Ćwiczenie nr 1

Wprowadzenie do programowania w języku asemblera

1.1 Wstęp

Postępy elektroniki ostatniego półwiecza, a zwłaszcza skonstruowanie szybkich i tanich mikroprocesorów pozwoliły na wprowadzenie techniki komputerowej do wielu urządzeń technicznych. Tworzenie oprogramowania dla mikroprocesorów wbudowanych w urządzenia techniczne posiada pewną specyfikę, która odróżnia je od metod stosowanych powszechnie w informatyce. Często mikroprocesory takie współpracują z niewielką pamięcią operacyjną np. 4kB, a znajdujące się w niej programy intensywnie komunikują się z różnymi podzespołami obsługiwanego urządzenia. Nierzadko występują też ostre ograniczenia czasowe w odniesieniu do czasów obsługi rozmaitych zdarzeń.

Wymienione cechy powodują, że istotne elementy oprogramowania muszą być tworzone na poziomie pojedynczych instrukcji procesora. Ponieważ programowanie na tym poziomie wymaga sporego wysiłku, więc do kodowania mniej wymagających fragmentów oprogramowania używa się języków wysokiego poziomu, przede wszystkim języka C i C++. W rezultacie całe oprogramowanie urządzenia składa się z modułów w języku C (lub C++) i współdziałających z nimi modułów napisanych w języku instrukcji procesora.

Ze względu na istotne trudności w zakresie bezpośredniego kodowania instrukcji procesora za pomocą ciągów zerojedynkowych, powszechnie stosuje się języki asemblerowe, które pozwalają na kodowanie danych i instrukcji procesora w sposób wygodny dla programisty.

Celem podanego tu zestawu ćwiczeń laboratoryjnych jest przedstawienie techniki tworzenia programów w języku procesora (w asemblerze), a także interfejsu pozwalającego na integrację z kodem napisanym w języku wysokiego poziomu. Istotnym celem omawianego laboratorium jest także pokazanie mechanizmów wykonania programu przez procesor na poziomie rejestrowym oraz opis tych mechanizmów w kategoriach, jakimi posługują się programiści. Wszystkie podane opisy odnoszą się do procesorów rodziny Pentium (i poprzedników) pracujących w trybie rzeczywistym. Większość podanych przykładów może być wykonywana w trybie V86 (w okienku DOSowym w systemie Windows).

1.2 Kodowanie i uruchamianie programów laboratoryjnych

W systemie Windows dostępnych jest wiele różnych sposobów edycji tekstów programów i ich tłumaczenia. Jednakże mniej doświadczeni studenci próbują korzystać z tych sposobów dość chaotycznie, co w rezultacie bardzo utrudnia kodowanie i uruchamianie programów. Z tego względu wskazane jest posługiwanie opisana niżej technika, aczkolwiek bardziej zaawansowani studenci mogą używać innych narzędzi, np. Windows Commander.

Programy opracowane w ramach laboratorium „Oprogramowanie mikrokomputerów" wygodnie jest kodować i uruchamiać w oknie DOSowym. W tym celu należy wybrać `Start/Uruchom`, a następnie wpisać komendę `cmd` i nacisnąć `OK`. W niektórych komputerach konieczne jest wybranie zestawu znaków 852 – w tym celu, jako pierwsze polecenie w oknie DOSowym należy wpisać `chcp 852`. Ze względu na to, że w trakcie uruchamiania programu wielokrotnie wprowadza się te same polecenia, warto tez na początku sesji wcześniej napisać polecenie `doskey`, co pozwala później na łatwe powtórzenie wcześniej wykonywanych poleceń. Wszystkie wydane wcześniej polecenia można przeglądać posługując się klawiszami ↑ i ↓.

Użytkownicy „student” posiadają uprawnienia do zapisu plików wyłącznie w katalogu `d:\studenci`. Wskazane jest by każdy użytkownik utworzył podkatalog roboczy w tym katalogu, np. `d:\studenci\robot`. Przed zakończeniem zajęć pliki źródłowe należy skopiować na dyskietkę, zaś wcześniej utworzony katalog powinien zostać skasowany.

