

Programowanie niskopoziomowe

Wstęp do języka assemblerowego

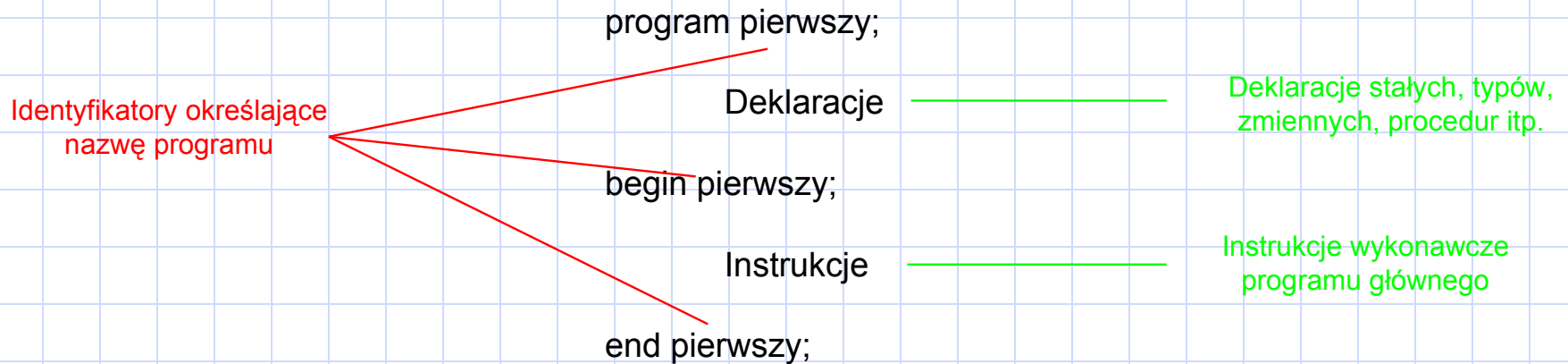
Prowadzący:
Piotr Kisała

Przegląd tematów

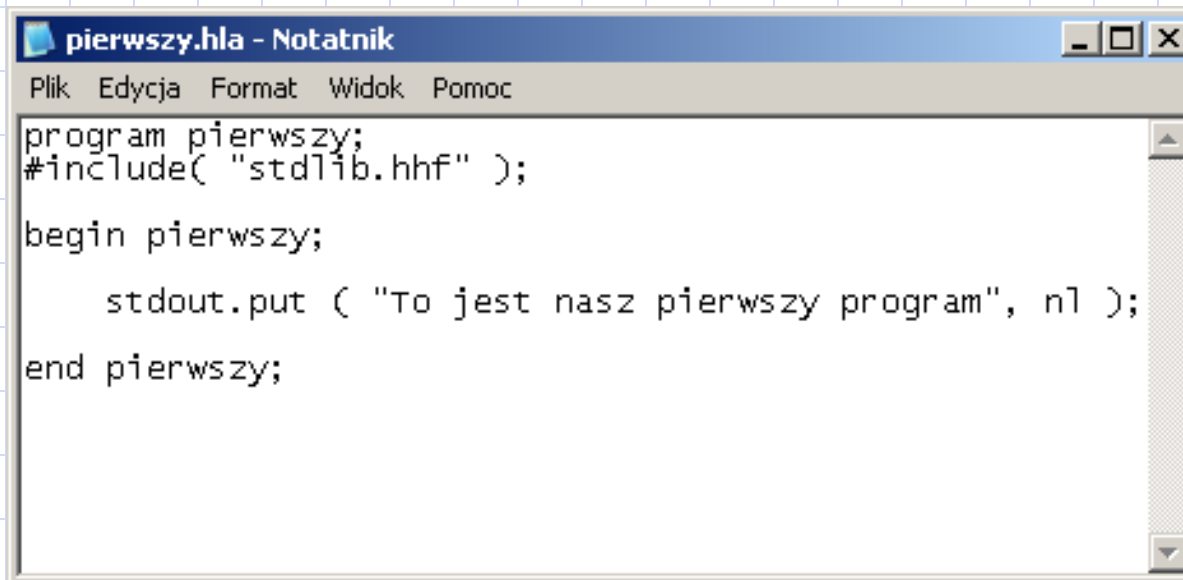
- ◆ Prezentacja podstaw składni programu HLA (High Level Assembly)
- ◆ Architektura procesorów INTEL
- ◆ Biblioteki HLA Standard
- ◆ Pierwsze programy w języku assemblerowym

Struktura programu

Ważne: zachowanie wielkości poszczególnych liter



Pierwszy program



```
program pierwszy;  
#include( "stdlib.hhf" );  
  
begin pierwszy;  
    stdout.put ( "To jest nasz pierwszy program", nl );  
end pierwszy;
```

Opis programu „pierwszy”

instrukcja `#include` instruuje kompilator języka HLA, aby ten włączył do kodu programu zestaw deklaracji zapożyczonych z pliku `stdlib.hhf` (pliku nagłówkowego biblioteki standardowej HLA).

Plik ten zawiera między innymi deklaracje kodu procedury `stdout.put` wykorzystywanej w kodzie programu.

Źródła

Całość oprogramowania
niezbędnego do skompilowania i
uruchamiania programów HLA
można znaleźć na stronie
internetowej:

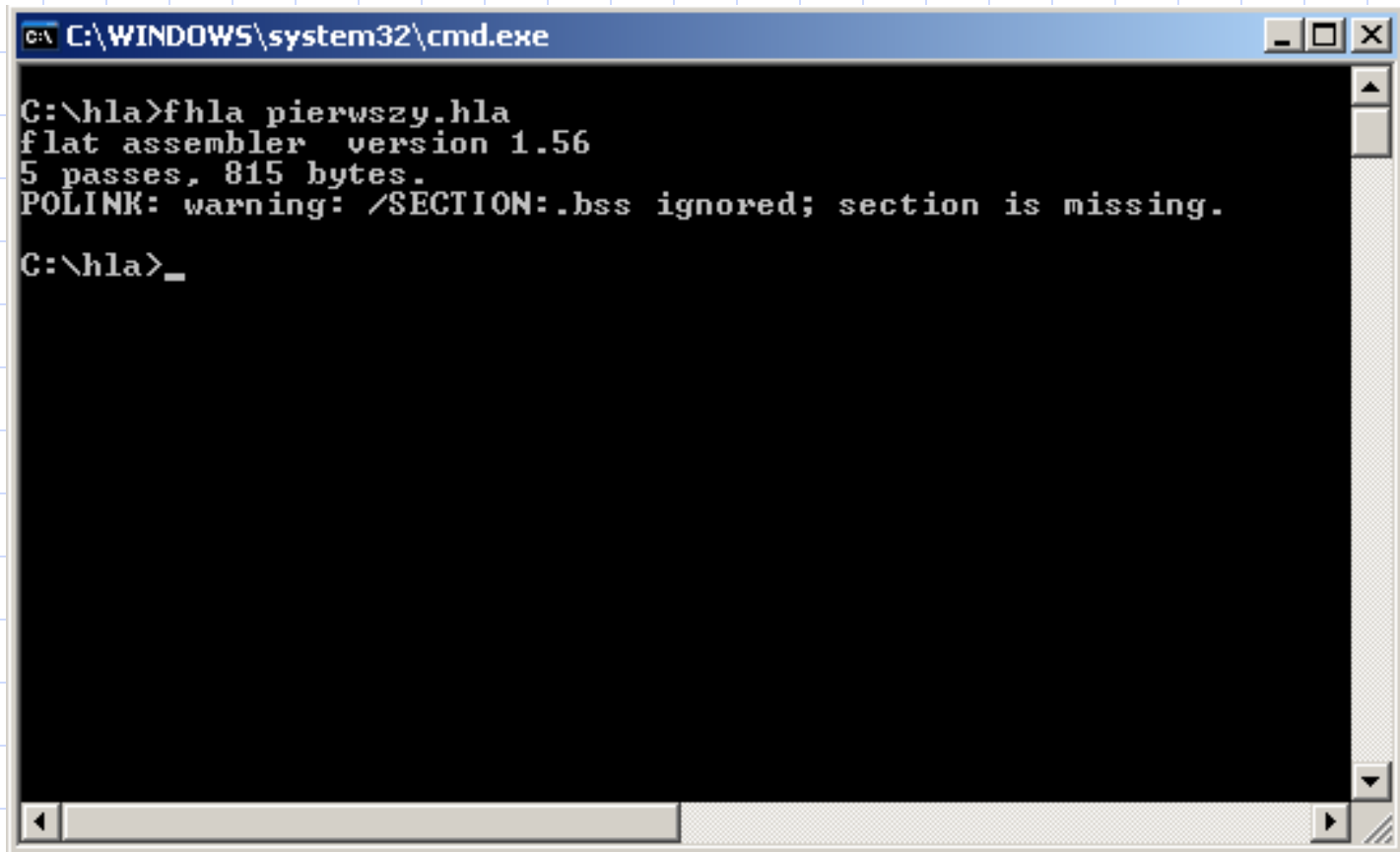
<http://webster/cs/ucr/edu>

- najnowsze wersje oprogramowania
- artykuły, opisy, instrukcje

Programy języka HLA

- ◆ Programy języka HLA można tworzyć wykorzystując edytory programistyczne stanowiące wyposażenie zintegrowanych środowisk programistycznych.
- ◆ Kompilator języka HLA jest typowym kompilatorem wiersza poleceń, może być zatem uruchamiany z poziomu wiersza poleceń (Windows) czy też powłoki (Linux).

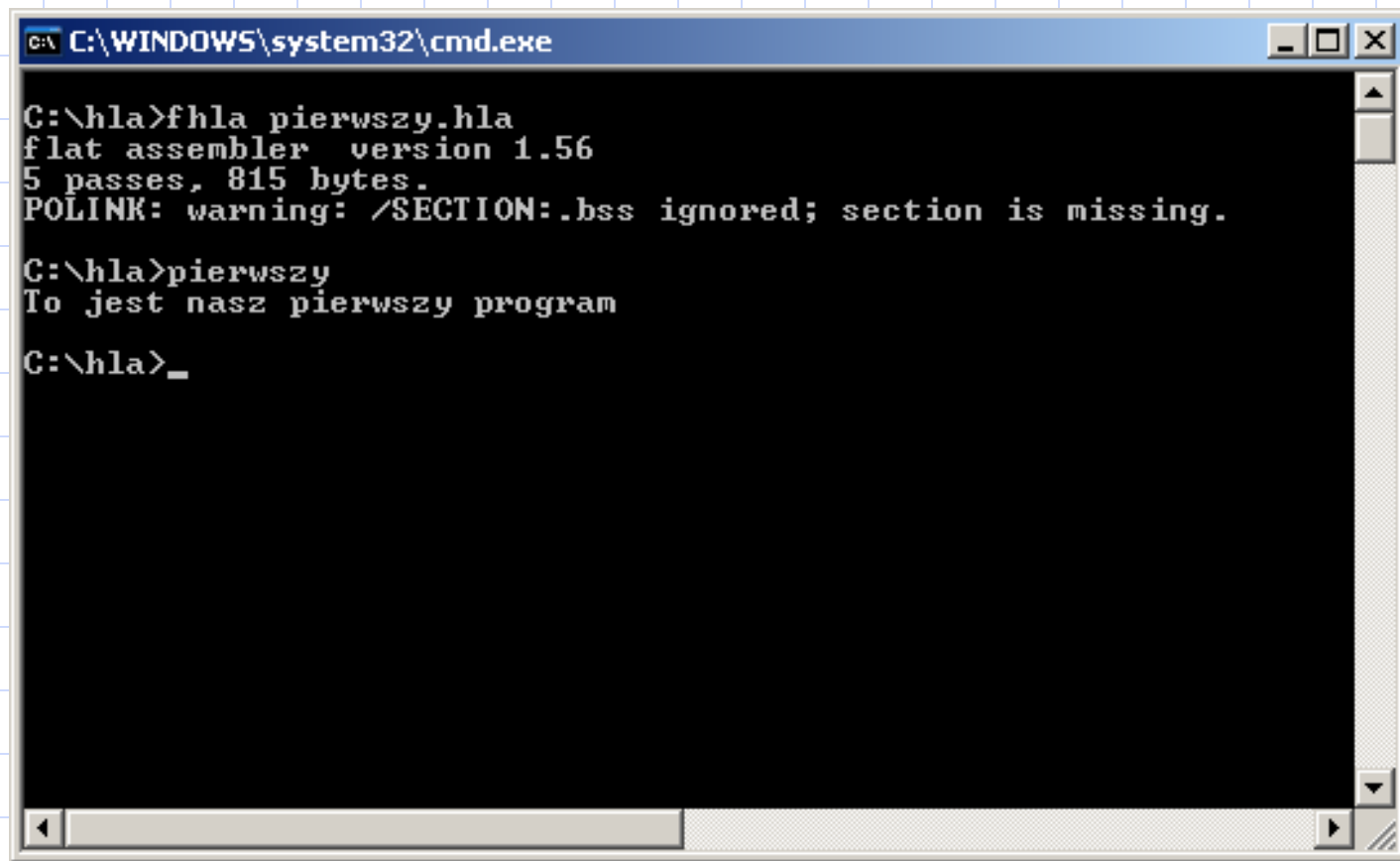
Kompilacja programu „pierwszy”



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

C:\hla>fhla pierwszy.hla
flat assembler  version 1.56
5 passes, 815 bytes.
POLINK: warning: /SECTION:.bss ignored; section is missing.
C:\hla>_
```


Uruchomienie programu „pierwszy”



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

C:\hla>fhla pierwszy.hla
flat assembler version 1.56
5 passes, 815 bytes.
POLINK: warning: /SECTION:.bss ignored; section is missing.

C:\hla>pierwszy
To jest nasz pierwszy program

C:\hla>_
```

Typy wartości całkowitych

- ◆ int8 - liczba całkowita 8-bitowa
- ◆ int16 - liczba całkowita 16-bitowa
- ◆ int32 - liczba całkowita 32-bitowa

