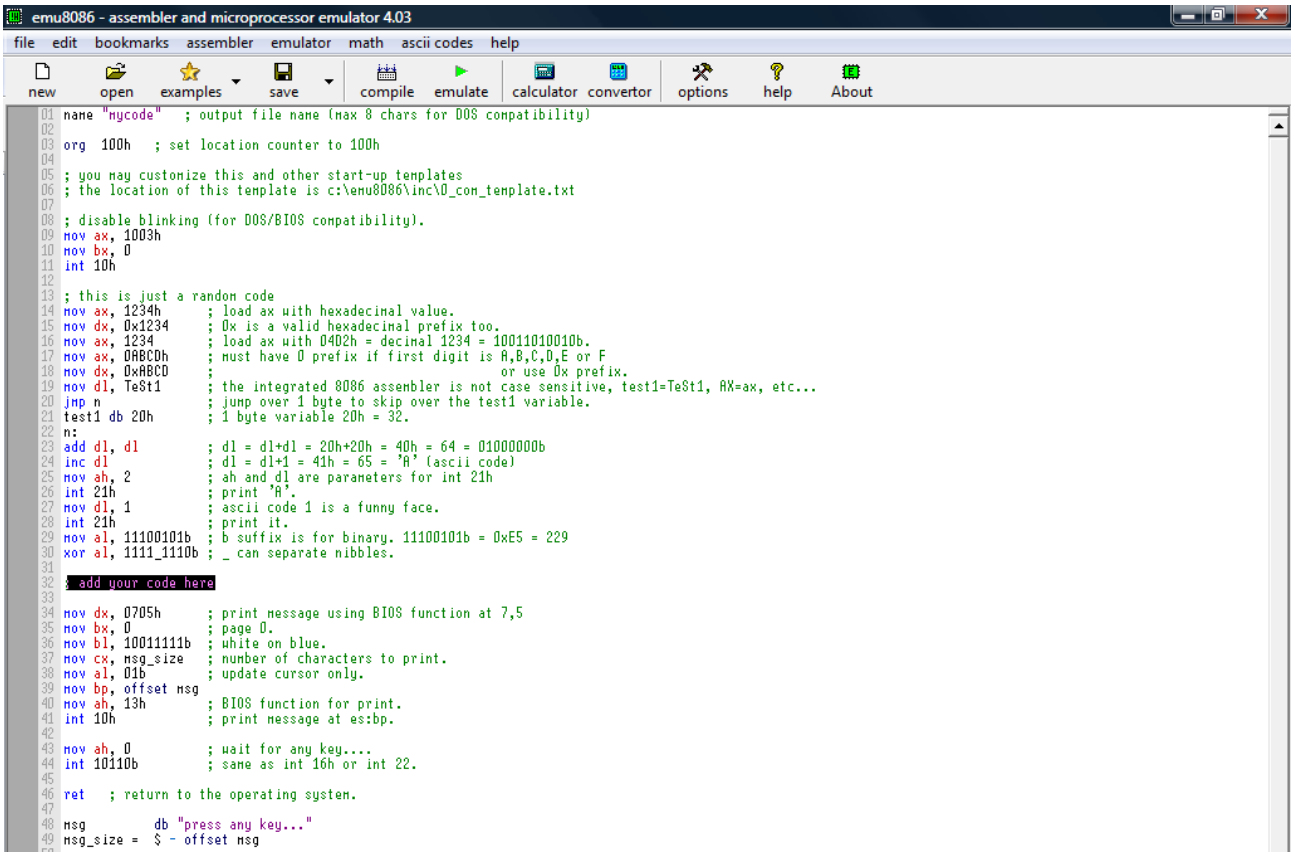
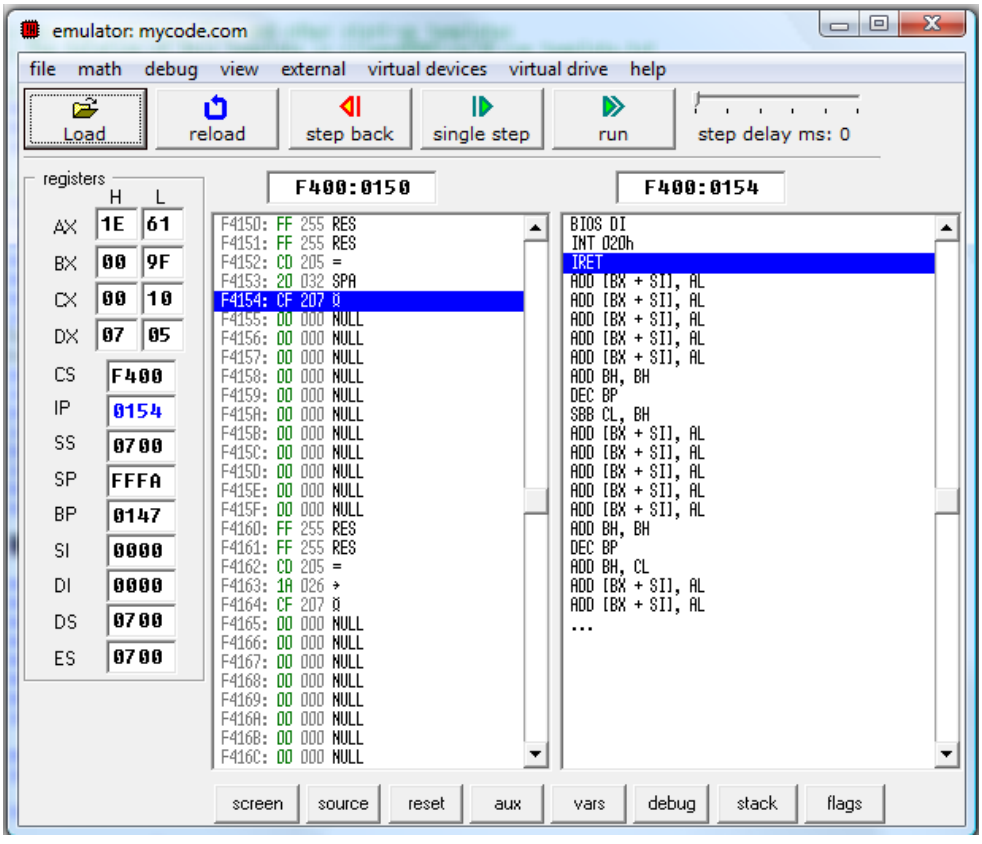
Zapoznanie z symulatorem procesora 8086



