określić ilość pamięci alokowanej dla naszego programu (i opuścić jeśli jest dostępna niewystarczająca ilość pamięci)

Trzecie pole jest pierwszą z wielu „dziur" pozostawionych przez Microsoft w PSP. Niech zgadnie ktoś dlaczego są one tu.

Czwarte pole jest to wywołanie funkcji DOS obsługującej funkcje systemowe. Celem tego (teraz przestarzałego) mechanizmu wywołania DOS było zezwolenie na dodatkową kompatybilność z programami CP/M.-80. W nowoczesnych programach DOS'a nie musimy się martwić o to pole.

Następne trzy pola są używane do przechowywania specjalnych adresów podczas wykonywania programów. Pola te zawierają domyślny wektor zakończenia, wektor przerwania i wektor krytycznych błędów. Są to wartości normalnie przechowywane w wektorach przerwań dla int 22h, int 23h i int 24h. Poprzez przechowywanie kopii tych wartości w wektorach dla tych przerwań, możemy zmienić te wektory aby wskazywały na nasz własny kod.. Kiedy program się kończy, DOS przywraca te trzy wektory z tych trzech pól w PSP.

Ósme pole w PSP jest innym zarezerwowanym polem, aktualnie niedostępnym do użytkowania przez programy.

Pole dziewiąte jest innym prawdziwym skarbem. Jest to adres segmentowy środowiska systemowego. Jest to dwu bajtowy wskaźnik, który zawiera adres segmentu obszaru pamięci środowiskowej. Łańcuchy środowiska zawsze zaczynają się od offsetu zero wewnątrz segmentu. Obszar łańcuchów środowiska składa się z sekwencji łańcuchów zakończonych zerem. Używa następującego formatu:

$string_1$ 0, $string_2$ 0, $string_3$ 0....0, $string_n$ 0 0

To znaczy , obszar środowiska składa się z listy łańcuchów zakończonych zerem , sama lista kończy się łańcuchem o długości zero (tj same zera w nim , albo dwa zera w wierszu). Łańcuchy są (zazwyczaj) umieszczone w obszarze środowiska poprzez polecenia DOS takie jak PATH, SET, itp. Ogólnie łańcuch przybiera w obszarze środowiska postać:

nazwa = parametry

Na przykład polecenie „SET IPATH = C:\ASSEMBLY\INCLUDE"kopiuje łańcuch „IPATH=C:\ASSEMBLY\INCLUDE" do obszaru pamięci środowiska systemowego.

Wiele języków przeszukuje obszar pamięci środowiska aby znaleźć domyślną nazwę pliku w ścieżce dostępu i inne fragmenty domyślnej informacji ustawionej przez DOS.. Nasze programy mogą również to wykorzystywać.

Następne pole w PSP jest innym blokiem zarezerwowanej pamięci, obecnie nie zdefiniowanej przez DOS.

Pole 11 w PSP jest inną procedurą DOS'a obsługującą funkcje systemowe. Dlaczego istnieje ta funkcja (kiedy pod lokacją 5 w PSP już istnieje i nikt rzeczywiście nie używa innego mechanizmu wywołania DOS) jest interesującym pytaniem. To pole powinno być ignorowane przez nasze programy.

Pole 12 jest innym blokiem nie używanych bajtów w PSP, który powinien być ignorowany.

Pola 13 i 14 w PSP są domyślnymi FCB'ami (Blok Kontrolny Pliku –FCB). Bloki kontrolne pliku są inną archaiczną strukturą danych przeniesiona z CP/M.-80. Są używane tylko wtedy, kiedy używamy przestarzałych podprogramów obsługi plików DOS'a 1.0 , więc są nieco interesujące dla nas. Ignorujemy jednak FCB'y w PSP.

Lokacja 80h na końcu PSP zawiera bardzo ważną część informacji – wpisane parametry lini poleceń DOS wraz z nazwą programu. Jeśli w lini poleceń jest wpisane:

MYPGM parametr1, parametr 2

W polu parametrów lini poleceń przechowujemy:

23, „parametr1, parametr2", 0Dh

Lokacja 80h zawiera 23_{10} ,długość parametrów następujących po nazwie programu. Lokacja 81h do 97h zawierają znaki stanowiące łańcuch parametrów . Lokacja 98h zawiera powrót karetki. Zauważmy, że znak powrotu karetki nie figuruje w długości łańcucha lini poleceń.

Przetwarzanie łańcucha lini poleceń jest takim ważnym aspektem programowania języku asemblera, że proces ten będzie omówiony w następnej sekcji.

Lokacje 80h..FFh w PSP również stanowią domyślny DTA. Dlatego też nie używajmy funkcji DOS 1Ah do zmiany DTA i wykonujmy FIND FIRST FILE, informacja o nazwie pliku będzie przechowana w początkowej lokacji 80h w PSP.

Ważnym szczegółem jaki pominęliśmy jest to jak dokładnie uzyskujemy dostęp do danych w PSP. Chociaż PSP jest ładowane do pamięci bezpośrednio przed programem, nie znaczy to ,ze pojawia się koniecznie 100h bajtów przed kodem. Nasz segment danych może być załadowany przed segmentem kodu, tym samym obala metodę umiejscawiania PSP. Adres segmentu PSP jest przekazany do programu w rejestrze ds. Przechowując adres PSP w segmencie danych , nasz program powinien zaczynać się od takiego kodu:

```
push    ds.             ;zachowanie wartości PSP
mov     ax, seg DSEG    ;DS. i ES wskazują na nasz segment
mov     ds., ax         ;danych
mov     es, ax
pop     PSP             ;przechowanie wartości PSP w zmiennej
-                       ;PSP
-
-
```

Innym sposobem uzyskania adresu PSP w DOS 5.0 i późniejszych, jest wykonanie wywołania DOS'a. Jeśli załadujemy do ah 51h i wykonamy instrukcję int 21h, MS-DOS zwróci adres segmentu aktualnego PSP w rejestrze bx.

Jest wiele sztuczek umożliwiających pracę z danymi w PSP. Peter Norton;s Progerammer's Guide dla IBM PC wylicza wszystkie rodzaje sztuczek. Takie działania nie będą tu omawiane ponieważ wychodzą one poza zakres tego materiału

---

13.3.12 DOSTĘP DO PARAMETRÓW LINI POLECEŃ

Większość programów takich jak MASM i LINK pozwala nam określić parametry lini poleceń, kiedy program jest wykonywany. Na przykład pisząc

ML MYPGM.ASM

Możemy poinstruować MASM aby zasemblował MYPGM bez żadnych dalszych interwencji z klawiatury. „MYPGM.ASM" jest dobrym przykładem parametru lini poleceń

Kiedy DOS'owskie polecenie COMMAND.COM interpretuje analizę lini poleceń, kopiuje większość następującego tekstu nazwy programu do lokacji 80h w PSP jak opisano w poprzedniej sekcji. Na przykład powyższa linia poleceń będzie przechowana w PSP:80h tak

11, 'MYPGM.ASM", 0

Przechowywany tekst w obszarze pamięci lini poleceń w PSP jest zazwyczaj dokładną kopią danej pojawiającej się w lini poleceń. Jest jednak parę wyjątków. Przede wszystkim parametry przekierowania I/O nie są przechowywane w buforze wejściowym. Inną sprawą pojawiającą się w lini poleceń, która jest nieobecna w danych pod PSP:80h jest nazwa programu. Jest to raczej nieszczęśliwe, ponieważ mając dostępną nazwę programu, możemy określić katalog zawierający program. Pomimo to, jest dużo użytecznych informacji przedstawionych w lini poleceń.

