

Programowanie niskopoziomowe

Prowadzący:
Piotr Kisała

LABORATORIUM 2

Przegląd tematów

- ◆ sposoby reprezentacji liczb
- ◆ liczby ze znakiem i bez znaku
- ◆ konwersje pomiędzy systemami liczb.
- ◆ Instrukcje rozszerzania
- ◆ Kopiowanie z rozszerzeniem
- ◆ Liczby zmiennoprzecinkowe
- ◆ Zestaw znaków ASCII

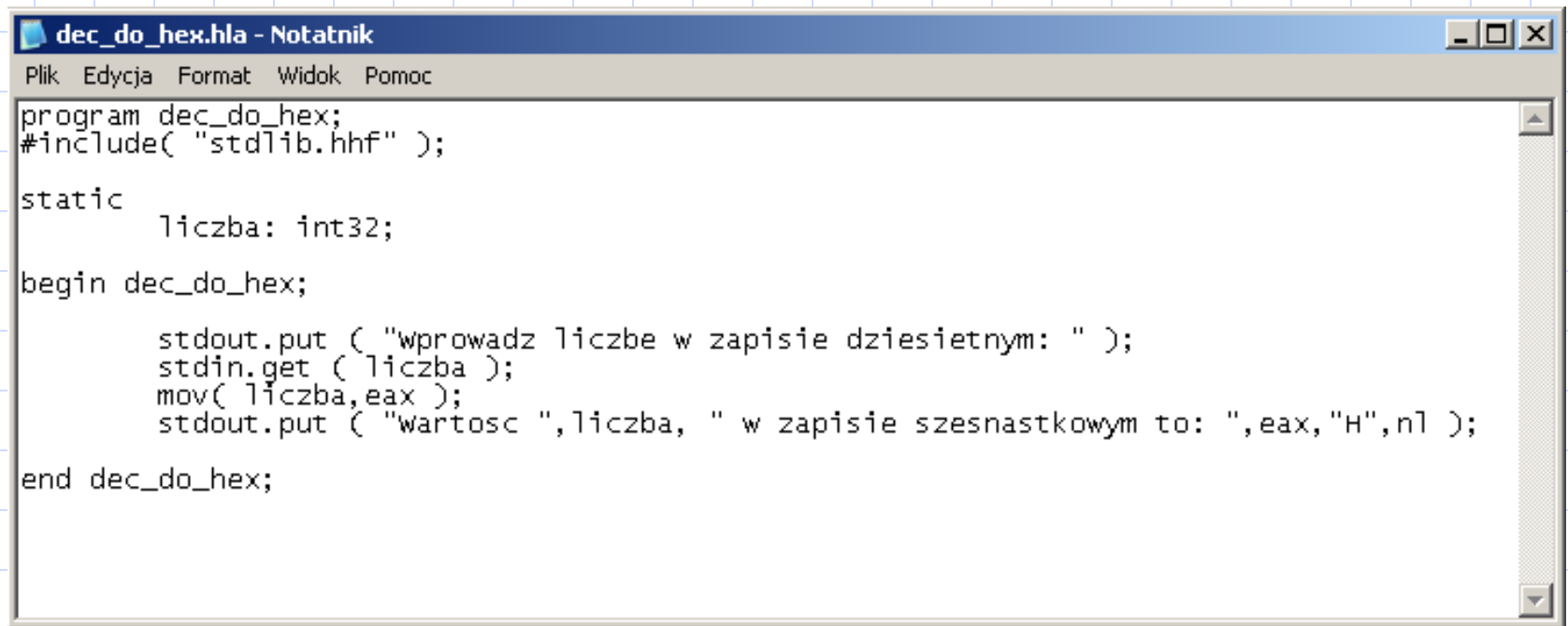
reprezentacja danych

- ◆ dane typu int – reprezentowane decymalnie za pomocą procedury `stdout.put`
- ◆ dane zawarte w rejestrach 8, 16, 32 bitowych – reprezentowane heksadecymalnie za pomocą procedury `stdout.put`
- ◆ dane typu byte, word, dword, qword, lword
– reprezentowane heksadecymalnie za pomocą procedury `stdout.put`

konwersja dec - hex

- ◆ Napisać program dokonujący konwersji pomiędzy systemem dziesiętnym, a szesnastkowym w oparciu o dowolny rejestr rozszerzony.

sposobów realizacji zadania



```
dec_do_hex.hla - Notatnik
Plik  Edycja  Format  Widok  Pomoc

program dec_do_hex;
#include( "stdlib.hhf" );

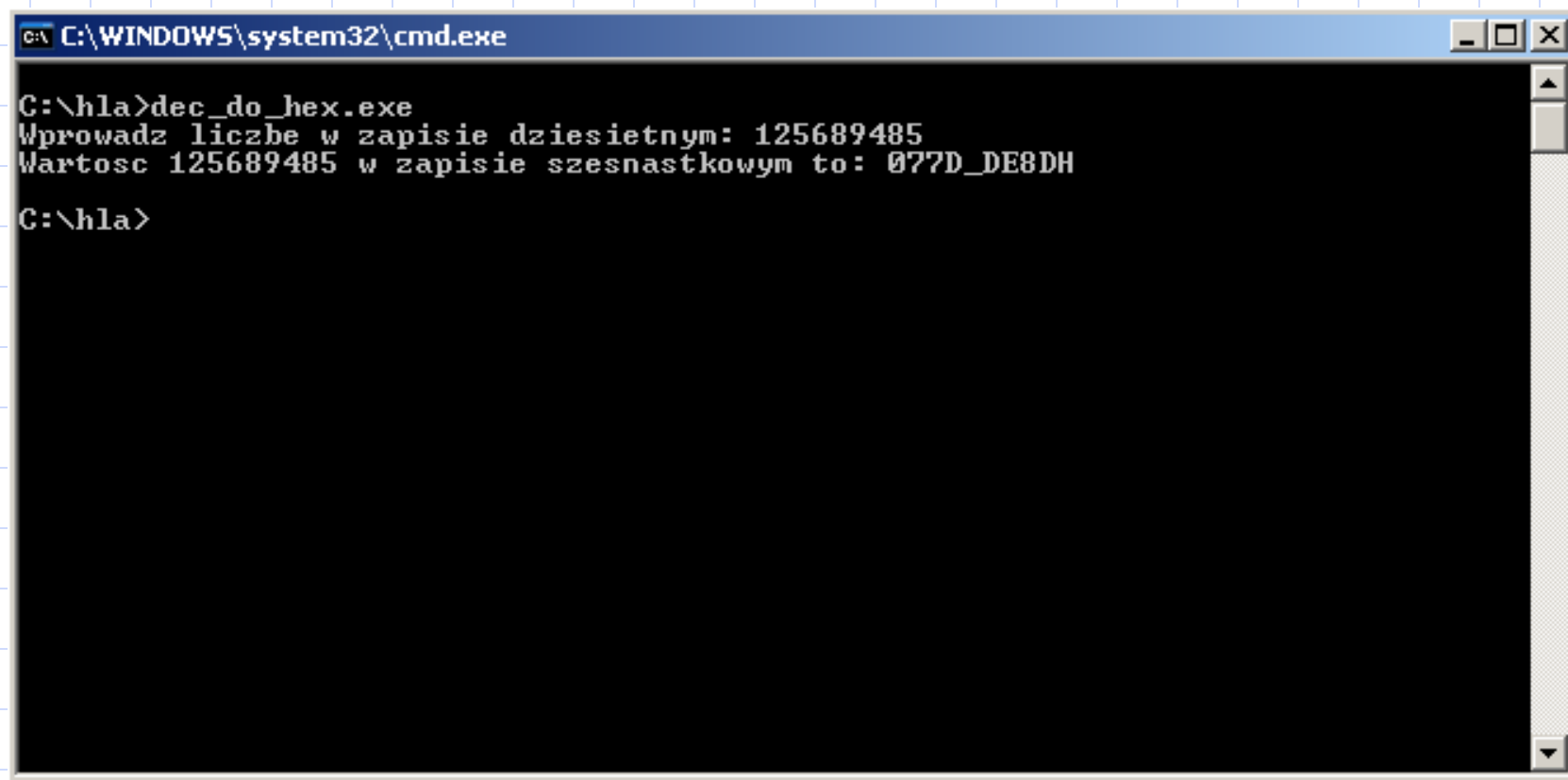
static
    liczba: int32;

begin dec_do_hex;

    stdout.put ( "wprowadz liczbe w zapisie dziesietnym: " );
    stdin.get ( liczba );
    mov( liczba,eax );
    stdout.put ( "wartosc ",liczba, " w zapisie szesnastkowym to: ",eax,"H",nl );

end dec_do_hex;
```

działanie konwertera



```
C:\WINDOWS\system32\cmd.exe

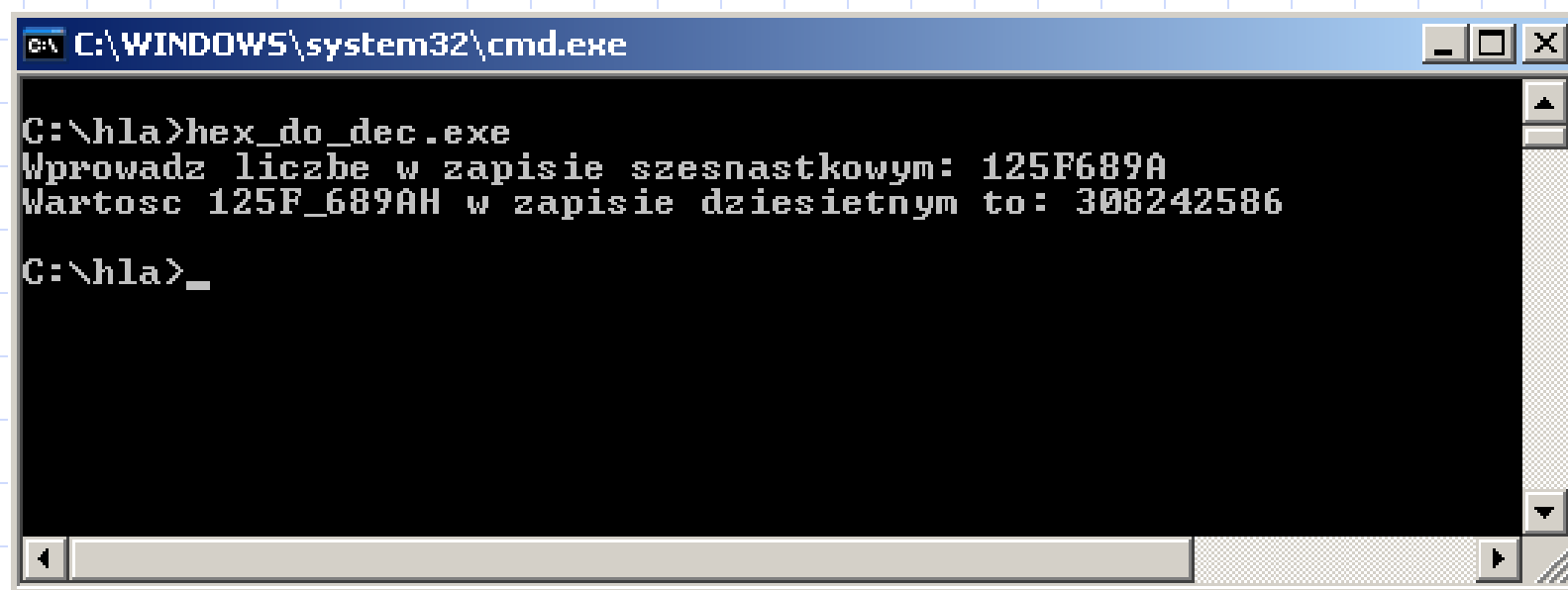
C:\hla>dec_do_hex.exe
Wprowadz liczbe w zapisie dziesietnym: 125689485
Wartosc 125689485 w zapisie szesnastkowym to: 077D_DE8DH

C:\hla>
```

konwersja hex - dec

- ◆ Napisać program dokonujący konwersji pomiędzy systemem szesnastkowym, a dziesiętnym w oparciu o dowolny rejestr rozszerzony.

