Architektura Systemów Komputerowych – Laboratorium nr 1: wprowadzenie do asemblera x86

Piotr Bilski

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z podstawowymi technikami programowania komputerów klasy IBM PC wyposażonych w procesor z rodziny x86. W szczególności chodzi tu o składnię wybranego asemblera (w tym przypadku jest to Netwide Assembler – NASM), zapoznanie się z podstawowymi instrukcjami maszynowymi i ich składnią, strukturą kodu, sposobami adresowania oraz podstawowymi wywołaniami systemu operacyjnego z rodziny DOS.

2. Program ćwiczenia
    a. Podstawowy program typu HelloWorld (plik **cw1.asm**)
    b. Zarządzanie sekcją danych, tworzenie zmiennych (plik **cw2.asm**)
    c. Komunikacja z pamięcią, aktualizacja jej zawartości i arytmetyka na adresach (pliki **cw4a.asm**, **cw4b.asm**, **cw5.asm**)
    d. Wykorzystanie funkcji systemowych BIOSa i DOSa. Wypisywanie w konsoli pojedynczych znaków (pliki **cw1.asm**, **cw3.asm**, **cw6.asm, cw7.asm**)
    e. Demonstracja „końcowości" procesorów Intel (plik **cw8.asm**)
    f. Operacja arytmetyczno-logiczne (plik **cw9.asm**)
3. Zadania do wykonania
    a. Zmodyfikować program **cw1.asm** tak, aby na ekranie wypisywane było własne imię i nazwisko. Ma ono być zdefiniowane w zmiennej jako ciąg kodów ASCII (a nie napis!). Należy stworzyć dwa warianty: wypisujący imię i nazwisko (oddzielone spacją) w jednej linii oraz wypisujący imię i nazwisko w dwóch oddzielnych liniach.
    b. Zmodyfikować program cw9.asm tak, aby mnożył dwie liczby i sprawdzał, czy wynik jest większy od (wpisanej na sztywno) wartości progowej. W drugiej kolejności należy zmodyfikować program, aby wykonywał dzielenie całkowitoliczbowe (liczb całkowitych bez znaku) i sprawdzał, czy wynik jest większy od zadanego progu oraz czy liczba jest parzysta.
    c. Napisać program, który umożliwi utworzenie hasła, które zostanie zapisane w pamięci. Hasło powinno być „ukryte" w konsoli, tzn. podczas wprowadzania kolejnych znaków powinny pojawiać się na ekranie znaki '*'. Następnie program powinien umożliwić wprowadzenie hasła oraz sprawdzić, czy jest ono zgodne z ustawionym wcześniej.
4. Wprowadzenie teoretyczne

4.1. Asemblery

Jest to podstawowa rodzina języków dla procesorów w systemach komputerowych. Charakteryzują się one największymi możliwościami kontroli działań procesora, ponieważ programista operuje bezpośrednio na mnemonikach odpowiadających instrukcjom maszynowym rozpoznawanym przez procesor. Z tego powodu, aby poprawnie tworzyć programy w asemblerze, należy znać listę instrukcji maszynowych procesora oraz ich składnię. W przeciwieństwie do wszystkich innych języków programowania, kod źródłowy asemblera nie wymaga kompilacji, a jedynie przetłumaczenia „jeden-do-jeden" mnemoników oraz ich argumentów na odpowiednie binarne instrukcje maszynowe. Proces ten nazywany jest asemblacją. Dla architektury x86 powstało wiele różnorodnych asemblerów w przeszłości (np. Turbo Assembler firmy Borland), obecnie również istnieje szereg narzędzi służących do tworzenia kodu maszynowego na najniższym możliwym poziomie. Należy przy tym dodać, że termin „asembler" oznacza nie tylko język, ale cały zestaw narzędzi służący do generowania binarnego kodu wykonywalnego przez komputer: moduł łączący (ang. linker), ładujący (ang. loader) itp. Stąd wynika różnorodność asemblerów dostępnych dla jednej architektury sprzętowej. Obecnie do popularnych asemblerów x86 zaliczyć można Netwide Assembler (NASM), Flat Assembler (FASM), czy też Microsoft Macro Assembler (MASM). Na potrzeby laboratorium wybrany został NASM ze względu na prostotę tworzenia kodu oraz dużą popularność wśród programistów. W zależności od zastosowanej składni, rozróżnia się dwa typy składni, tzw. „Intel" oraz „AT&T": pierwsza charakterystyczna dla asemblerów projektowanych dla systemu operacyjnego Windows, druga wykorzystywana przez asemblery tworzone przede wszystkim dla systemów Linux. Różnice między nimi wyrażają się w kolejności interpretacji argumentów instrukcji maszynowej, konstrukcji adresu efektywnego, obliczania długości argumentu oraz oznaczania rejestrów i argumentów natychmiastowych. W przypadku narzędzia NASM stosowana jest składnia typu „Intel" i taka będzie zaprezentowana w przykładach.