Informacje w lini poleceń mogą być użyte do prawie każdego celu jak uznamy za stosowny. Jednakże, większość programów, oczekuje dwóch typów parametrów w buforze parametrów lini poleceń – nazwy pliku i przełączników. Cel nazwy pliku jest raczej oczywisty, pozwala programowi uzyskać dostęp do pliku. Przełączniki, z drugiej strony, są dowolnymi parametrami programu. Przez konwencję, przełączniki są poprzedzone ukośnikiem lub łącznikiem w lini poleceń. Zastanówmy się co robić z informacjami w lini poleceń zwanymi analizą składniową lini poleceń. Jeśli programy manipulują danymi w lini poleceń, musimy zanalizować linię poleceń wewnątrz kodu.

Zanim linia poleceń zostanie zanalizowana, każda pozycja lini poleceń musi być oddzielona od innych. To znaczy, każde słowo (lub bardziej poprawnie – leksem) musi być zidentyfikowane w lini poleceń. Oddzielenie leksemów w lini poleceń jest stosunkowo łatwe, wszystko co musimy zrobić to wyszukać separatory ograniczające w lini poleceń. Separatory są specjalnymi symbolami używanymi do oddzielania znaczników w lini poleceń. DOS wspiera sześć różnych znaków ograniczników: spacja, przecinek, średnik, znak równości, tabulator lub powrót karetki.

Generalnie każda liczba znaków ograniczających może pojawić się pomiędzy dwoma znacznikami w lini poleceń. Dlatego też, wszystkie takie wystąpienia muszą być pominięte, kiesy przeszukujemy linie poleceń. Poniższy kod asemblerowy przeszukuje całą linie poleceń i drukuje wszystkie znaczniki tam się pojawiające:

```
                include  stdlib.a
                includelib stdlib.lib
cseg            segment  byte  public  'CODE'
                assume  cs: cseg, ds :dseg, es : dseg,  ss: sseg
;Przyrównywanie w lini poleceń
CmdLnLen        equ     byte   ptr  es: [80h]          ;długość lini poleceń
CmdLn           equ     byte   ptr  es:[81h]           ;dane lini poleceń

Tab             equ     09

MainPgm         proc    far
;właściwe ustawienie rejestrów segmentowych:
                push    ds.                            ;zachowanie PSP
                mov     ax, seg  dseg
                mov     ds., ax
                pop     PSP
;-------------------------------------------------------------------------------------------------------------------------------

                print
                byte    cr, lf
                byte    'Pozycja w tej lini:', cr, lf, lf, 0

                mov     es, PSP                        ;ES wskazuje PSP
                lea     bx, CmdLn                      ;wskazuje linię poleceń
PrintLoop:      print
                byte    cr, lf, 'Pozycja: ', 0
                call    SkipDelimiters                 ;przeskoczenie głównych ograniczników
PrtLoop2:       mov     al., es:[bx]                   ;pobranie następnego znaku
                call    TestDelimiter                  ;czy jest ogranicznik?
                jz      EndofToken                     ;wyjście z pętli jeśli jest
                putc                                   ;wydruk znaku jeśli nie
                inc     bx                             ;przesunięcie na następny znak
                jmp     PrtLoop2

EndofToken:     cmp     al., cr                        ;powrót karetki?
                jne     PrintLoop                      ;powtórz jeśli nie koniec lini

                print
                byte    cr, lf, lf
                byte    'Koniec lini poleceń', cr, lf,lf ,0
                ExitPgm

MainPgm         endp
;Poniższy podprogram ustawia flagę zera jeśli znak w rejestrze AL. to jeden z sześciu ograniczników
;DOS'owych, w przeciwnym razie flaga zera jest zerowana. Pozwala to nam  później na użycie instrukcji
;JE/JNE do testowania ograniczników

TestDelimiter   proc    near
                cmp     al., ' '
                jz      ItsOne
                cmp     al., ' , '
                jz      ItsOne
                cmp     al., Tab
                jz      ItsOne
                cmp     al., ' ; '
                jz      ItsOne
                cmp     al., ' = '
```

```
                    jz      ItsOne
                    cmp     al., cr
ItsOne:             ret
TestDelimiter       endp

;SkipDelimiters przeskakuje główne ograniczniki w lini poleceń. Nie przeskakuje jednak powrotu karetki i
;końca lini ponieważ znak ten jest używany jako kończący program główny

SkipDelimiters      proc    near
                    dec     bx                  ;Offset BX poniżej
SDLoop:             inc     bx                  ;przesunięcie na następny znak
                    mov     al., es;[bx]        ;pobranie następnego znaku
                    cmp     al., 0dh            ;nie przeskakuj jeśli CR
                    jz      QuitSD
                    call    TestDelimiter       ;zobacz czy jest jakiś inny
                    jz      SDLoop              ;ogranicznik i powtórz
QuitSD:             ret
SkipDelimiters      endp
cseg                ends
dseg                segment  byte  public ‘data’
PSP                 word    ?                   ;przedrostek segmentu programu
dseg                ends
sseg                segment  byte  stack  ‘stack’
stk                 word    0ffh  dup  (?)
sseg                ends

zzzzzzseg           segment  para public ‘zzzzzz’
LastBytes           byte    16   dup (?)
Zzzzzzseg           ends
                    End     MainPgm
```

Ponieważ możemy przeszukać linie poleceń (to znaczy, oddzielić leksemy), następnym krokiem jest jego analiza składniowa. Dla większości programów analiza składniowa lini poleceń jest nadzwyczajnie trywialnym działaniem. Jeśli pogram akceptuje tylko pojedynczą nazwę pliku, wszystko co musimy zrobić jest uchwycenie pierwszego leksema w lini poleceń, dodanie zerowego bajta na jego koniec (być może przesuniecie go w segmencie danych) i użycie jako nazwy pliku. Poniższy program asemblerowy modyfikuje podprogram zrzutu hex prezentowanego wcześniej, żeby pobrać jego nazwę pliku z lini poleceń zamiast ustalenia nazwy pliku w programie

```
                    include     stdlib.a
                    includelib  stdlib.lib
cseg                segment     byte  public  ‘CODE’
                    assume      cs: cseg, ds.: dseg, es : dseg, ss :sseg
;zanotuj, że CR i LF są już zdefiniowane w STDLIB.A

tab                 equ     09h
MainPgm             proc    far
;właściwe ustawienie rejestrów segmentowych:
                    mov     ax, seg   dseg
                    mov     es, ax                  ;DS. wskazuje na PSP
;-------------------------------------------------------------------------------------------------------------------------------
;
; Najpierw analizujemy linie poleceń aby pobrać nazwę pliku:
                    mov     si, 81h                 ;wskazuje na linię poleceń
                    lea     di, FileName            ;wskazuje bufor FileName
SkipDelimiters:
                    lodsb                           ;pobranie następnego znaku
                    call    TestDelimiter
                    je      SkipDelimiters
; Zakładamy, że to co nastąpi jest  faktyczną nazwą pliku
```

```
                dec     si                              ;wskazuje pierwszy znak nazwy
GetFName:       lodsb
                cmp     al., 0dh
                je      GotName
                call    TestDelimiter
                je      GotName
                stosb                                   ;zachowanie znaku nazwy pliku
                jmp     GetFName
;Jesteśmy na końcu pliku, wiec jest wymagane przez DOS zakończenie zerem.

GotName:        mov     byte  ptr  es:[di], 0
                mov     ax, es
                mov     ds., ax
;teraz działamy na pliku
                mov     ah, 3dh
                mov     al., 0                          ;otwarcie pliku do odczytu
                lea     dx, FileName                    ;Plik do otwarcia
                int     21h
                jnc     GoodOpen
                print
                byte    „Nie można otworzyć  pliku...program przerwany...', cr, 0
                jmp     PgmExit
GoodOpen:       mov     FileHandle, ax                  ;zachowanie uchwytu pliku
                mov     Position, 0                     ;inicjalizacja pozycji pliku
ReadFileLp:     mov     al., byte  ptr Position
                and     al., 0Fh                        ;obliczanie (Position MOD 16)
                jnz     NotNewLn                        ;każde 16 bajtów zaczyna się w lini
                putcr
                mov     ax, Position                    ;wydruk offsetu pliku
                xchg    al., ah
                puth
                xchg    al., ah
                puth
                print
                byte    ' : ',0

NotNewLn:       inc     Position                        ;zwiększenie licznika znaków
                mov     bx, FileHandle
                mov     cx, 1                           ;odczyt jednego bajta
                lea     dx, buffer                      ;miejsce do przechowywania tego bajta
                mov     ah, 3Fh                         ;operacja odczytu
                int     21h
                jc      BadRead
                cmp     ax, 1                           ;osiągnięty EOF?
                jnz     AtEOF
                mov     al., Buffer                     ;pobranie odczytanego znaku i
                puth                                    ;jego wydruk w hex
                mov     al., ' '                        ;wydruk spacji między wartościami
                putc
                jmp     ReadFileLp

BadRead:        print
                byte    cr, lf
                byte    'Błąd odczytu danej z pliku, przerwanie'
                byte    cr, lf, 0

AtEOF:          mov     bx, FileHandle                  ;zamknięcie pliku
                mov     ah,  3Eh
                int     21h
;-------------------------------------------------------------------------------------------------------------------------------
```

```
PgmExit:        ExitPgm
MainPgm         endp

TestDelimiter   proc    near
                cmp     al., ‘ ‘
                je      xit
                cmp     al., ‘ , ‘
                je      xit
                cmp     al., Tab
                je      xit
                cmp     al., ‘ ; ‘
                je      xit
                cmp     al., ‘ = ‘
xit:            ret
TestDelimiter   endp
cseg            ends

dseg            segment  byte  public ‘data’

PSP             word    ?
Filename        byte    64  dup (?)             ;plik do zrzutu
FileHandle      word    ?
Buffer          byte    ?
Position        word    0

dseg            ends

sseg            segment byte  stack  ‘stack’
stk     word    0ffh  dup (?)
sseg            ends

zzzzzzseg       segment  para public ‘zzzzzz’
LastBytes       byte    16  dup (?)
zzzzzzseg       ends
                end     MainPgm
```