działanie programu

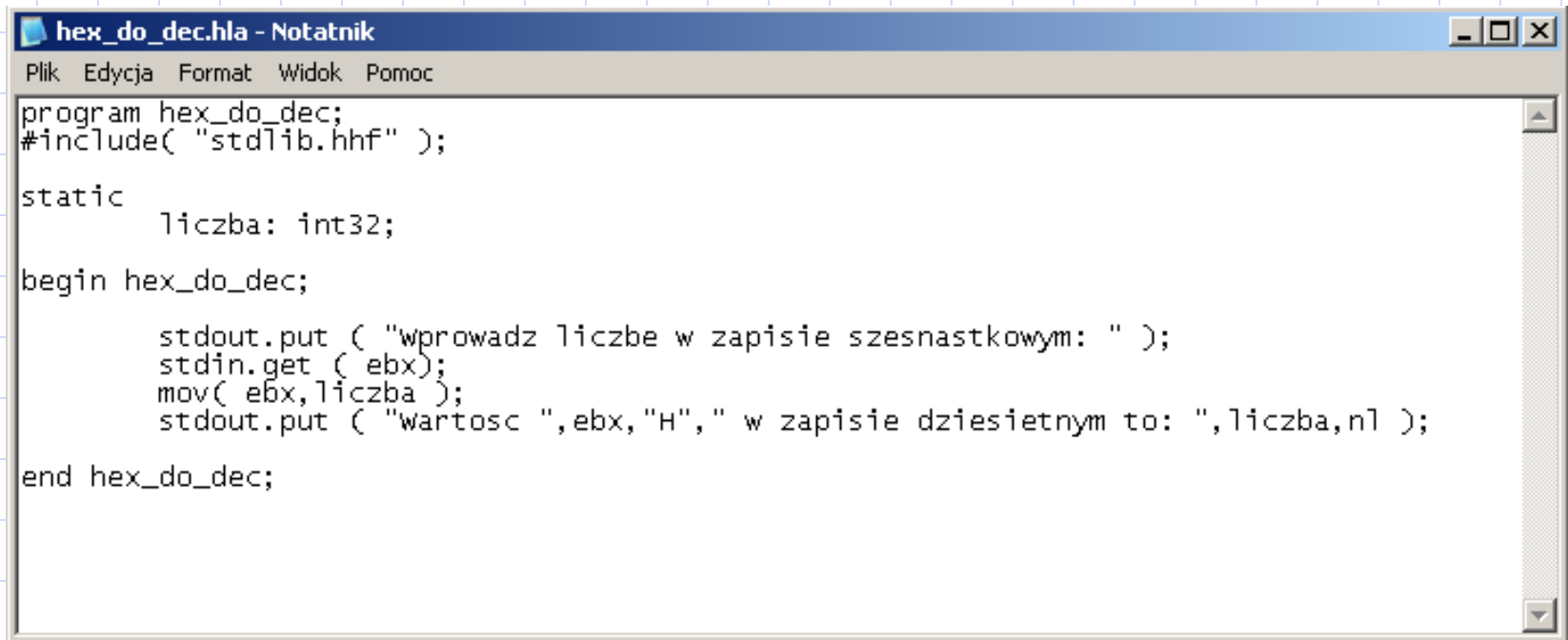


```
C:\WINDOWS\system32\cmd.exe

C:\hla>hex_do_dec.exe
Wprowadz liczbe w zapisie szesnastkowym: 125F689A
Wartosc 125F_689AH w zapisie dziesietnym to: 308242586

C:\hla>_
```


sposobów realizacji zadania



```
hex_do_dec.hla - Notatnik
Plik Edycja Format Widok Pomoc

program hex_do_dec;
#include( "stdlib.hhf" );

static
    liczba: int32;

begin hex_do_dec;

    stdout.put ( "wprowadz liczbe w zapisie szesnastkowym: " );
    stdin.get ( ebx);
    mov( ebx,liczba );
    stdout.put ( "wartosc ",ebx,"H"," w zapisie dziesietnym to: ",liczba,nl );

end hex_do_dec;
```

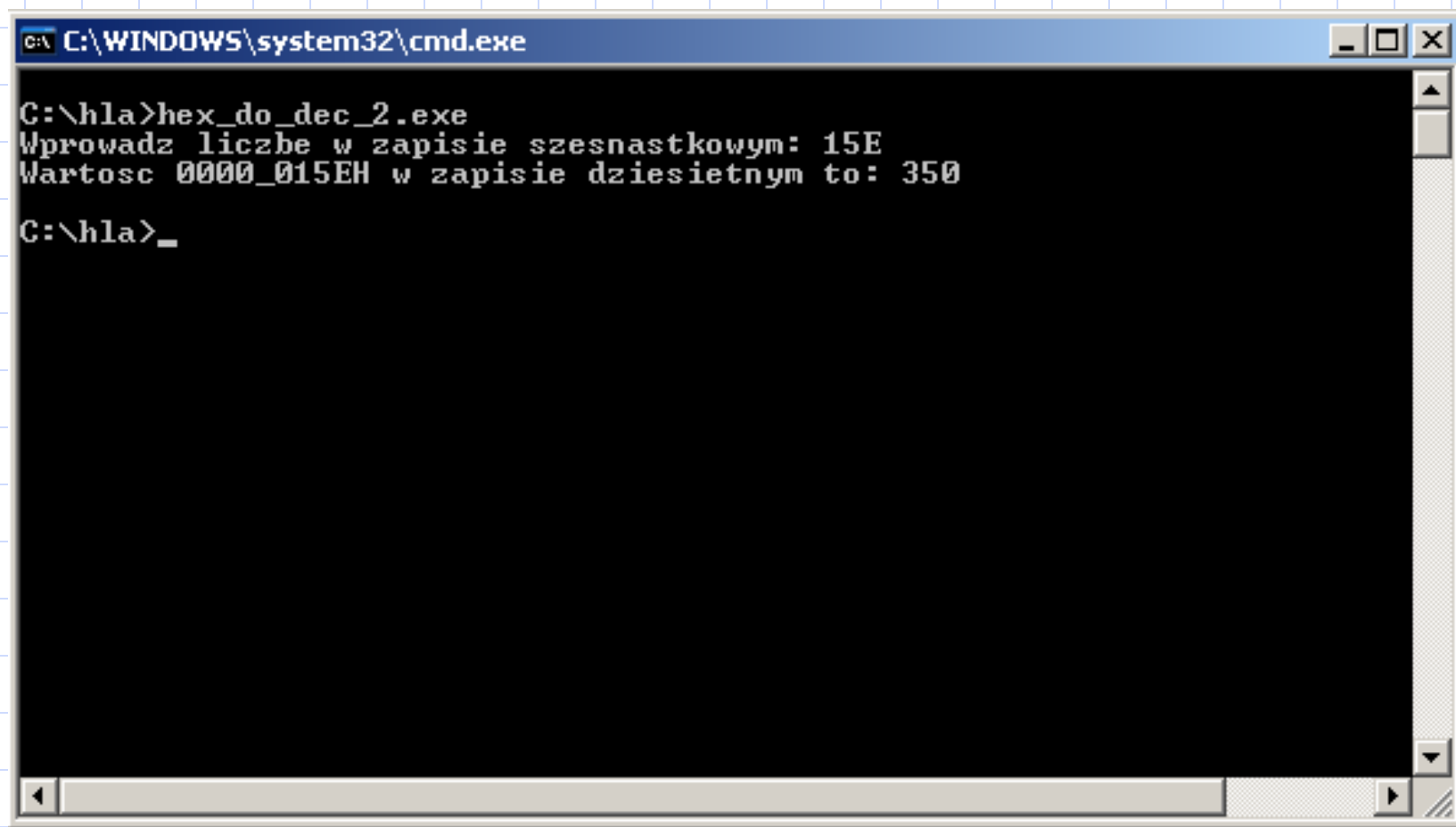
Reprezentacja dziesiętna

- ◆ Istnieje możliwość wyświetlania zawartości rejestrów w postaci dziesiętnej. Służy do tego procedura **stdout.putiN**
- ◆ Na przykład procedura **stdout.puti8** traktuje przekazany w wywołaniu argument jako liczbę całkowitą ze znakiem i wyprowadza ją w zapisie dziesiętnym. Do procedury można przekazać 8-bitowy argument, również 8-bitowy rejestr.
- ◆ dla obiektów 16 i 32-bitowych – procedury **stdout.puti16** oraz **stdout.puti32**

konwersja z hex do dec

- ◆ Napisać program dokonujący konwersji pomiędzy systemem szesnastkowym a dziesiętnym bez angażowania dodatkowych zmiennych.