Okno główne symulatora procesora 8086



Okno symulatora procesora 8086 w trybie emulacji

Struktura programu typu program.com

```
; program według modelu tiny
name "mycode"   ; nazwa pliku wyjściowego (maksymalnie 8 znaków)

org  100h          ; początek programu od adresu IP = 100h


; kod programu


ret   ; koniec programu i powrót do systemu operacyjnego.

msg        db "press any key..."    ; deklaracje zmiennych
```

Struktura programy typu program.exe

```
; program wykonywalny wielosegmentowy.

data segment
; tutaj umieszczamy deklaracje danych
ends

stack segment
   dw   128  dup(0)        ; określenie wielkości stosu
ends

code segment
start:              ; początek programu
; ustawienie wartości rejestrów segmentowych:
   mov ax, data
   mov ds, ax
   mov es, ax

; kod programu


   mov ax, 4c00h ; zakończenie programu i powrót do system operacyjnego.
   int 21h
ends

end start ; ustawienie punktu startu programu i koniec instrukcji w assemblerze.
```

Program w assemblerze składa się z dyrektyw kompilatora, instrukcji, deklaracji zmiennych i stałych.

Dyrektywa kompilatora nie jest tłumaczona na kod wynikowy – określa tylko jak ma przebiegać kompilacja i jak zarządzać kodem wynikowym. Dyrektywa ORG 100h

w programach typu .com określa, że instrukcje programu powinny zaczynać się od adresu 100h w segmencie kodu. Pierwsze 256 B (100h) zarezerwowane są dla potrzeb systemu operacyjnego w celu komunikacji z programem (np. przekazanie parametrów uruchomienia programu). Dyrektywy mogą sterować warunkową kompilacją programu, definiować makroinstrukcje i in.