Poniższy przykład demonstruje kilka koncepcji działania z parametrami lini poleceń. Program ten kopiuje jeden plik do innego.. jeśli przełącznik „/U” jest (gdzieś) w lini poleceń włączony, wszystkie małe litery w pliku są konwertowane na duże przed zapisaniem do pliku przeznaczenia. Inna cecha tego kodu to to ,że zachęca użytkownika do wprowadzenia zaginionej nazwy pliku, podobnie jak programy MASM i LINK zachęcają nas do wprowadzenia nazwy pliku jeśli nie dostarczyliśmy żadnej.

```
                include     stdlib.a
                includelib  stdlib.lib
cseg            segment  byte  public  ‘CODE’
                assume  cs: cseg, ds: nothing, es: dseg, ss: sseg
;Notka: Stałe CR (0dh) i LF (0ah) pojawiają się wewnątrz  pliku stdlib.a

tab             equ     09h

MainPgm         proc    far
;właściwe ustawienie rejestrów segmentowych:
                mov     ax, seg  dseg
                mov     es, ax                  ;DS. wskazuje na PSP
;-------------------------------------------------------------------------------------------------------------------------
;Najpierw analizujemy linię poleceń aby pobrać nazwę pliku:
                mov     es: GotName1, 0         ;inicjalizacja flag, które mówią nam czy
                mov     es: GotName2, 0         ;analizujemy nazwę pliku
                mov     es: ConvertLc, 0        ;i włączony jest przełącznik „/U”
;Okay, zaczynamy przeszukiwanie i analizowanie lini poleceń
                mov     si, 81h                 ;wskazuje linię poleceń
```

```
SkipDelimiters:
                    lodsb                               ;pobranie następnego znaku
                    call    TestDelimiter
                    je      SkipDelimiters
;Określenie czy to jest nazwa pliku lub przełącznik /U
                    cmp     al., '/ '
                    jnz     MustBeFN
;zobacz czy jest tu „/U"
                    lodsb
                    and     al., 5fh                    ;konwersja „u" do „U"
                    cmp     al., 'U'
                    jnz     NotGoodSwitch
                    lodsb                               ;upewnij się, że następny znak
                    cmp     al., cr                     ;jest ograniczony jakoś
                    jz      GoodSwitch
                    call    TestDelimiter
                    jne     NotGoodSwitch
;Okay, tu jest "/U'
GoodSwitch:         mov     es: ConvertLC , 1           ;konwersja LC na UC
                    dec     si
                    jmp     SkipDelimiters              ;przesunięcie na następną pozycję
;jeśli został znaleziony zły przełącznik w lini poleceń, wydruk błędu i przerwanie
NotGoodSwitch:
                    print
                    byte    cr, lf
                    byte    'Nieprawidłowy przełącznik, dozwolony jest tylko /U!', cr, lf
                    byte    'Przerwane wykonywanie programu', cr. lf, 0
                    jmp     PgmExit
;jeśli to nie jest przełącznik zakładamy, że jest to poprawna nazwa pliku i dany tu uchwyt
MustBeFN:           cmp     al., cr                     ;zobacz czy koniec lini cmd
                    je      EndOfCmdLn
;Zobacz czy jest jedna, dwie lub więcej nazw plików , które zostały określone
                    cmp     es: GotName1, 0
                    jz      Is1stName
                    cmp     es: GotName2
                    jz      Is2ndName
;więcej niż dwa wprowadzone pliki, wydruk błędu i przerwanie
                    print
                    byte    cr, lf
                    byte    'Zbyt wiele określono nazw plików.', cr, lf
                    byte    'Program przerwany...', cr, lf, lf, 0
                    jmp     PgmExit
;skok tu jeśli to jest pierwsza przetwarzana nazwa pliku
Is1stName:          lea     dl, FileName1
                    mov     es: GotName1, 1
                    jmp     ProcessName
Is2ndName:          lea     dl, FileName2
                    mov     es: GotName2, 1
ProcessName:
                    stosb                               ;przechowanie znaku z nazwy
                    lodsb                               ;pobranie kolejnego znaku z lini cmd
                    cmp     al., cr
                    je      NameIsDone
                    call    TestDelimiter
                    jne     ProcessName

NameIsDone:         mov     al., 0                      ;nazwa pliku zakończona zerem
                    stosb
                    dec     si                          ;wskazuje na poprzedni znak
                    jmp     SkipDelimiters              ;spróbuj  ponownie
```

```
;kiedy osiągnięto koniec lini poleceń, zobacz czy obie nazwy pliku zostały określone
                        assume      ds.: dseg

EndOfCmdLn:             mov     ax, es              ;DS. wskazuje DSEG
                        mov     ds., ax
;Jesteśmy na końcu pliku wiec przez DOS jest wymagane zakończenie zerem
GotName:                mov     ax, es              ;DS. wskazuje DSEG
                        mov     ds., ax
;zobacz czy nazwa pliku została dostarczona do lini poleceń
;jeśli nie zachęć użytkownika do podania i odczytaj ją z klawiatury
                        cmp     GotName1, 0         ;dostarczono nazwę numer 1?
                        jnz     HasName1
                        mov     al., '1'            ;nazwa pliku numer 1
                        lea     si, FileName1
                        call    GetName             ;pobranie pliku numer 1
HasName1:               cmp     GotName2, 0         ;dostarczono nazwę pliku numer 2?
                        jnz     HasName2
                        mov     al., '2'            ;jeśli nie odczyt z klawiatury
                        lea     si, FileName2
                        call    GetName
;Okay, mamy obie nazwy plików, teraz otwieramy pliki i kopiujemy  plik źródłowy do pliku przeznaczenia
HasName2:               mov     ah, 3dh
                        mov     al., 0              ;otwarcie pliku do odczytu
                        lea     dx, FileName1       ;plik do otwarcia
                        int     21h
                        jnc     Goodopen1

                        print
                        byte    'Nie można otworzyć pliku, program przerwany...', cr, lf, 0
                        jmp     PgmExit
;jeśli plik źródłowy został otwarty z powodzeniem, zachowujemy uchwyt pliku
GoodOpen1:              mov     FileHandle1, ax     ;zachowanie uchwytu pliku
;Otwarcie (właściwie CREATE) drugiego pliku
                        mov     ah, 3ch             ;tworzenie pliku
                        mov     cx, 0               ;standardowy atrybut
                        lea     dx, Filename2       ;plik do otwarcia
                        int     21h
                        jnc     GoodCreate
;Notka: poniższe kody błędów zależą od tego, że DOS automatycznie zamyka każdy otwarty plik źródłowy
;kiedy program  się kończy
                        print
                        byte    cr, lf
                        byte    'Nie można stworzyć nowego pliku, operacja przerwana'
                        byte    cr, lf, lf, 0
                        jmp     PgmExit
GoodCreate:             mov     FileHandle2, ax             ;zachowanie uchwytu pliku
;teraz  przetwarzamy pliki
CopyLoop:               mov     ah,3Fh                      ;DOS'owy opcod odczytu
                        mov     bx, fileHandle1             ;odczyt z pliku numer 1
                        mov     cx,512                      ;odczyt 512 bajtów
                        lea     dx, buffer                  ;bufor dla pamięci
                        int     21h
                        jc      BadRead
                        mov     bp, ax                      ; zachowanie # odczytanego bajtu

                        cmp     ConvertLC, 0                ;opcja konwersji aktywna?
                        jz      NoConversion
;Konwersja wszystkich małych liter w buforze na duże
                        mov     cx, 512
                        lea     si, Buffer
```

```
                    mov     di, si
ConvertLC2UC:
                    lodsb
                    cmp     al., ‘a’
                    jb      NoConv
                    cmp     al., ‘z’
                    ja      NoConv
                    and     al., 5fh
NoConv:             stosb
                    loop    ConvertLC2UC
NoConversion:
                    mov     ah, 40h             ;DOS’owy opcod zapisu
                    mov     bx, FileHandle2     ;zapis do pliku numer 2
                    mov     cx, bp              ;zapis jednak wielu bajtów
                    lea     dx, buffer          ;bufor na pamięć
                    int     21h
                    jc      BadWrite
                    cmp     ax, bp              ;zapisano wszystkie bajty?
                    jnz     jDiskFull
                    cmp     bp, 512             ;czy odczytano te 512 bajtów
                    jz      CopyLoop
                    jmp     AtEOF
jDiskFull:          jmp     DiskFull
;różne komunikaty o błędach
BadRead             print
                    byte    cr, lf
                    byte    ‘Błąd podczas odczytu pliku źródłowego, przerwanie ‘
                    byte    ‘operacji ‘, cr, lf, 0
                    jmp     AtEOF

BadWrite:           print
                    byte    cr, lf
                    byte    „Błąd podczas zapisu pliku przeznaczenia, przerwana ‘
                    byte    ‘operacja’, cr, lf, 0
                    jmp     AtEOF

DiskFull:           print
                    byte    cr, lf
                    byte    „Błąd, dysk pełny. Operacja przerwana’, cr, lf,0
AtEOF               mov     bx, FileHandle1             ;zamknięcie pierwszego pliku
                    mov     ah, 3Eh
                    int     21h
                    mov     bx, FileHandle2             ;zamknięcie drugiego pliku
                    mov     ah, 3Eh
                    int     21h
PgmExit:            ExitPgm
MainPgm             endp

TestDelimiter       proc    near
                    cmp     al., ‘ ‘
                    je      xit
                    cmp     al., ‘ , ‘
                    je      xit
                    cmp     al., Tab
                    je      xit
                    cmp     al., ‘ ; ‘
                    je      xit
                    cmp     al., ‘ = ‘
xit:                ret
TestDelimiter       endp
```

```
;GetName – odczytuje nazwę pliku z klawiatury, Na wejściu, AL. zawiera numer nazwy pliku a DI wskazuje
;bufor w ES gdzie musi być przechowywany plik zakończony zerem.
GetName         proc    near
                print
                byte    'Wprowadź  numer nazwy pliku:' , 0
                putc
                mov     al., ' : '
                putc
                gets
                ret
GotName         endp
cseg            ends

dseg            segment  byte  public 'data'

PSP             word    ?
FileName1       byte    128   dup (?)          ;nazwa pliku źródłowego
FileName2       byte    128   dup (?)          ;nazwa pliku przeznaczenia
FileHandle1     word    ?
FileHandle2     word    ?
GotName1        byte    ?
GotName2        byte    ?
ConvertLC       byte    ?
Buffer          byte    512   dup (?)

dseg            ends
sseg            segment  byte stack 'stack'
stk             word    0ffh  dup(?)
sseg            ends

zzzzzzseg       segment  para public 'zzzzzz'
LastBytes       byte    16  dup (?)
Zzzzzzseg       ends
                end   MainPgm
```