wykonanie programu

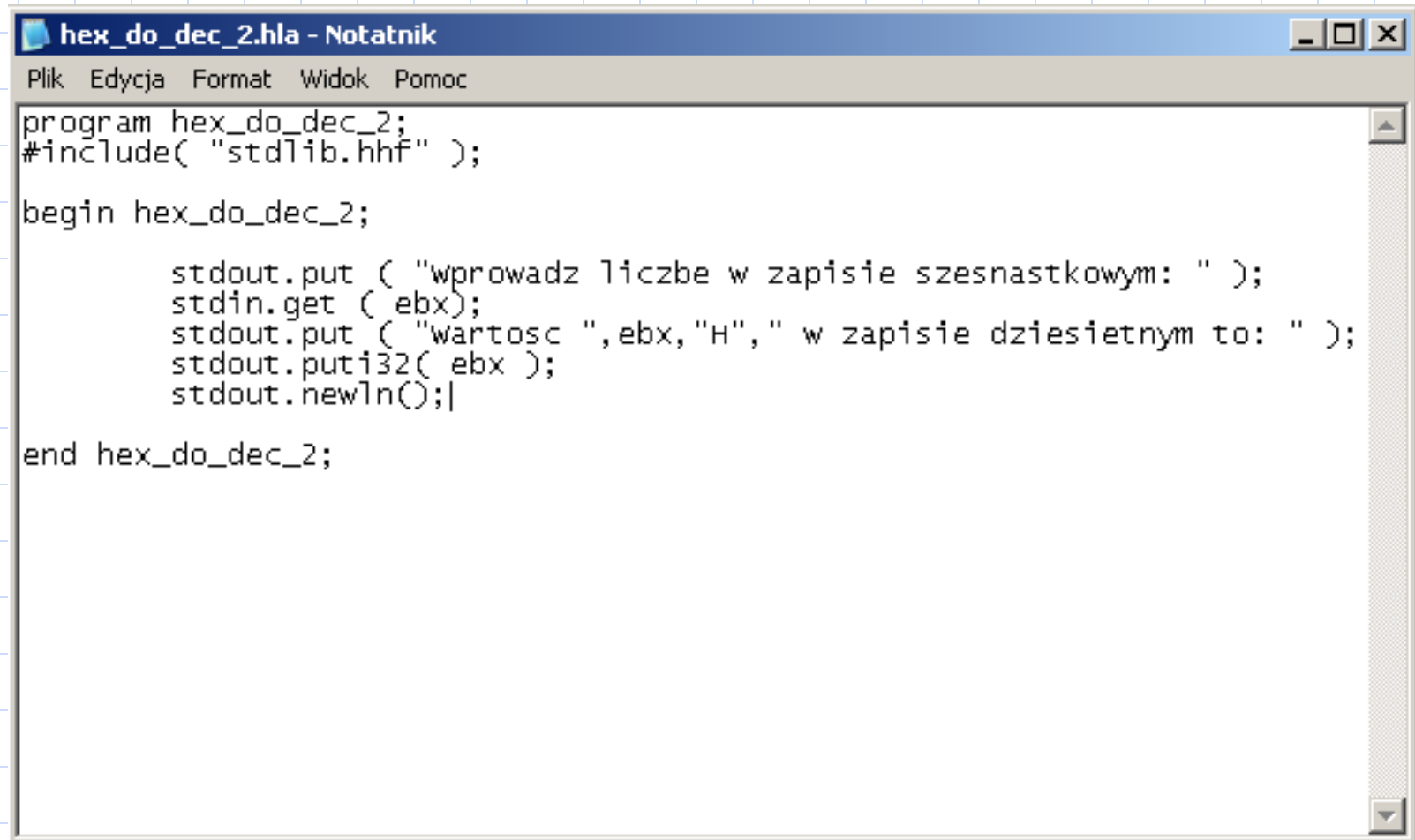


A screenshot of a Windows command prompt window. The title bar at the top reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
C:\hla>hex_do_dec_2.exe
Wprowadz liczbe w zapisie szesnastkowym: 15E
Wartosc 0000_015EH w zapisie dziesietnym to: 350
C:\hla>_
```

The window has a standard Windows interface with a title bar, maximize, minimize, and close buttons, and a scroll bar on the right side.

sposobów realizacji zadania



```
hex_do_dec_2.hla - Notatnik
Plik  Edycja  Format  Widok  Pomoc

program hex_do_dec_2;
#include( "stdlib.hhf" );

begin hex_do_dec_2;

    stdout.put ( "wprowadz liczbe w zapisie szesnastkowym: " );
    stdin.get ( ebx);
    stdout.put ( "wartosc ",ebx,"H"," w zapisie dziesietnym to: " );
    stdout.puti32( ebx );
    stdout.newln();

end hex_do_dec_2;
```

sposoby reprezentacji procedura stdin.getiV

Procedura `stdin.get` przyjmuje taką samą podstawę systemu liczbowego dla danych wejściowych jak `proc.stdout.put` dla danych wyprowadzanych. Jeśli więc chodzi o wczytanie wartości do umieszczenia w zmiennej `int8,16,32` procedura spodziewa się wprowadzenia wartości w zapisie `dec`. Gdy wczytywana zmienna ma być umieszczona w rejestrze lub zmiennej typu `byte, word, dword`, spodziewana jest wartość w zapisie szesnastkowym.

sposoby reprezentacji procedura stdin.getiN

◆ Aby zmienić przyjmowaną domyślnie podstawę systemu liczbowego dla danych mających trafić do rejestrów i zmiennych typu byte, word, dword, qword, lword należy skorzystać z wywołań:

- `stdin.geti8`
- `stdin.geti16`
- `stdin.geti32`
- `stdin.geti64`
- `stdin.geti128`

sposoby reprezentacji

procedura `stdout.putX` `stdin.getX`

- ◆ Jeżeli zachodzi potrzeba odwrotna polegająca na wprowadzeniu bądź wyprowadzeniu zmiennej typu `int8`, `int16`, `int32` `int64` bądź `int128` w postaci szesnastkowej należy posłużyć się procedurami

- `stdout.putb`, `stdout.putw`, `stdout.putq` ...
- `stdin.getb`, `stdin.getw`, `stdin.getq` ...

wczytane wartości umieszczane są w rejestrach `AL`,

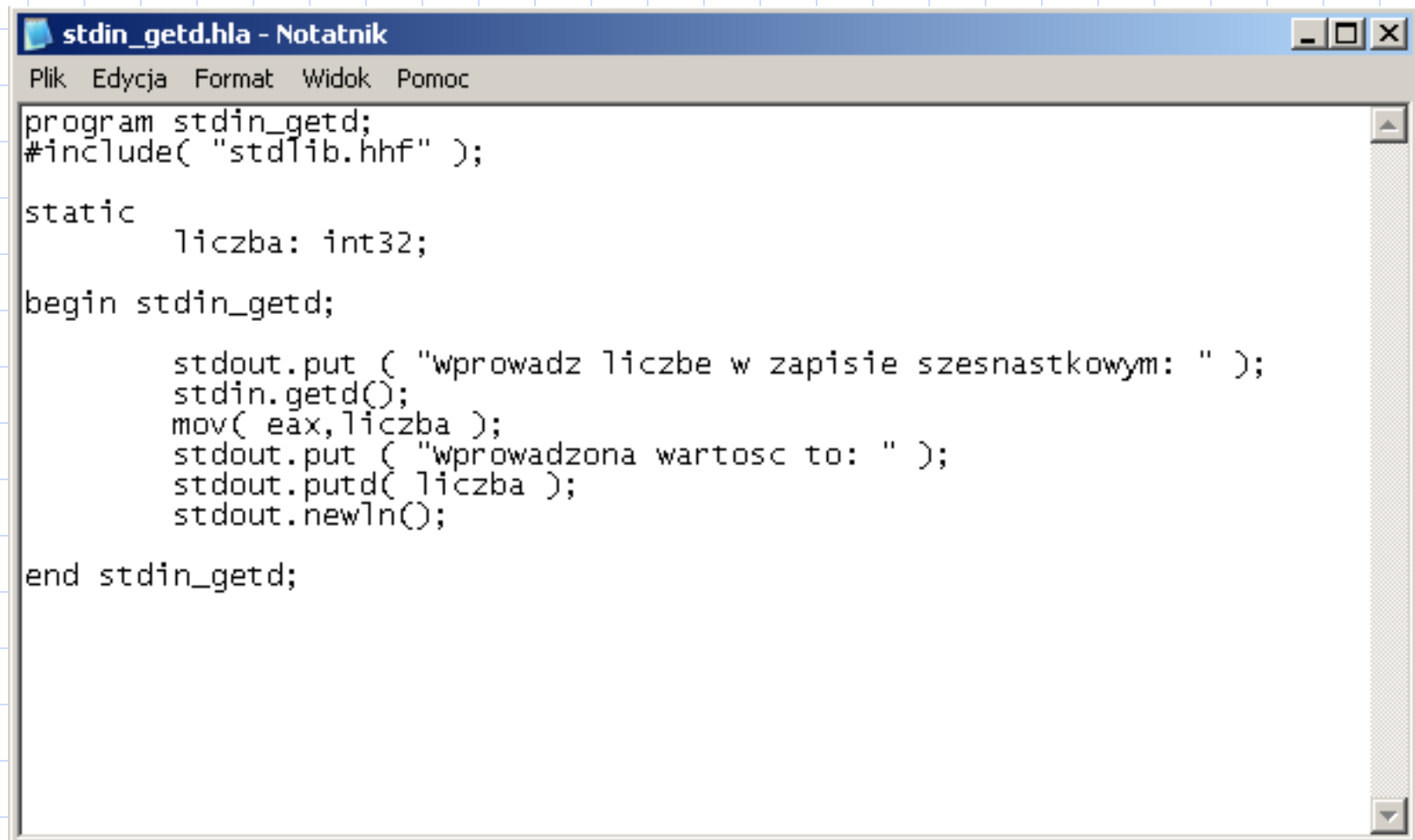
`AX`, `EAX` bądź – w przypadku wartości 64 i 128-bitowych – w miejscu określonym

argumentem wywołania procedury.

wykorzystanie stdout.putd i stdin.getd

- ◆ Zaproponować program, który dokona zapisu 32-bitowej liczby wprowadzonej w postaci szesnastkowej a następnie wyświetli tą liczbę w postaci szesnastkowej w oparciu o procedury stdout.putd i stdin.getd
- ◆ Wprowadzaną liczbę umieścić w zmiennej typu **int**

sposobów realizacji zadania



