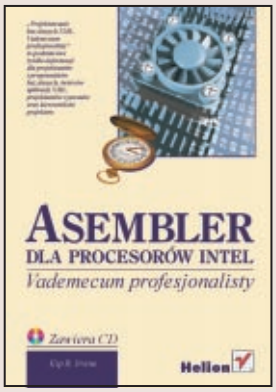
Assembler dla procesorów Intel. Vademecum profesjonalisty



Autor: Kip R. Irvine
Tłumaczenie: Marcin Jędrysiak
ISBN: 83-7197-910-X
Tytuł oryginału: Assembly Language for Intel-Based Computers
Format: B5, stron: 640

Chociaż świat programowania nieodwołalnie zdominowany jest przez języki wyższego poziomu (takie jak C, C++ czy Java), znajomość asemblera jest nadal wysoko ceniona. Użycie języka maszynowego pozwala na pisanie niezwykle wydajnych programów, odwołujących się bezpośrednio do sprzętu, wykorzystujących w 100% możliwości hardware'u.

Książka „Asembler dla procesorów Intel. Vademecum profesjonalisty" to kompletny przewodnik po języku programowania najpopularniejszej rodziny procesorów. Możesz ją traktować jako podręcznik asemblera lub jako kompendium wiedzy, do którego zawsze będziesz mógł się odwołać, gdy zajdzie taka potrzeba. Znajdziesz w niej informacje dotyczące:

- architektury mikroprocesorów i zestawu rozkazów procesorów z rodziny Intel 80x86
- dyrektyw, makr i operatorów języka asembler oraz podstawowej struktury programu
- metodologii programowania oraz sposobów użycia języka asembler do tworzenia narzędzi systemowych i aplikacji
- sposobów pracy z urządzeniami sprzętowymi
- interakcji między programem napisanym w języku asembler, systemem operacyjnym a innymi aplikacjami
- śledzenia przebiegu wykonania programu za pomocą debugera

Oprócz krótkich przykładów książka „Asembler dla procesorów Intel. Vademecum profesjonalisty" zawiera 75 gotowych do uruchomienia programów, które realizują koncepcje prezentowane w tekście. Znajdujące się na końcu książki dodatki stanowią przewodniki po przerwaniach MS-DOS i mnemonikach kodu maszynowego.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Podziękowania

Gorące podziękowania przekazuję redaktorce w wydawnictwie Prentice Hall — Laurze Steele — która była moim przewodnikiem w czasie pisania tej książki.

Dziękuję także Johnowi Griffiniowi oraz Ronowi Harrisowi, którzy pracują jako redaktorzy w Macmillan Publishing.

Moje specjalne podziękowania i wdzięczność przekazuję trzem profesorom. Są to: Gerald Calhill z Antelope Valley Collage, który dokonał wielu cennych poprawek, Tim Downey z Florida Internation University, który podzielił się ze mną cennymi koncepcjami na temat architektury komputerów, oraz James Brink z Pacific Lutheran University, który utworzył świetną bibliotekę konsolidowaną z programami pracującymi w płaskim trybie pamięci.

Microsoft pozwolił na dołączenie do książki swojego Macro Assemblera za umiarkowaną cenę.

Osoby pomagające w wydaniu tej książki

Bardzo dziękuję następującym osobom: Donnie Sullivan, menedżerowi projektu w Prentice Hall, Rayowi Robinsonowi, dyrektorowi projektu i Kelly Dobbs, kierownikowi projektu (obie osoby z D & G Limited, LLC) za doskonałą pomoc w czasie przygotowywania tej książki do druku. Michael Brumitt z tej samej firmy zajął się korektą tej książki. David Irvine wykonał świetną pracę, przygotowując wszystkie ilustracje i wiele tabel. Za korektę końcowego rękopisu odpowiedzialni byli także Bill Dever, Alejandro Ferro i Raymond Lim.

Recenzenci

- Kathy Blicharz (Pima Community College);
- Patricia Nettnin (Finger Lakes Community College);
- Michael J. Walron, Barry Brosch, Bruce DeSautel i Richard White (Miami-Dade Community College);
- Richard A. Beebe (Simpson College);
- John V. Erhart i Gonshin Liu (University of Bridgeport);
- S.K. Sachdev (Eastern Michigan University);
- Douglas W. Knight (University of Southern Colorado);
- Don Retzlaff (University of North Texas);
- Robert Galivan, konsultant w zakresie oprogramowania;
- George Kamenz, świetny programista, który przeczytał rękopis tej książki i przedstawił wiele cennych sugestii;
- Diego Escala napisał wspaniały program do przeglądania plików bitmapowych z rozdziału 12.

Chciałbym także podziękować setkom studentów z Miami-Dade Community College, którzy w latach 1990 – 99 wykazali wiele determinacji ucząc się z tej książki, i którzy często przekraczali oczekiwania wykładowców.

Rozdział 12.
Przetwarzanie plików

12.1. Manipulacja plikami

12.1.1. Wprowadzenie

W poprzednim rozdziale omówiono sposoby organizacji plików i katalogów na dysku. Teraz nadszedł czas na przedstawienie wywołań funkcji służących do pracy z plikami. W celu uzyskania dostępu do plików i urządzeń DOS stosuje technikę zapożyczoną z systemu operacyjnego UNIX, a mianowicie *uchwyty*. W większości przypadków uchwyty odnoszą się zarówno do plików, jak i do urządzeń typu klawiatura czy monitor. *Uchwyt* to 16-bitowy numer używany do identyfikacji otwartego pliku lub urządzenia. DOS rozpoznaje pięć standardowych uchwytów urządzeń. Każdy z nich (z wyjątkiem urządzenia wyjścia błędu) umożliwia przekierowanie wejścia-wyjścia poprzez wiersz poleceń:

- 0 — klawiatura (standardowe wejście),
- 1 — konsola (standardowe wyjście),
- 2 — wyjście błędów,
- 3 — urządzenie pomocnicze (asynchroniczne),
- 4 — drukarka.

Uchwyty te są wstępnie zdefiniowane i nie ma potrzeby ich tworzenia przed użyciem. Możliwy jest na przykład zapis do konsoli poprzez uchwyt 1 bez żadnych przygotowań. Wszystkie funkcje związane z uchwytami mają pewną wspólną cechę — jeśli operacja nie powiedzie się, ustawiany jest znacznik `Carry`, a w rejestrze `AX` zwracany jest kod błędu, który może być użyty do wyświetlenia odpowiedniego komunikatu dla użytkownika.

Podstawowe funkcje plików

Prezentację funkcji rozpoczniemy od najbardziej podstawowych funkcji plików, które są definiowane poprzez umieszczenie numeru funkcji w rejestrze `AH`. Wszystkie funkcje plików są dostępne w językach wysokiego poziomu (tabela 12.1).

Kolejny zestaw procedur zapewnia ogromną kontrolę nad plikami. Programista uzyskuje funkcje, które często nie są dozwolone w wierszu poleceń DOS, takie jak ukrycie i przywrócenie pliku, ustawienie atrybutu tylko do odczytu oraz zmiana znacznika czasowego pliku. Możliwe jest także wyszukiwanie plików z użyciem znaków zastępczych, na przykład **.ASM*.

Tabela 12.1. *Podstawowe funkcje plików*

Funkcja	Opis
1Ah	Powoduje ustawienie adresu transferu dysku
3Ch	Przeznaczona jest do tworzenia pliku. Powoduje utworzenie nowego pliku lub ustawienie wielkości już istniejącego pliku na 0 bajtów
3Dh	Służy do otwierania pliku. Otwiera istniejący plik dla operacji wejścia, wyjścia lub wejścia-wyjścia
3Eh	Powoduje zamknięcie uchwytu pliku
3Fh	Powoduje odczyt z pliku lub urządzenia. Umożliwia odczytanie określonej liczby bajtów z pliku do buforu wejściowego
40h	Służy do zapisu do pliku lub urządzenia. Umożliwia zapisanie określonej liczby bajtów z buforu wejściowego do pliku
41h	Usuwa plik
42h	Służy do przesuwania wskaźnika pliku. Powoduje umieszczenie wskaźnika pliku przed wykonaniem operacji odczytu lub zapisu do pliku
43h	Powoduje uzyskanie lub ustawienie atrybutu pliku
4Eh	Służy do wyszukiwania pierwszego żądanego pliku
4Fh	Służy do wyszukiwania kolejnego żądanego pliku
56h	Powoduje zmianę nazwy pliku
57h	Powoduje uzyskanie lub ustawienie daty i czasu pliku

12.1.2. Uzyskanie lub ustawienie atrybutu pliku (43h)

Funkcja 43h może być użyta do uzyskania lub zmiany atrybutu pliku. Wybór operacji jest zależny od znacznika w rejestrze AL. Używane są następujące rejestry wejściowe:

- AH — wartość 43h,
- AL — wartość 0 oznacza uzyskanie atrybutu, 1 to ustawienie atrybutu,
- CX — nowy atrybut, jeśli AL = 1,
- DS:DX — wskazuje ciąg *ASCIIZ* ze specyfikacją pliku.