Program źródłowy można napisać korzystając z dowolnego edytora, który nie wprowadza znaków formatujących. Może to być więc „NOTATNIK” (ang. *NOTEPAD*) czy „EDIT”, ale „WORD” czy „WRITE” nie jest odpowiedni. Istotne jest także by edytor wyświetlał numer wiersza – własność tę ma m.in. edytor „EDIT”. Nazwa pliku zawierającego kod źródłowy powinna mieć rozszerzenie ASM. Ze względu na używane asemblery, nazwy plików nie powinny zawierać więcej niż 8 znaków, a także należy unikać stosowania liter specyficznych dla alfabetu polskiego. Praktyczne jest wywołanie edytora z nazwą pliku podaną w linii zlecenia, np. `edit cw7.asm`.

Po utworzeniu pliku źródłowego należy poddać go asemblacji i linkowaniu. W wyniku asemblacji uzyskuje się plik z rozszerzeniem .OBJ (o ile program nie zawierał błędów formalnych). Kod zawarty w pliku .OBJ (tzw. kod półskompilowany) zawiera już instrukcje programu zakodowane w języku maszyny, ale nie jest jeszcze całkowicie przygotowany do wykonania przez procesor. Ostateczne przygotowanie kodu, a także włączenie programów bibliotecznych czy innych programów, jeśli jest to konieczne, następuje w fazie zwanej linkowaniem lub konsolidacją. Wykonuje to program zwany linkerem (np. LINK).

Wygodnie jest utworzyć plik wsadowy z rozszerzeniem .BAT zawierający polecenia asemblacji i linkowania. Jeszcze lepszy sposób polega na użyciu programu narzędziowego MAKE, którego znajomość może być przydatna także w ramach innych przedmiotów. Oba te sposoby opisane są poniżej.

Dostępne są także rozmaite środowiska zintegrowane, które łączą w sobie edytor, asembler, linker i debbuger (np. Borland). Edycja i uruchamianie programów w takich systemach może być łatwiejsza, ale z drugiej strony powodują one pewne „zamazanie” realizowanych procesów, co z dydaktycznego punktu widzenia jest niewskazane. Z tego względu posługiwać się będziemy oddzielnym edytorem, oddzielnym asemblerem, itd.

Przykładowy program

Poniżej przedstawiono prosty program w asemblerze. Program ten należy wpisać do pliku z rozszerzeniem .ASM, np. `pierwszy.asm`.

```
dane SEGMENT      ;segment danych
tekst       db 'Nazywam sie ...', 13, 10
            db 'moj pierwszy program asemblerowy'
            db 13,10
koniec_txt  db ?
dane ENDS

rozkazy SEGMENT   'CODE' use16      ;segment zawierający rozkazy programu
                  ASSUME cs:rozkazy, ds:dane
      wystartuj:
                  mov ax, SEG dane
                  mov ds, ax
                  mov cx, koniec_txt-tekst
                  mov bx, OFFSET tekst    ;wpisanie do rejestru BX obszaru
                                          ;zawierającego wyswietlany tekst
            ptl:
                  mov dl, [bx]      ;wpisanie do rejestru DL kodu ASCII
                                    ;kolejnego wyświetlanego znaku
                  mov ah, 2
                  int 21H     ;wyświetlenie znaku za pomocą funkcji nr 2 DOS
                  inc bx            ;inkrementacja adresu kolejnego znaku
```

```
                    loop ptl    ;sterowanie pętlą

                    mov al, 0   ;kod powrotu programu (przekazywany przez
                                ;rejestr AL) stanowi syntetyczny opis programu
                                ;przekazywany do systemu operacyjnego
                                ;(zazwyczaj kod 0 oznacza, że program został
                                ;wykonany poprawnie)

                    mov ah, 4CH ;zakończenie programu – przekazanie sterowania
                                ;do systemu, za pomocą funkcji 4CH DOS
                    int 21H
rozkazy ENDS

nasz_stos   SEGMENT stack     ;segment stosu
            dw 128 dup (?)
nasz_stos   ENDS

END         wystartuj         ;wykonanie programu zacznie się od rozkazu
                              ;opatrzonego etykietą wystartuj
```

Po wpisaniu programu do pliku należy go poddać asemblacji, np.:

```
D:\MASM611\BIN\ml /Fl /Zm /Zi /c pierwszy.asm
```

W wyniku asemblacji, jeśli tłumaczony program nie zawierał błędów, zostaje utworzony plik z rozszerzeniem .OBJ. Z kolei plik ten należy poddać linkowaniu, np.:

```
D:\MASM611\BINR\link /CODEVIEW pierwszy.obj
```

W trakcie uruchamiania programu trzeba zazwyczaj wielokrotnie zmieniać jego tekst, a następnie poddawać go asemblacji i linkowaniu. Z tego względu warto posługiwać się opisanym niżej plikiem wsadowym (.BAT)