```
stdin_getd.hla - Notatnik
Plik Edycja Format Widok Pomoc

program stdin_getd;
#include( "stdlib.hhf" );

static
    liczba: int32;

begin stdin_getd;

    stdout.put ( "wprowadz liczbe w zapisie szesnastkowym: " );
    stdin.getd();
    mov( eax,liczba );
    stdout.put ( "wprowadzona wartosc to: " );
    stdout.putd( liczba );
    stdout.newln();

end stdin_getd;
```

Liczby ze znakiem i bez znaku

- ◆ W procesorach z rodziny 80x86 przyjęto notację z uzupełnieniem do dwóch.
- ◆ Najstarszy bit jest określa znak
- ◆ 0 – liczba jest dodatnia
- ◆ 1 – liczba jest ujemna
 - jeśli bit najstarszy ma wartość zero, to liczba traktowana jest jako dodatnia i interpretuje się ją zgodnie z regułą systemu dwójkowego.
W przypadku kiedy najstarszy bit ma wartość jeden, liczba jest traktowana jako ujemna a jej zapis odpowiada notacji U2.

przykłady liczb 16-bitowych

100H dodatnia

8000 ujemna

FFF dodatnia

FFFF ujemna

U2 w 80x86

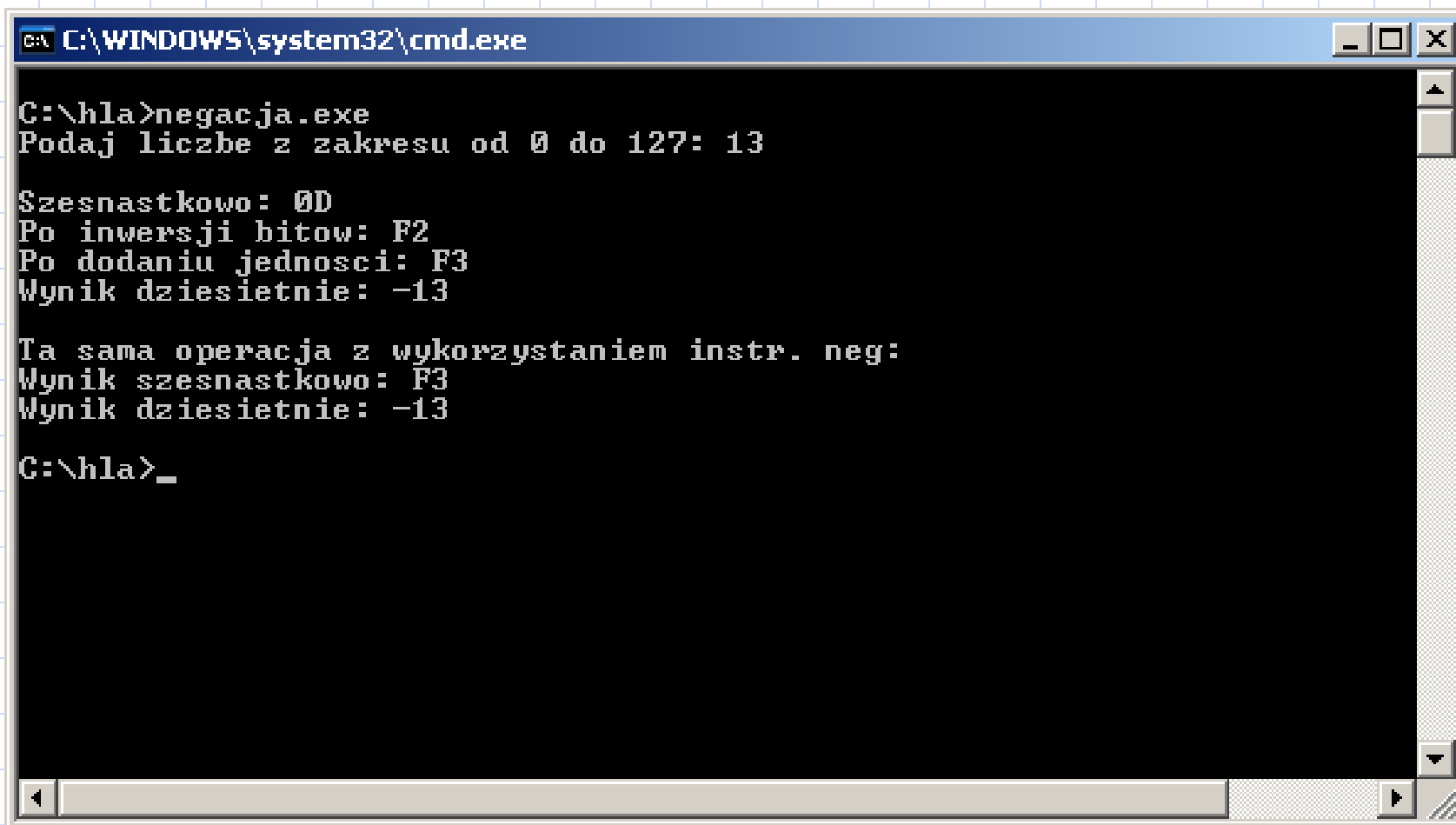
- ◆ Procesory rodziny 80x86 udostępniają specjalną instrukcję o nazwie **neg**, która realizuje konwersję do liczby odwrotnej w kodzie uzupełnienia do dwóch składnia polecenia neg:
neg(operand-docelowy);

operand musi być zmienną w pamięci lub rejestrem

neg - weryfikacja

- ◆ Napisać program który: (patrz nast. slajd)
 - wczytuje liczby jako zmienne 1-bajtowe typu int.
 - wyświetla wprowadzone liczby w postaci szesnastkowej
 - wyświetla wprowadzone liczby w postaci hex po inwersji bitów
 - wyświetla wprowadzone liczby w postaci hex po inwersji bitów i dodaniu jedności
 - wyświetla wprowadzone liczby w postaci dec po inwersji bitów i dodaniu jedności
 - wyświetla wprowadzone liczby w postaci dec w kodzie U2 przy wykorzystaniu

wynik działania programu



```
C:\WINDOWS\system32\cmd.exe

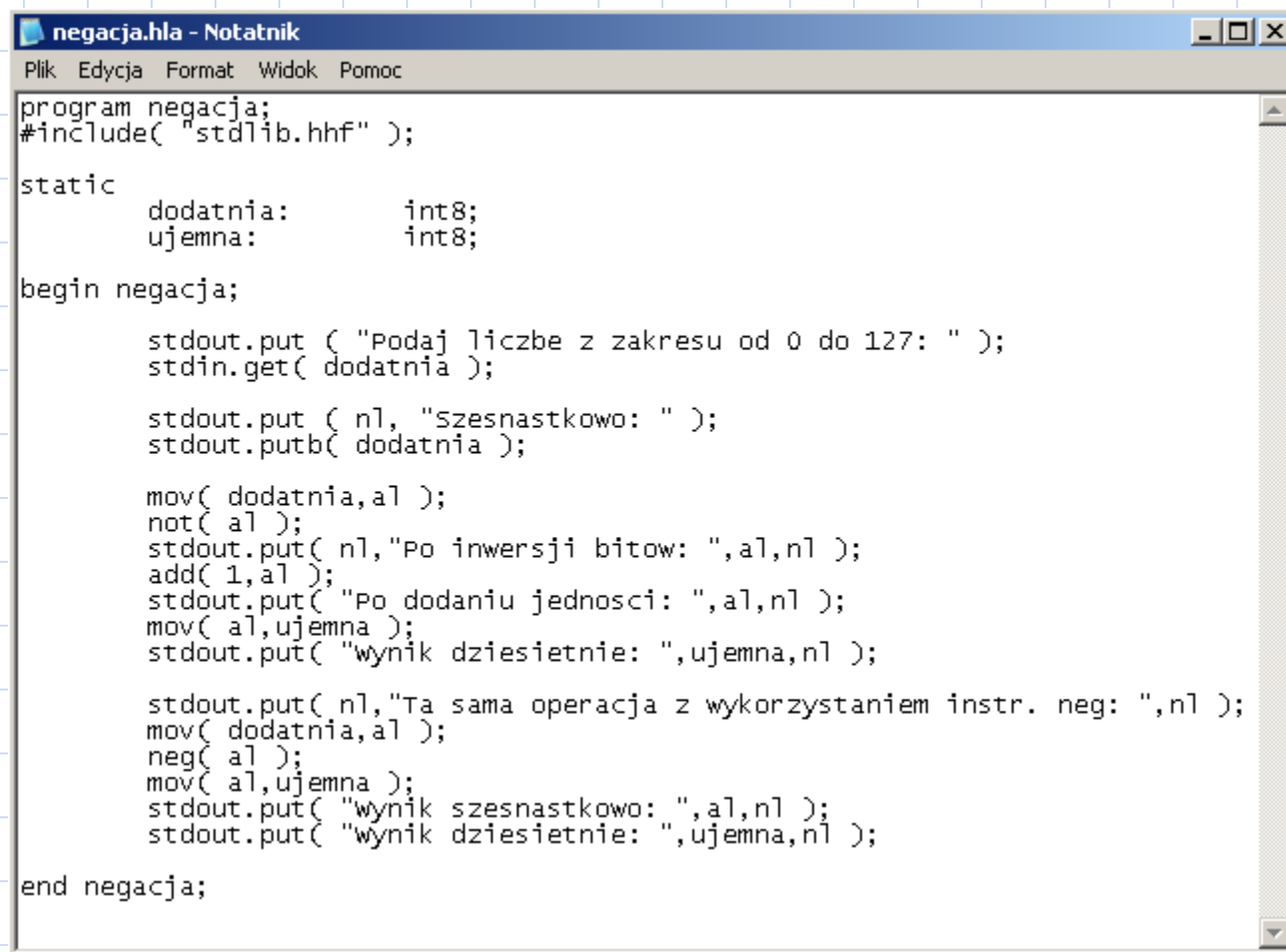
C:\hla>negacja.exe
Podaj liczbe z zakresu od 0 do 127: 13

Szesnastkowo: 0D
Po inwersji bitow: F2
Po dodaniu jednosci: F3
Wynik dziesietnie: -13

Ta sama operacja z wykorzystaniem instr. neg:
Wynik szesnastkowo: F3
Wynik dziesietnie: -13