Znacznik Carry jest ustawiany w przypadku niepowodzenia, a zwracany kod błędu może mieć wartość: 1 (niewłaściwy kod funkcji), 2 (nie odnaleziono pliku), 3 (nie odnaleziono ścieżki) lub 5 (dostęp zastrzeżony). Jeśli AL ma wartość 0 (uzyskanie atrybutu), to atrybut pliku jest zwracany w rejestrze CX. Ten atrybut może także wskazywać etykietę woluminu (08h) lub podkatalog (10h). Na podstawie poniższego fragmentu kodu zostają ustawione atrybuty pliku typu „ukryty" i „tylko do odczytu":

```
.data
filename   db "TEST.DOC",0
.code
mov   ah,43h
mov   al,1              ; ustawienie atrybutu pliku
mov   cx,3              ; ukryty, tylko do odczytu
```

```
mov   dx,offset filename
int   21h
jc    display_error
```

Atrybuty pliku zostały omówione bardziej szczegółowo w podrozdziale 11.1.4. Przykładowe wartości atrybutów znajdują się w poniższej tabeli. Plik może posiadać także ustawiony bit archiwalny (5).

Atrybut	Wartość
Normalny plik	00
Plik tylko do odczytu	01
Plik ukryty	02
Plik ukryty, tylko do odczytu	03
Plik systemowy	04
Ukryty plik systemowy, tylko do odczytu	07

Jedną z największych zalet tej funkcji jest możliwość ukrycia pliku, dzięki czemu nie będzie widoczny dla poleceń `DIR`, `DEL` i `COPY`. Atrybut tylko do odczytu uniemożliwia dokonanie zmian w pliku. Aby usunąć lub zmienić taki plik poprzez wiersz poleceń, należy najpierw ustawić normalny atrybut pliku.

12.1.3. Usunięcie pliku (41h)

Aby usunąć plik, ustaw `DS:DX` na adres ciągu *ASCIIZ* zawierającego specyfikację pliku. Specyfikacja może zawierać literę dysku i nazwę ścieżki, ale znaki zastępcze nie są dozwolone. Poniższy kod powoduje usunięcie pliku *SAMPLE.OBJ* z dysku *B*:

```
.data
filespec   db "B:SAMPLE.OBJ",0

.code
mov   ah,41              ; usunięcie pliku
mov   dx,offset filespec
int   21h
jc    display_error
```

Znacznik `Carry` jest ustawiany w przypadku niepowodzenia, a zwracany kod błędu może mieć wartość 2 (nie odnaleziono pliku), 3 (nie odnaleziono ścieżki) lub 5 (dostęp zastrzeżony, ponieważ plik ma atrybut tylko do odczytu). Aby usunąć plik tylko do odczytu, należy wywołać funkcję `43589h` w celu zmiany jego atrybutu.

12.1.4. Zmiana nazwy pliku (56h)

Do funkcji `56h` należy przekazać w `DS:DX` wskaźnik do aktualnej nazwy oraz w `ES:DI` wskaźnik do nowej nazwy pliku. Obie nazwy muszą mieć postać ciągów ASCIIZ i nie mogą zawierać znaków zastępczych. Ta funkcja może być także użyta do przeniesienia pliku do innego katalogu, ponieważ dla każdej nazwy pliku można podać odrębną ścieżkę.

Przeniesienie pliku różni się od kopiowania, ponieważ oryginalny plik jest usuwany z katalogu źródłowego. Znacznik Carry jest ustawiany w przypadku niepowodzenia, a zwracany kod błędu może mieć wartość 2 (nie odnaleziono pliku), 3 (nie odnaleziono ścieżki), 5 (dostęp zastrzeżony) lub 11h (niezgodność urządzeń). Błąd 11h jest zgłaszany w przypadku użycia nazw plików na różnych dyskach. Poniższa procedura zmienia nazwę pliku z *prog1.asm* na *prog2.asm*:

```
.data
oldname   db "prog1.asm",0
newname   db "prog2.asm",0

.code
mov   ah,56h            ; zmiana nazwy pliku
mov   dx,offset oldname
mov   di,offset newname
int   21h
jc    display_error
```

Poniższy kod powoduje przeniesienie pliku *prog1.asm* z aktualnego katalogu do podkatalogu *\asm\progs*:

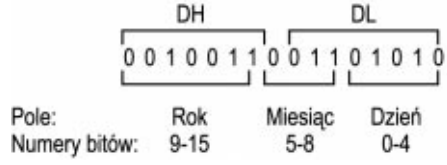
```
.data
oldname   db "prog1.asm",0
newname   db "\asm\progs\prog2.asm",0

.code
mov   ah,56h            ; zmiana nazwy pliku
mov   dx,offset oldname
mov   di,offset newname
int   21h
jc    display_error
```

12.1.5. Uzyskanie i ustawienie daty i czasu pliku

Funkcja 57h może być użyta do odczytania lub modyfikacji znacznika czasu lub daty. Oba te znaczniki są automatycznie aktualizowane w przypadku modyfikacji pliku, ale w niektórych przypadkach konieczne jest ręczne dopasowanie tych wartości.

Przed wywołaniem tej funkcji plik musi być już otwarty. Aby odczytać datę i czas pliku, należy ustawić AL na 0, a BX na uchwyt pliku. Aby ustawić datę i czas, rejestr AL powinien zawierać 1, BX — uchwyt pliku, CX — czas, a DX — datę. Wartości daty i czasu są odwzorowane bitowo dokładnie w taki sam sposób, jak w przypadku katalogu. Na rysunku przedstawiono datę pliku:



Liczba sekund jest zapisywana z odstępem 2 sekund. Wartość czasu 10:02:02 zostanie odwzorowana jako:

```
0101000001000001
```

Wartość roku jest dodawana do roku 1980, przez co data 16 kwietnia 1992 (920416) zostanie zapisana jako:

```
0001100010010000
```

Jeśli wystarczy jedynie uzyskanie daty i czasu pliku, to zalecam użycie prostszej funkcji 4Eh (odnalezienie pierwszego szukanego pliku), która nie wymaga otworzenia pliku.

12.1.6. Odnalezienie pierwszego szukanego pliku (4Eh)

Funkcja 4Eh służy do wyszukiwania plików w określonym katalogu. W celu jej użycia należy przekazać w DS:DX wskaźnik do specyfikacji pliku *ASCIIZ* oraz ustawić CX na atrybut szukanego pliku. Ciąg specyfikacji pliku może zawierać znaki zastępcze (* i ?), dzięki czemu funkcja 4Eh świetnie się sprawdza przy wyszukiwaniu większej grupy plików. Poniższy kod wyszukuje wszystkie pliki z rozszerzeniem *.ASM* w katalogu *C:\ASM\PROGS*:

```
.data
filespec   db "C:\ASM\PROGS\*.ASM",0

.code
mov   ah,4Eh        ; odnalezienie pierwszego szukanego pliku
mov   cx,0          ; pliki z normalnym atrybutem
mov   dx,filespec
int   21h
jc    display_error
```

Po odnalezieniu żądanego pliku w pamięci pod aktualnym *adresem transferu dysku* (*DTA*) tworzony jest opis pliku o długości 43 bajtów. Domyślnie adres ten ma przesunięcie 80h od PSP, ale zwykle jest ustawiany na adres wewnątrz segmentu danych. Do tego celu służy funkcja 1Ah. W tabeli przedstawiono zawartość DTA po odnalezieniu pliku:

Przesunięcie	Informacje o pliku
0 – 20	Zarezerwowane przez DOS
21	Atrybut
22 – 23	Znacznik czasu
24 – 25	Znacznik daty
26 – 29	Wielkość (dwusłowo)
30 – 42	Nazwa pliku (ciąg zakończony zerem)

Funkcja 4Eh pozwala na uzyskanie znaczników daty i czasu pliku bez konieczności otwarcia tego pliku. W przypadku niepowodzenia wyszukiwania ustawiany jest znacznik Carry, a rejestr AX będzie zawierał wartość 2 (nieprawidłowa ścieżka) lub 12 (brak dalszych plików). Ten drugi kod błędu oznacza, iż nie odnaleziono szukanych plików.