Sekcja deklaracji zmiennych

static

```
zmienna1:    int8;  
zmienna2:    int16;  
zmienna3:    int32;
```

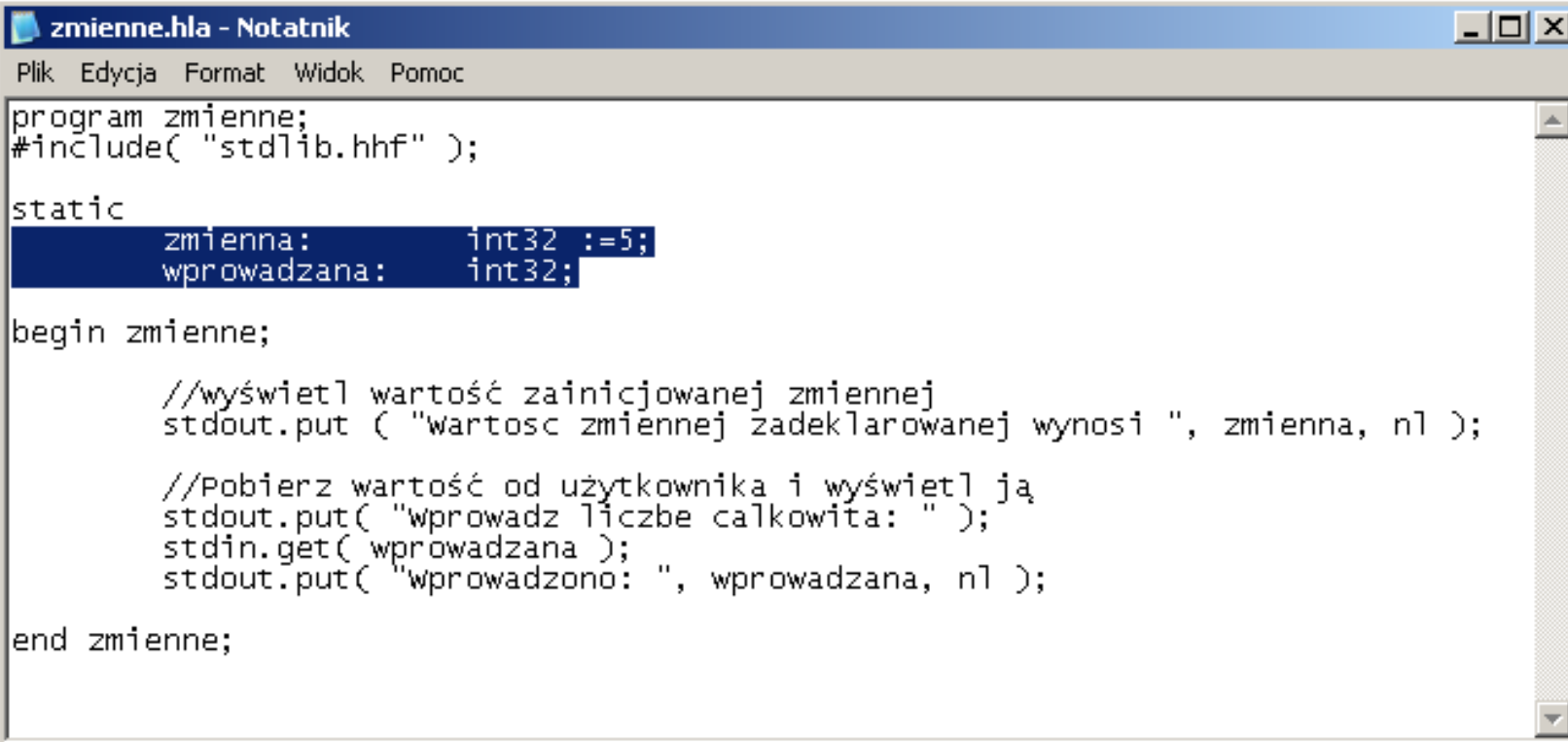
Deklaracja wartości początkowych

W sekcji deklaracji zmiennych statycznych można zainicjować deklarowaną zmienną wartością początkową. Wartość ta zostanie przypisana do zmiennej podczas wczytywania programu do pamięci przez system operacyjny.

static

```
zmienna1: int8  := 8;  
zmienna2: int16 := 1580;  
zmienna3: int32 := -311256;
```

Wprowadzanie i wyprowadzanie zmiennych



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

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

static
    zmienna:      int32 :=5;
    wprowadzana:  int32;

begin zmienne;

    //wyświetl wartość zainicjowanej zmiennej
    stdout.put ( "wartosc zmiennej zadeklarowanej wynosi ", zmienna, nl );

    //Pobierz wartość od użytkownika i wyświetl ją
    stdout.put( "wprowadz liczbę całkowitą: " );
    stdin.get( wprowadzana );
    stdout.put( "wprowadzono: ", wprowadzana, nl );

end zmienne;
```

stdout.put

Procedura wyprowadzania napisów na wyjście programu, dostępna w module obsługi standardowego wyjścia.

Składnia:

`stdout.put (lista wyprowadzanych wartości);`

Lista argumentów wywołania procedury `stdout.put` może zostać konstruowana ze stałych, rejestrów i zmiennych. Kolejne argumenty oddziela się przecinkami.

stdin.get

Procedura odczytuje wartość wprowadzaną ze standardowego urządzenia wejściowego (zwykle klawiatura), konwertuje ją do postaci całkowitej i przypisuje otrzymaną wartość do zmiennej określonej parametrem wywołania – w naszym przypadku jest to zmienna o nazwie „wprowadzana”.

Składnia:

stdin.get (nazwa zmiennej);

Wartości logiczne

Zmienną logiczną deklaruje się, określając w miejsce typu typ **boolean**.

static

zmiennalogsyczna:	boolean;
falszlogiczny:	boolean := false;
prawdalogsyczna:	boolean := true;

Jako że zmienne logiczne są obiektami jednobajtowymi, można nimi manipulować przy wykorzystaniu dowolnych instrukcji operujących bezpośrednio na operandach ośmiobitowych.

Wartości znakowe

Wartości znakowe są obiektami jednobajtowymi. Typ obiektów, którymi są wartości znakowe to: **char**. Zmienne znakowe można inicjalizować literałami znakowymi; literały takie należy ograniczyć znakami pojedynczego cudzysłowu. Przykład:

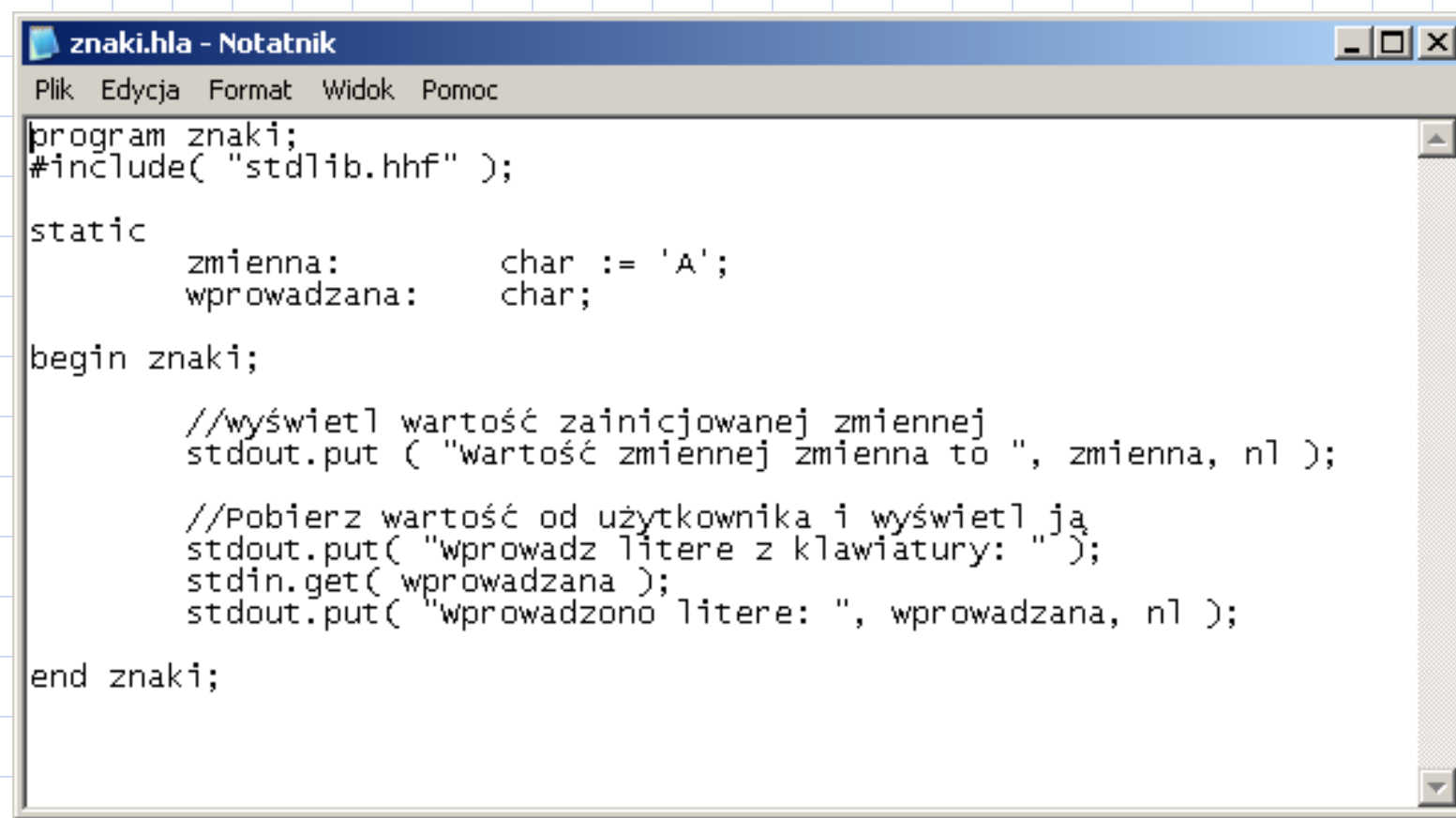
static