C:\hla>_
```

jeden z możliwych sposobów realizacji zadania



```
negacja.hla - Notatnik
Plik Edycja Format Widok Pomoc

program negacja;
#include( "stdlib.hhf" );

static
    dodatnia:      int8;
    ujemna:        int8;

begin negacja;

    stdout.put ( "Podaj liczbe z zakresu od 0 do 127: " );
    stdin.get( dodatnia );

    stdout.put ( nl, "Szesnastkowo: " );
    stdout.putb( dodatnia );

    mov( dodatnia,a1 );
    not( a1 );
    stdout.put( nl,"Po inwersji bitow: ",a1,nl );
    add( 1,a1 );
    stdout.put( "Po dodaniu jednosci: ",a1,nl );
    mov( a1,ujemna );
    stdout.put( "wynik dziesietnie: ",ujemna,nl );

    stdout.put( nl,"Ta sama operacja z wykorzystaniem instr. neg: ",nl );
    mov( dodatnia,a1 );
    neg( a1 );
    mov( a1,ujemna );
    stdout.put( "wynik szesnastkowo: ",a1,nl );
    stdout.put( "wynik dziesietnie: ",ujemna,nl );

end negacja;
```


konwersja liczby 8-bitowej do liczby 16 bitowej

Aby rozszerzyć wartość interpretowaną jako liczbę ze znakiem do dowolnej większej liczby bitów, wystarczy skopiować bit znaku do wszystkich nadmiarowych bitów nowego formatu

8 bitów	16 bitów	32 bity
80H	FF80H	FFFF_FF80H
28H	0028H	0000_0028H
9A	FF9AH	FFFF_FF9A
7F	007F	0000_007F

Rozszerzenie zerem

Aby rozszerzyć wartość bez znaku, należy wykonać tak zwane rozszerzenie zerem.

Starszy bajt (bajty) większego operandu są po prostu zerowane:

8 bitów	16 bitów	32 bity
80H	0080H	0000_0080H
28H	0028H	0000_0028H
9A	009AH	0000_009A
7F	007F	0000_007F

Instrukcje rozszerzania rejestrów AL, AX, EAX

Procesory rodziny 80x86 udostępniają programiście szereg instrukcji rozszerzania znakiem i zerem.

instrukcja	działanie
cbw();	rozszerza znakiem bajt z rejestru AL na rejestr AX
cwd(); DX:AX	rozszerza znakiem słowo z rejestru AX na rejestr DX:AX
cdq(); na	rozszerza znakiem podwójne słowo z rejestru EAX na rejestry EDI:ESI
cwde(); EAX	rozszerza znakiem słowo z rejestru EAX na rejestr EDI:ESI

zapis DX:AX oznacza wartość 32-bitową, której starsze słowo umieszczone jest w rejestrze DX, a młodsze w rejestrze AX

Kopiowanie z rozszerzeniem

Instrukcja movsx przy kopiowaniu operandu źródłowego automatycznie rozszerza go znakiem do rozmiaru operandu docelowego.

movsx(operand-źródłowy,operand-docelowy);

operand docelowy musi mieć rozmiar większy od operandu rozmiaru źródłowego,
operand docelowy musi być rejestrem; tylko operand źródłowy może być zmienną w pamięci.

Rozszerzenie zerem

Instrukcja **movzx** ma identyczną składnię i nakłada identyczne ograniczenia na operandy jak instrukcja movsx.

Rozszerzenie zerem niektórych 8-bitowych rejestrów (AL,BL,CL oraz DL) na odpowiednie rejestry 16-bitowe można łatwo osiągnąć bez pośrednictwa instrukcji movzx – wystarczy proste wyzerowanie starszych połówek tych rejestrów (AH, BH, CH i DH).

program - rozszerzanie

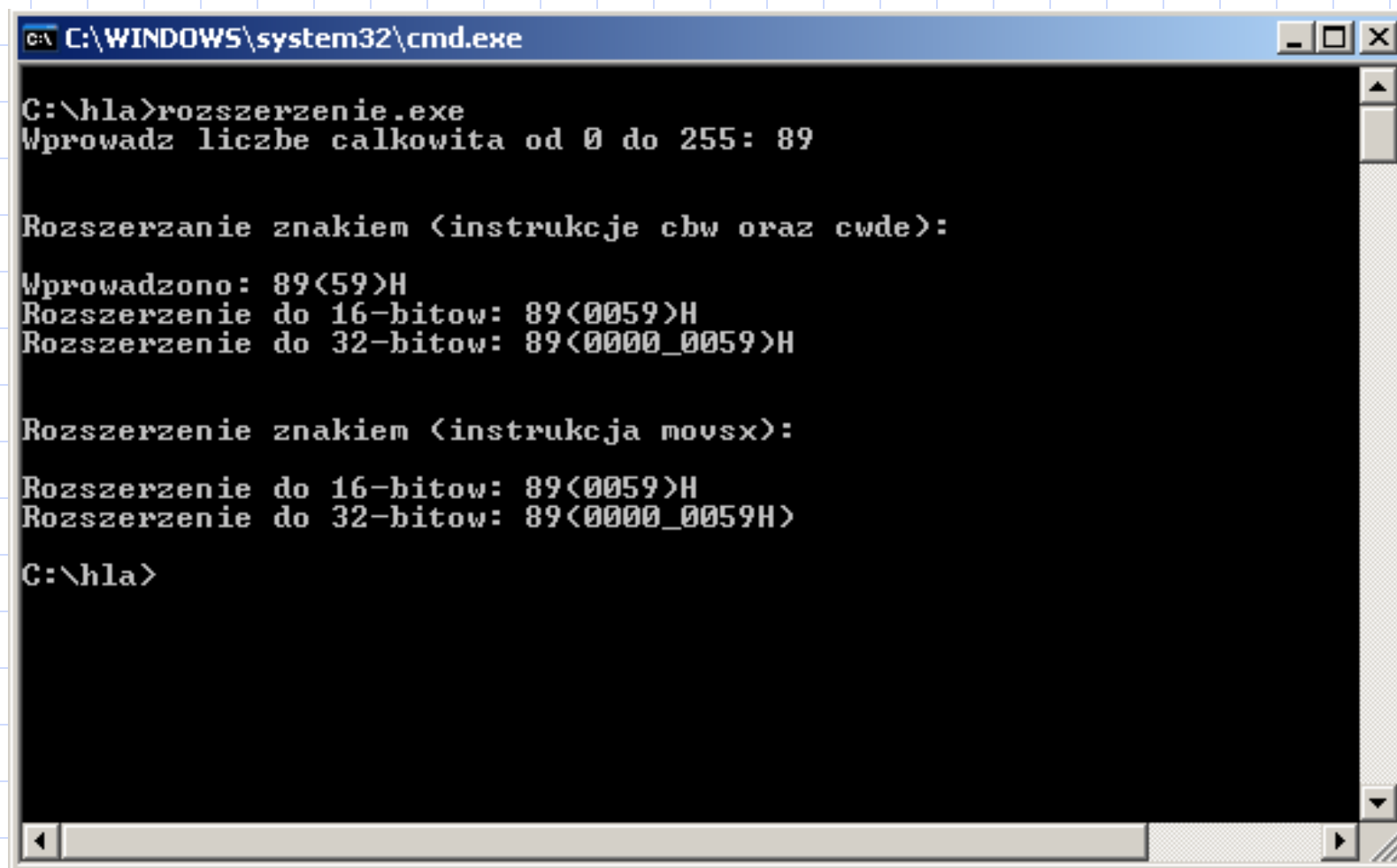
Napisać program wyświetlający (w postaci dziesiętnej) wprowadzoną liczbę jednobajtową:

1. bez rozszerzenia
2. z rozszerzeniem do 16-bitów
3. z rozszerzeniem do 32-bitów

za pomocą:

1. instrukcji `cbw`, `cwde`
2. instrukcji `movsx`

Działanie programu



```
C:\WINDOWS\system32\cmd.exe

C:\hla>rozszerzenie.exe
Wprowadz liczbe calkowita od 0 do 255: 89

Rozszerzanie znakiem (instrukcje cbw oraz cwde):

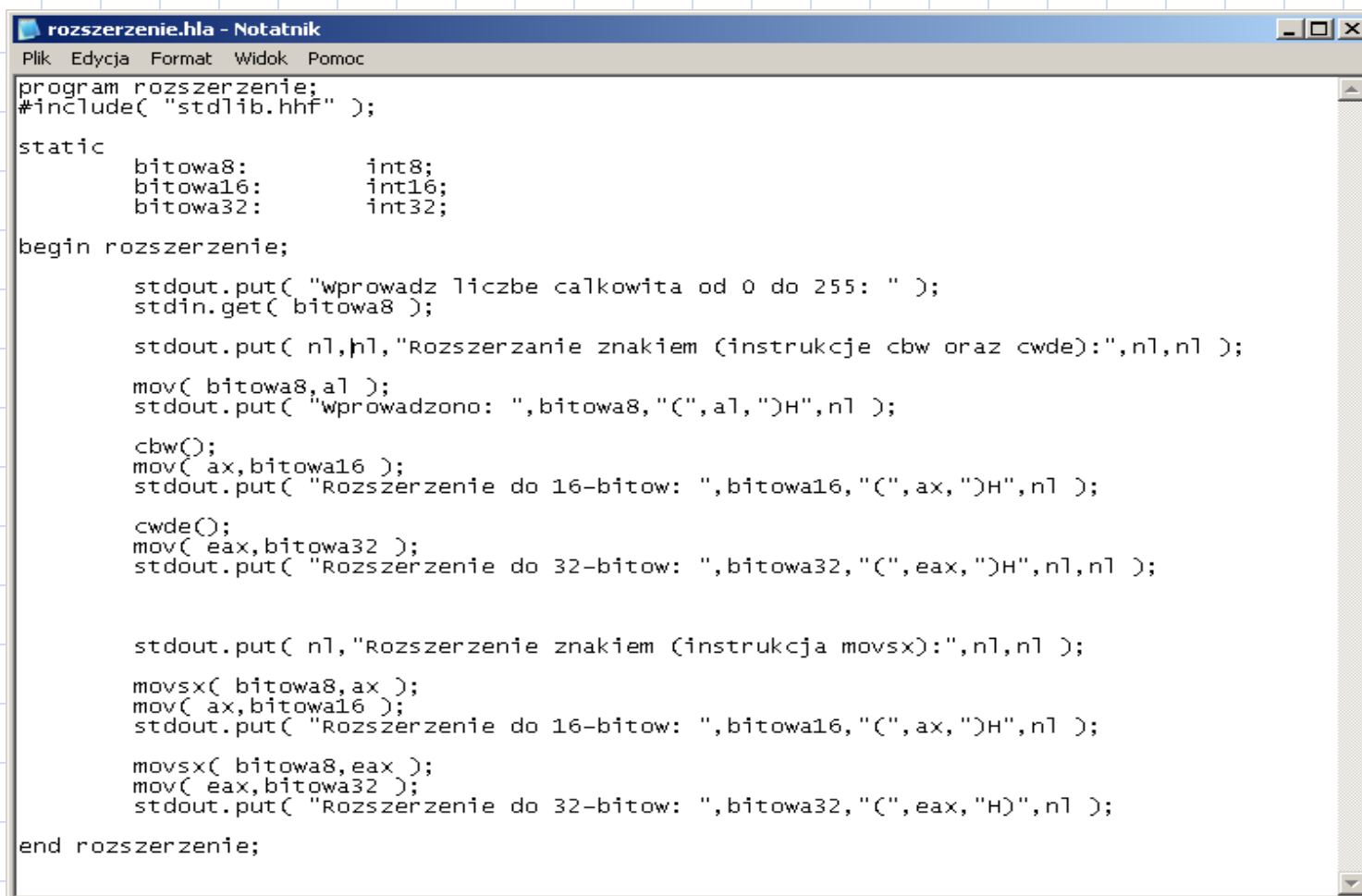
Wprowadzono: 89<59>H
Rozszerzenie do 16-bitow: 89<0059>H
Rozszerzenie do 32-bitow: 89<0000_0059>H

Rozszerzenie znakiem (instrukcja movsx):