4.2. Architektura x86

Począwszy od końca lat 70. XX w. firma Intel rozwija rodzinę mikroprocesorów ogólnego przeznaczenia, głównie, choć nie tylko, stosowanych w komputerach klasy IBM PC. Rodzina ta jest produkowana nadal, w przeszłości przechodziła jednak kilka transformacji, począwszy od prostych procesorów 16-bitowych, a skończywszy na obecnie rozwijanych 64-bitowych układach powoli zaczynających przypominać rozwiązania typu System-on-a-Chip. Wszystkie jednostki należące do rodziny x86 charakteryzują się tzn. kompatybilnością wsteczną. Umożliwia ona uruchamianie nawet najstarszych programów na najnowszych procesorach, ponieważ każda kolejna generacja procesora zawiera w sobie wszystkie poprzednie (zgodność na poziomie ISA, czyli Instruction Set Architecture), zatem instrukcje maszynowe raz wprowadzone do procesora zostają tam, zaś wszelkie unowocześnienia są rozszerzeniem już istniejących rozwiązań. Na poziomie ISA z procesorami Intela zgodne są rodzinny procesorów innych producentów. Dla przykładu, firma AMD produkuje i rozwija własne układy, które

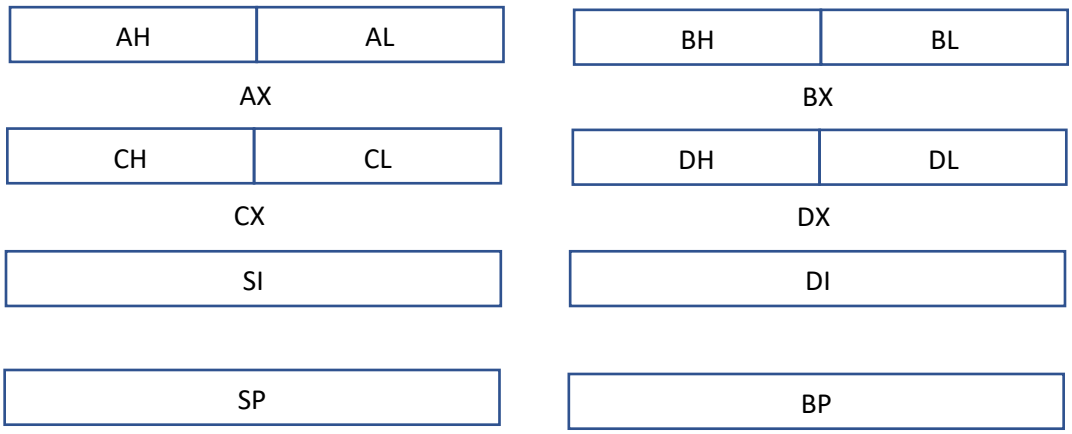
choć różnią się strukturą (mikroarchitekturą), implementują te same zestawy instrukcji. Podobnie, w przeszłości

W ogólności, architektura obejmuje trzy rodziny procesorów:

a. Procesory 16-bitowe (od Intel 8088/8086 do Intel 286)
b. Procesory 32-bitowe (od Intel 386 do Intel Pentium IV)
c. Procesory 64-bitowe, w tym wielordzeniowe (od Pentium IV Extreme do Core i)

Niniejsze laboratorium dotyczy w szczególności architektur 16-bitowych, reprezentowanych przez procesory Intel 8088/8086, 80188/80186, wreszcie Intel 80286. Organizacje procesorów są różne, jednak listy rozkazowe – podobne (z czasem rozwijane, zatem w kolejnych generacjach procesorów pojawiały się nowe instrukcje). Tematem laboratorium jest podstawowy zestaw instrukcji, którego szczegóły można znaleźć pod adresem: https://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm.