Jak można zauważyć, jest wymagającej więcej wysiłku przetwarzanie parametrów lini poleceń niż rzeczywiste kopiowanie plików.

13.3.13 ARGC I ARGV

Standardowa Biblioteka UCR dostarcza dwóch podprogramów, argc i argv, które pozwalają na łatwy dostęp do parametrów lini poleceń. Argc (licznik argumentów) zwraca liczbę pozycji w lini poleceń. Argv (wektor argumentów0 zwraca wskaźnik do określonej pozycji w lini poleceń

Podprogramy te rozrywają linię poleceń w leksemach używając standardowych ograniczników. W konwencji przed MS-DOS , argc i argv traktowane były jako łańcuchy otoczone cudzysłowami w lini poleceń, jako pojedyncze pozycje lini poleceń.

Argc zwróci w cx liczbę pozycji lini poleceń. Ponieważ MS-DOS nie wprowadza nazwy programu do lini poleceń, ten licznik również nie wlicza nazwy programu. Co więcej, operandy przekierowania („>nazwa pliku" i „<nazwa pliku" i pozycje na prawo od „ | polecenie" również nie pojawiają się w lini poleceń. Tak więc argc nie zlicza ich również.

Argv zwraca wskaźnik do łańcucha (alokowanego na stercie) określonego pozycją lini poleceń. Użycie argv polega po prostu na załadowaniu ax wartością między jeden a liczbą zwracaną przez argc i wykonaniem podprogramu argv. Przy zwrocie es:di wskazuje ciąg zawierający określoną opcję lini poleceń. Jeśli liczba w ax jest większa niż liczba argumentów lini poleceń, wtedy argv zwraca wskaźnik do pustego ciągu (tj. zero bajtowego). Ponieważ argv wywołuje malloc do alokowania pamięci na stercie, jest możliwość, że wystąpi błąd alokowania pamięci. Pamiętajmy o zwolnieniu zaalokowanej pamięci. Jeśli wystąpi błąd , argv ustawi flagę przeniesienia. Pamiętajmy o zwolnieniu pamięci zaalokowanej dla parametrów lini poleceń po wszystkim.