```
naszznak:  char;  
literaA:   char := 'A';
```


Program znaki.exe

- ◆ Wykorzystując wiadomości zdobyte przy pisaniu programu zmienne:
 - stworzyć program o nazwie znaki
 - zadeklarować znak A
 - wyświetlić znak A na ekranie
 - Pobrać znak z klawiatury i wyświetlić go również na ekranie

Przykładowe rozwiązanie



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

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

static
    zmienna:      char := 'A';
    wprowadzana:  char;

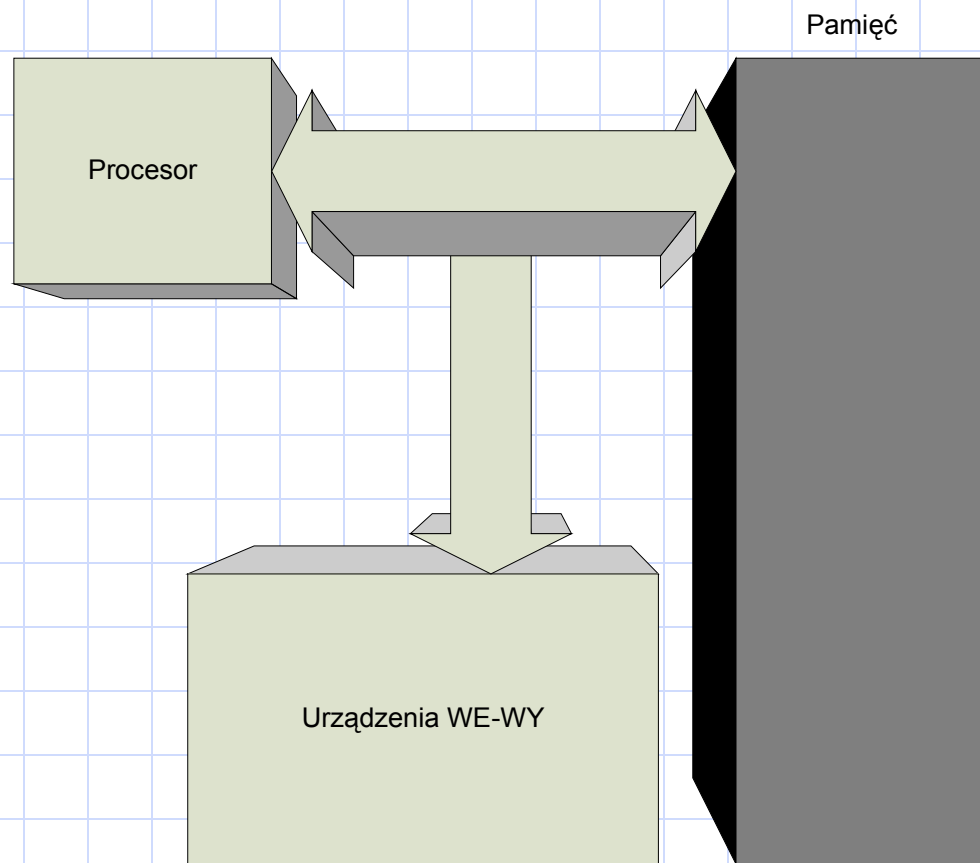
begin znaki;

    //wyświetl wartość zainicjowanej zmiennej
    stdout.put ( "wartość zmiennej zmienna to ", zmienna, nl );

    //Pobierz wartość od użytkownika i wyświetl ją
    stdout.put( "wprowadz literę z klawiatury: " );
    stdin.get( wprowadzana );
    stdout.put( "wprowadzono literę: ", wprowadzana, nl );

end znaki;
```

Architektura proc. 80x86



Magistrala systemowa

- ◆ linie danych
- ◆ linie adresowe
- ◆ linie sterujące

CPU komunikuj się z pamięcią i urządzeniami wejścia-wyjścia przez

odpowiednie wysterowanie linii adresowych określające adres pamięci

bądź numer urządzenia wejścia-wyjścia; każda z komórek pamięci i każde

z urządzeń wejścia wyjścia dysponuje własnym, unikalnym adresem.

Następnie procesor wymienia dane z pamięcią lub urz. we-wy odpowiednio

sterując stanami linii danych. Stan linii sterujących określa

Rejestry procesorów 80x86

- ◆ rejestry ogólnego przeznaczenia
- ◆ rejestry specjalne trybu użytkownika
- ◆ rejestry segmentowe
- ◆ rejestry specjalne trybu nadzoru

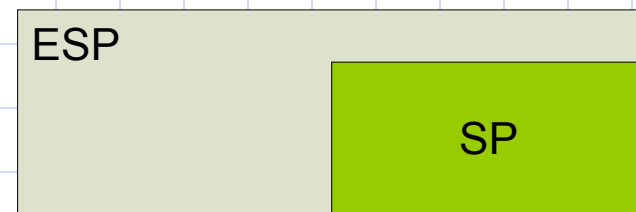
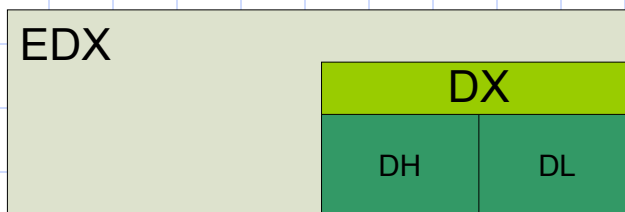
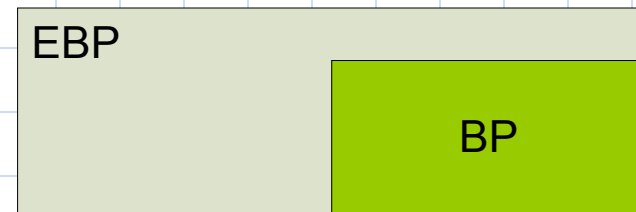
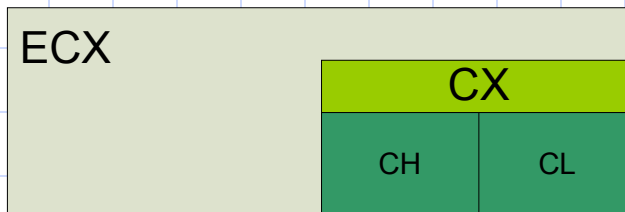
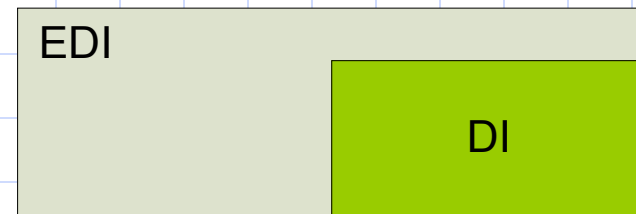
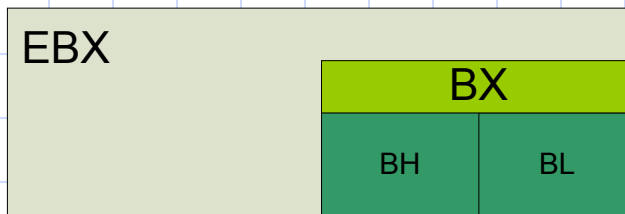
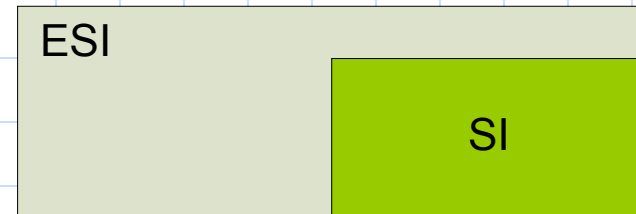
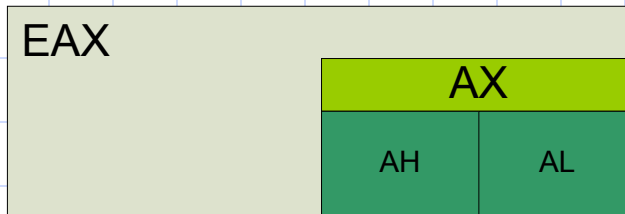
Segmentowe – nie wykorzystywane w 32-bitowych syst. oper. (jak Windows, Linux).

Specjalne trybu nadzoru – wykorzystywane tylko przez twórców systemów operacyjnych, debuggerów i innych specjalistycznych narzędzi systemowych.