12.1.7. Odnalezienie kolejnego szukanego pliku (4Fh)

Po odnalezieniu pierwszego żądanego pliku przez funkcję 4Eh możliwe jest odnalezienie kolejnych plików zgodnych ze wzorcem wyszukiwania. Do tego zadania służy funkcja 4Fh. W tym przypadku zakłada się, iż wzorzec wyszukiwania zawiera znaki zastępcze, na przykład

PROG?.EXE lub **.ASM*. Funkcja `4Fh` wykorzystuje ten sam adres transferu dysku (DTA), który wykorzystuje funkcja `4Eh`; ponadto funkcja `4Fh` aktualizuje go o informacje dotyczące kolejnych plików. Po odnalezieniu wszystkich plików ustawiany jest znacznik `Carry`. Przedstawienie informacji zawartych w DTA znajduje się w opisie funkcji `4Eh` (podrozdział 12.1.6). W celu wywołania funkcji `4Fh` wystarczy umieścić jej numer w rejestrze `AH`:

```
mov   ah,4Fh        ; wyszukanie kolejnego pliku
int   21h
jc    no_more_files
```

12.1.8. Ustawienie adresu DTA (1Ah)

Adres transferu danych (*DTA*) to obszar pamięci przeznaczony do transferu pliku do pamięci. Początkowo był używany przez funkcje plików we wczesnych wersjach systemu DOS, kiedy w celu uzyskania dostępu do plików niezbędne było wykorzystanie bloków sterujących. Od pewnego momentu DTA jest stosowany przede wszystkim jako bufor dla funkcji `4Eh` (wyszukanie pierwszego pliku) i `4Fh` (wyszukanie kolejnych plików).

Funkcja `1Ah` może być użyta do ustawienia adresu DTA na położenie w segmencie danych. Domyślnie DTA wskazuje położenie mające przesunięcie 80h od początku PSP. W większości przypadków ten adres jest ustawiany na bufor wewnątrz segmentu danych, ponieważ domyślne położenie w PSP jest wykorzystywane do innych celów (na przykład do przechowywania parametrów wiersza poleceń dla programu). Poniższe instrukcje powodują ustawienie DTA na bufor o nazwie `myDTA`:

```
mov   ah,1Ah            ; ustawienie DTA
mov   dx,offset myDTA   ; na bufor w segmencie danych
int   21h
```

12.2. Zastosowanie — wyświetlenie nazw plików i informacji o dacie

Wykorzystując przedstawione w tym rozdziale informacje o sposobach wyszukiwania plików, a także informacje o formatach daty i czasu, możemy napisać program o nazwie *Date Stamp* (listing 12.1), który wyszukuje plik lub grupę plików, a następnie wyświetla nazwy plików i ich daty. W pewnym sensie odpowiada to zadaniom wykonywanym przez polecenie `DIR` w systemie DOS. Uruchamiając program poprzez wiersz poleceń, możliwe jest wprowadzenie specyfikacji pliku wraz ze znakami zastępczymi. Program *Date Stamp* powinien wykonywać następujące operacje:

- Odczytanie nazwy pliku wprowadzonej w wierszu poleceń. Jeśli nie wprowadzono żadnej nazwy, powinien być wyświetlany komunikat informujący o sposobie uruchamiania programu.
- Wyszukanie pierwszego pliku zgodnego ze wzorcem. Jeśli żaden plik nie zostanie odnaleziony, program powinien wyświetlać odpowiedni komunikat przed powrotem do systemu.
- Zdekodowanie znacznika daty i zachowanie w odpowiednich zmiennych wartości dnia, miesiąca i roku.

- Wyświetlenie nazwy pliku i jego daty utworzenia.
- Wyszukanie kolejnego pliku zgodnego ze wzorcem. Trzy ostatnie kroki powinny być powtarzane aż do momentu odnalezienia ostatniego pliku.

Listing 12.1. *Program Date Stamp*

```
title Program Date Stamp       (DAT.ASM)

; Program wyświetla nazwę i znacznik daty dla każdego
; pliku zgodnego ze specyfikacją pliku wprowadzoną
; w wierszu poleceń DOS. Program wykorzystuje makra
; i strukturę.

.model small
.stack 100h
.286
EOLN EQU <0dh,0ah>

FileControlBlock struc
           db 22 dup(?)        ; informacje nagłówka - nieużywane
  fileTime dw ?                ; znacznik czasu pliku
  fileDate dw ?                ; znacznik daty pliku
  fileSize dd ?                ; wielkość pliku - nieużywana
  fileName db 13 dup(0)        ; nazwa pliku odnaleziona przez DOS
FileControlBlock ends

mWriteint macro value, radix:=<10>
    push  ax
    push  bx
    mov   ax,value
    mov   bx,radix
    call  Writeint
    pop   bx
    pop   ax
endm

mWritestring macro aString
    push  dx
    mov   dx,offset aString
    call  Writestring
    pop   dx
endm
;-----------------------------------------------------
.data
filespec db  40 dup(0)         ; wiersz poleceń DOS
heading  db "Program Date Stamp       (DAT.EXE)"
         db  EOLN,EOLN,0
helpMsg  db "Prawidłowa składnia to:  "
         db "DAT [d:][path]filename[.ext]",EOLN,0
DTA      FileControlBlock <>
;-----------------------------------------------------
.code
extrn DOS_error:proc, Get_Commandtail:proc,         Str_length:proc, Writeint:proc,
Writestring:proc, Crlf:proc

main proc
     mov   bx,ds
     mov   ax,@data            ; inicjalizacja DS i ES
     mov   ds,ax
     mov   es,ax
```

```
     mov   dx,offset filespec  ; uzyskanie specyfikacji pliku
     call  Get_Commandtail     ; z wiersza poleceń
     jc    A2                  ; wyjście, jeśli nie odnaleziono
     mWritestring heading
     call  findFirst           ; wyszukanie pierwszego pliku
     jc    A3                  ; wyjście, jeśli nie odnaleziono

A1:  call  decodeDate          ; oddzielenie znacznika daty
     call  display_filename
     mov   ah,4Fh              ; wyszukanie kolejnego pliku
     int   21h
     jnc   A1                  ; kontynuacja wyszukiwania
     jmp   A3                  ; aż do odnalezienia wszystkich plików

A2:  mWritestring helpMsg      ; wyświetlenie pomocy

A3:  mov   ax,4C00h            ; wyjście z programu
     int   21h
main endp

; Odnalezienie pierwszego pliku zgodnego ze specyfikacją
; wprowadzoną w wierszu poleceń.

findFirst proc
     mov   ah,1Ah              ; ustawienie adresu transferu
     mov   dx,offset DTA
     int   21h
     mov   ah,4Eh              ; wyszukanie pierwszego pliku
     mov   cx,0                ; tylko normalny atrybut
     mov   dx,offset filespec
     int   21h
     jnc   B1                  ; w przypadku błędu DOS
     call  DOS_error           ; wyświetl komunikat
B1:  ret
findFirst endp

; Translacja zakodowanego formatu do znacznika daty pliku.

.data
month   dw  ?                  ; tymczasowe miejsce
day     dw  ?                  ; dla miesiąca, dnia i roku
year    dw  ?
.code
decodeDate proc
     mov   bx,offset DTA.fileDate
     mov   dx,[bx]             ; uzyskanie dnia
     mov   ax,dx
     and   ax,001Fh            ; skasowanie bitów 5-15
     mov   day,ax
     mov   ax,dx               ; uzyskanie miesiąca
     shr   ax,5                ; przesunięcie w prawo o 5 bitów
     and   ax,000Fh            ; skasowanie bitów 4-15
     mov   month,ax
     mov   ax,dx               ; uzyskanie roku
     shr   ax,9                ; przesunięcie w prawo o 9 bitów
     add   ax,80               ; rok jest relatywny do 1980
     mov   year,ax
     ret
```

```
decodeDate endp

; Zapisanie nazwy pliku i znacznika daty do konsoli.

display_filename proc
     mWritestring DTA.fileName
     call  fill_with_spaces
     mWriteint month
     call  write_dash            ; wyświetlenie znaku "-"
     mWriteint day
     call  write_dash            ; wyświetlenie znaku "-"
     mWriteint year
     call  Crlf
     ret
display_filename endp

; Uzupełnienie nazwy pliku o spacje po prawej stronie.

fill_with_spaces proc
     mov   cx,15                 ; maksymalna długość nazwy plus 3 spacje
     mov   di,offset DTA.fileName ; uzyskanie długości nazwy
     call  Str_length            ; AX = długość nazwy pliku
     sub   cx,ax
     mov   ah,2                  ; wyświetlenie znaku
     mov   dl,20h                ; spacja
E1:  int   21h                   ; zapisanie spacji
     loop  E1                    ; aż do momentu, w którym CX = 0
     ret
fill_with_spaces endp

write_dash proc                  ; zapisanie łącznika
     push  ax
     push  dx
     mov   ah,2
     mov   dl,'-'
     int   21h
     pop   dx
     pop   ax
     ret
write_dash endp
end main
```

Procedura main

Procedura `main` wywołuje podprocedury służące do uzyskania ogona poleceń i wyszukania pierwszego pliku zgodnego ze wzorcem. Od tego momentu program działa praktycznie w pętli, która dekoduje i wyświetla datę oraz wyszukuje kolejne pliki.

Procedura FindFirst

Procedura `FindFirst` wywołuje funkcję `1Ah` w celu ustawienia adresu DTA, gdzie zapisywane są informacje odnoszące się do odnalezionych plików. Funkcja `4Eh` wyszukuje pierwszy plik zgodny ze wzorcem i powraca do `main`. Znacznik `Carry` jest ustawiany po odnalezieniu ostatniego pliku.