1.3 Tłumaczenie programu za pomocą pliku wsadowego .BAT oraz programu MAKE

Poniżej podano treść podano treść pliku wsadowego `asembluj.bat` przy założeniu, że programy ML i LINK znajdują się odpowiednio w katalogach `D:\ MASM611\BIN` oraz `C:\ MASM611\BINR`.

```
D:\MASM611\BIN\ml /Fl /Zm /Zi /c %1.asm
if errorlevel 1 goto koniec
D:\MASM611\BINR\link /CODEVIEW %1.obj
:koniec
```

Jeśli kod źródłowy programu znajduje się, np. w pliku `pierwszy.asm`, to wywołanie pliku wsadowego powinno mieć postać `asembluj pierwszy`. Zauważmy, że nie należy podawać rozszerzenia nazwy pliku .ASM. W pliku wsadowym warto zwrócić uwagę na polecenie „`if errorlevel 1 goto koniec`”. Polecenie to testuje kod powrotu zwrócony przez ostatnio wykonany program. Jeśli kod powrotu jest równy lub większy od podanej liczby (tu: 1), to następuje wykonanie podanej instrukcji (tu: instrukcji skoku do etykiety koniec). Zwyczajowo programy zwracają kod powrotu równy zero, jeśli zostały wykonane poprawnie, a wartość niezerową gdy występowały błędy. Zatem jeśli asembler MASM wykrył błędy w programie źródłowym, to zwróci kod powrotu większy od zera. W tym

przypadku warunek testowany w wierszu „`if errorlevel`” będzie spełniony, wskutek czego linkowanie wykonywane przez program LINK zostanie pominięte. Omawiany tu mechanizm powoduje, że linkowanie wykonywane jest tylko wówczas, jeśli asemblacja została wykonana poprawnie.
Poszczególne opcje dla programu asemblującego:

- /Zi - dołączenie informacji do debuggera
- /Fl - utworzenie pliku wydruku (*.lst)
- /Fm - utworzenie pliku mapy (*.map)
- /c - przeprowadzenie tylko kompilacji (*.obj)

Opisane powyżej tłumaczenie programu w asemblerze nie przedstawia żadnych trudności. Jednak w przypadku bardziej złożonych programów, zapisanych w kilku czy nawet kilkuset plikach źródłowych, tłumaczenie staje się znacznie bardziej skomplikowane. W takich przypadkach powszechnie używany jest program narzędziowy MAKE, który znakomicie ułatwia przeprowadzenie nawet skomplikowanej translacji. Zalety tego programu są mało widoczne w przypadku tłumaczenia prostych programów asemblerowych, ale mimo to warto zapoznać się z jego działaniem, tak by w przyszłości można było go wykorzystać przy bardziej skomplikowanych zadaniach. Program narzędziowy MAKE dostępny jest w wielu systemach, m.in. w Windows, Unix (Linux) i wielu innych. Program MAKE stanowi narzędzie wspomagające proces translacji programu. MAKE buduje program docelowy (który zazwyczaj ma format .EXE) na podstawie szczegółowego opisu postępowania.

Przypuśćmy, że kod źródłowy opracowanego programu umieszczono w pliku `pierwszy.asm`.
W celu uzyskania wersji EXE tego programu (np. `pierwszy.exe`) należy przeprowadzić asemblację (kompilację) pliku za pomocą asemblera MASM. Następnie uzyskany plik `pierwszy.obj` należy poddać konsolidacji za pomocą programu LINK – w rezultacie uzyskamy plik EXE. Wychodząc od końca tego postępowania można powiedzieć, że plik EXE stanowi wynik działań na pliku `pierwszy.obj`, co można zapisać w formalizmie stosowanym przez MAKE:

```
pierwszy.exe : pierwszy.obj
      D:\MASM611\BINR\link pierwszy.obj
```

Pierwszy z tych wierszy opisuje „składowe”, z których tworzy się plik `pierwszy.exe` . Drugi wiersz, obowiązkowo zaczynający się od znaku tabulacji, zawiera polecenie, które należy wykonać w celu uzyskania pliku EXE. W analogiczny sposób można opisać reguły tworzenia pliku OBJ

```
pierwszy.obj: pierwszy.asm
      D:\MASM611\BIN\ml /c pierwszy.asm
```