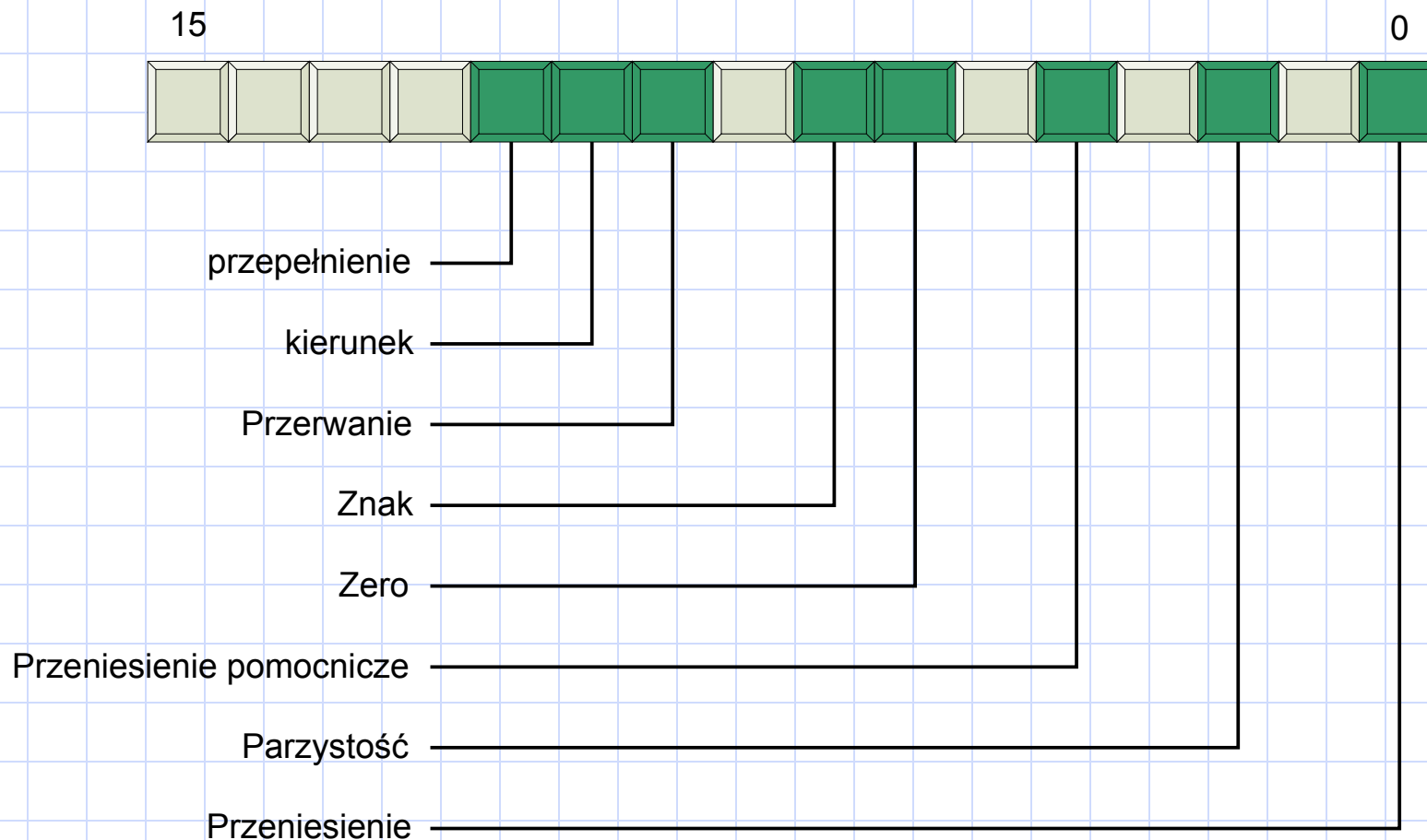
Rejestry ogólnego przeznaczenia

- ◆ Sposób wykorzystania zależny wyłącznie od programisty.
- ◆ 8 rejestrów 32-bitowych:
EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.
Przedrostek E od extended, rozróżnia on rej. 32 od 16-bitowych
- ◆ 8 rejestrów 16-bitowych:
AX, CX, DX, SI, DI, BP, SP.
- ◆ 8 rejestrów 8-bitowych:
AL, AH, BL, BH, CL, CH, DL, DH.

Rejestry ogólnego przeznaczenia



Rejestr EFLAGS



Rejestr EFLAGS

- ◆ Większość bitów (znaczników) tego rejestru zarezerwowana jest dla trybu nadzoru (czyli dla kodu syst. oper.). Programistów interesuje jedynie 8 bitów tego rejestru.
- ◆ 4 znaczniki są szczególnie ważne:
 - przepełnienia
 - przeniesienia
 - znaku
 - zera

Funkcje rejestrów

Każda operacja angażuje rejestry. Aby dodać do siebie 2 wartości i umieścić ich sumę w trzeciej, należy załadować jeden ze składników do rejestru, dodać do niego (w rejestrze) drugi składnik sumy i dopiero potem wynik skopiować z rejestru do miejsca przechowywania sumy.

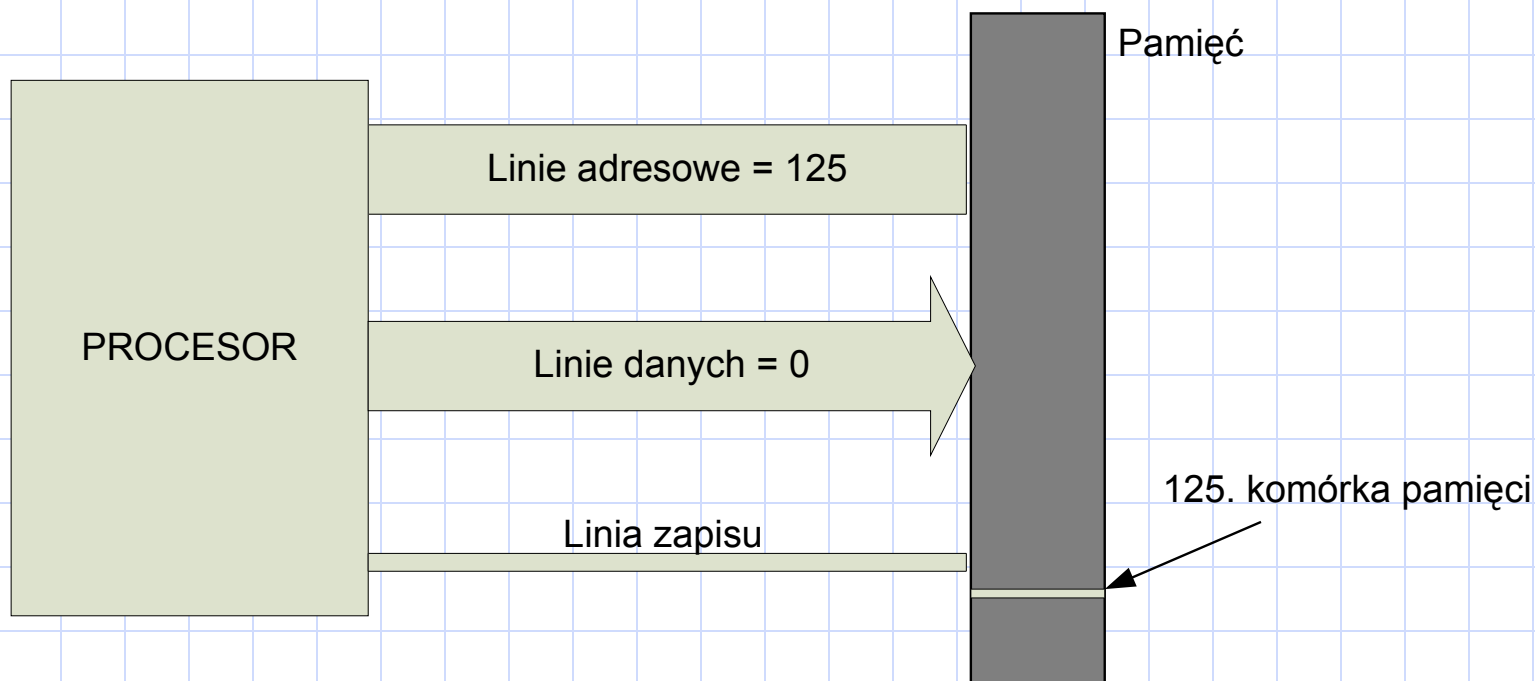
Rejestry stanowią bazę wszelkich obliczeń.