Procedura DecodeDate

Procedura DecodeDate jest najbardziej złożona, ponieważ dla każdego pola (dzień, miesiąc i rok) należy zastosować maskę, a następnie przesunąć bity w prawo. Poszczególne wartości są zapisywane w oddzielnych zmiennych. Dzień tygodnia zajmuje bity 0. – 4; aby uzyskać tę wartość, wystarczy skasować bity 5. – 15. i zapisać wynik w day. Numer miesiąca znajduje się w bitach 5. – 8. Aby go uzyskać, należy przesunąć AX o 5 bitów w prawo, a następnie skasować pozostałe bity i zachować wynik w month. Numer roku można odnaleźć w bitach 9. – 15., a więc należy przesunąć rejestr AX o 9 bitów w prawo. Do wyniku w zmiennej year dodawana jest liczba 80, ponieważ data jest zawsze relatywna wobec roku 1980.

12.3. Usługi wejścia-wyjścia

12.3.1. Tworzenie pliku (3Ch)

Funkcja 3Ch umożliwia utworzenie nowego pliku lub zmniejszenie rozmiaru już istniejącego pliku do 0 bajtów. Plik jest automatycznie otwierany do odczytu i zapisu, ale można zmienić to poprzez wywołanie funkcji 43h (zmiana trybu pliku) natychmiast po jego otwarciu. DS:DX musi wskazywać ciąg *ASCIIZ* z nazwą pliku, a rejestr CX powinien zawierać wybrane wartości atrybutu:

- 00h — normalny plik,
- 01h — plik tylko do odczytu,
- 02h — plik ukryty,
- 04h — plik systemowy (rzadko używany atrybut).

Poniżej przedstawiono procedurę, która tworzy plik z normalnym atrybutem. Plik jest umieszczany na domyślnym dysku w aktualnym katalogu. Do przekazania przesunięcia nazwy pliku do procedury używany jest rejestr DX:

```
CreateFile proc           ; wejście: DX wskazuje nazwę pliku
   push  cx
   push  dx
   mov   ah,3Ch           ; funkcja: tworzenie pliku
   mov   cx,0             ; normalny atrybut
   int   21h              ; wywołanie DOS
   pop   dx
   pop   cx
   ret
CreateFile endp
```

Poniższy kod przedstawia sposób wywołania procedury CreateFile:

```
.data
newfile  db "NEWFILE.DOC",0
handle   dw ?

.code
mov    dx,offset newfile  ; przekazanie przesunięcia nazwy pliku
call   CreateFile         ; utworzenie pliku
jc     display_error      ; wyświetlenie komunikatu o błędzie
mov    handle,ax          ; brak błędu: zapisanie uchwytu
```

Jeśli otwarcie pliku powiodło się, to w rejestrze AX zwracany jest 16-bitowy uchwyt pliku. Uchwyt ma wartość 5, jeśli jest to pierwszy otwarty plik. W przypadku większej liczby plików ta wartość wzrasta.

Ochrona istniejących plików

Jedną z wad funkcji 3Ch jest możliwość przypadkowego skasowania istniejącego pliku poprzez utworzenie nowego pliku o tej samej nazwie. Na szczęście dostępnych jest kilka sposobów rozwiązania tego problemu. Pierwszy z nich to próba otwarcia pliku do odczytu przy użyciu funkcji 3Dh. Jeśli w wyniku tej operacji zostanie ustawiony rejestr Carry, a AX będzie miał wartość 2 (nie odnaleziono pliku), to możliwe jest bezpieczne użycie funkcji do tworzenia plików.

Innym rozwiązaniem jest użycie funkcji 5Bh (utworzenie nowego pliku). Funkcja przerwie swoje działanie i zgłosi błąd 50h, jeśli plik już istnieje. Poniżej przedstawiono odpowiedni przykład zastosowania tej funkcji:

```
.data
newfile   db "FILE1.DOC",0

.code
mov   ah,5Bh            ; utworzenie nowego pliku
mov   cx,0              ; normalny atrybut
mov   dx,offset newfile
int   21h
jc    error_routine
```

Kody błędów

Jeśli DOS ustawi znacznik Carry, to zwracany znacznik błędu może mieć wartość 3, 4 lub 5. Błąd 3. (nie odnaleziono ścieżki) oznacza, iż wskazywany przez rejestr DX specyfikator pliku prawdopodobnie zawiera nieistniejącą nazwę katalogu. W poniższym poleceniu zamiast prawidłowej nazwy podkatalogu *ASM* podano *ASMS*:

```
File1   db 'C:\ASMS\FILE1.ASM',0
```

Błąd 4. (zbyt dużo otwartych plików) występuje, jeśli przekroczono maksymalną liczbę otwartych plików, jaka została zdefiniowana w DOS. Domyślnie DOS pozwala na otwarcie tylko ośmiu plików. Ponieważ pierwsze pięć jest używane przez DOS (jako standardowe uchwyty plików), to pozostałe aplikacje mogą wykorzystać tylko trzy pozostałe. Zmiana dopuszczalnej liczby otwartych plików jest możliwa poprzez polecenie FILES znajdujące się w pliku *CONFIG.SYS*, który jest odczytywany w czasie startu komputera, na przykład:

```
files=32
```

Po odliczeniu pięciu uchwytów DOS dostępnych będzie 27 uchwytów dla innych aplikacji. Każdy program może otworzyć jednak tylko 20 plików. Zmiana tego domyślnego ustawienia jest możliwa poprzez wywołanie funkcji 67h przerwania INT 21h. W rejestrze BX należy umieścić liczbę uchwytów (od 1 do 65534). Poniższy fragment kodu ustawia 30 jako maksymalną liczbę plików dla pojedynczego programu:

```
mov   ah,67h
mov   bx,30
int   21h
```

Błąd 5. (dostęp zastrzeżony) wskazuje, iż wykonano próbę utworzenia już istniejącego pliku z atrybutem tylko do odczytu. Ten błąd pojawia się jako wynik próby utworzenia pliku o tej samej nazwie, jaką posiada podkatalog na dysku lub jako wynik dodania nowego wpisu do katalogu głównego, który jest już pełny.

W niektórych wersjach systemu DOS błąd 2. jest generowany w przypadku umieszczenia znaku powrotu karetki na końcu nazwy pliku.

12.3.2. Otwarcie pliku (3Dh)

Funkcja 3Dh otwiera istniejący plik w jednym z trzech trybów: wejścia, wyjścia lub wejścia-wyjścia. Rejestr AL zawiera tryb, a DS:DX wskazuje nazwę pliku. Możliwe jest otworzenie plików normalnych i ukrytych. Jeśli próba otwarcia pliku powiedzie się, to jego uchwyt jest zwracany w rejestrze AX:

```
.data
filename      db 'A:\FILE1.DOC',0
infilehandle  dw ?

.code
mov   ah,30h              ; funkcja: otwarcie pliku
mov   al,0                ; tryb pracy
mov   dx,offset filename
int   21h                 ; wywołanie DOS
jc    display_error       ; wyświetlenie komunikatu o błędzie
mov   infilehandle,ax     ; brak błędu: zapisanie uchwytu
```

Tryb pliku

Umieszczany w rejestrze AL tryb pliku może przyjąć jedną z następujących wartości:

AL	Tryb
0	Wejście (tylko do odczytu)
1	Wyjście (tylko zapis)
2	Wejście-wyjście

Do otwarcia pliku w trybie wyjścia w celu dokonania sekwencyjnego zapisu najlepszym wyborem będzie funkcja 3Ch (utwórz plik). Z drugiej strony, funkcja 3Dh najlepiej sprawdza się w czasie odczytu i zapisu danych do pliku. Funkcje wejścia-wyjścia swobodnego dostępu także wymagają funkcji 3Dh.

Kody błędów

Jeśli znacznik CF ma wartość 1, to rejestr AX zawiera jeden z kodów błędów. Błąd 1. (niewłaściwy numer funkcji) oznacza próbę udostępnienia pliku bez załadowania programu *SHARE*. Błąd 2. (nie odnaleziono pliku) sygnalizuje, iż DOS nie był w stanie odnaleźć żądanego pliku. Błąd 3. (nie odnaleziono ścieżki) oznacza, iż w ścieżce pliku użyto nieprawidłowej nazwy katalogu. Błąd 4. wskazuje zbyt dużą liczbę otwartych plików. Z kolei błąd 5. (dostęp zastrzeżony) oznacza, że plik ma atrybut tylko do odczytu lub podana nazwa jest nazwą podkatalogu lub woluminu.

12.3.3. Zamknięcie pliku (3Eh)

Aby zamknąć plik, należy umieścić uchwyt pliku w rejestrze BX i wywołać funkcję 3Eh. Ta funkcja opróżnia wewnętrzny bufor plików i zapisuje odpowiednie dane na dysk, a następnie zwalnia uchwyt pliku. Jeśli plik został zapisany, to zmieniana jest jego wielkość oraz znaczniki daty i czasu. Poniższy fragment kodu powoduje zamknięcie pliku identyfikowanego przez infilehandle:

```
.data
infile         db 'B:\FILE1.DOC',0
infilehandle   dw ?

.code
mov   ah,3Ah           ; zamknięcie uchwytu pliku
mov   bx,infilehandle
int   21h
jc    display_error
```

Jedyny błąd, jaki może się pojawić, to błąd 6. (*niewłaściwy uchwyt*), sygnalizujący, że uchwyt pliku w rejestrze BX nie odnosi się do otwartego pliku.