Zmienne określają miejsce przechowywania danych i ich typ. Dane te mogą znajdować się w rejestrach procesora, ale ze względu na ograniczoną liczbę rejestrów, miejscem ich przechowywania jest głównie pamięć operacyjne. Procesor korzysta z pamięci operacyjnej podając adres komórki z którą chce się skontaktować. Adresy są długimi liczbami binarnymi (np. 20 bitów) i ze względów praktycznych podczas deklaracji zmiennych adresom komórek przyporządkowuje się nazwy symboliczne. Określa się również typ zmiennych, który decyduje o tym jak dużo miejsca zajmuje zmienna w pamięci oraz jakie operacje można na niej wykonywać. Będziemy korzystać ze zmiennych o długości 1 bajtu i 1 słowa (2 bajty).

nazwa **DB** *wartość* ; deklaracja zmiennej o długości 1B
nazwa **DW** *wartość* ; deklaracja zmiennej o długości 2B

nazwa jest dowolnym ciągiem liter i cyfr zaczynającym się od litery, duże i małe litery nie są rozróżniane. Można zadeklarować zmienną bez podania nazwy lecz tracimy możliwość bezpośredniego odwołania się do zmiennej za pośrednictwem nazwy.
Wartość początkowa zmiennej może być określona w systemie binarnym, dziesiętnym, szesnastkowym, w postaci kodu ASCII. Można też nie podawać wartości początkowej (zmienne nie zainicjalizowana) pisząc znak **?**.

Przykłady:

```
Zm1   DB   8                ; zapis dziesiętny
Zm2   DW   1234h, 0A5A5h    ; zapis szesnastkowy
Zm3   DB   01011100b        ; zapis binarny
Zm4   DB   'A'              ; kod ASCII
Zm5   DB   'Witaj', 0       ; ciąg znaków ASCII zakończony 0
Zm6   DW   ?                ; zmienna bez podanej wartości początkowej
```

Zmienna typu WORD zajmuje dwa bajty. Stosowana jest zasada, że młodszy bajt zapisywany jest pod niższym adresem w pamięci. Poniższe dwie deklaracje są równoważne:

```
Zm16b    DW   1234h
Zm8b     DB   34h, 12h
```

Wartość szesnastkowa rozpoczynająca się znakiem A..F musi być poprzedzona 0.

Można również oprócz pojedynczych zmiennych deklarować tablice. Tablicą nazywamy zmienną złożoną z określonej liczby elementów jednakowego typu. Dostęp do elementów tablicy uzyskujemy korzystając z indeksu elementu przechowywanego w rejestrze indeksowym. Jeśli wartości elementów tablicy się powtarzają możemy skorzystać z operatora DUP.

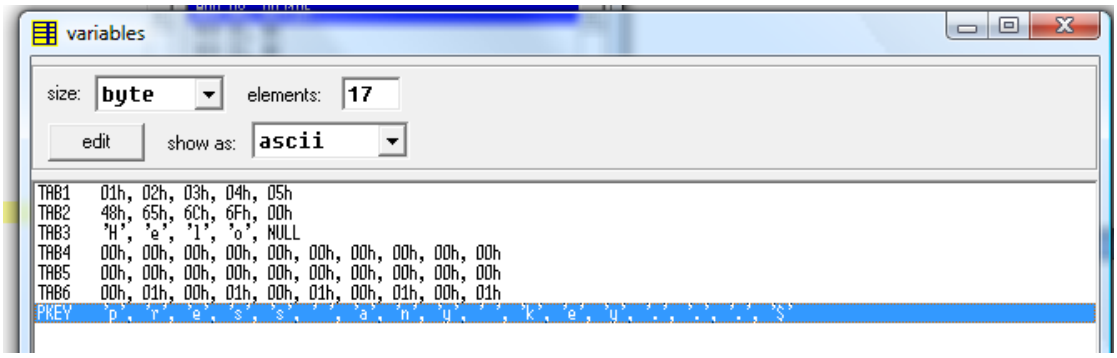
Przykłady:

```
Tab1  DB   1, 2, 3, 4, 5    ; tablica 5-elementowa zawiera liczby 1-5
```

```
Tab2  DB   'Hello' , 0            ; tablica zawiera napis zakończony 0
Tab3  DB   48h, 65h, 6Ch, 6Fh, 00h   ; zapis równoważny Tab2
Tab4  DB   10 DUP (?)             ; tablica 10-elementowa nie zainicjalizowana
Tab5  DB   10 DUP (0)             ; tablica 10-elementowa o wartościach 0
Tab6  DB   5 DUP (0,1)  ; tablica 10-elementowa o wartościach 0,1,0,1 …
```

Uwaga: DW nie może być używane przy deklaracji napisów.