Przykład: Poniższy kod potwierdza parametry lini poleceń na ekranie

```
        include  stdlib.a
        includelib       stdlib.lib
```

```
dseg            segment  para  public  'data'
ArgCnt          word     0
dseg            ends
cseg            segment para   public 'code'
                assume   cs: cseg, ds.: dseg
Main            proc
                mov      ax, dseg
                mov      ds., ax
                mov      es, ax
;Musimy wywołać podprogram inicjalizacyjny menadżera pamięci jeśli używamy jakiegoś podprogramu który
;wywołuje malloc! ARGV jest dobrym przykładem podprogramu który wywołuje malloc.

                meminit

                argc                                ;pobrania licznika argumentów lini poleceń
                jcxz     Quit                       ;Wyjście jeśli nie ma argumentów lini poleceń
                mov      ArgCnt , 1                 ;inicjalizacja licznika lini poleceń
PrintCmds:      printf                              ;drukuj pozycję
                byte     „\n%2d: „, 0
                dword    ArgCnt

                mov      ax, ArgCnt                 ;pobranie następnego argumentów
                argv
                puts
                inc      ArgCnt                     ;Przesunięcie na następny argument
                loop     PrintCmds                  ;powtarzanie dla każdego argumentu
                putcr
Quit:           ExitPgm                             ;makro DOS dla wyjścia z programu
Main            endp
cseg            ends

sseg            segment  para   stack ' stack'
stk             byte     1024   dup („stack")
sseg            ends

;zzzzzzseg jest wymagany przez podprogram standardowej biblioteki

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       byte     16  dup (?)
Zzzzzzseg       ends
                end      Main
```

13.4 PODPROGRAMY PLIKÓW I/O STANDARDOWEJ BIBLIOTEKI UCR

Chociaż programy plików I/O MS-DOS nie są złe, Standardowa Biblioteka UCR dostarcza pakietu plików I/O, które czynią zblokowanie sekwencyjne tak łatwym jak symbole w plikach I/O. Ponadto , przy małym wysiłku możemy użyć wszystkich podprogramów StdLib takich jak printf, print, puti, puth, putc, getc, gets itd. Kiedy wykonujemy plik I/O. Znacznie upraszcza to działanie na plikach tekstowych w języku asemblera.

Zauważmy, że zapis ukierunkowany lub binarny I/O jest prawdopodobnie najlepsze co zostało po czystym DOS'ie, jeśli w dowolnym czasie chcesz wykonać dostęp swobodny wewnątrz pliku. Podprogramy Biblioteki Standardowej w rzeczywistości tylko wspomagają sekwencyjne pliki tekstowe I/O. Niemniej jednak jest to najpowszechniejsza postać plików I/O, więc podprogramy Biblioteki Standardowej są rzeczywiście całkiem użyteczne

Biblioteka Standardowa. UCR dostarcza ośmiu podprogramów plików I/O: fopen, fcreate, fgetc, fread, fputc i fwrite. Fgetc i fputc odtwarzają po kolei znaki I/O, fread i fwrite pozwalają nam odczytać i zapisać bloki danych, pozostałe cztery funkcje wykonują oczywiste działania DOS

Standardowa Biblioteka UCR używa specjalnych zmiennych plikowych do śledzenia operacji pliku. Jest to specjalne typy rekordowe, FileVar zadeklarowany w stdlib.a. Kiedy używamy podprogramów

plików I/O StdLib musimy stworzyć zmienną typu FileVar dla każdego pliku, który musimy otworzyć w tym samym czasie. Jest to bardzo proste, używamy definicji w postaci:

MyFileVar FileVar {}

Proszę zauważyć, że zmienna plikowa Biblioteki Standardowej nie jest tym samym co uchwyt pliku DOS'a. Jest to struktura, która zawiera uchwyt pliku DOS, bufor 9dla zblokowanych I/O i różne indeksy i zmienne stanu. Wewnętrzna struktura tego typu nie jest interesująca (pamiętajmy o hermetyzacji danych!) z wyjątkiem implementatora podprogramów pliku. Będziemy przekazywali adres tej zmiennej plikowej do różnych podprogramów plików I/O Standardowej Biblioteki.

13.4.1 FOPEN

Parametry wejściowe: ax – tryb otwarcia pliku
0 – plik otwarty do odczytu
1 – plik otwarty do zapisu
dx:si – wskazuje łańcuch zakończony zerem zawierający nazwę pliku
es:di – wskazuje na zmienną plikową StdLib
Parametry wyjściowe: Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu

Fopen otwiera sekwencyjny plik tekstowy do odczytu lub zapisu. W odróżnieniu od DOS'a , nie możemy otworzyć pliku do zapisu lub odczytu. Co więcej, jest to sekwencyjny plik tekstowy , który nie wspiera dostępu swobodnego. Zauważmy, że plik musi istnieć lub fopen zwróci błąd. Jest to prawda, nawet kiedy otworzymy plik do zapisu.

Zauważ, że jeśli otworzymy plik do zapisu a plik ten już istnieje , dane zapisane do tego pliku napiszą dane istniejące Kiedy zamykamy plik, dane pojawiające się w pliku po zapisanych danych będą tam jeszcze. Jeśli chcemy usunąć istniejące plik przed zapisaniem danych do niego, użyjemy funkcji fcreate.

13.4.2 FCREATE

Parametry wejściowe: dx:si - wskazuje łańcuch zakończony zerem zawierający nazwę pliku
es:di – wskazuje zmienną plikową StdLib
Parametry wyjściowe: jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu

Fcreate tworzy nowy plik i otwiera go do zapisu. Jeśli plik już istnieje, fcreate usuwa istniejący plik i tworzy nowy. Inicjalizuje zmienną plikową wyjściową, ale poza tym jest identyczny z funkcją fopen

---

13.4.3 FCLOSE

Parametry wejściowe: es:di – wskazuje zmienną plikową StdLib
Parametry wyjściowe: Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu

Fclose zamyka plik i uaktualnia informacje porządkowe.. jest bardzo ważne aby zamknąć wszystkie pliki otwarte przez fopen lub fcreate używając tej funkcji. Kiedy robimy wywołanie pliku DOS, i zapomnimy zamknąć plik, DOS automatycznie zrobi to za nas kiedy nasz program się skończy. Jednak, podprogramy StdLib chowają dane w buforach wewnętrznych, a funkcja fclose automatycznie czyści te bufory na dysku. Jeśli zakończymy program bez wywołania fclose, możemy zgubić jakieś zapisane dane w pliku ale jeszcze nie przekazane z wewnętrznego bufora na dysk.

Jeśli jesteśmy w środowisku gdzie jest możliwość przerwania przez kogoś programu ,bez dania nam możliwości zamknięcia pliku, powinniśmy wywołać podprogram fflush (następna sekcja)c dla regularnego unikania gubienia dużej ilości danych.

13.4.4 FFLUSH

Parametry wejściowe: es:di – wskazuje zmienną plikową StdLib
Parametry wyjściowe: jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu

Podprogram ten bezpośrednio zapisuje dane w wewnętrznym buforze na dysku. Zauważmy, że powinniśmy tylko używać tego podprogramu wraz z plikami otwartymi do zapisu (lub otwartymi przez fcreate). Jeśli zapiszemy dane do pliku a potem musimy zostawić plik otwarty, ale nieaktywny, w tym samym okresie czasu powinniśmy wykonać operację opróżnienia w przypadku programu zamkniętego nienormalnie.

13.4.5 FGETC

Parametry wejściowe: es:di – wskazuje zmienną plikową StdLib
Parametry wyjściowe: jeśli flaga przeniesienia jest wyzerowana, al. zawiera odczytany znak z pliku.
Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu
Ax będzie zawierał zero jeśli spróbujemy odczytać poza końcem pliku.