12.3.4. Odczyt z pliku lub urządzenia (3Fh)

W rozdziale 5. pokazałem sposób użycia funkcji 3Fh do odczytu ze standardowego wejścia, którym zwykle jest klawiatura. Ta funkcja jest bardzo elastyczna, ponieważ pozwala także na odczyt danych z pliku dyskowego. Najpierw jednak należy wywołać funkcję 3Dh w celu otworzenia pliku w trybie wejścia. Po uzyskaniu uchwytu pliku można wywołać funkcję 3Fh i rozpocząć odczyt danych.

Jeśli wywołanie funkcji spowoduje ustawienie znacznika Carry, to w rejestrze AX znajdzie się kod błędu 5. lub 6. Błąd 5. (*dostęp zastrzeżony*) oznacza zwykle, iż plik został otwarty w trybie wyjścia. Błąd 6. (*nieprawidłowy uchwyt*) wskazuje, iż przekazany w rejestrze BX uchwyt pliku nie odnosi się do otwartego pliku. Jeśli po wykonaniu funkcji 3Fh znacznik Carry nie został ustawiony, to rejestr AX zawiera liczbę odczytanych bajtów.

Informacje zwracane przez funkcję 3Fh są przydatne, jeśli program sprawdza osiągnięcie końca pliku. Jeżeli w pliku nie ma już więcej danych do odczytania, to wartość AX jest mniejsza niż liczba żądanych bajtów (rejestr CX). Poniższy fragment kodu powoduje skok do etykiety Exit, jeśli osiągnięto koniec pliku:

```
 .data
bufferSize = 512
filehandle   dw ?
buffer   db bufferSize dup(0)

.code
mov   ah,3Fh               ; odczyt z pliku lub urządzenia
mov   bx,filehandle        ; BX = uchwyt pliku
mov   cx,buffersize        ; liczba bajtów do odczytania
mov   dx,offset buffer     ; wskazanie buforu
int   21h                  ; odczytanie danych
jc    Display_error        ; błąd jeśli CF = 1
cmp   ax,cx                ; porównanie z żądaną liczbą bajtów
jb    Exit                 ; osiągnięto koniec pliku
```

12.3.5. Zapis do pliku lub urządzenia (40h)

Funkcja 40h jest używana w czasie zapisywania danych do urządzenia lub pliku. W celu użycia tej funkcji należy umieścić poprawny uchwyt pliku w rejestrze BX i liczbę zapisywanych bajtów w CX. DS:DX musi wskazywać bufor zawierający zapisywane dane. DOS automatycznie zaktualizuje wskaźnik pliku po wykonaniu operacji zapisu, przez co następne wywołanie funkcji 40h spowoduje zapis danych od kolejnej pozycji. Kod z poniższego przykładu zapisuje zawartość buforu buffer do pliku identyfikowanego przez uchwyt handle:

```
.data
buffer   db 100h dup(?)     ; bufor wyjściowy
handle   dw ?               ; uchwyt pliku

.code
write_to_file:
   mov   ah,40h             ; zapis do pliku lub urządzenia
   mov   bx,handle          ; uchwyt pliku zwracany przez OPEN
   mov   cx,100h            ; liczba bajtów do zapisania
   mov   dx,offset buffer   ; DX wskazuje bufor wyjściowy
   int   21h                ; wywołanie DOS
   jc    display_error      ; wyświetlenie komunikatu o błędzie
   cmp   ax,100h            ; zapisano wszystkie dane?
   jne   close_file         ; dysk jest pełny
```

Jeśli wywołanie funkcji spowoduje ustawienie znacznika Carry, to w rejestrze AX znajdzie się kod błędu 5. lub 6. Błąd 5. (dostęp zastrzeżony) oznacza zwykle, iż plik został otwarty w trybie wejścia lub posiada atrybut tylko do odczytu. Błąd 6. (nieprawidłowy uchwyt) wskazuje, iż przekazany w rejestrze BX uchwyt pliku nie odnosi się do otwartego pliku. Jeśli Carry nie został ustawiony, ale AX zawiera liczbę mniejszą niż żądana liczba bajtów, to prawdopodobnie oznacza to wystąpienie błędu wejścia-wyjścia, na przykład na dysku nie ma już wolnego miejsca.

12.4. Bezpośredni dostęp do plików

Bezpośrednie przetwarzanie plików jest zaskakująco proste w języku asembler. Do wcześniej przedstawionej grupy funkcji wystarczy dodać funkcję 42h (przesunięcie wskaźnika pliku), która umożliwia odnalezienie dowolnego rekordu w pliku. Poszczególne języki wysokiego poziomu wymagają zastosowania różnego rodzaju składni. Z kolei w systemie DOS różnice między sekwencyjnym i bezpośrednim dostępem do plików są minimalne.

Bezpośredni dostęp jest możliwy tylko wtedy, gdy poszczególne rekordy w pliku mają *stałą wielkość*. Dzieje się tak, gdyż wielkość rekordu jest używana do obliczenia przesunięcia rekordów od początku pliku. Plik tekstowy ma zwykle rekordy o zmiennej długości. Rekordy te są rozdzielone symbolami końca wiersza (0Dh i 0Ah). Z tego powodu nie jest możliwe ustalenie położenia poszczególnych rekordów, ponieważ ich przesunięcie nie jest zależne od ich wielkości.

W poniższym przykładzie *Plik1* ma rekordy o stałej długości. Początek każdego rekordu jest obliczany poprzez odjęcie 1 od numeru rekordu i pomnożenie wyniku przez 20. *Plik2* zawiera te same dane, ale są one przechowywane w pliku tekstowym rozgraniczonym przecinkami. Pomiędzy poszczególnymi polami znajdują się przecinki, a na końcu każdego

rekordu znaczniki końca wiersza (0Dh i 0Ah). Położenie rekordu nie może być obliczone, ponieważ rekordy mają zmienną wielkość. Rekord 2. rozpoczyna się od przesunięcia 000F, rekord 3. od przesunięcia 0022 itd.

Plik1. Przesunięcia rekordów (szesnastkowo) 0000, 0014, 0028, 003C:

```
          1               2               3               4
0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
1000AU  00300H1003BAKER  02000B2001DAVIDSON  40000H3000GONZALEZ  5000A
```

Plik2. Przesunięcia rekordów (szesnastkowo) 0000, 0014, 0028, 003C:

```
          1               2               3               4
0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
1000,AU,300,H..1003,BAKER,2000,B..2001,DAVIDSON,40000,H..3000,GONZALEZ,5000,A..
```

12.4.1. Przesunięcie wskaźnika pliku (42h)

Funkcja `42h` powoduje przeniesienie wskaźnika pliku do nowego położenia. Plik musi być otwarty przed wywołaniem tej funkcji. Parametry wejściowe to:

- `AH` — wartość 42h,
- `AL` — kod metody (typ przesunięcia),
- `BX` — uchwyt pliku,
- `CX` — przesunięcie, górna część,
- `DX` — przesunięcie, dolna część.

Przesunięcie może być relatywne do początku pliku, końca pliku lub aktualnego położenia w pliku. W czasie wywołania funkcji `AL` zawiera tak zwany *kod metody*, który identyfikuje sposób ustawienia wskaźnika. W `CX:DX` znajduje się 32-bitowe przesunięcie:

AL	Zawartość CX:DX
0	Przesunięcie od początku pliku
1	Przesunięcie od aktualnego położenia
2	Przesunięcie od końca pliku

Wyniki funkcji

Jeśli funkcja spowoduje ustawienie znacznika `Carry`, to DOS zwróci błąd 1. (niewłaściwy numer funkcji) lub błąd 6. (niepoprawny uchwyt pliku). Jeśli działanie funkcji zakończy się sukcesem, to znacznik `Carry` zostanie skasowany, a w `DX:AX` znajdzie się nowe położenie wskaźnika pliku, które będzie relatywne do początku pliku niezależnie od użytego kodu metody.

Przykład: wyszukanie rekordu

Załóżmy, iż przetwarzany jest plik z rekordami o wielkości 80 bajtów każdy i konieczne jest odnalezienie konkretnego rekordu. Procedura `Lseek` (listing 12.2) przesuwa wskaźnik pliku do położenia zależnego od numeru rekordu przekazanego w rejestrze `AX`. Jeśli numery rekordów rozpoczynają się od 0, to w celu uzyskania przesunięcia rekordu należy pomnożyć jego numer przez jego wielkość.