Możemy dokonywać podglądu zmiennych podczas pracy symulatora otwierając **okienko variables** (przycisk vars).



Możliwy jest wybór typu zmiennej (byte, word, dword, qword, tword), ilości elementów zmiennej, sposobu interpretacji wyświetlanych danych (hex, bin, octal, signed, unsigned, ASCII). Dostępna jest również edycja zawartości pamięci danych podczas pracy symulatora.

Zadanie:
Zadeklarować w programie zmienne jak w powyższych przykładach, skompilować program i przejść do podglądu zawartości zmiennych. Zlokalizować położenie segmentu danych w stosunku do segmentu kodu. Sprawdzić w jakiej kolejności zapisywane są bajty słowa 16-bitowego w pamięci operacyjnej.

Stałe

Stałe deklarujemy za pomocą dyrektywy **EQU**.

```
nazwa_stałej  EQU wyrażenie
```

Istnieją one tylko do momentu kompilacji – w tym momencie każda stała zamieniana jest na odpowiadającą jej wartość wyrażenia.

Przykład:

```
k     EQU  5      ; definicja stałej
mov   AX, k       ; przesłanie do rejestru AX wartości 5
```

Instrukcje przesyłania danych

Podstawową instrukcją przesyłania danych jest instrukcja **mov dst, src**
Pozwala ona przesłać do miejsca przeznaczenia (rejestru, komórki pamięci) zawartość ze źródła (rejestru, komórki pamięci lub wartość liczbowa). Istnieją przy tym pewne ograniczenia, np. nie można wpisać bezpośrednio wartości liczbowej do rejestru segmentowego, dlatego używaliśmy powyżej dwóch instrukcji:

mov AX, data
mov DS, AX

Nie można wpisać wartości do licznika rozkazów (rejestru CS:IP) – służą do tego rozkazy skoku lub wywołania podprogramu.

Rozkazy mikroprocesora 8086 są wielobajtowe. Liczba bajtów każdego rozkazu zależy od jego rodzaju i może wynosić od jednego do sześciu.

7	6	5	4	3	2	1	0
kod operacji						D	W
MOD		REG			R/M		

Pierwszy bajt zawiera sześciobitowy kod operacji oraz dwa bity (kierunku i szerokości). Bit *D* określa kierunek transmisji (*0* - wynik operacji jest przesyłany z rejestru do pamięci, *1* - z pamięci do rejestru). W zależności od wartości tego bitu w rozkazie rozróżniane są operandy źródłowe i operandy przeznaczenia. Bit *W* określa szerokość operandu danego rozkazu (0 - operacje bajtowe, 1 - operacje na słowie 16-bitowym).

Jeżeli rozkaz jest wielobajtowy, to drugi bajt rozkazu określa sposób adresowania argumentów. Zawiera on trzy grupy bitów

- dwubitowa grupa *MOD* określa tryb adresowania
- trzybitowa grupa *REG* określa numer rejestru, w którym znajduje się operand
- trzybitowa grupa *R/M* określa sposób wyznaczenia miejsca operandu

Jeżeli operandy znajdują się w rejestrach mikroprocesora (*MOD* = 11), to pola *REG* i *R/M* stanowią ich numery (odpowiednio pierwszego i drugiego operandu)

REG **R/M**	**W = 0**	**W = 1**
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Jeżeli jeden z operandów znajduje się w pamięci, to pola *MOD* i *R/M* określają jego adres

R/M	**MOD = 00**	**MOD = 01**	**MOD = 10**
000	BX+SI	BX+SI+p8	BX+SI+p16
001	BX+DI	BX+DI+p8	BX+DI+p16
010	BP+SI	BP+SI+p8	BP+SI+p16
011	BP+DI	BP+DI+p8	BP+DI+p16
100	SI	SI+p8	SI+p16
101	DI	DI+p8	DI+p16
110	p16	BP+p8	BP+p16
111	BX	BX+p8	BX+p16