Powyższy sformalizowany opis postępowania umieszcza się w zwykłym pliku tekstowym, zwanym plikiem reguł. Plik taki, z rozszerzeniem .MAK, zawiera pozycje opisujące, jakie narzędzia należy wywołać, aby wygenerować lub uaktualnić plik docelowy. Dodatkowo, wiersze zaczynające się od znaku # zawierają komentarze. Przykładowo, dla rozpatrywanego zadania można utworzyć plik `opis.mak`, zawierający poniższe wiersze:

```
# Opis asemblacji i linkowania programu źródłowego pierwszy.asm
pierwszy.exe : pierwszy.obj
      D:\MASM611\BINR\link pierwszy.obj
pierwszy.obj: pierwszy.asm
      D:\MASM611\BIN\ml /c pierwszy.asm
```

Jeśli teraz wywołamy program MAKE z parametrem `make -f opis.mak` program MAKE wykonana wskazane polecenia, w wyniku czego uzyskamy plik pierwszy.exe. Zauważmy, że istnienie pliku docelowego (np. programu wynikowego) zależy od istnienia pewnych innych plików (np. plików z postacią półskompilowaną i bibliotek); z kolei te pliki mogą być wygenerowane pod warunkiem istnienia odpowiadających im plików źródłowych. Stąd lista poleceń w pliku reguł jest de facto listą zależności, warunkujących możliwość wykonania danego polecenia. W ten sposób powstaje drzewo zależności, którego korzeniem jest plik docelowy, gałęziami zbiory generowane na różnych etapach kompilacji, liśćmi zaś pliki źródłowe. To drzewo jest zapisane w określony sposób w pliku reguł, począwszy od korzenia, i jest analizowane przez program MAKE.

1.4 Uruchamianie programów z wykorzystaniem CodeView

Nieodłącznym elementem praktyki programowania jest występowanie różnych typów błędów. Nawet doświadczonym programistom zdarza się popełniać omyłki. Z tego powodu we współczesnej informatyce rozwinięto szereg zasad i reguł postępowania w zakresie tworzenia oprogramowania, tak by ograniczyć błędy do minimum. Zidentyfikowanie błędu może być znacznie łatwiejsze jeśli dysponujemy programem narzędziowym pozwalającym na wykonywanie pod nadzorem poszczególnych fragmentów analizowanego programu, czyli debuggerem. Jednym takich programów jest m.in. program CodeView dostarczany wraz z MASM-em.

Stosunkowo najprostsze do znalezienia są błędy formalne polegające na niezgodności kodu programu ze składnią języka. Kompilatory sygnalizują takie błędy podając numer wiersza w programie, co pozwala na szybkie ich odnalezienie i usunięcie. Trudniejsze do wykrycia są błędy wykonania programu (ang. *run-time errors*) jak też błędy logiczne. Błędy wykonania programu ujawniają się dopiero podczas wykonywania jego wykonywania – próba wykonania niedozwolonej operacji jest wykrywana przez sprzęt lub oprogramowanie, a ślad za tym wykonywanie programu zostaje zawieszone, czemu towarzyszy odpowiedni komunikat. Błędy logiczne nie są sygnalizowane przez system operacyjny, ale ich objawami jest niepoprawne działanie programu, np. program podaje błędne wartości, wykres na ekranie ma niewłaściwy kształt, dźwięki odtwarzane są nieprawidłowo, itp.

CodeView może stanowić istotną pomoc w odnalezieniu przyczyn występowania błędów wykonania programu jak też błędów logicznych. Warunkiem uruchomienia debuggera jest posiadanie wersji wykonywalnej programu w formacie pliku .EXE. Oznacza to, że wcześniej trzeba usunąć ewentualne błędy składniowe, tak można było uzyskać plik .EXE.

Podane dalej zasady używania CodeView obejmują tylko najbardziej podstawowe operacje. Pełny opis debuggera nierzadko ma postać oddzielnej książki. W celu uruchomienia debuggera należy wywołać go w poniższy sposób:

```
D:\MASM611\BINR\cv pierwszy.exe
```

W ślad za tym na ekranie pojawi się okno debuggera zawierające instrukcje programu.

Debugger pozwala na śledzenie programów, które nie zostały specjalnie przygotowane do śledzenia – w takim przypadku możliwe jest śledzenie jedynie na poziomie instrukcji (rozkazów) procesora. Taka właśnie technika opisana jest w 1.4.1. Debuggowanie jest znacznie wygodniejsze w przypadku, gdy w kodzie programu umieszczono dodatkowe informacje wspomagające pracę debuggera – szczegóły opisane są. 1.4.2.