Obsługa pamięci

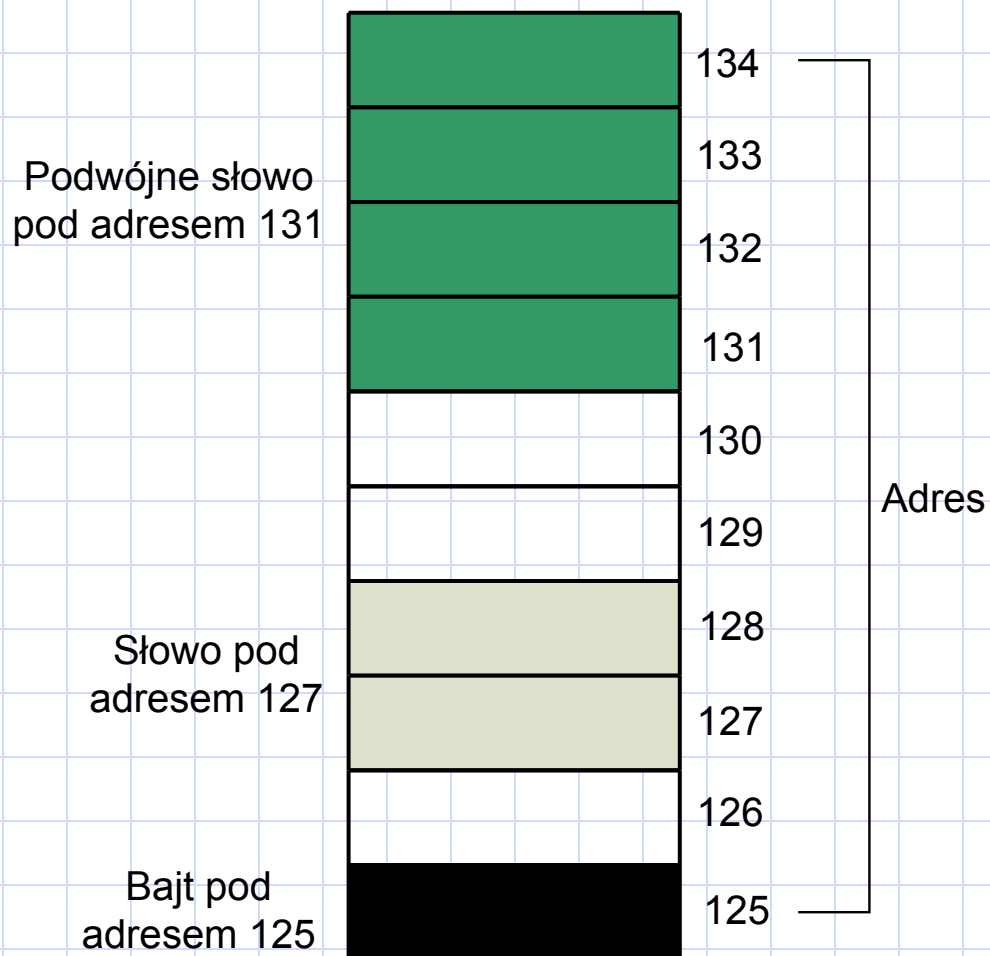
- ◆ Typowy procesor działający pod kontrolą 32-bitowego systemu operacyjnego może odwoływać się najwyżej do 2^{32} różnych adresów pamięci, czyli do nieco ponad czterech miliardów komórek pamięci – bajtów.
- ◆ Procesory 80x86 obsługują pamięć adresowaną bajtowo. Podstawową jednostką pamięci jest bajt, który wystarcza do zakodowania pojedynczego znaku bądź niewielkiej liczby całkowitej.
- ◆ Pamięć jest liniową tablicą bajtów. Adresem pierwszego bajta w tej tablicy jest 0, ostatni ma adres $2^{32} - 1$.

Operacja zapisu do pamięci

Aby zapisać do komórki pamięci o adresie 125 wartości 0 procesor umieszcza wartość zero na magistrali danych, wartość 125 na magistrali adresowej oraz ustawia linię zapisu (zapis to zwykle wyzerowanie).



Zapis/odczyt słowa i słowa podwójnego



Instrukcja maszynowa

mov

Służy do przemieszczania danych pomiędzy lokacjami

mov(operand źródłowy, operand docelowy);

operandem źródłowy: zmienna, rejestr, stała

operand docelowy: zmienna, rejestr

operandy muszą być tych samych rozmiarów !!!

Instrukcja maszynowa *add*, *sub*

dodawanie danych:

add(operand źródłowy, operand
docelowy);

odejmowanie danych:

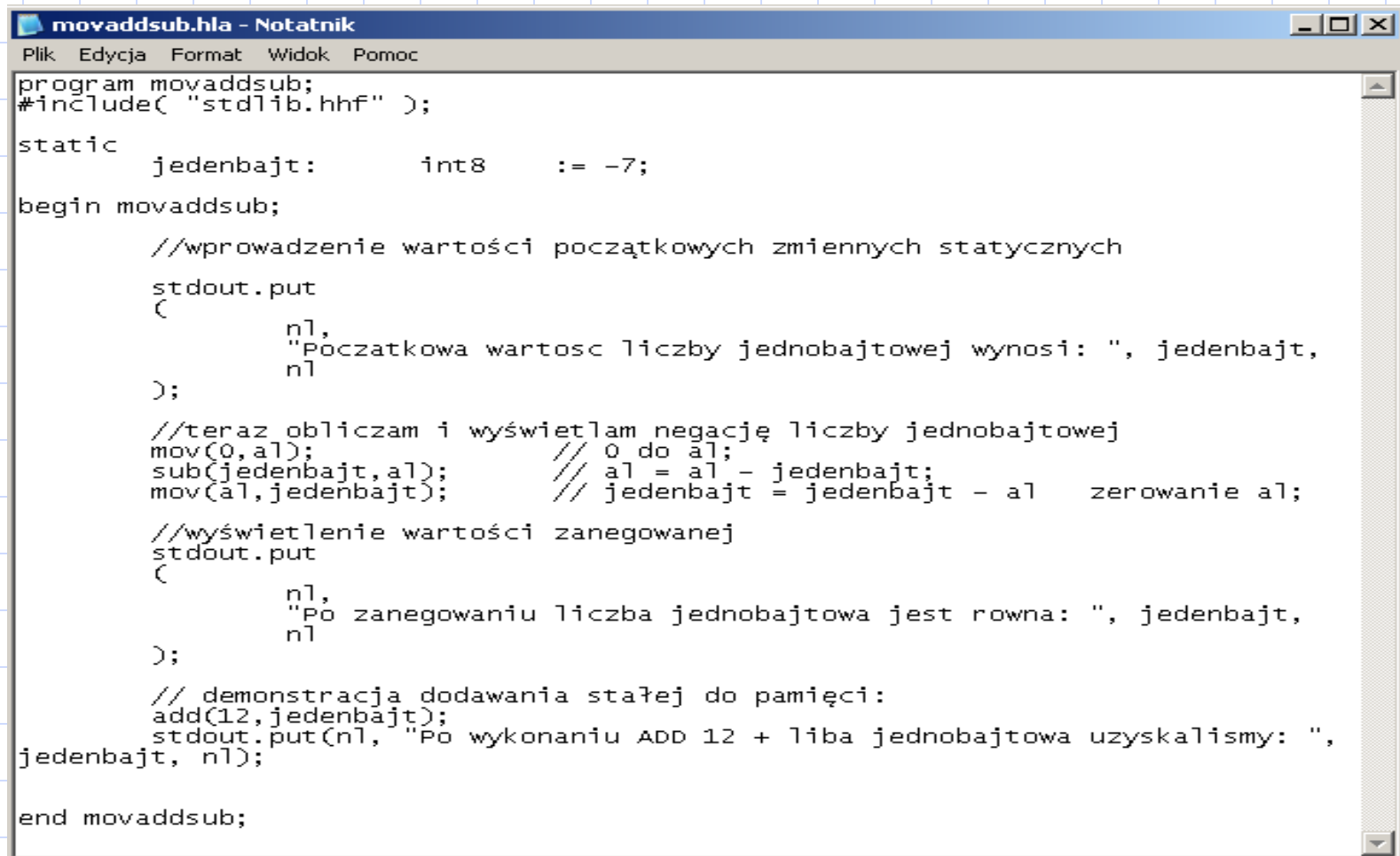
sub(operand źródłowy, operand
docelowy);

Wykorzystanie mov, add, sub

- ◆ Program o nazwie movaddsub.hla
- ◆ Wprowadzić stałą „jedenbajt” o długości 1-bajta i wartości -7.
- ◆ Wyświetlić stałą na ekranie.
- ◆ Dokonać negacji wpisanej liczby wykorzystując rejestr 8 bitowy rejestr ogólnego przeznaczenia AL.
- ◆ Wyświetlić wartość zanegowanej stałej.
- ◆ Do zanegowanej stałej dodać wartość 12.
- ◆ Wyświetlić wynik dodawania.