Wszystkie wymienione procesory mają cechy, wspólne, do których należy zestaw rejestrów roboczych. Są one 16-bitowe, uznaje się za rejestry ogólnego przeznaczenia (choć niektóre, jak wskaźnik stosu, mają ściśle określoną funkcję). Część rejestrów jest podzielona na dwie ośmiobitowe połówki (dzięki temu możliwe jest bardziej ekonomiczne przechowywanie krótszych liczb, dla których wystarczy rejestr ośmiobitowy; wówczas procesor dysponuje ośmioma lokacjami dla takich wartości).



Rejestry te to, odpowiednio:

a. AX – akumulator, czyli podstawowy rejestr do przechowywania argumentów i wyników obliczeń. Jest domyślnym argumentem w operacjach mnożenia i dzielenia. Składa się on z dwóch rejestrów ośmiobitowych: górnej (AH) oraz dolnej (AL).

b. BX – rejestr bazowy, przechowujący podstawę adresu w przypadku wykonywania arytmetyki na adresach.
c. CX – rejestr zliczający (ang. counting register), domyślnie wykorzystywany do przechowywania zmiennych iteracyjnych w pętlach.
d. DC – rejestr danych, który wykorzystany jest do przechowywania znaków, także domyślnie przechowujący wyniki mnożenia liczb 16-bitowych i argument dzielenia (wyższa połówka tzw. dzielnej).
e. SI – rejestr indeksowy źródła (ang. Source Index), służący do budowania adresu w czasie indeksowania.
f. DI – rejestr indeksowy przemieszczenia (ang. Displacement Index), również służący do budowania adresu w czasie indeksowania.
g. SP – wskaźnik stosu zawierający adres „szczytu" stosu.
h. BP – wskaźnik bazowy, czyli drugi rejestr adresowy obsługujący stos.

Oprócz tego w procesorze znajdują się cztery rejestry segmentowe, służące do przechowywania adresów początkowych segmentów przydzielonych działającemu programowi użytkownika. Jest to związane z określonym sposobem zarządzania pamięcią w systemie operacyjnym DOS (chodzi o segmentację). Rejestry te to: CS (przechowujący adres początkowy segmentu programu), DS. (przechowujący adres początkowy segmentu danych), SS (przechowujący adres początkowy segmentu stosu), wreszcie ES (przechowujący adres początkowy dodatkowego segmentu, tzw. extra segment).

4.3. Składnia NASM

Kod źródłowy NASMa jest plikiem tekstowym zawierającym symbole w kodzie ASCII. Program dla komputera 16-bitowego asemblera jest zwykle podzielony na sekcje, co jest związane ze wsparciem dla niego przez system operacyjny (patrz punkt 4.4). Ogólna postać kodu wygląda następująco:

```
; Pierwszy program w asemblerze NASM - w pliku cw1.asm
;Asemblacja: nasm -o cw1.com -f bin cw1.asm
;To jest komentarz
;To jest sekcja kodu
section .text
org 100h ;tu adres początku programu względem początku programu (selektor + deskryptor segmentu na niego ws

start:  ;To jest etykieta - na razie służy do zupełnie niczego
    mov ah, 9
    mov dx, label
    int 21h

    mov ah, 0
    int 16h

    mov ax, 4C00h
    int 21h


section .data ; Sekcja danych - właściwie "stałych" (bo można je podmieniać)

    label   db  "Pierwszy program w asemblerze.$"


section .bss
    ; Tu będą zmienne bez wartości
```

segment kodu

segment danych

segment zarezerwowanej przestrzeni w pamięci

UWAGA: Pomimo iż NASM nie wymaga jawnego deklarowania poszczególnych sekcji, zaleca się ich stosowanie, co może być przydatne podczas tworzeniu kodu 32- i 64-bitowego.