1.4.1 Śledzenie wykonywania instrukcji programu

CodeView oferuje kilka różnych sposobów wykonywania programów. W najprostszym przypadku posługujemy się klawiszem `F8`, którego naciśnięcie powoduje wykonanie pojedynczej zaznaczonej ("podświetlanej") instrukcji. Skutkiem jej wykonania może być zmiana zawartości rejestru, zmiana zawartości komórki pamięci lub inna operacja. Identyczny skutek ma naciśnięcie klawisza `F10`, o ile wykonywany rozkaz nie wywołuje podprogramu (`CALL`). W takim przypadku naciśnięcie `F10` powoduje wykonanie całego podprogramu. Jeśli w trakcie śledzenia programu zachodzi konieczność uruchomienia go od nowa, to program można przełączyć (ang. *reset*) do stanu początkowego poprzez naciśnięcie kombinacji klawiszy `Ctrl F2`. Przesuwanie "podświetlanej" instrukcji za pomocą klawiszy strzałek nie powoduje wykonywania rozkazów. W takim przypadku naciśnięcie klawisza `F7` powoduje wykonanie kolejnego rozkazu programu, a nie rozkazu zaznaczonego ("podświetlanego"). Poprzez naciśnięcie klawisza `F4` można jednak spowodować wykonanie kolejnych instrukcji programu aż do instrukcji aktualnie zaznaczonej (wyłącznie). Inna możliwość związana jest z kombinacją klawiszy `Alt F9` – naciśnięcie tej kombinacji powoduje pojawienie się na ekranie niewielkiego okna dialogowego, do którego należy wpisać adres instrukcji (np. `2E`), do której ma zostać wykonany program (wyłącznie). Zatem debugger rozpocznie wykonywanie kolejnych instrukcji programu, i zatrzyma się po dojściu do instrukcji o podanym adresie. Jeszcze inna opcja powoduje automatyczne i bardzo spowolnione wykonywanie kolejnych instrukcji programu. Zwykle kolejne instrukcje wykonywane co 0.3 s, ale wartość ta może być zmieniona.

W trudniejszych przypadkach może być celowa rejestracja kolejnych wykonywanych instrukcji – służy do tego opcja `View/Execution history`. Po wybraniu tej opcji na ekranie pojawi się okno o tej samej nazwie. Wówczas, w polu tego okna należy kliknąć prawym klawiszem myszy (lub nacisnąć `Alt F10`), co spowoduje rozwinięcie menu – wybór opcji `Full history Yes` zainicjuje rozpoczęcie rejestracji wykonywanych instrukcji. Każda wykonana instrukcja (wskutek naciśnięcia `F7` lub `F8`) zostanie zapisana w oknie historii wykonania.

W omawianym przypadku pojawia się dodatkowa możliwość wykonywania programu wstecz, czyli powrotu do sytuacji przed wykonaniem instrukcji – działania takie realizuje się za pomocą kombinacji klawiszy `Alt F4`.

1.4.2 Śledzenie programów zawierających informację symboliczną

Opisane wcześniej polecenie asemblacji programu można rozszerzyć o dodatkową opcję:

```
D:\MASM611\BIN\ml /Zi pierwszy.asm
```

Opcja `/Zi` powoduje włączenie do pliku `.OBJ` informacji symbolicznych wspomagających debuggowanie. Analogiczne znaczenie ma opcja `/CODEVIEW` w przypadku linkowania:

```
D:\MASM611\BINR\link /CODEVIEW pierwszy.obj
```

Jeśli asemblacja i linkowanie zostaną przeprowadzone z podanymi opcjami, i dostępny jest jest plik zawierający kod źródłowy programu, to po uruchomieniu debuggera na ekranie pojawi się pełny kod źródłowy programu. Możliwości debuggowania opisane w p. 1.4.1 są nadal dostępne. Zazwyczaj celowe jest otwarcie okna podającego zawartości rejestrów procesora (opcja `Windows/Registers` Ogólnie: proces debuggowania w opisywanym przypadku jest znacznie ułatwiony. Dodajmy, że opisana technika dotyczy także programów

napisanych w językach wysokiego poziomu, m.in. w języku C – należy wówczas podać opcję kompilacji także `/CODEVIEW`.