Komentarze ułatwiają wykrywanie błędów.

jeden z możliwych sposobów realizacji zadania „movaddsub”



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

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

static
    jedenbajt:      int8      := -7;
begin movaddsub;

    //wprowadzenie wartości początkowych zmiennych statycznych
    stdout.put
    (
        nl,
        "Początkowa wartość liczby jednobajtowej wynosi: ", jedenbajt,
        nl
    );

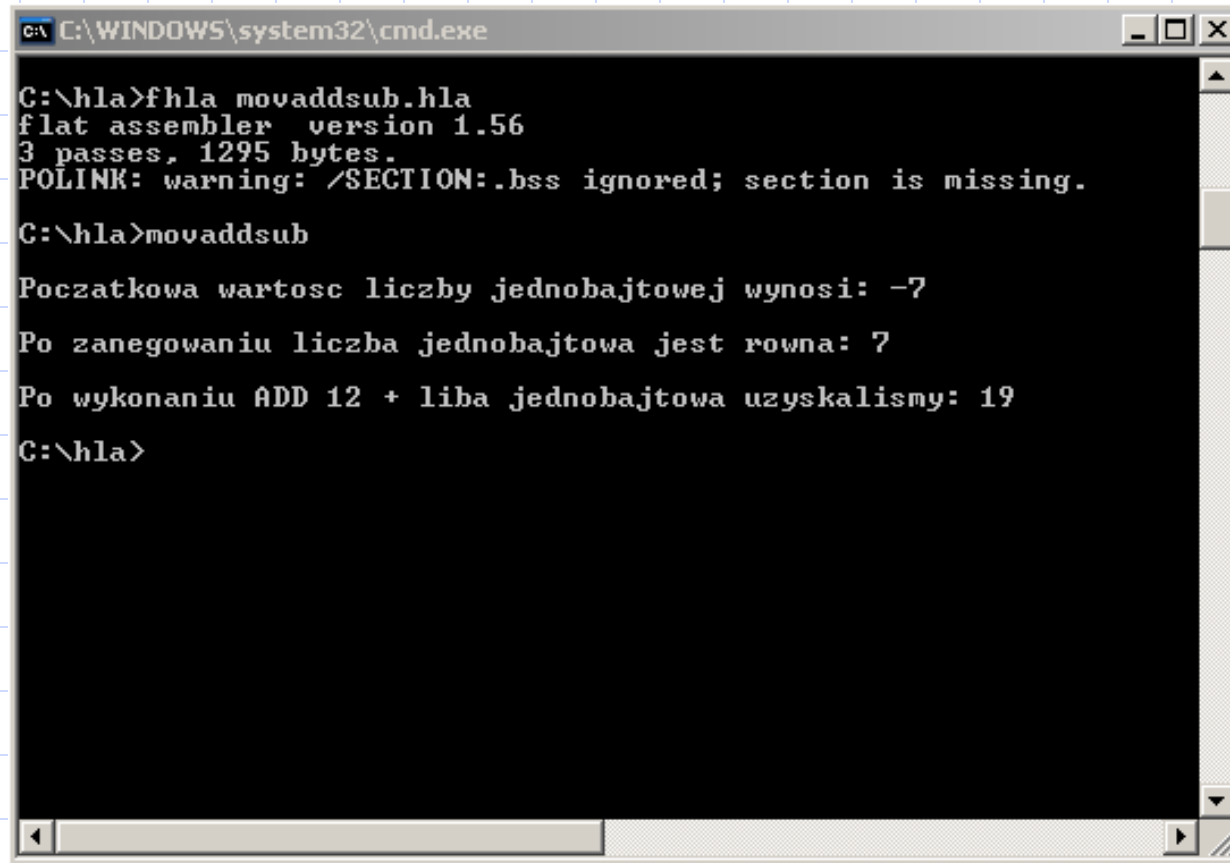
    //teraz obliczam i wyświetlam negację liczby jednobajtowej
    mov(0,al);           // 0 do al;
    sub(jedenbajt,al);    // al = al - jedenbajt;
    mov(al,jedenbajt);    // jedenbajt = jedenbajt - al   zerowanie al;

    //wyświetlenie wartości zanegowanej
    stdout.put
    (
        nl,
        "Po zanegowaniu liczba jednobajtowa jest równa: ", jedenbajt,
        nl
    );

    // demonstracja dodawania stałej do pamięci:
    add(12,jedenbajt);
    stdout.put(nl, "Po wykonaniu ADD 12 + 1ba jednobajtowa uzyskalismy: ",
jedenbajt, nl);

end movaddsub;
```

programu „movaddsub”



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

C:\hla>fhla movaddsub.hla
flat assembler  version 1.56
3 passes, 1295 bytes.
POLINK: warning: /SECTION:.bss ignored; section is missing.

C:\hla>movaddsub

Poczatkowa wartosc liczby jednobajtowej wynosi: -7
Po zanegowaniu liczba jednobajtowa jest rowna: 7
Po wykonaniu ADD 12 + liba jednobajtowa uzyskalismy: 19
C:\hla>
```

Działania na liczbach 1,2,3-bajtowych

- ◆ Zmodyfikować program movaddsub.exe
- ◆ Zadeklarować stałe o długości 1,2 i 3 bajtów
- ◆ Wartości poszczególnych zmiennych:
 - zmienna 1-bajtowa = -7
 - zmienna 2-bajtowa = -277
 - zmienna 3-bajtowa = -66000
- ◆ Dokonać ich negacji oparciu o rejestry ogólnego przeznaczenia:
odpowiednio AL, AX, EAX.
Dodać do wartości zanegowanej zmiennej 3-bajtowej wartość 666

Procedura stdout.put – c.d.