Znaczenie bitów *R/M* zależy od wartości bitów w polu MOD. Jeżeli *MOD* != 11, to grupa *R/M* określa rejestry adresujące. Jeżeli natomiast *MOD* == 11, to *R/M* określa rejestr (podobnie jak *REG*) drugiego operandu. Jeżeli rozkaz tego wymaga, to po drugim bajcie rozkazu może występować jeden lub dwa bajty przemieszczenia. Jeden bajt przemieszczenia występuje w sytuacji, gdy *MOD* == 01, natomiast przemieszczenie dwubajtowe występuje, gdy *MOD* == 10.

Przykłady wykorzystania rozkazu ***mov dst, src***

```
mov   AX, 0B00h
mov   DS, AX
mov   CL, 'A'
mov   CH, 01101001b
mov   BX, 15Eh
mov   [BX], CX
```

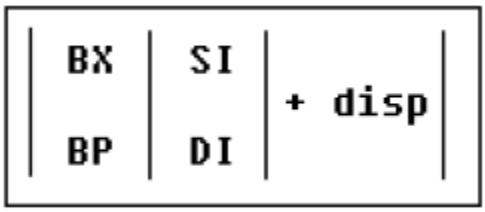
Zadanie:

Sprawdzić w symulatorze kodowanie i wykonanie powyższych rozkazów.

Tryby adresowania

Trybem adresowania nazywamy sposób wyznaczania adresu operandu, którego to mianem określamy argumenty i wyniki operacji. Adres operandu obliczany jest zgodnie z równaniem

EA = BR + IR + p



gdzie:

- ***EA*** - adres efektywny
- ***BR*** - rejestr bazowy
- ***IR*** - rejestr indeksowy
- ***p*** - przemieszczenie

Adres efektywny *EA* jest adresem logicznym "widzianym" przez program. Na podstawie *EA* układy segmentacji obliczają adres rzeczywisty w pamięci operacyjnej. Rejestrem bazowym może być rejestr *BP* lub *BX*, a rejestrem indeksowym może być rejestr *SI* lub *DI*. Przemieszczenie jest zawarte w rozkazie i może mieć długość ośmiu lub szesnastu bitów.

Mikroprocesor 8086 realizuje następujące tryby adresowania:

- natychmiastowe
- bezpośrednie
- rejestrowe
- pośrednie
- bazowe
- indeksowe
- indeksowo-bazowe

Adresowanie natychmiastowe

W adresowaniu natychmiastowym argument pobierany jest bezpośrednio z rozkazu. Np. **mov AX, 100** – w rejestrze *AX* zostanie zapisana liczba 100.

Adresowanie bezpośrednie

W adresowaniu bezpośrednim adres operandu znajduje się bezpośrednio w rozkazie. Np. **mov AX, [100]** – w rejestrze *AX* zostanie zapisana zawartość komórki pamięci (segment danych) o adresie 100.

Standardowy adres operandu jest przesunięciem w segmencie danych (*DS*), można to nadpisać poprzez wskazanie innego segmentu. Np. mov AX, CS:[40] – w rejestrze *AX* zostanie zapisana zawartość z komórki pamięci (segment programu(kodu)) o offsecie 40.

Adresowanie rejestrowe

W adresowaniu rejestrowym operandy znajdują się w rejestrach wewnętrznych mikroprocesora.
Np. **mov AX, BX** – w rejestrze *AX* zostanie zapisana zawartość rejestru *BX*.

Adresowanie pośrednie

W trybie adresowania pośredniego w rozkazie znajduje się nie adres argumentu lecz adres miejsca, gdzie znajduje się argument operacji.
Np. **mov AX, [BX]** – w rejestrze AX zostanie zapisana zawartość komórki pamięci o adresie, który znajduje się w rejestrze *BX*.

Wszystkie rejestry wskazują offset w segmencie danych (*DS*), poza rejestrem *BP*, który jest przesunięciem w segmencie stosu (*SS*). Można to zmienić określając segment w rozkazie.