Listing 12.2. *Użycie procedury Lseek przeznaczonej do wyszukania rekordu*

```
Lseek proc                  ; AX = numer rekordu
   push  bx
   puxh  cx
   mov   bx,80              ; DX:AX = (AX * 80)
   mul   bx
   mov   cx,dx              ; górna połowa przesunięcia w CX
   mov   dx,ax              ; dolna połowa przesunięcia w CX
   mov   ah,42h
   mov   al,0               ; metoda: przesunięcie od początku
   mov   bx,handle
   int   21h                ; przesunięcie wskaźnika pliku
   pop   cx
   pop   bx
   ret
Lseek endp
```

Na przykład rekord 9. będzie miał przesunięcie 720, a rekord 0. — przesunięcie 0:

```
przesunięcie = 9 * 80 = 720
przesunięcie = 0 * 80 = 0
```

Procedura `ReadRecord` z listingu 12.3 wykorzystuje funkcję `3Fh` do odczytania 80 bajtów z pliku. Aby tego dokonać, należy umieścić w rejestrze `AX` numer żądanego rekordu, a następnie wywołać kolejno procedury `Lseek` i `ReadRecord`:

```
mov    ax,numer_rekordu
call   Lseek
call   ReadRecord
```

Listing 12.3. *Procedura ReadRecord*

```
ReadRecord proc
   pusha
   mov   ax,3Fh             ; odczyt z pliku lub urządzenia
   mov   bx,handle          ; uchwyt pliku lub urządzenia
   mov   cx,80              ; liczba bajtów do odczytania
   mov   dx,offset buffer
   int   21h
   popa
   rer
ReadRecord endp
```

Przykład: dołączenie danych do pliku

Funkcja `42h` może być użyta także do dołączenia danych do pliku. W tym przypadku można użyć plik tekstowy z rekordami o zmiennej wielkości lub plik z rekordami o stałej wielkości.

Należy pamiętać, aby przed dołączeniem nowych rekordów użyć kodu metody 2. do umieszczenia wskaźnika pliku na końcu tego pliku. Odpowiedni kod stanowi część procedury `SeekEOF` z listingu 12.4.

Listing 12.4. *Procedura SeekEOF*

```
SeekEOF proc
   pusha
   mov   ah,42h          ; ustawienie wskaźnika pliku
   mov   al,2            ; w odniesieniu do końca pliku
   mov   bx,handle
   mov   cx,0            ; górna część przesunięcia
   mov   dx,0            ; dolna część przesunięcia
   int   21h
   popa
   ret
SeekEOF proc
```

Użycie ujemnego przesunięcia

Jeśli kod metody w rejestrze AL ma wartość 1 lub 2, to wartość przesunięcia może być dodatnia lub ujemna, co zapewnia kilka ciekawych możliwości. Program może na przykład cofnąć wskaźnik pliku (przy użyciu metody 1.) i ponownie odczytać rekord. Ta metoda działa poprawnie nawet w przypadku pliku tekstowego o rekordach zmiennej wielkości:

```
mov   ah,42h            ; funkcja: umieszczenie wskaźnika
mov   al,1              ; relatywnie do aktualnego położenia
mov   bx,handle
mov   cx,0
mov   dx,-10            ; cofnięcie o 10 bajtów
int   21h
jc    error_routine     ; wyjście w przypadku błędu
mov   ah,3Fh            ; funkcja: odczyt pliku
mov   cx,10             ; odczyt 10 bajtów
mov   dx,offset input
int   21h
```

12.5. Odczyt pliku bitmapowego

W tej części rozdziału zostanie przedstawiona procedura o nazwie ShowBMP, która pozwala na załadowanie z pliku *rysunku bitmapowego* i wyświetlenie go na ekranie. Rysunek bitmapowy może mieć maksymalną rozdzielczość 320×200 z 256 kolorami. Kod procedury przedstawiono w listingu 12.5.

Listing 12.5. *Odczytanie i wyświetlenie pliku bitmapy*

```
; Program  wyświetlający bitmapy (bitmap.asm)

; Procedura ShowBMP ładuje plik BMP typu Windows
; i wyświetla go na ekranie. Parametry wejściowe:
; DS:DX wskazuje ciąg ASCIIZ zawierający ścieżkę
; do pliku BMP. Maksymalna rozdzielczość rysunku
; to 320×200 z 256 kolorami

.model  small
.186
.code
```

```
extrn   OpenInputFile:proc, CloseFile:proc

PUBLIC  ShowBMP

ShowBMP proc
pusha                           ; Zachowanie rejestrów
call    OpenInputFile           ; Otwarcie pliku wskazywanego przez DS:DX
jc      FileErr                 ; Błąd? Wyświetlenie komunikatu i wyjście
mov     bx,ax                   ; Umieszczenie uchwytu pliku w BX
call    ReadHeader              ; Odczyt 54-bajtowego nagłówka z informacjami o pliku
jc      InvalidBMP              ; Plik BMP nieprawidłowy? Komunikat błędu i wyjście
call    ReadPal                 ; Odczytanie palety BMP i umieszczenie w buforze
push    es
call    InitVid                 ; Ustawienie trybu VGA 320x200
call    SendPal                 ; Wysłanie palety do rejestrów wideo
call    LoadBMP                 ; Załadowanie i wyświetlenie grafiki
call    CloseFile               ; Zamknięcie pliku
pop     es

jmp     ProcDone

FileErr:
mov     ah,9
mov     dx,offset msgFileErr
int     21h
jmp     ProcDone

InvalidBMP:
mov     ah,9
mov     dx,offset msgInvBMP
int     21h

ProcDone:
popa                            ; Przywrócenie rejestrów
ret
ShowBMP endp

; Sprawdzenie dwóch pierwszych bajtów pliku. Jeśli nie
; są zgodne ze standardowym nagłówkiem BMP ("BM"),
; to ustawiany jest znacznik Carry.

CheckValid proc
clc
mov     si,offset Header
mov     di,offset BMPStart
mov     cx,2                    ; identyfikator BMP ma dwa bajty
CVloop:
mov     al,[si]                 ; uzyskanie bajta z nagłówka
mov     dl,[di]
cmp     al,dl                   ; Czy jest to właściwy nagłówek?
jne     NotValid                ; Jeśli nie, to ustawienie Carry
inc     si
inc     di
loop    CVloop

jmp     CVdone

NotValid:
stc
```

```
CVdone:
ret
CheckValid      endp

GetBMPInfo      proc
; Ta procedura pobiera z nagłówka ważne informacje o pliku BMP
; i umieszcza je w odpowiednich zmiennych.

mov     ax,header[0Ah]        ; AX = przesunięcie początku danych graficznych
sub     ax,54                 ; Odjęcie długości nagłówka
shr     ax,2                  ; i podzielenie przez 4,
mov     PalSize,ax            ; aby uzyskać liczbę kolorów w pliku
                              ; (Każdy wpis palety ma 4 bajty)
mov     ax,header[12h]        ; AX = rozdzielczość pozioma (szerokość)
mov     BMPWidth,ax           ; Zapisanie rozdzielczości
mov     ax,header[16h]        ; AX = rozdzielczość pionowa (wysokość)
mov     BMPHeight,ax          ; Zapisanie rozdzielczości
ret
GetBMPInfo      endp

InitVid proc
; Ta procedura inicjalizuje tryb wideo i ustawia ES
; na pamięć wideo.

mov     ax,13h
int     10h                   ; Ustawienie trybu wideo na 320x200x256.
push    0A000h
pop     es                    ; ES = A000h (segment wideo)
ret
InitVid endp

LoadBMP proc
; Pliki BMP są zapisywane do góry nogami. Ta procedura odczytuje
; kolejne wiersze i wyświetla je od dołu do góry.
; Początkowy wierszy jest zależny od rozdzielczości pionowej,
; a lewy górny wierzchołek grafiki zawsze znajdzie się w lewym
; górnym rogu ekranu.

; Pamięć wideo jest dwumiarową tablicą bajtów pamięci. Poszczególne
; bajty można modyfikować oddzielnie.  Każdy bajt reprezentuje
; piksel na ekranie i zawiera kolor punktu w tym położeniu.

mov     cx,BMPHeight           ; Wyświetlenie wierszy pliku BMP
ShowLoop:
push    cx
mov     di,cx                 ; Utworzenie kopii CX
shl     cx,6                  ; Mnożenie CX przez 64
shl     di,8                  ; Mnożenie DI przez 256
add     di,cx                 ; DI = CX * 320; DX wskazuje
                              ; piksel w żądanym wierszu

mov     ah,3fh
mov     cx,BMPWidth
mov     dx,offset ScrLine
int     21h                   ; Odczytanie jednego wiersza do buforu

cld                           ; Skasowanie znacznika kierunku dla movsb
mov     cx,BMPWidth
mov     si,offset ScrLine
rep     movsb                 ; Skopiowanie wiersza z buforu na ekran
```

```
pop     cx
loop    ShowLoop
ret
LoadBMP endp

; Ta procedura sprawdza, czy dany plik jest poprawnym plikiem BMP,
; a także pobiera pewne informacje o rysunku.

ReadHeader proc
mov     ah,3fh
mov     cx,54
mov     dx,offset Header
int     21h                     ; Odczytanie nagłówka pliku do buforu.

call    CheckValid              ; Czy poprawny plik BMP?
jc      RHdone                  ; Nie? Wyjście
call    GetBMPInfo              ; Tak: Przetwarzanie nagłówka

RHdone:
ret
ReadHeader endp

; Odczytanie palety kolorów

ReadPal proc
mov     ah,3fh
mov     cx,PalSize              ; CX = liczba kolorów w palecie
shl     cx,2                    ; CX = mnożenie przez 4, aby uzyskać
                                ; wielkość palety (w bajtach)
mov     dx,offset palBuff
int     21h                     ; Umieszczenie palety w buforze
ret
ReadPal endp

SendPal proc
; Ta procedura odczytuje bufor palety i przesyła odpowiednie
; informacje do rejestrów wideo. Jeden bajt jest wysyłany przez
; port 3C8h i zawiera numer pierwszego koloru w palecie (0 = pierwszy ; kolor).
Kolejne informacje o kolorach RGB (dowolne numery kolorów)
; są przesyłane poprzez 3C9h.

mov     si,offset palBuff       ; Wskazanie buforu zawierającego paletę
mov     cx,PalSize              ; CX = Liczba kolorów do wysłania
mov     dx,3c8h
mov     al,0                    ; Rozpoczęcie od koloru 0
out     dx,al
inc     dx                      ; DX = 3C9h.
sndLoop:
; Uwaga: Kolory w pliku BMP są zapisywane jako wartości BGR,
; a nie RGB.

mov     al,[si+2]               ; Uzyskanie wartości czerwonego koloru
shr     al,2                    ; Maksymalnie 255, ale możliwe jest
                               ; wyświetlenie wartości tylko do 63.
                               ; Dzielenie przez 4 zapewnia dobry wynik
out     dx,al                   ; Wysłanie wartości
mov     al,[si+1]               ; Uzyskanie wartości zielonego koloru
```

```
shr     al,2
out     dx,al                ; Wysłanie wartości
mov     al,[si]              ; Uzyskanie wartości niebieskiego koloru
shr     al,2
out     dx,al                ; Wysłanie wartości

add     si,4                 ; Wskazanie kolejnego koloru
                             ; (Po każdym kolorze znajduje się
                             ; pusty znak)
loop    sndLoop
ret
SendPal endp

.data
Header          label word
HeadBuff        db 54 dup('H')
palBuff         db 1024 dup('P')
ScrLine         db 320 dup(0)

BMPStart        db 'BM'

PalSize         dw ?
BMPHeight       dw ?
BMPWidth        dw ?

msgInvBMP       db "Nieprawidłowy plik BMP.",7,0Dh,0Ah,24h
msgFileErr      db "Błąd w czasie otwierania pliku.",7,0Dh,0Ah,24h
end
```