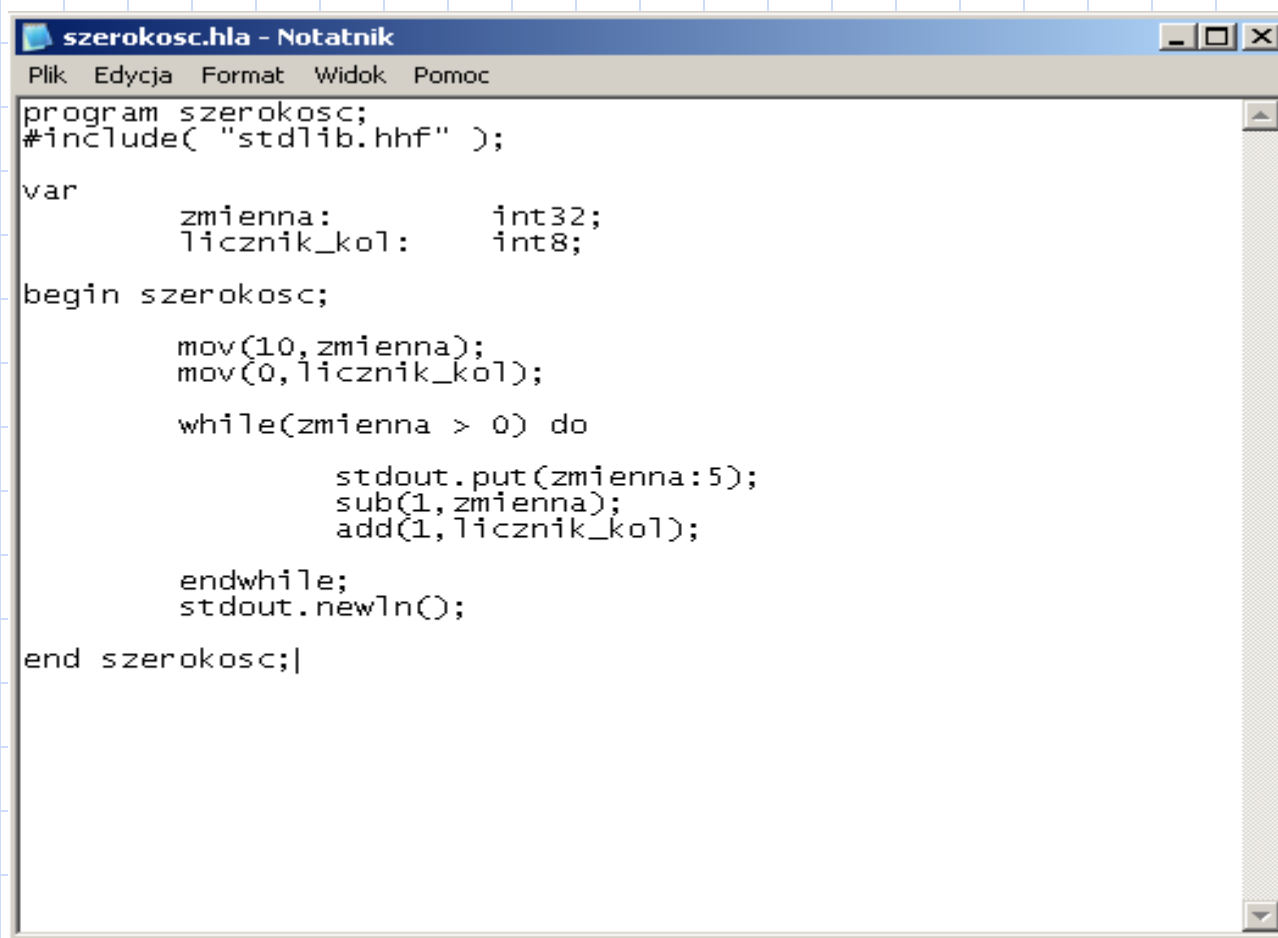
- ◆ Lista argumentów może zostać skonstruowana ze stałych, rejestrów i zmiennych, kolejne argumenty oddziela się przecinkami.
- ◆ Każdy z argumentów wywołania może być zadany w jednej z dwóch postaci:

wartość

wartość:szerokość

jest to minimalna szerokość napisu reprezentującego wartość

Przykład wykorzystania operandu wartość:szerokość



```
program szerokosc;
#include( "stdlib.hhf" );

var
    zmienna:      int32;
    licznik_kol:  int8;

begin szerokosc;

    mov(10,zmienna);
    mov(0,licznik_kol);

    while(zmienna > 0) do
        stdout.put(zmienna:5);
        sub(1,zmienna);
        add(1,licznik_kol);

    endwhile;
    stdout.newln();

end szerokosc;
```

Pętla *while*

while(warunek) do

instrukcja

 bądź ich

cały blok

endwhile;

Instrukcja *for*

for(wyrażenie inicjalizujące:warunek:instrukcja licznika) do

instrukcja

bądź ich

cały blok

endfor;

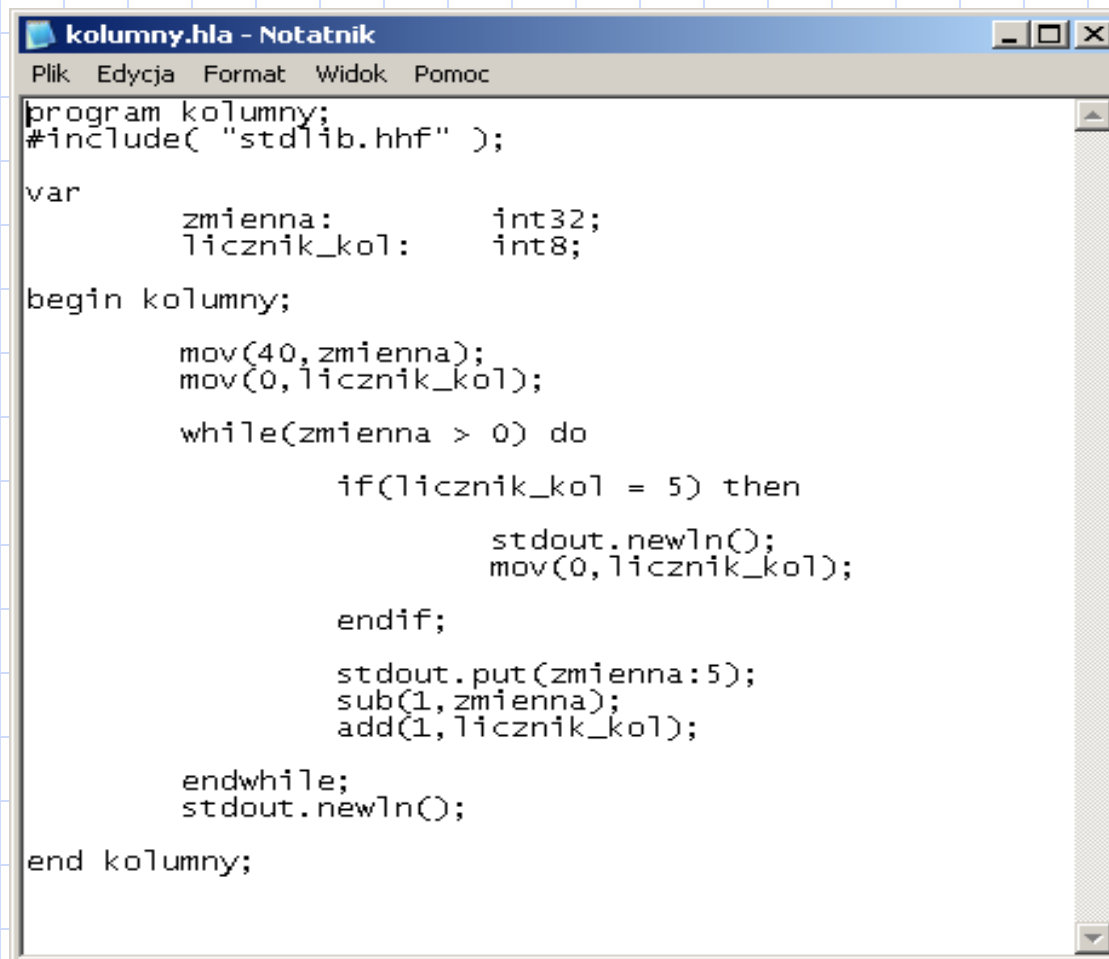
Kolumny

- ◆ Wykorzystując instrukcje *while* i *for* ułożyć liczby od 40-1 w 5 kolumnach w 8 wierszach

podpowiedź – nowa linia

`stdout.newln();`

Możliwa realizacja programu kolumny



```
program kolumny;
#include( "stdlib.hhf" );

var
    zmienna:      int32;
    licznik_kol:  int8;

begin kolumny;

    mov(40,zmienna);
    mov(0,licznik_kol);

    while(zmienna > 0) do
        if(licznik_kol = 5) then
            stdout.newln();
            mov(0,licznik_kol);
        endif;

        stdout.put(zmienna:5);
        sub(1,zmienna);
        add(1,licznik_kol);

    endwhile;
    stdout.newln();

end kolumny;
```

Moduł stdio

- ◆ `stdio.bell` – znak dzwonka (głośniczek systemowy)
- ◆ `stdio.bs` – znak cofania kursora
- ◆ `stdio.tab` – znak tabulacji
- ◆ `stdio.lf` – znak wysuwu wiersza
- ◆ `stdio.cr` – znak powrotu karetki

występowanie – jak „`nl`” w procedurze `stdout.put`

zmodyfikować program kolumny dodając
dzwonek po
każdym wierszu

Procedura zakończenia

end laboratorium;