Rozszerzenie do 16-bitow: 89<0059>H
Rozszerzenie do 32-bitow: 89<0000_0059H>

C:\hla>
```

jeden z możliwych sposobów realizacji zadania



```
rozszerzenie.hla - Notatnik
Plik Edycja Format Widok Pomoc

program rozszerzenie;
#include( "stdlib.hhf" );

static
    bitowa8:      int8;
    bitowa16:     int16;
    bitowa32:     int32;

begin rozszerzenie;

    stdout.put( "wprowadz liczbe calkowita od 0 do 255: " );
    stdin.get( bitowa8 );

    stdout.put( nl,nl,"Rozszerzanie znakiem (instrukcje cbw oraz cwde):",nl,nl );

    mov( bitowa8,al );
    stdout.put( "wprowadzono: ",bitowa8,"(",al,")H",nl );

    cbw();
    mov( ax,bitowa16 );
    stdout.put( "Rozszerzenie do 16-bitow: ",bitowa16,"(",ax,")H",nl );

    cwd();
    mov( eax,bitowa32 );
    stdout.put( "Rozszerzenie do 32-bitow: ",bitowa32,"(",eax,")H",nl,nl );

    stdout.put( nl,"Rozszerzenie znakiem (instrukcja movsx):",nl,nl );

    movsx( bitowa8,ax );
    mov( ax,bitowa16 );
    stdout.put( "Rozszerzenie do 16-bitow: ",bitowa16,"(",ax,")H",nl );

    movsx( bitowa8,eax );
    mov( eax,bitowa32 );
    stdout.put( "Rozszerzenie do 32-bitow: ",bitowa32,"(",eax,"H)",nl );

end rozszerzenie;
```


Liczby zmiennoprzecinkowe

reprezentowane są za pomocą literałów zmiennoprzecinkowych:
liczbę może poprzedzać nieobowiązkowy znak (plus lub minus),
w przypadku braku znaku zakłada się, że liczba jest dodatnią,

za ewentualnym znakiem występuje jedna bądź więcej cyfr
dziesiętnych,

cyfry te uzupełnione są przecinkiem dziesiętnym (kropka) oraz
jedną większą liczbą cyfr dziesiętnych.

całość może być uzupełniona literą „e” lub „E”,
nieobowiązkowym znakiem oraz kolejnymi cyframi dziesiętnymi.

1.234 3.75e2 -1.0 1.1e-1 1e+4 0.1 -123.456e789
25e0

deklaracje liczb zmiennoprzecinkowych

Do wykorzystania są następujące typy: real32, real64, real80. Podobnie jak w ich odpowiednikach całkowitych liczba kończąca nazwę typu określa rozmiar (w bitach) zmiennych tego typu.

real32 – odpowiada liczbom o pojedynczej precyzji

real64 – odpowiada liczbom o podwójnej precyzji

real80 – odpowiada liczbom o rozszerzonej precyzji

Wyprowadzanie liczb zmiennoprzecinkowych

służy do tego jedna z procedur:

`stdout.putr32`, `stdout.putr64`, `stdout.putr80`

składnia wywołania jest identyczna dla wszystkich procedur:

`stdout.putr32(lzmennoprz, szerokość, liczba-cyfr-po-przecinku)`

pierwszym argumentem powinna być wartość zmiennoprzecinkowa, która ma zostać wyprowadzona na standardowe wyjście programu

argument musi mieć odpowiedni rozmiar – w wywołaniu `stdout.putr80` należy przekazywać wartości zmiennoprzecinkowe rozszerzonej precyzji

Wyprowadzanie liczb zmiennoprzecinkowych

Wywołanie:

```
stdout.putr32( pi,10,4 );
```

powoduje wyprowadzenie napisu:

___ 3.1416

Wyprowadzanie liczb zmiennoprzecinkowych w postaci wykładniczej

służą do tego procedury:

`stdout.pute32`, `stdout.pute64`,
`stdout.pute80`

wywoływane następująco:

`stdout.pute32(liczba-zmiennoprz ,
szerokość);`

szerokość określa wszystkie pozycje po
przecinku (razem z pozycjami wykładnika)

Wyprowadzanie liczb zmiennoprzecinkowych w postaci wykładniczej

Można również wykorzystać procedurę `stdout.put`, jeśli w wywołaniu tej procedury znajdzie się nazwa obiektu zmiennoprzecinkowego jego wartość zostanie skonwertowana do postaci napisu zawierającego notację wykładniczą liczby.

Szerokość ustalana jest automatycznie.

Jeśli procedura `stdout.put` ma konwertować liczby zmiennoprz. do zapisu dziesiętnego należy argument `zmiennoprz.` określać następująco:

`nazwa:szerokość:liczba-cyfr-po-przecinku`

np.

```
stdout.put( „Pi = ”, pi:5:3 );
```

spowoduje wyprowadzenie na wyjście napisu:

Pi = 3.141

Zestaw znaków ASCII

Zestaw kodów ASCII (American Standard Code for Information Interchange) definiuje odwzorowanie 128 znaków do interpretowanych bez znaku wartości całkowitych z zakresu od 0 do 127.

Standard przyjęty przez znaczną część przemysłu. Jeżeli więc do reprezentowania znaku „A” wykorzystywana jest liczba 65, zgodnie z zestawem znaków ASCII, wtedy można mieć pewność że liczba ta zostanie zinterpretowana jako znak „A” również przez drukarkę czy terminal.

Grupy znaków ASCII

Zestaw znaków ASCII podzielony jest na 4 grupy po 32 znaki.

- ◆ Pierwsze 32 są to znaki niedrukowane (sterujące), np. powrót karetki, znak wysuwu wiersza, znak cofania.
- ◆ Kolejne 32 to znaki specjalne, przystankowe i cyfry.
- ◆ Trzecia grupa to znaki wielkich liter alfabetu. Jako, że znaków w alfabecie łacińskim jest 26, sześć kodów przypisanych zostało do różnych znaków specjalnych.
- ◆ Czwartą grupę stanowią kody 26 małych liter alfabetu, pięć znaków specjalnych i znak sterujący DELETE.

ASCII w asemblerze

Literały znakowe w języku HLA mogą przyjmować jedną z dwóch postaci:
pojedynczego znaku otoczonego znakami
pojedynczego cudzysłowu albo wartości
(z zakresu 0 do 127) określającej kod ASCII
poprzedzonej znakiem kratki (#):

`'A'` `#65` `#41` `#0100_0001`

wszystkie powyższe literały reprezentują ten sam znak „A”

ASCII w HLA

Warto przyjąć zasadę, że znaki drukowalne określane są w kodzie programu literałami w pierwszej z prezentowanych form, czyli jako znaki **ujęte w znaki pojedynczego cudzysłowu**.

Znak kratki i literał liczbowy należy stosować wyłącznie do określania znaków niedrukowalnych: specjalnych i sterujących, ewentualnie do określenia znaków z rozszerzonej części zestawu ASCII, które w kodzie źródłowym mogą być wyświetlane **niepoprawnie**.

literał łańcuchowy \neq literał znakowy

- ◆ Literały łańcuchowe zawierają zero bądź więcej znaków i są ograniczane znakami podwójnego cudzysłowu
- ◆ Literały znakowe zawierają zaś pojedynczy znak i są ograniczane znakami pojedynczego cudzysłowu. W szczególności należy uświadomić sobie fakt, że:

`'A'` \neq `"A"`

ASCII w HLA

Aby zadeklarować w HLA zmienną znakową należy skorzystać z typu danych o nazwie char. Zmienne znakowe można przy deklaracji od razu inicjalizować:

static

znakA:

char:='A'

znak_rozszerzony:

char:=#128

jako że zmienne są obiektami 8-bitowymi, można nimi manipulować za pośrednictwem ośmiobitowych rejestrów i odwrotnie – zawartość takiego rejestru można skopiować do zmiennej znakowej

Wprowadzanie i wyprowadzanie znaków

służą temu procedury:

`stdout.putc`, `stdout.putSize`, `stdout.put`,
`stdin.getc`, `stdin.get`

`stdout.putc(zmienna-znakowa);`

- procedura wyprowadza na standardowe wyjście programu pojedynczy znak określony wartością argumentu wywołania procedury
- argument może zostać określony jako stała lub zmienna znakowa bądź rejestr 8-bitowy

stdout.putcSize

procedura pozwala na wyprowadzenie znaków z określeniem szerokości wyprowadzanego napisu i wypełnienia

```
stdout.putcSize( zmienna-znakowa, szerokosc, wypełnienie );
```

procedura wyprowadza znak określony zmienną znakową, umieszczając go w napisie o określonej szerokości. Jeżeli bezwzględna wartość argumentu szerokość jest większa niż jeden, napis zostanie uzupełniony znakami *wypełnienia*. Jeśli argument szerokość zostanie określony jako ujemny, wyprowadzany znak będzie wyrównany do lewej krawędzi napisu. Wartość dodatnia powoduje wyrównanie do prawej krawędzi napisu.

Znaki są zwykle wyrównane do lewej.

stdout.put a wartości znakowe

Wartości znakowe mogą być też wyprowadzane za pośrednictwem uniwersalnej procedury wyjścia stdout.put. Jeśli na liście wywołania tej procedury znajduje się zmienna znakowa, kod procedury automatycznie wyświetli odpowiadający jej znak, np.:

```
stdout.put( "Znak c = ",c," ' ",nl );
```

Pobieranie znaków

za pomocą procedur `stdin.getc` i `stdin.get`.
Procedura `stdin.getc` nie przyjmuje żadnych argumentów. Jej działanie ogranicza się do wczytania z bufora urządzenia standardowego wejścia pojedynczego znaku i umieszczenia go w AL. Po wczytaniu do rejestru można tą wartością manipulować na miejscu albo skopiować do zmiennej w pamięci.

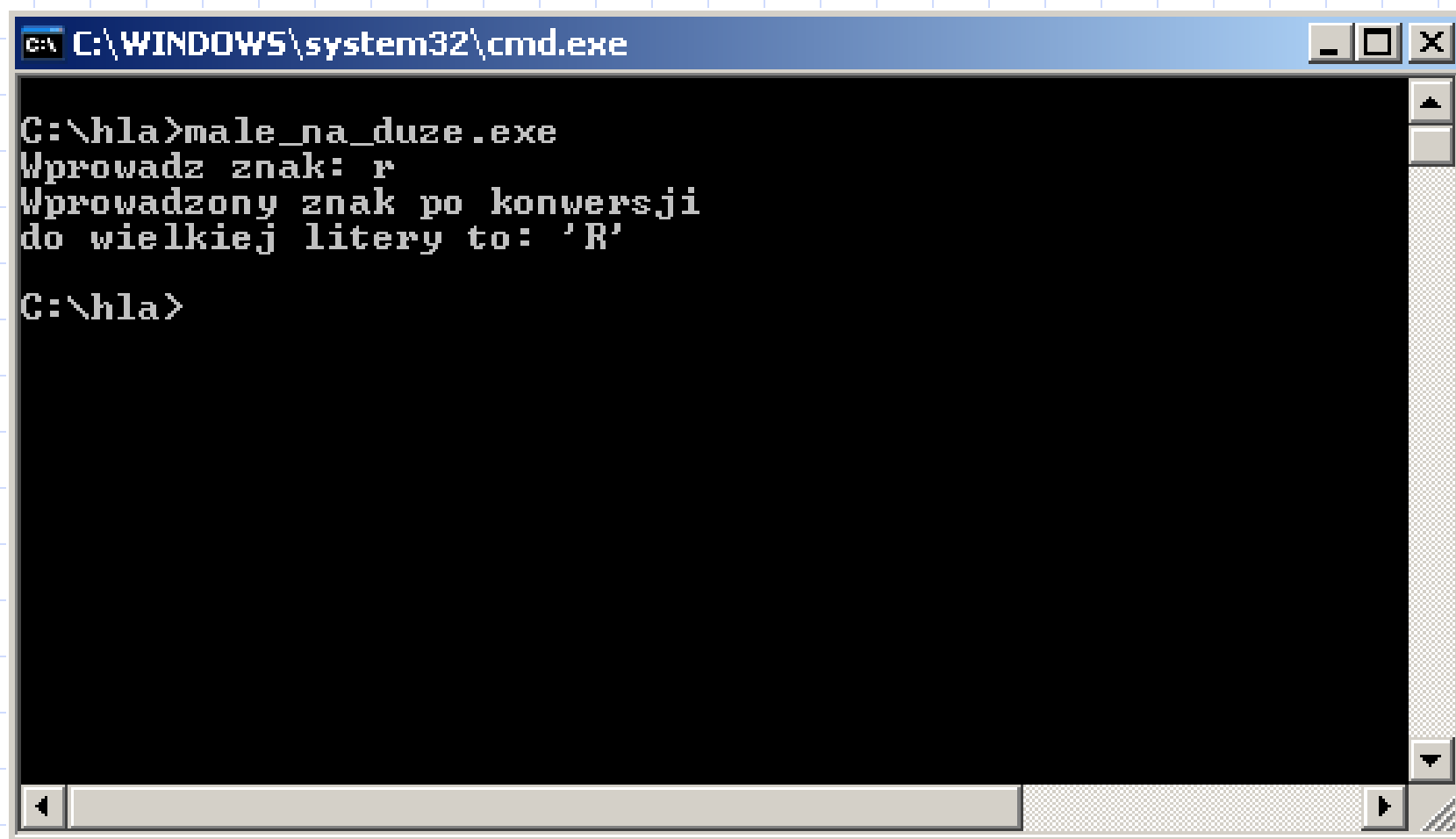
Wczytywanie i wyprowadzanie znaków

`stdin.getc()`

wiedząc, że małe i duże litery różnią się w kodzie ASCII pozycją 6-go bitu napisać program który wczytuje małe litery z klawiatury, zamienia je na duże i wyświetla na ekranie.

wykorzystać procedurę **`stdin.getc()`**

działanie programu

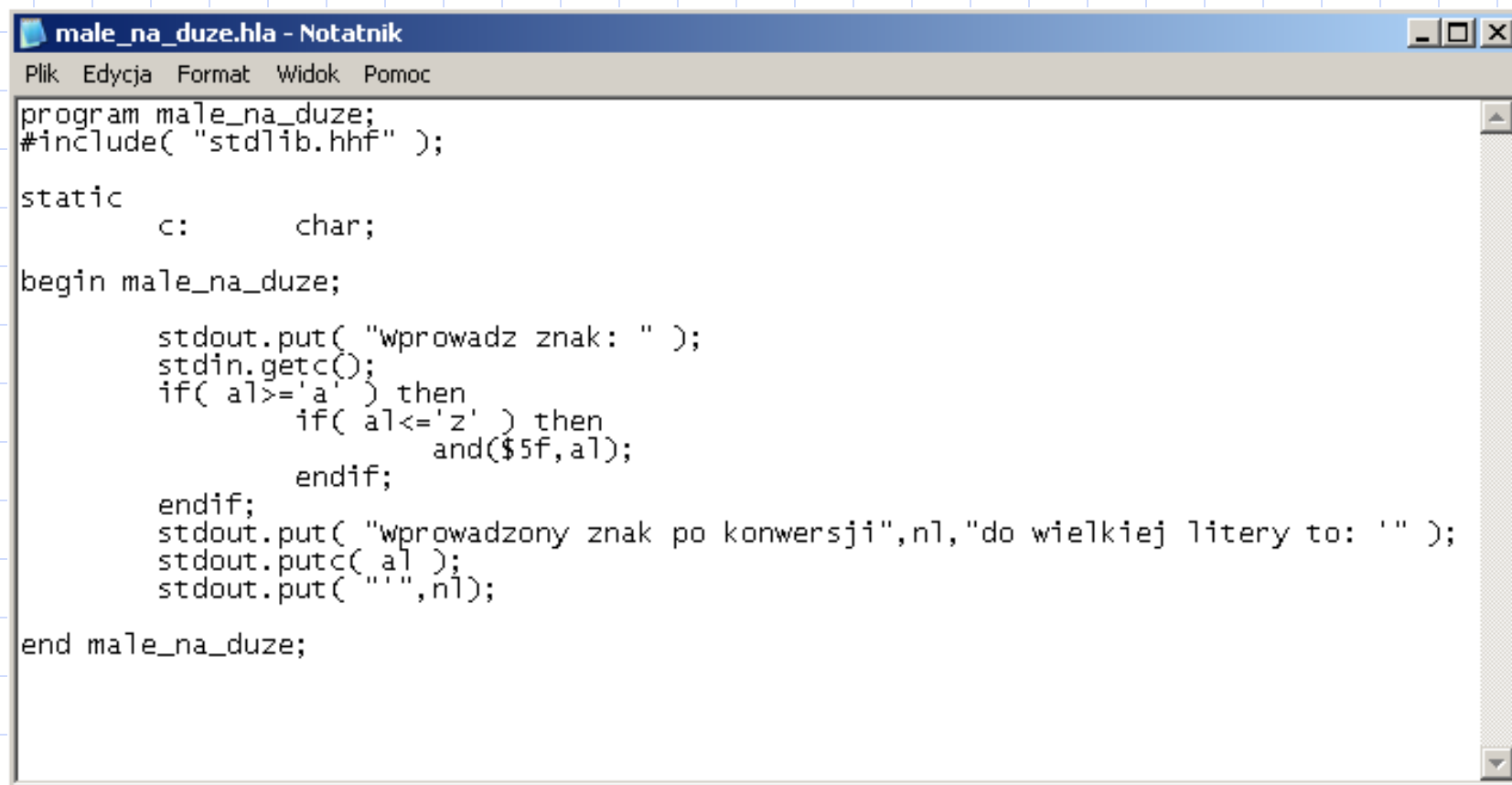


A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
C:\hla>male_na_duze.exe  
Wprowadz znak: r  
Wprowadzony znak po konwersji  
do wielkiej litery to: 'R'  
C:\hla>
```

The window has a standard Windows interface with a title bar, maximize, minimize, and close buttons, and a scroll bar on the right side.

jeden z możliwych sposobów realizacji zadania



```
male_na_duze.hla - Notatnik
Plik  Edycja  Format  Widok  Pomoc

program male_na_duze;
#include( "stdlib.hhf" );

static
    c:      char;

begin male_na_duze;

    stdout.put( "wprowadz znak: " );
    stdin.getc();
    if( a1>='a' ) then
        if( a1<='z' ) then
            and($5f,a1);
        endif;
    endif;
    stdout.put( "wprowadzony znak po konwersji",nl,"do wielkiej litery to: " );
    stdout.putc( a1 );
    stdout.put( "'",nl);

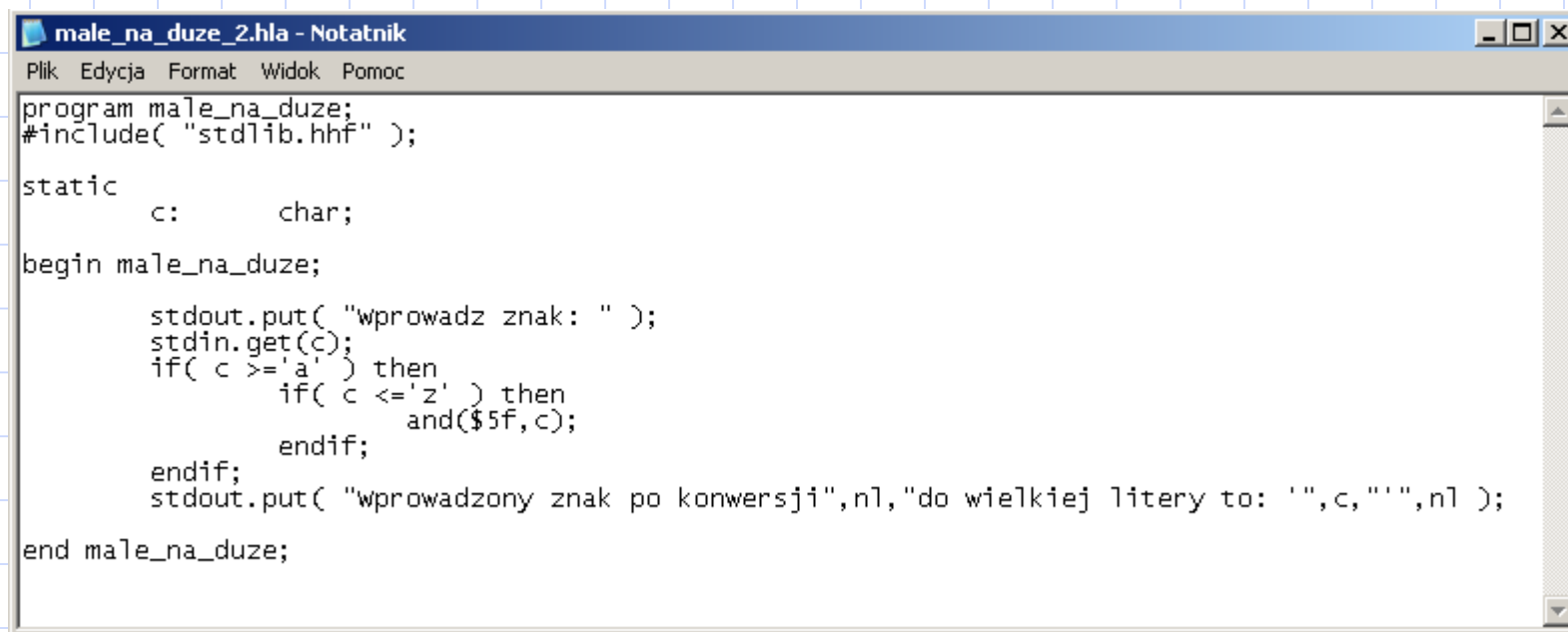
end male_na_duze;
```

Wczytywanie i wyprowadzanie znaków

`stdin.get`

Zmodyfikować program tak, aby wczytywanie znaków odbywało się za pomocą zwykłej procedury `stdin.get`

jeden z możliwych sposobów realizacji zadania



```
male_na_duze_2.hla - Notatnik
Plik  Edycja  Format  Widok  Pomoc

program male_na_duze;
#include( "stdlib.hhf" );

static
    c:      char;
begin male_na_duze;

    stdout.put( "wprowadz znak: " );
    stdin.get(c);
    if( c >='a' ) then
        if( c <='z' ) then
            and($5f,c);
        endif;
    endif;
    stdout.put( "wprowadzony znak po konwersji",nl,"do wielkiej litery to: '",c,"'",nl );
end male_na_duze;
```

stdin.flushInput

Aby wymusić każdorazowe wczytywanie nowego wiersza danych przy pobieraniu kolejnych znaków, należy wywołanie procedury wczytującej znak poprzedzić wywołaniem procedury **stdin.flushInput()**.