Netwide Assembler implementuje składnię Intela, zatem ogólna postać instrukcji maszynowej w kodzie źródłowym ma następującą postać:

kod argumenty

gdzie **kod** oznacza mnemonik operacji do wykonania, zaś argumenty oznaczają lokalizacje w pamięci, gdzie znajdują się operandy dla instrukcji. W przypadku, gdy argument jest tylko jeden, stanowi on zarówno wartość początkową, jak i końcową. Np. instrukcja inkrementacji zawartości rejestru AX wygląda następująco:

```
INC AX ; AX = AX + 1
```

W przypadku, gdy argumenty są dwa, składnia definiuje pierwszy argument jako docelowy, zaś drugi – jako źródłowy. Np. operacja przemieszczenia liczby 24 do rejestru AX wygląda następująco:

```
MOV AX, 24 ; Ogólnie: MOV cel, źródło, tutaj: AX = 24
```

Z kolei działanie arytmetyczne dodawania dla dwóch liczb przechowywanych w rejestrach AX oraz BX wygląda następująco:

```
ADD AX, BX ; AX = AX + BX
```

4.4. Rola systemu operacyjnego

Kompletna lista funkcji systemowych znajduje się pod adresem: http://spike.scu.edu.au/~barry/interrupts.html. Służą one do wykonywania typowych operacji wejścia-wyjścia związane z obsługą takich urządzeń, jak klawiatura, ekran terminala, dysk twardy, czy czasomierz systemowy.

Ponadto system operacyjny

5. Instrukcja instalacji i konfiguracji niezbędnych programów

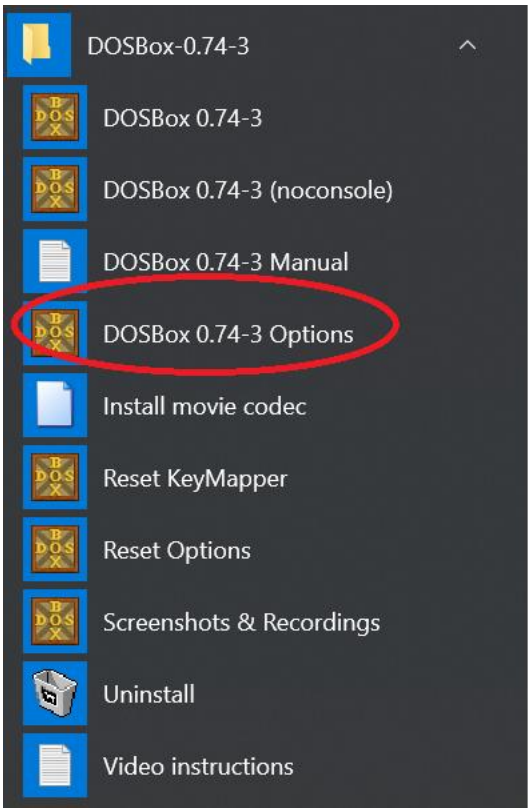
Laboratorium zaplanowane jest do przeprowadzenia pod kontrolą systemu operacyjnego Windows. Ponieważ jego przedmiotem są programy 16-bitowe (pliki wykonywalne o rozszerzeniu **com**, tj. zajmujące dokładnie jeden segment pamięci o wielkości 64kB), możliwość ich uruchomienia zależy od rodzaju systemu operacyjnego. W przypadku systemów 32-bitowych stworzone programy będą mogły być uruchomione bez żadnych modyfikacji. Niestety, w przypadku systemów 64-bitowych wycofano w nich wsparcie dla aplikacji 16-bitowych (tzw. podsystem Win16), zatem uruchamianie tego typu

aplikacji nie jest już możliwe. W tym przypadku konieczne jest wykorzystanie emulatora systemu operacyjnego DOS (takiego jak DOSBox). Niniejsza instrukcja dotyczy instalacji i konfiguracji niezbędnych programów pod kontrolą wiodących obecnie systemów operacyjnych 64-bitowych. W przypadku użycia systemu operacyjnego 32-bitowego można pominąć krok d.