Np. mov AX, SS:[BX] – w rejestrze *AX* zostanie zapisana zawartość komórki pamięci z segmentu stosu o adresie, który znajduje się w rejestrze *BX*.

Z reguły kompilator automatycznie określa długość argumentu w oparciu o rejestry, które biorą udział w operacji. Czasami zachodzi jednak potrzeba użycia prefiksu **byte ptr** lub **word ptr** w celu określenia typu danej np.

```
mov  word ptr [BX], 25      ; zapisanie w pamięci dwóch bajtów 0025
mov  byte ptr [BX], 25      ; zapisanie w pamięci jednego bajta 25
```

Można używać skróconego zapisu **b.** lub **w.**

Pobieranie adresu zmiennej

Istnieje możliwość pobrania adresu zmiennej za pomocą instrukcji **LEA** (Load Effective Address) lub operatora **OFFSET**. Jest to wykorzystywane przy adresowaniu pośrednim oraz przy przekazywaniu parametrów do procedur.

Przykład:

```
org 100h
mov  AL, zmX
lea  BX, zmX
mov  byte ptr [BX], 55h
mov  AL, zmX
ret
zmX  DB  22h
end
```

lub z operatorem OFFSET

```
org 100h
mov  AL, zmX
mov  BX, offset  zmX
mov  byte ptr [BX], 55h
mov  AL, zmX
ret
zmX  DB  22h
end
```

Instrukcja LEA jest bardziej elastyczna niż operator OFFSET gdyż pozwala pobrać adres zmiennej z indeksem np. lea BX, tablica[4]

Zadanie:

Sprawdzić wykonanie powyższych programów

Adresowanie bazowe

Adresowanie bazowe jest to rodzaj adresowania pośredniego, gdzie rozkaz wskazuje na jeden z rejestrów bazowych *BX* lub *BP* i może zawierać 8- lub 16-bitową wartość stanowiącą lokalne przemieszczenie. Adresem efektywnym jest suma zawartości rejestru bazowego i lokalnego przemieszczenia. Np. **mov AX, [BP+2]**.

Adresowanie indeksowe

Adresowanie indeksowe jest rodzajem adresowania pośredniego, gdzie adres efektywny jest sumą zawartości rejestru indeksowego *SI* lub *DI* i lokalnego przemieszczenia. Np. **mov AX, [SI+3].**

Adresowanie bazowo-indeksowe

W adresowaniu bazowo-indeksowym, adres efektywny jest sumą zawartości jednego z rejestrów bazowych, jednego z rejestrów indeksowych i lokalnego przemieszczenia. Np. **mov AX, [SI+BP+4].**

Rozkazy operujące na ciągach słów

Rozkazy operujące na ciągach słów posługują się rejestrami indeksowymi. Rejestry *SI* i *DI* zawierają adresy efektywne pierwszego słowa odpowiednio w ciągu źródłowym i wynikowym. Po każdej transmisji rejestry indeksowe są automatycznie inkrementowane lub dekrementowane w zależności od ustawienia bitu *DF* w rejestrze znaczników.

Rozkazy operujące na rejestrach WE/WY

Rozkazy operujące na rejestrach WE/WY (**in** oraz **out**) zawierają adres WE/WY (adres natychmiastowy) lub posługują się zawartością rejestru *DX* (adresowanie pośrednie).

Instrukcja XCHG dokonuje zamiany miejscami dwóch argumentów np.

```
MOV AL, 5
MOV AH, 2
XCHG AL, AH   ; AL = 2, AH = 5
XCHG AL, AH   ; AL = 5, AH = 2
```

Zamiana może nastąpić pomiędzy dwoma rejestrami lub między rejestrem i komórką pamięci.

Zadanie

Napisać program złożony z instrukcji przesyłania danych różnych typów pomiędzy różnymi lokalizacjami (rejestry, pamięć adresowana w różnych trybach). Zadeklarować do tego celu odpowiednie zmienne. Sprawdzić sposób kodowania tych rozkazów, ich długość i wykonanie. Dokonywać podglądu zawartości rejestrów oraz zmiennych, a także sprawdzać zachowanie licznika rozkazów podczas pracy krokowej z programem.