Fgetc odczytuje pojedynczy znak z pliku i zwraca ten znak w rejestrze al. w odróżnieniu od funkcji DOS, pojedynczy znak I/O używający fgetc jest względnie szybszy ponieważ podprogramy StdLib używają zblokowanych I/O. Oczywiście wielokrotne wywołanie fgetc nie będzie szybsze niż wywołanie fread, ale wydajność nie jest zła. Fgetc jest bardzo elastyczna. Jak zobaczymy, możemy przekierować podprogramy wejściowe StdLib ,aby odczytywały dane z pliku używając fgetc. Pozwala to nam zastosować podprogramy wyższego poziomu takie jak gets i getsm kiedy odczytujemy dane z pliku.

13.4.6 FREAD

Parametry wejściowe: es:di – wskazuje zmienną plikową StdLib
dx:si – wskazuje bufor danych wejściowych
cx - zawiera bajt licznika
Parametry wyjściowe: Jeśli flaga przeniesienia jest wyzerowana, ax zawiera rzeczywistą liczbę bajtów odczytanych z pliku. Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędów.

Fread jest bardzo podobna do DOS'owego polecenia read. Pozwala nam odczytać blok bajtów, zamiast jednego bajtu, z pliku. Zauważmy, że jeśli wszystko co chcemy zrobić to odczytanie bloku bajtów z pliku, DOS'owska funkcja jest odrobinę wydajna niż fread. Jednak jeśli mamy do odczytania mieszankę bajtów pojedynczych i wielu bajtów, kombinacja fread i fgetc działa bardzo dobrze.

Podobnie jak przy operacji odczytu DOS, jeśli liczony bajt w ax nie zgadza się z wartością przekazaną w rejestrze cx, wtedy odczytamy pozostałe bajty w pliku. Jeśli tak się zdarzy , następne wywołanie fread lub fgetc zwróci błąd końca pliku (flaga przeniesienia będzie ustawiona a ax będzie zawierał zero). Zauważmy, ze fread nie zwraca EOF, chyba że zostało odczytane zero bajtów z pliku.

13.4.7 FPUTC

Parametry wejściowe: es:di – wskazuje zmienna plikową StdLib
al. – zawiera znak do zapisania w pliku
Parametry wyjściowe: Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS kod błędu

Fputc zapisuje pojedynczy znak (a al) do pliku określonego przez zmienną plikową , której adres jest w es:di. Wywołanie to po prostu dodaje znak w al. do wewnętrznego bufora (część zmiennej plikowej) dopóki bufor nie będzie pełny. Gdy tylko bufor jest wypełniony lub wywołujesz fflush (zamykamy plik fclose), podprogramy plików I/O zapisują dane na dysk.

13.4.8 FWRITE

Parametry wejściowe: es:di – wskazuje zmienną plikową StdLib
dx:si – wskazuje bufor danych wyjściowych
cx – zawiera zliczony bajt
Parametry wyjściowe: Jeśli flaga przeniesienia jest wyzerowana, ax zawiera rzeczywistą liczbę bajtów zapisanych do pliku
Jeśli flaga przeniesienia jest ustawiona, ax zawiera zwracany przez DOS, kod błędów
Podobnie jak fread, fwrite działa na blokach bajtów. Pozwala nam zapisać blok bajtów do pliku otwartego do zapisu przez fopen lub fcreate .

13.4.9 PRZEKIEROWANIE NA PORTY I/O PRZEZ PODPROGRAMY PLIKÓW I/O STDBLIB

Standardowa biblioteka UCR dostarcza niewielu podprogramów plików I/O .Na przykład Fputc i fwrite są jedynie dwoma podprogramami wyjściowymi .Standardowa biblioteka języka C (na której oparta jest Standardowa Biblioteka UCR) dostarcza wiele podprogramów takich jak fprintf, fputs, fscanf itp. Żaden z nich nie jest potrzebny w Standardowej Bibliotece UCR , ponieważ biblioteka UCR dostarcza mechanizmu przekierowania na porty I/O , który pozwala nam na ponowne użycie wszystkich istniejących podprogramów I/O dla wykonania pliku I/O.

Podprogram putc Standardowej Biblioteki UCR składa się z pojedynczej instrukcji jmp. Instrukcja ta przekazuje sterowanie do jakiegoś rzeczywistego podprogramu wyjściowego poprzez pośredni adres wewnętrzny do kodu putc. Zwykle ta zmienna wskaźnikowa wskazuje kawałek kodu który zapisuje znak w rejestrze al. do standardowego urządzenia wyjściowego DOS. Jednak Biblioteka Standardowa dostarcza również czterech podprogramów, które pozwalają nam manipulować tym pośrednim wskaźnikiem . Przez zmianę tego wskaźnika, możemy przekierować wyjście z jego aktualnego podprogramu do podprogramu jaki wybraliśmy, Wszystkie podprogramy wyjściowe Biblioteki Standardowej (tj. printf, puti, puth, puts) wywołują putc do wyprowadzenia pojedynczego znaku. Dlatego też przekierowanie podprogramu putc wpływa na wszystkie podprogramy wyjściowe.

Podobnie podprogram getc nie jest niczym innym jak skokiem pośrednim jmp, którego zmienna wskaźnikowa zwykle wskazuje kawałek kodu który odczytuje dane ze standardowego wejścia. Ponieważ wszystkie podprogramy wejściowe Biblioteki Standardowej wywołują funkcję getc do odczytu każdego znaku, możemy przekierować plik wejściowy w sposób identyczny do pliku wyjściowego.

GetOutAdrs, SetOutAdrs, PushOutAdrs i PopOutAdrs z Bibliotek Standardowej są czterema głównymi podprogramami, które manipulują wskaźnikiem przekierowania wyjścia. GetOutAdrs zwraca adres bieżącego podprogramu wyjściowego w rejestrach es:di. SetOutAdrs odwrotnie, oczekuje, że przekażemy mu adres nowego podprogramu wyjściowego w rejestrach es:di i przechowuje ten adres we wskaźniku wyjściowym. PushOutAdrs i PopOutAdrs odkładają i zdejmują wskaźnik z wewnętrznego stosu. Nie używają one stosu sprzętowego 80x86. Jesteśmy ograniczeni do małej ilości odłożeń i zdjęć. Generalnie nie powinniśmy liczyć na możliwość odłożenia więcej niż czterech tych adresów na wewnętrznym stosie bez jego przepełnienia.

GetInAdrs, SetInAdrs, PushInAdrs i PopInAdrs są uzupełniającymi podprogramami dla wektora wejściowego. Pozwalają nam one na manipulowanie wskaźnikiem podprogramów wejściowych. Zauważmy, że stos dla PushInAdrs / PopInAdrs nie jest taki sam jak stos dla PushOutAdrs / PopOutAdrs. Odkładanie i zdejmowanie z tych dwóch stosów jest niezależne jedno od drugiego.

Zazwyczaj, wskaźnik wyjściowy (do którego będziemy się odtąd odnosili jako do łącza wyjściowego) wskazuje podprogram PutcStdOut Biblioteki Standardowej. Dlatego też możemy przywrócić łącze wyjściowe do jego normalnego stanu inicjalizującego w każdej chwili poprzez wykonanie instrukcji:

```
mov     di, seg  SL_PutcStdOut
mov     es, di
mov     di, offset SL_PutcStdOut
SetOutAdrs
```

Podprogram PutcStdOut zapisuje znak z rejestru al. do standardowego wyjścia DOS'a, które samo może być przekierowane do jakiegoś pliku lub urządzenia (używając DOS'owego operatora przekierowania „>"). Jeśli chcemy upewnić się, że naszym urządzeniem wyjściowym jest monitor, możemy zawsze wywołać podprogram PutcBIOS, który wywołuje BIOS bezpośrednio dla znaku wyjściowego. Możemy zmusić wszystkie wyjścia Biblioteki Standardowej do urządzenia standardowego błędu używając takiej sekwencji kodu:

```
mov     di, seg SL_PutcBIOS
mov     es, di
mov     di, offset SL_PutcBIOS
SetOutAdrs
```