Spowoduje to opróżnienie dotychczasowej zawartości bufora i wymusi wprowadzenie nowego wiersza danych w ramach realizacji wywołania `stdin.getc` albo `stdin.get`

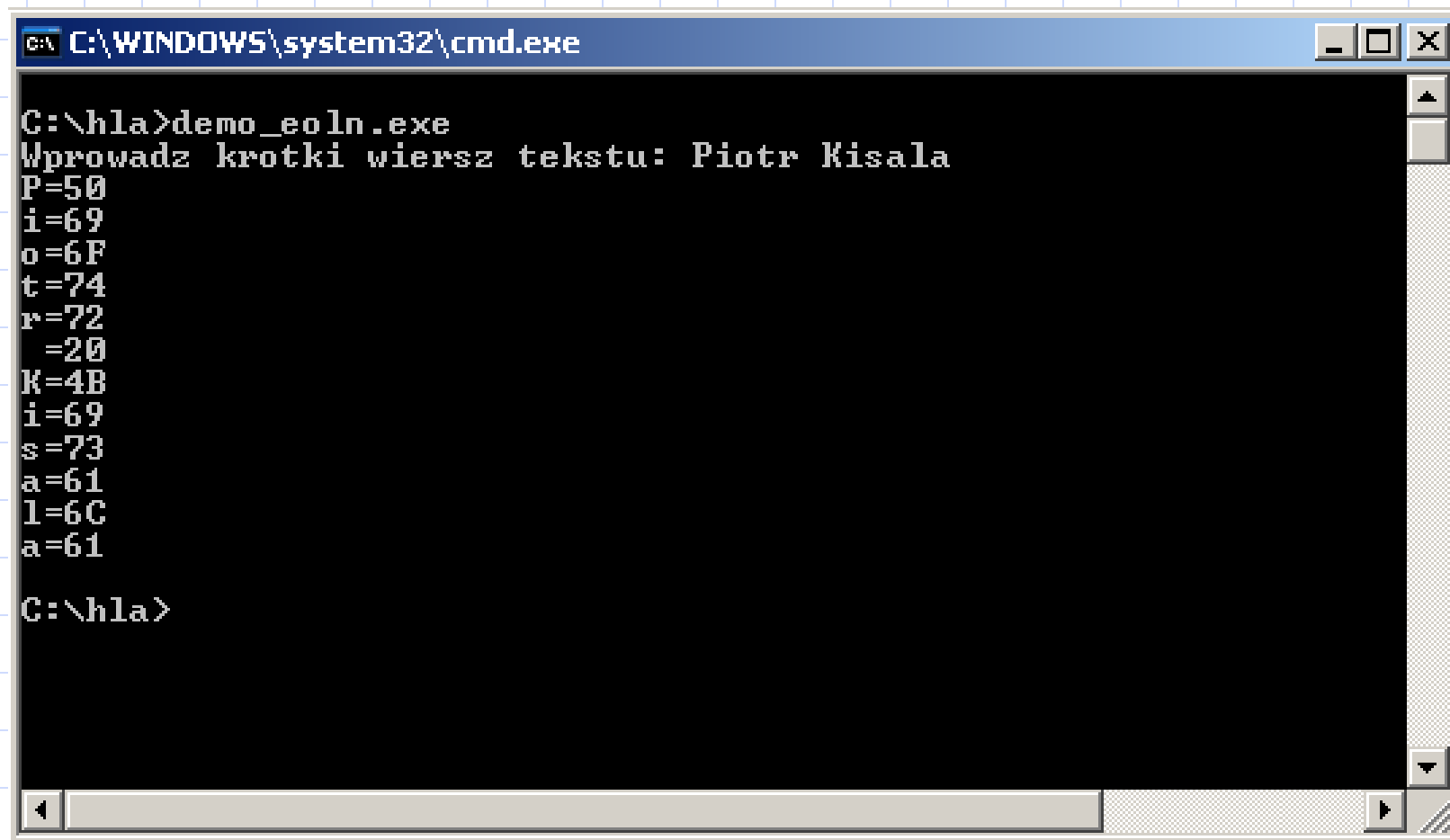
stdin.eoln jako znak Enter

Biblioteka standardowa języka HLA zawiera definicję "końca wiersza". Procedura **stdin.eoln()** zwraca w rejestrze AL wartość jeden, jeśli bieżący wiersz znaków wejściowych został wyczerpany (w innym przypadku rejestr AL zawiera po wywołaniu wartość zero).

wczytywanie kilku znaków

Napisać program wczytujący z klawiatury ciąg znaków (np. "Imię Nazwisko"), wyprowadzający następnie każdy ze znaków w kolumnie pionowej jednocześnie dekodując wartości poszczególnych znaków na postać liczbową szesnastkową, zgodnie z następnym slajdem.

działanie programu



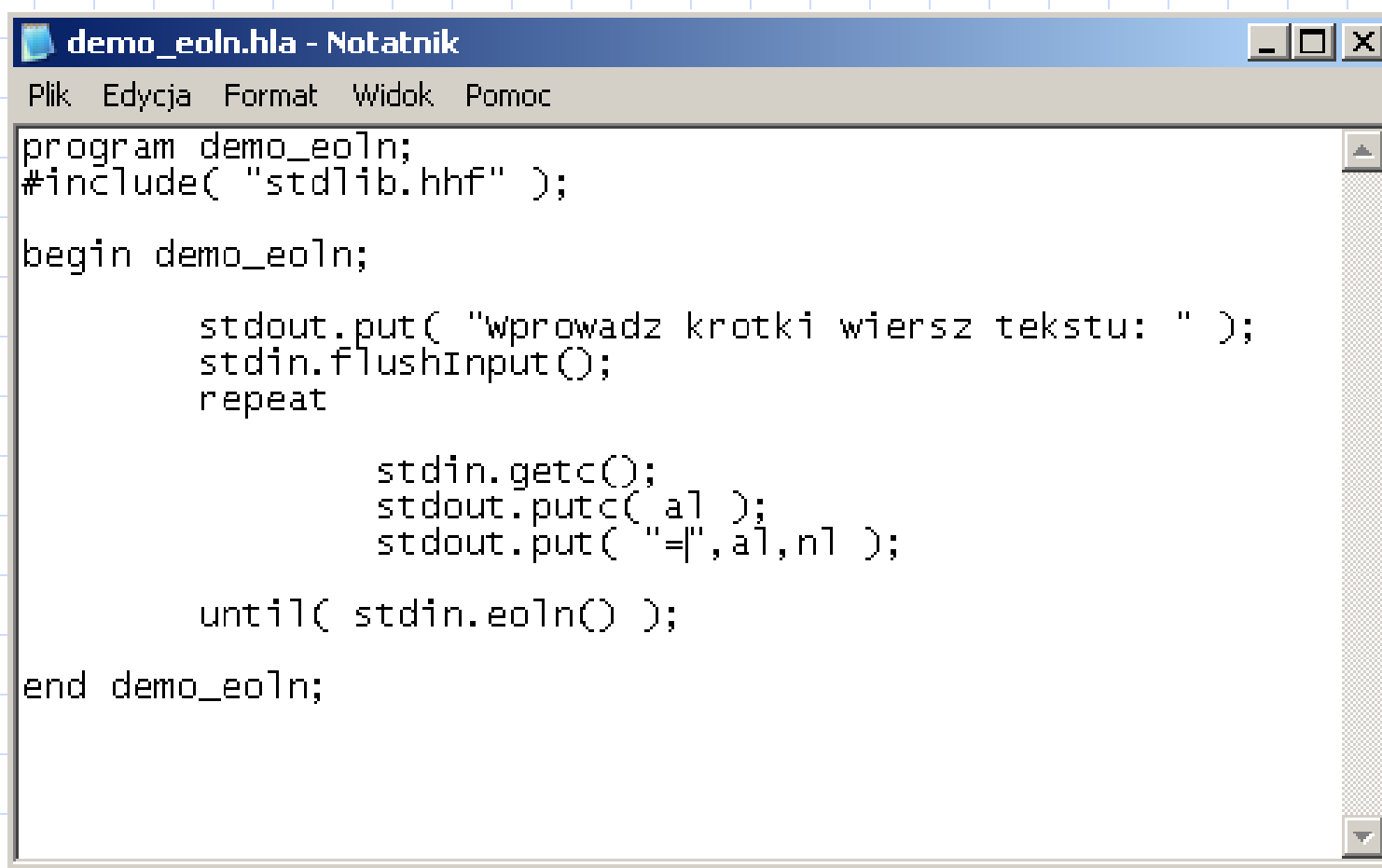
A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
C:\hla>demo_eoln.exe
Wprowadz krotki wiersz tekstu: Piotr Kisala
P=50
i=69
o=6F
t=74
r=72
=20
K=4B
i=69
s=73
a=61
l=6C
a=61

C:\hla>
```

The output consists of a single line of text followed by several lines of hexadecimal values. The window has a scroll bar on the right and a status bar at the bottom.

jeden z możliwych sposobów realizacji zadania



```
demo_eoln.hla - Notatnik
Plik  Edycja  Format  Widok  Pomoc

program demo_eoln;
#include( "stdlib.hhf" );

begin demo_eoln;

    stdout.put( "wprowadz krotki wiersz tekstu: " );
    stdin.flushInput();
    repeat

        stdin.getc();
        stdout.putc( a1 );
        stdout.put( "=", a1, nl );

    until( stdin.eoln() );

end demo_eoln;
```

na dzisiaj
koniec