W czasie wywołania procedury `ShowBMP` rejestr `DS:DX` musi wskazywać nazwę pliku zakończoną pustym bajtem. Wewnątrz tej procedury wywoływana jest podprocedura `OpenInputFile` z biblioteki konsolidowanej. Praca programu jest przerywana, jeśli procedura `OpenInputFile` nie może otworzyć pliku. Kolejnym krokiem jest odczytanie rekordu nagłówka bitmapy. Procedura `ReadHeader` zapisuje 54 bajty do buforu i wywołuje `CheckValid` w celu sprawdzenia poprawności nagłówka.

Procedura `CheckValid` szuka ciągu `BM` na początku pliku. Program wywołuje teraz `GetBMPInfo` w celu odczytania *pełnego rekordu* nagłówka bitmapy. Można tu odnaleźć informacje o przesunięciu początku danych rysunku, liczbie kolorów oraz o rozdzielczości pionowej i poziomej.

Procedura `ReadPal` wczytuje do pamięci paletę graficzną. Polega to na sprawdzeniu liczby kolorów i załadowaniu pełnej palety do zmiennej. Procedura `InitVid` inicjalizuje kolorowy tryb graficzny, a `LoadBMP` powoduje wyświetlenie rysunku bitmapowego. W czasie tego procesu uwzględniany jest fakt, iż dane wewnątrz plików BMP są zapisywane w odwrotnej kolejności. Program odczytuje kolejne wiersze rysunku, co oznacza jednak pewne spowolnienie jego pracy.

Program odczytujący i wyświetlający plik bitmapy stanowi jedynie zarys techniki wyświetlania bitmap, ale po rozbudowaniu tego programu o dodatkowe funkcje możliwe jest na przykład umieszczenie rysunku w dowolnym miejscu ekranu.

12.6. Pytania kontrolne

1. Jeśli dany plik jeszcze nie istnieje, to co się stanie, jeśli za pomocą funkcji 3Dh plik zostanie otwarty w trybie wyjścia?
2. Jeśli plik zostanie utworzony przez funkcję 3Ch, to czy będzie możliwy odczyt i zapis do tego pliku przed jego zamknięciem? Co się stanie, jeśli plik zostanie utworzony z atrybutem tylko do odczytu?
3. Jakie kroki należy wykonać, aby utworzyć nowy plik i mieć pewność, że istniejący plik o tej samej nazwie nie zostanie skasowany?
4. Dla każdego z poniższych kodów błędów, jakie są zwracane przez wywołanie INT 21h, napisz krótkie wyjaśnienie przyczyny błędu:

Numer błędu	Wywoływana funkcja
03h	56h (zmiana nazwy pliku)
05h	41h (usunięcie pliku)
06h	57h (ustawienie daty i czasu)
10h	3Ah (usunięcie katalogu)
11h	56h (zmiana nazwy pliku)
12h	4Eh (odnalezienie pierwszego szukanego pliku)

5. Co się stanie po uruchomieniu następującego kodu:

```
.data
filename   db 'FIRST.RND',0

.code
mov   ah,3Dh
mov   al,2
mov   dx,offset filename
int   21h
```

6. Czy w celu zamknięcia pliku należy użyć rejestru DX do wskazania jego nazwy?
7. Jaki będzie efekt użycia następujących instrukcji?

```
mov   ah,3Eh
mov   bx,0
int   21h
```

8. Wyobraź sobie, iż wywołano funkcję 3Eh (odczyt z pliku lub urządzenia). Znacznik Carry został ustawiony, a rejestr AX ma wartość 6. Co to oznacza?
9. Wyobraź sobie, iż wywołano funkcję 3Eh z CX = 80h. DOS skasował znacznik Carry i zwrócił wartość 20h w rejestrze AX. Co to oznacza?
10. Jeśli użyto funkcji 3Eh do odczytu danych z klawiatury, a CX = 0Ah, to jaka będzie zawartość buforu wejściowego po wprowadzeniu poniższego ciągu:

```
1234567890
```

11. Czy ciąg zapisywany do konsoli przez funkcję 40h musi być zakończony bajtem o wartości zero?

12. Wyobraź sobie, że użyto funkcji 40h do zapisania danych do pliku. Czy DOS automatycznie zaktualizuje wskaźnik pliku?
13. Jakie kroki należy wykonać, jeśli odczytano rekord z pliku i konieczne jest ponowne zapisanie go w tym samym miejscu?
14. Czy jest możliwe przeniesienie wskaźnika pliku wewnątrz pliku tekstowego?
15. Napisz kod pozwalający na umieszczenie wskaźnika pliku w położeniu 20 bajtów od końca pliku identyfikowanego przez uchwyt filehandle.
16. Jaka jest wartość przesunięcia dwudziestego rekordu w pliku, zawierającego rekordy o stałej wielkości 50 bajtów?
17. Jaki jest cel buforowania rekordów wejściowych?
18. Zakładając, iż w poniższych polach odwzorowanych bitowo bity 0. – 4. zawierają numer bloku, a bity 5. – 7. stanowią numer piętra, uzupełnij poniższe wartości:

```
11000101   piętro =   budynek =
00101001   piętro =   budynek =
01010101   piętro =   budynek =
```

19. Celem poniższej procedury Write_Buffer jest zapisanie zawartości buforu buffer do pliku identyfikowanego przez uchwyt filehandle. Zmienna buflen zawiera aktualną wielkość buforu. Jeśli na dysku nie ma już miejsca, to procedura powinna wyświetlić odpowiedni komunikat. Jakie błędy logiczne popełniono w tej procedurze?

```
.data
filehandle  dw ?
buflen      dw ?
buffer   db 80 dup(?)
message  db 'Na dysku nie ma miejsca,$'

.code
write_buffer proc
    mov   ah,40h
    mov   bx,filehandle
    mov   cx,buflen
    mov   dx,offset buffer
    int   21h
    jnc   L1
    mov   dx,offset message
    call  display
L1: ret
write_buffer endp
```

12.7. Zadania programistyczne

1. Narzędzie touch.

   Już od wielu lat programiści używają narzędzia touch, które odczytuje specyfikator pliku z wiersza poleceń (włącznie ze znakami zastępczymi) i zmienia znaczniki daty oraz czasu plików na aktualną datę i godzinę. Napisz taki program w języku asembler. Jeśli użytkownik wpisze na przykład następujące polecenie, to zostaną zaktualizowane wszystkie pliki z rozszerzeniem *.ASM* w aktualnym katalogu:

```
touch *.asm
```

Narzędzie może być użyte na przykład do ustawienia wszystkich plików aplikacji wysyłanych do klienta na tę samą datę i godzinę.