Generalnie nie możemy oswobodzić łącza wyjściowego poprzez przechowanie wskaźnika do podprogramu na szczycie jakiegoś wskaźnika ,który tam był a potem przywrócić łącze do PutcStdOut po ukończeniu. Kto wie czy łącze przede wszystkim wskazywałoby na PutcStdOut.? Najlepszym rozwiązaniem jest użycie podprogramów Biblioteki Standardowej PushOutAdrs i PopOutAdrs do przechowania i przywrócenia poprzedniego łącza. Poniższy kod demonstruje łagodniejszy sposób modyfikacji łącza wyjściowego:

```
        PushOutAdrs                     ;zachowanie bieżącego podprogramu wyjściowego
        mov     di, seg Output_Routine
        mov     es, di
        mov     di, offset Output_Routine
        SetOutAdrs
<wykonanie wszystkich wyjść Output_Routine >
        PopOutAdrs                      ;przywrócenie poprzedniego podprogramu wyjścia
```

Obsługa wejścia w podobny sposób używa odpowiednich łączy wejściowych dla dostępu do podprogramu i podprogramów SL_GetStdOut i SL_GetBIOS. Pamiętajmy zawsze, że jest ograniczona liczba wejść do łączy stosu wejścia i wyjścia , i ile pozycji odłożymy na te stosy bez zdejmowania czegokolwiek.

Dla przekierowania wyjścia do pliku (lub przekierowania wejścia z pliku) musimy najpierw napisać krótki podprogram, który zapisze (odczyta) pojedynczy znak z (do) pliku. Jest to bardzo łatwe.. Kod dla podprogramu dla danych wyjściowych do pliku opisany przez zmienną plikową OutputFile to:

```
ToOutput        proc    far
                push    es
                push    di
;Ładuje ES:DI adresem zmiennej OutputFile. Kod ten zakłada, że OutputFile jest typu FileVar, a nie
;wskaźnikiem do zmiennej typu FileVar
                mov     di, seg OutputFile
                mov     es, di
                mov     di, offset OutputFile
;wprowadzany znak z AL. do pliku jest opisany przez „OutputFile”
                fputc

                pop     di
                pop     es
                ret
ToOutput        endp
```

Teraz, tylko z jednym dodatkowym kawałkiem kodu, możemy zacząć zapisywać dane do pliku wyjściowego używając podprogramów wyjściowych Biblioteki Standardowej. Poniżej mamy krótki fragment kodu, który przekierowywuje łącze wyjściowe do powyższego podprogramu „ToOutput”:

```
SetOutFile      proc
                push    es
                push    di

                PushOutAdrs                     ;zachowanie bieżącego łącza wyjściowego
                mov     di, seg  ToOutput
                mov     es, di
                mov     di, offset ToOutput
                SetOutAdrs

                pop     di
                pop     es
                ret
SetOutFile      endp
```

Nie ma potrzeby oddzielnego podprogramu przywracającego łącze wyjściowe do jego poprzedniej wartości; PopOutAdrs wykona to zadanie.

13.4.10 PRZYKŁAD PLIKU I/O

Poniższy przykład łączy wszystko razem z poprzednich kilku sekcji .Jest to prosty program , który dodaje numery lini do pliku tekstowego. Program ten wymaga dwóch parametrów lini poleceń: pliku wejściowego i pliku wyjściowego. Kopiuje plik wejściowy do pliku wyjściowego dołączając numery lini na początku każdej lini w pliku wyjściowym. Kod ten demonstruje zastosowanie argc, argv, podprogramów plików I/O Biblioteki Standardowej i przekierowania I/O

```
;Program ten kopiuje plik wejściowy do pliku wyjściowego i dodaje numery lini podczas kopiowania pliku

                include  stdlib.a
                includelib   stdlib.lib

dseg            segment para  public  ‘data’

ArgCnt          word    0
LineNumber      word    0
DOSErrorCode    word    0
InFile          dword   ?               ;wskaźnik do nazwy pliku wejściowego
OutFile         dword   ?               ;wskaźnik do nazwy pliku wyjściowego
```

```
InputLine       byte    1024   dup  (0)          ;bufor danych Wejścia / Wyjścia
OutputFile      FileVar {}
InputFile       FileVar {}

dseg            ends

cseg            segment para  public  'code'
                assume  cs : cseg, ds. : dseg
;ReadLn – Odczyt lini tekstu z pliku wejściowego i przechowanie danych w buforze InputLine:

ReadLn          proc
                push    ds.
                push    es
                push    di
                push    si
                push    ax

                mov     si, dseg
                mov     ds., si
                mov     si, offset  InputLine
                lesi    InputFile

GetLnLp:
                fgetc
                jc      RdLnDone                ;jeśli jakiś dziwaczny błąd
                cmp     ah, 0                   ;sprawdzenie dla końca pliku
                je      RdLnDone                ;Notka: przeniesienie jest ustawione
                mov     ds.:[si], al.
                inc     si
                cmp     al., lf                 ;EOLN?
                jne     GetLnLp
                dec     si                      ;cofnięcie przed LF
                cmp     byte   ptr ds.:[si –1], cr  ;CR przed LF?
                jne     RdLnDone
                dec     si                      ;jeśli tak, przeskocz także

RdLnDone:       mov     byte ptr ds.:[si], 0    ;zakończenie zerem
                pop     ax
                pop     si
                pop     di
                pop     es
                po      ds.
                ret
ReadLn          endp
;MyOutput – zapis pojedynczego znaku z AL. do pliku wyjściowego
MyOutput        proc    far
                push    es
                push    di
                lesi    OutputFile
                fputc
                pop     di
                pop     es
                ret
MyOutput        endp
;Program główny , który wykonuje całą pracę
Main            proc
                mov     ax, dseg
                mov     ds., ax
                mov     es, ax
```

```
;Musimy wywołać podprogram inicjalizujący menadżera pamięci, jeśli używamy jakiegoś podprogramu, który
;wywołuje malloc! ARGV jest dobrym przykładem podprogramu wywołującego malloc
                meminit
;Oczekujemy, że program będzie wywoływany jak następuje
;               fileio file1, file2
;w przeciwnym razie mamy błąd
                argc
                cmp     cx, 2                   ;musimy mieć dwa parametry
                je      Got2Parms
BadParms:       print
                byte    „Użycie : FILEIO infile, outfile”, cr, lf, 0
                jmp     Quit
;Okay, mamy dwa parametry, szczęśliwie mają poprawne nazwy.
‘Pobieramy kopię nazw plików i przechowujemy jako wskaźnik do nich
Got2Parms:      mov     ax, 1                   ;pobranie nazwy pliku wejściowego
                argv
                mov     word   ptr  InFile, di
                mov     word   ptr  InFile+2, es

                mov     ax, 2                   ;pobranie nazwy pliku wyjściowego
                argv
                mov     word  ptr  OutFile, di
                mov     word  ptr OutFile+2, es
;Wyprowadzenie nazwy pliku na standardowe urządzenie wyjściowe
                printf
                byte    „Plik wejściowy:  %^s\n”
                byte    „plik wyjściowy:  %^s\n”, 0
                dword   InFile, OutFile
;Otwarcie pliku wejściowego:
                lesi    InputFile
                mov     dx, word   ptr InFile+2
                mov     si, word   ptr InFile
                mov     ax, 0
                fopen
                jnc     GoodOpen
                mov     DOSErrorCode, ax
                printf
                byte    „Nie można otworzyć pliku wejściowego, DOS:  %d\n”, 0
                dword   DOSErrorCode
                jmp     Quit
;Stworzenie nowego pliku dla wyjścia:
GoodOpen:       lesi    OutputFile
                mov     dx, word  ptr  OutFile+2
                mov     si, word   ptr OutFile
                fcreate
                jnc     GoodCreate
                mov     DOSErrorCode, AX
                printf
                byte    „Nie można otworzyć pliku wyjściowego, DOS :  %d\n”, 0
                dword   DOSErrorCode
                jmp     Quit
;Okay, zachowamy łącze wyjściowe i przekierowanie wyjścia
GoodCreate:     PushOutAdrs
                lesi    MyOutput
                SetOutAdrs
WhlNotEOF:      inc     LineNumber
;Okay odczytujemy linię wejściową od użytkownika:
                call    ReadLn
                jc      BadInput
;okay, przekierowanie wyjścia do naszego pliku wyjściowego i zapis ostatniej  odczytanej lini
```

```
;poprzedzonej numerem lini
                printf
                byte    „%4d:   %s\n”, 0
                dword   LineNumber, InputLine
                jmp     WhlNotEOF
BadInput:       push    ax                      ;zachowanie kodu błędu
                PopOutAdrs                      ;przywrócenie łącza wyjściowego
                pop     ax                      ;odzyskanie kodu błędu
                test    ax, ax                  ;błąd EOF? (AX = 0)
                jz      CloseFiles
                mov     DOSErrorCode, ax
                printf
                byte    „Błąd wejścia, DOS:  %d\n”, 0
                dword   LineNumber
;okay, zamykamy plik  i wychodzimy:
CloseFiles:     lesi    OutputFile
                fclose
                lesi     InputFile
                fclose

Quit:           ExitPgm                         ;makro DOS’a do zamknięcia programu
Main            endp
cseg            ends

sseg            segment para stack  ‘stack’
stk             byte    1024   dup („stack   „)
sseg            ends

zzzzzzseg       segment para   public ‘zzzzzz’
LastBytes       byte    16  dup  (?)
Zzzzzzseg       ends
                end     Main
```