1.4.3 Dostrajanie informacji wyświetlanych w oknach debuggera

Jak już wspomnieliśmy, po uruchomieniu debuggera (jeśli plik `.EXE` nie zawiera informacji symbolicznej) ekran zawiera cztery podstawowe okna: instrukcji (rozkazów), rejestrów procesora, obszaru danych i obszaru stosu. Rodzaj informacji i sposób jej wyświetlania w poszczególnych oknach można zmodyfikować poprzez wybranie okna (kliknięcie lewym klawiszem), a następnie otwarcie menu specyficznego dla danego okna (kliknięcie prawym klawiszem myszki). Przykładowo, menu dla okna rejestrów procesora pozwala wybrać wyświetlanie rejestrów 16- albo 32-bitowych. Menu dla segmentu danych m.in. pozwala wybrać najbardziej odpowiedni sposób prezentacji informacji: w postaci bajtów, słów, słów podwójnych czy też liczby w formacie zmiennoprzecinkowym.

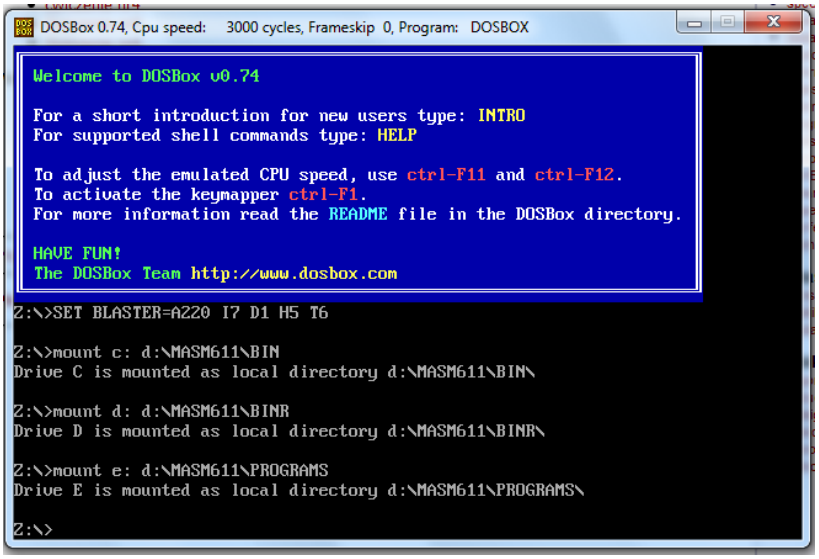
Opcja `Windows` pozwala również na wyświetlanie różnych rodzajów okien. Można też łatwo zmienić rozmiary okien wyświetlane aktualnie na ekranie dostosowując do aktualnej sytuacji. Więcej informacji o programie CodeView można znaleźć pod następującymi linkami:

- http://kipirvine.com/asm/cv/codeView_01.htm
- http://web.sau.edu/lilliskevinm/csci240/masmdocs/envtools/16LMAETC08.pdf

1.5 Praca z programem DOSBOX

W przypadku systemów Windows Vista i Windows 7 należy do asemblacji, konsolidacji i debagowania wykorzystać program DOSBOX umożliwiający uruchamianie programów 16-bitowych. Załóżmy, że katalog z programem MASM znajduje się w następujący miejscu na dysku `D:\MASM611`.

Można zatem zamontować w DOSBOX-ie poleceniem `mount` odpowiednie katalogi. Taką sytuację przedstawia poniższy Rys.1. Zostały zamontowane trzy katalogi do odpowiadających im wirtualnym napędom `C: D: i E:`.



Rys. 1. Okno DOSBOX-a z zamontowanymi katalogami

Następnie należy zmienić dysk `Z:` na `E:` i uruchomić program wsadowy `asembluj.bat`. Przykładowo jeżeli w katalogu `D:\MASM611\PROGRAMS` znajduje się plik źródłowy `prog1.asm` należy w wierszu poleceń DOSBOX-a uruchomić poniższe polenienie

```
E:\asembluj prog1
```

Jak widać nie podajemy w poleceniu rozszerzenia pliku źródłowego. Wynika to z przyjętej postaci pliku wsadowego `asembluj.bat`, w którym .ASM jest domyślnym rozszerzeniem. Jeżeli program zawierał błędy zostaną one wyświetlone na ekranie i nastąpi zakończenie kompilacji. W przeciwny wypadku, program zostanie skonsolidowany do pliku wykonywalnego. W miejscu tym MASM będzie prosił użytkownika o podanie nazw powstającym plikom, tzn. wykonywalnemu, listingu, mapy itp. Domyślne nazwy – identyczne jak nazwa pliku źródłowego – będą nadawane poprzez naciśnięcie klawisza ENTER.