2. Wyszukiwanie ciągów tekstowych.

   Napisz program, który otwiera plik tekstowy zawierający co najmniej 60 kB tekstu i wyszukuje w nim określony ciąg (niezależnie od wielkości liter). Ciąg i nazwa pliku są wprowadzane w wierszu poleceń. Wyświetl na ekranie wszystkie wiersze pliku tekstowego, w których pojawia się dany ciąg. Poszczególne wiersze powinny być poprzedzone numerem danego wiersza, na przykład:

```
>search wiersz plik1.txt

 2: To jest wiersz 2.
10: W wierszu 10 jest jeszcze więcej znaków.
11: To jest bardzo długi wiersz, w którym jest dużo znaków.
```

3. Ulepszone wyszukiwanie ciągu.

   Rozbuduj program z poprzedniego zadania o następujące funkcje:

   - Pozwól na użycie znaków zastępczych w specyfikacji pliku, dzięki czemu ciąg będzie wyszukiwany w wielu plikach tekstowych.
   - Dołącz opcję wiersza poleceń do wyświetlania jedynie nazw plików. Ta opcja powinna mieć postać `+/-` w celu zachowania zgodności z narzędziem `grep` dostarczanym wraz z Turbo Assemblerem. Poniżej przedstawiono przykład polecenia, które spowoduje wyświetlenie wszystkich plików *.ASM* zawierających ciąg `xlat`:

     ```
     search -l+ xlat *.asm
     ```

4. Wyświetlanie zawartości pliku.

   Napisz program, który zapisuje plik tekstowy do pamięci i wyświetla pierwsze 24 wiersze tekstu. Zawartość buforu powinna być zapisywana bezpośrednio do pamięci wideo, co pozwoli na znaczne zwiększenie wydajności. Program musi obsługiwać następujące polecenia klawiaturowe:

Klawisz	Funkcja
PgUp	Przewinięcie w górę o 24 wiersze
PgDown	Przewinięcie w dół o 24 wiersze
Strzałka w górę	Przewinięcie w górę o jeden wiersz
Strzałka w dół	Przewinięcie w dół o jeden wiersz
Esc	Wyjście do systemu

5. Tworzenie pliku umożliwiającego bezpośredni dostęp.

   Napisz program tworzący plik zawierający informacje o studentach, które zostaną wprowadzone poprzez konsolę. Każdy rekord ma wielkość 27 bajtów. W pliku powinno być co najmniej 20 rekordów. Format rekordu przedstawiono poniżej:

```
Pole                    Kolumna
Numer studenta          1
Nazwisko                6
Kierunek                19
Liczba punktów          27
Ocena                   28
```

Oto przykładowe dane dla programu. Proszę dodać co najmniej 12 nowych rekordów:

```
10024ADAMS      ENG 11003A
10123BEAZLIE    CIS 23-14B
10200BOOKER     MAC 11325A
10201BOZEK      BUS 30023B
10330CHARLES    MUS 23003C
10405DANIELS    ART 10022A
10524GONZALEZ   CHM 40004A
10645HART       ENG 11003B
```

6. Program do obsługi rekordów studentów.

   Używając pliku z wcześniejszego zadania napisz program aktualizujący taki plik. Program powinien wyświetlać następujące menu:

```
DANE O STUDENTACH

P   Pokaż rekord studenta
D   Dodaj nowy rekord studenta
Z   Zmień rekord studenta
U   Usuń rekord studenta
Q   Wyjście z programu
```

   Użytkownik może wybierać rekordy według ich numerów. Po wykonaniu zadania określonego przez wybrane polecenie następuje powrót do głównego menu. Przetestuj program, wykonując wiele operacji dodawania, edycji i usuwania rekordów.

7. Rozbudowany program *Sector Display*.

   W rozdziale 11. przedstawiono program *Sector Display*, który służył do wyświetlania sektorów dysku. Dodaj teraz nową funkcję: wciśnięcie klawisza *F3* przez użytkownika ma powodować zapisanie sektora do pliku wyjściowego. Program powinien poprosić o podanie nazwy pliku. Jeśli ten plik już istnieje, to dane należy dołączyć na jego końcu. Po dokonaniu tych zmian program może być wykorzystany do przywracania uszkodzonych sektorów na dysku, ponieważ możliwa jest ich konwersja na pliki.

12.7.1. Manipulacja katalogami dyskowymi

1. Wyszukiwanie podkatalogów.

   Napisz procedurę wyszukującą w głównym katalogu dysku wszystkie wpisy z atrybutem 10h (nazwa podkatalogu). Wyświetl nazwy na ekranie.

2. Wyświetlenie podkatalogu.

   Napisz procedurę wyszukującą pierwszy wpis podkatalogu w katalogu głównym. Procedura ma przechodzić do tego podkatalogu, a następnie wyświetlać listę wszystkich jego plików.

3. Wyświetlanie podkatalogu z rekurencją.

   To zadanie wymaga znajomości metod przeszukiwania drzewa. Napisz rekurencyjną procedurę o nazwie `ShowTree`, która wyszukuje i wyświetla nazwy wszystkich podkatalogów w danym katalogu. Proces jest powtarzany dla wszystkich odnalezionych podkatalogów. Drzewo katalogu powinno być przedstawione w następującej formie:

```
A1
   A1B1
      A1B1C1
      A1B1C2
   A1B2
   A1B3
      A1B3C1
      A1B3C2
A2
   A2B1
   A2B2
A3
   A3B1
```

   Według tego schematu katalog główny zawiera podkatalogi `A1`, `A2` i `A3`. Katalog `A1` zawiera podkatalogi `A1B1`, `A1B2` i `A1B3`. Katalog `A1B1` zawiera podkatalogi `A1B1C1` i `A1B1C2`.

4. Przedstawienie znaczników daty i czasu.

   Zmodyfikuj program *Date Stamp* z listingu 12.1, aby wyświetlał znaczniki daty i czasu dla każdego pliku.

5. Sortowanie według nazw plików.

   Zmodyfikuj program *Date Stamp* z listingu 12.1, tak aby odczytywany katalog został zapisany do tablicy. Posortuj tablicę według nazw plików i wyświetl ją na ekranie.

6. Sortowanie według daty i godziny.

   Zmodyfikuj program *Date Stamp* z listingu 12.1, tak aby odczytany katalog został zapisany do tablicy. Posortuj tablicę według daty i godziny, a następnie wyświetl ją na ekranie.

7. Usuwanie wielu plików.

   Napisz program, który pobiera specyfikację pliku z wiersza poleceń, wyświetla nazwy wszystkich zgodnych plików i pyta o chęć usunięcia danego pliku. Wciśnięcie klawisza *T* powoduje skasowanie pliku.

8. Wyszukiwanie plików według daty.

   Napisz program, który wyszukuje w aktualnym katalogu wszystkie pliki mające znaczniki wskazujące datę wcześniejszą niż data systemowa. Wyświetl nazwy tych plików. Aby uzyskać datę systemową, wywołaj funkcję `2Ah` przerwania `INT 21`. Rok jest zwracany w rejestrze `CX`, miesiąc w `DH`, a dzień w `DL`. Data 12 października 1990 zostanie zwrócona jako:

```
CX = 07C6h, DH = 0Ah, DL = 0Ch
```

9. Ukrywanie i ujawnianie pliku.

   Napisz dwa programy. Pierwszy z nich, *hide.exe*, ukrywa wszystkie pliki zgodne ze specyfikatorem pliku. Program *unhide.exe* powoduje ujawnienie wszystkich plików zgodnych ze wzorcem. Oba programy działają tylko w aktualnym katalogu. Po zakończeniu pracy programy powinny wyświetlić listę ukrytych lub ujawnionych plików.

   Programy tego typu, często dostępne jako oprogramowanie typu shareware, mają wiele cennych zastosowań. Ich główną zaletą jest możliwość ochrony ważnych plików przed usunięciem poprzez polecenie DOS `DEL`. Przeciętny użytkownik komputera nie będzie mógł zapoznać się z zawartością ukrytych plików. Kolejnym ciekawym sposobem użycia takich programów jest usunięcie wszystkich plików w katalogu z *wyjątkiem* ukrytego pliku. Najpierw należy ukryć plik, a następnie skasować pozostałe pliki w katalogu i przywrócić zabezpieczony plik.

   Oba programy z tego zadania odczytują specyfikator pliku z wiersza poleceń. Może to być nazwa pliku, kompletna ścieżka lub nazwa ze znakami zastępczymi, na przykład **.ZIP*.