W celu przygotowania środowiska do pracy należy:

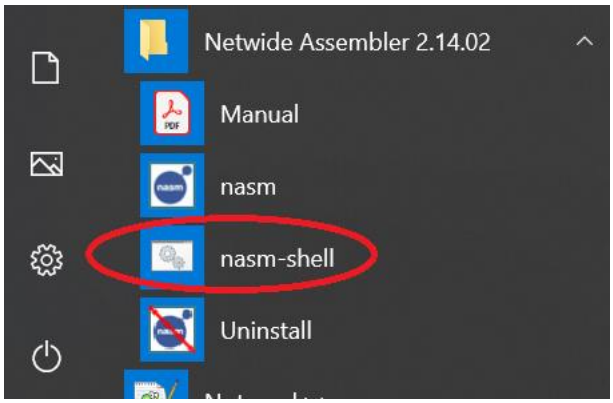
a. Stworzyć katalog roboczy, w którym będą umieszczane pliki źródłowe oraz binarny kod maszynowy, np. **C:\Projekty\ASK\Lab1**.
b. Pobrać Netwide Assembler ze strony domowej, tj. https://nasm.us (obecnie stabilna wersja to 2.14.02). W zależności od systemu operacyjnego można pobrać program w wersji 32- albo 64-bitowej. Następnie należy go zainstalować w systemie.
c. Pobrać program Notepad++ (np. ze strony https://notepad-plus-plus.org/downloads/) i zainstalować go, a potem uruchomić. Plik źródłowy asemblera tworzy się jako nowy, który od razu należy zapisać w katalogu roboczym z rozszerzeniem asm i o typie (Assembly Language Source File)
d. Pobrać program DOSBox (ze strony https://www.dosbox.com/, najnowsza wersja to 0.74-3) i zainstalować go. Aby możliwe było uruchamianie programów 16-bitowych wytworzonych podczas ćwiczenia, konieczna jest konfiguracja programu (polecenie DOSBox Options w grupie DOSBox). Pojawi się wówczas plik konfiguracyjny, w którym należy przejść do samego końca (grupa [autoexec]) i tam w nowej linii wpisać polecenie podłączające katalog roboczy jako dysk, np.:

mount c c:\Projekty\ASK\Lab1




```
[dos]
#            xms: Enable XMS support.
#            ems: Enable EMS support.
#            umb: Enable UMB support.
# keyboardlayout: Language code of the keyboard layout (or none).

xms=true
ems=true
umb=true
keyboardlayout=auto

[ipx]
# ipx: Enable ipx over UDP/IP emulation.

ipx=false

[autoexec]
# Lines in this section will be run at startup.
# You can put your MOUNT lines here.

mount c c:\Projekty\ASK\Lab1
```

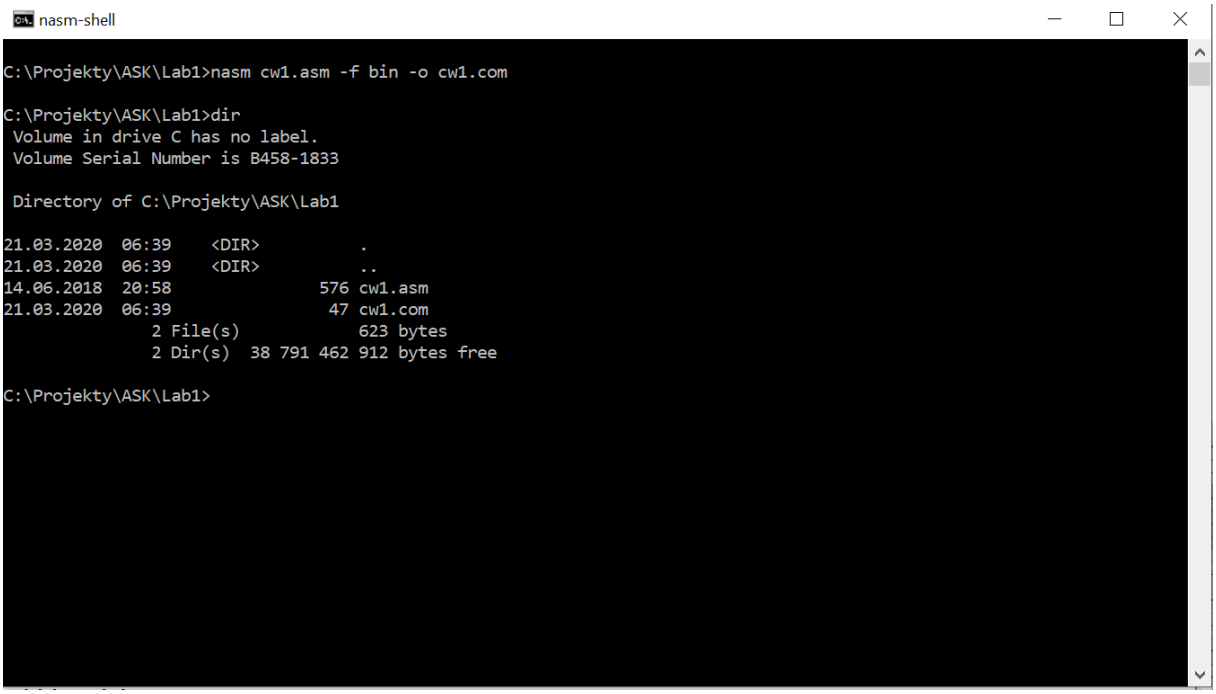
e. Uruchomić program nasm-shell (w grupie programów Netwide Assembler), w efekcie czego pojawi się konsola. Jest ona potrzebna, aby można było dokonywać asemblacji kodu źródłowego. Po jej uruchomieniu należy przejść do roboczego katalogu, np. **cd /d C:\Projekty\ASK\Lab1**.