13.8 PODSUMOWANIE

MS-DOS i BIOS dostarczają wiele usług systemowych, które sterują sprzętem na PC. Dostarczają one niezależnego od sprzętu i elastycznego interfejsu. Niestety, PC rozwinął się trochę od dni oryginalnego 5 Mhz 8088 IBM PC. Wiele funkcji DOS i BIOS jest teraz przestarzałych, wypieranych przez nowsze funkcje. Zakładając wsteczną kompatybilność, MS-DOS i BIOS generalnie wspierają wszystkie starsze i przestarzałe funkcje tak samo jak nowsze. Jednakże nasz program nie powinien używać funkcji przestarzałych.

BIOS dostarcza wiele usług powiązanych ze sterowaniem urządzeniami takimi jak monitor; port drukarki, klawiatura, port szeregowy, zegar czasu rzeczywistego itp. Opis usług BIOS dla tych urządzeń pojawia się w poniższych sekcjach:

- „INT 5 - Zrzut Ekranowy”
- „INT 10h - Usługi Video”
- INT 11h – Konfiguracja komputera’
- „INT 12h – Określenie rozmiaru pamięci”
- „INT 13h – Usługa obsługi dysków”
- „INT 14h – Obsługa złącza szeregowego I/O”
- „INT 15h – Usługi Dodatkowe”
- „INT 16h – Obsługa klawiatury”
- „INT 17h – Obsługa drukarki”
- „INT 18h – BASIC”
- „INT 19h – Gorący restart systemu”
- „INT 1Ah – Usługa zegara czasu rzeczywistego”

MS-DOS dostarcza kilka różnych typów usług. Ten rozdział koncentrował się na usługach plików I/O dostarczonych przez MS-DOS. W szczególności rozdział ten zajmował się implementacją działania wydajnych plików I/O używających zblokowanych I/O. Uczy jak wykonać pliki I/O i inne operacje MS-DOS.

- „Sekwencja wywołania MS-DOS”
- „Funkcje zorientowane znakowo MS-DOS”
- „Przestarzałe funkcje zapisujące dane MS-DOS”
- „Funkcje daty i czasu MS-DOS”
- „Funkcje zarządzania pamięcią MS-DOS’
- „Funkcje sterowania procesem MS-DOS”
- „Nowe funkcje zapisujące dane MS-DOS”
- „Przykłady pliku I/O”
- „Zblokowane pliki I/O”

Dostęp do parametrów lini poleceń jest ważną operacją wewnątrz aplikacji MS-DOS. DOS’owy PSP zawiera linię poleceń i kilka innych kawałków ważnych informacji. Uczy o różnych polach w PSP i pokazuje jak uzyskać dostęp do parametrów lini poleceń.

- „Program Segment Prefix”
- „Dostęp do parametrów lini poleceń”
- „ARGC i ARGV”

Oczywiście, Standardowa Biblioteka UCR dostarcza również kilka podprogramów pliku I/O. Rozdział ten zamyka się przez opisanie kilku podprogramów pliku I/O StdLib wraz z ich zaletami i wadami

- „Fopen”
- „Fcreate”
- „Fclose”
- „Fflush”
- „Fgetc”
- „Fread”
- „Fputc
- „Fwrite”
- „Przekierowanie I/O przez podprogramy I/O StdLib
- „Przykład pliku I/O”

13.9 PYTANIA

1) Jak są wywoływane podprogramy BIOS’a?
2) Jaki podprogram BIOS jest używany do zapisu znaku do:
   a) monitora b) portu szeregowego c) portu drukarki
3) Kiedy usługa transmisji szeregowej lub odbioru wraca do kodu wywołującego, stan błędu jest zwracany w rejestrze AH. Jednakże jest problem z wartością zwracaną. Co to za problem?
4) Wyjaśnij jak można przetestować klawiaturę, aby zobaczyć czy klawisz jest dostępny.
5) Co jest złego w stanie funkcji przesunięcia klawiatury/
6) Jak specjalne kody kluczy (nie zwracające kodów ASCII) są zwracane przez funkcję odczytu klawiatury ?
7) Jak wysyłamy znak do drukarki?
8) Jak odczytujemy czas rzeczywisty zegara systemowego?
9) Podano, że RTC zwiększa 32 bitowy licznik co 55 ms, jak długo będzie działał system zanim wystąpi przepełnienie licznika?
10) Dlaczego powinniśmy resetować zegar ,kiedy odczytujemy zegar, określając, że licznik został przepełniony?
11) Jak program asemblerowy wywołuje MS-DOS
12) Gdzie generalnie są przekazywane parametry do MS-DOS?
13) Dlaczego są dwa zbiory funkcji zapisujących dane na dysku w MS-DOS?
14) Gdzie mogą być znaleziona linia poleceń DOS’a?
15) Jakie jest zadanie środowiska obszaru ciągów?
16) Jak można określić ilość pamięci dostępnej do zastosowania przez nasz program?
17) Co jest bardziej wydajne : znakowe I/O lub zblokowane I/O? Dlaczego?
18) Jaki jest dobry rozmiar bloku dla zblokowanych I/O?
19) Dlaczego nie możemy użyć zblokowanych I/O w plikach o dostępie swobodnym?
20) Wyjaśnij jak zastosować polecenie SEEK do przesunięcia wskaźnika pliku 128 bajtó wstecz w pliku z bieżącej pozycji pliku.

21) Gdzie zazwyczaj zwracany jest stan błędu po wywołaniu DOS?
22) Dlaczego jest trudno użyć zblokowanych I/O w plikach o dostępie swobodnym?
23) Opisz jak można zaimplementować zblokowane I/O w plikach otwartych z sekwencyjnym dostępem do odczytu i zapisu.
24) Jakie są dwa sposoby w jaki możemy uzyskać adres PSP?
25) Jak możemy określić, że osiągnęliśmy koniec pliku kiedy używamy funkcji pliku I/O MS-DOS? Kiedy używamy funkcji Biblioteki Standardowej UCR?