f. Asemblacja programów odbywa się poprzez następujące polecenie:

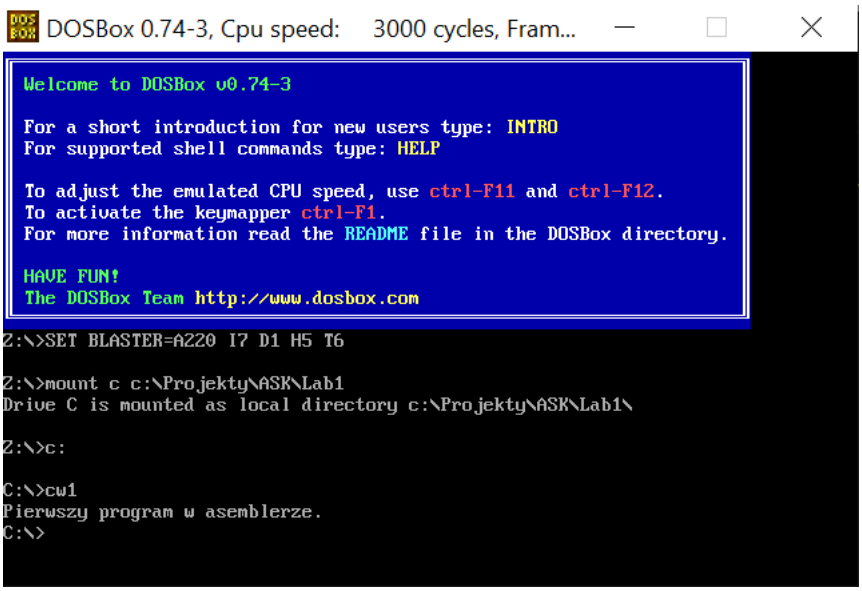
nasm plik_źródłowy.asm -f bin -o plik_wynikowy.com

Czyli np. nasm cw1.asm -f bin -o cw1.com. Jeśli proces tworzenia pliku binarnego się powiedzie, nie powinny pokazać się w konsoli żadne inne komunikaty. Wówczas można sprawdzić, czy plik o rozszerzeniu com istnieje, wpisując polecenie **dir**.



g. W przypadku systemu 32-bitowego plik można uruchomić poprzez wpisanie w konsoli jego nazwy. W przypadku systemu 64-bitowego konieczne jest uruchomienie DOSBoxa, który, poprawnie skonfigurowany, powinien wystartować jak poniżej.

Wówczas należy zmienić aktualny dysk na c (za pomocą „polecenia” **C:**) i uruchomić program poprzez wpisanie jego nazwy.


DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
Welcome to DOSBox v0.74-3
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c:\Projekty\ASK\Lab1
Drive C is mounted as local directory c:\Projekty\ASK\Lab1\
Z:\>c:
C:\>cw1
Pierwszy program w asemblerze.
C:\>