

Факултет за информатички науки и компјутерско инженерство, Скопје

## **Микропроцесорски системи**

Подготовка за лабораториска вежба број 00

# **ОСНОВНИ ВО АСЕМБЛЕР 8086**

## Основни во асемблер 8086

Целта на оваа вежба е запознавање на регистрите за општа намена на процесорот 8086 и основните инструкции со нив во асемблер, и тоа и како 8 и како 16 битни регистри. Исто така, при извршувањето на секоја инструкция, со помош на симулаторот ќе ги следите вредностите на сите регистри, како и регистарот на знаменца, кога и како и при која операција истиот се менува.

Во оваа лабораториска вежба имате неколку примери кои се порачува да ги извршите самостојно на емулаторот.

### Предмет на работа

1. Запознавање на регистрите AX, BX, CX, DX, како и нивните „горни и долни половинки“ AH, AL, BH, BL, CH, CL, DH, DL.
2. Креирање на нова програма.
3. Извршување на инструкции. (пример MOV и ADD)

## Запознавање на регистрите на процесорот 8086

### Основни регистри

8086 CPU има 8 основни регистри за општа намена, кои се 16 битни, и секој регистар има свое име:

- AX - Accumulator register (поделен на 2 8-битни AH / AL)
- BX - Base address register (поделен на 2 8-битни BH / BL)
- CX - Count register (поделен на 2 8-битни CH / CL)
- DX - Data register (поделен на 2 8-битни DH / DL)
- SI - Source index register
- DI - Destination index register
- BP - Base pointer
- SP - Stack pointer

### Сегментни регистри

- CS – покажува на сегментот кој го содржи програмот
- DS – генерално покажува на сегментот каде што променливите се дефинираат
- ES – Екстра сегментен регистар, и се зависи од кодерот за што ќе се користи
- SS – Покажува на сегментот кој го содржи стекот

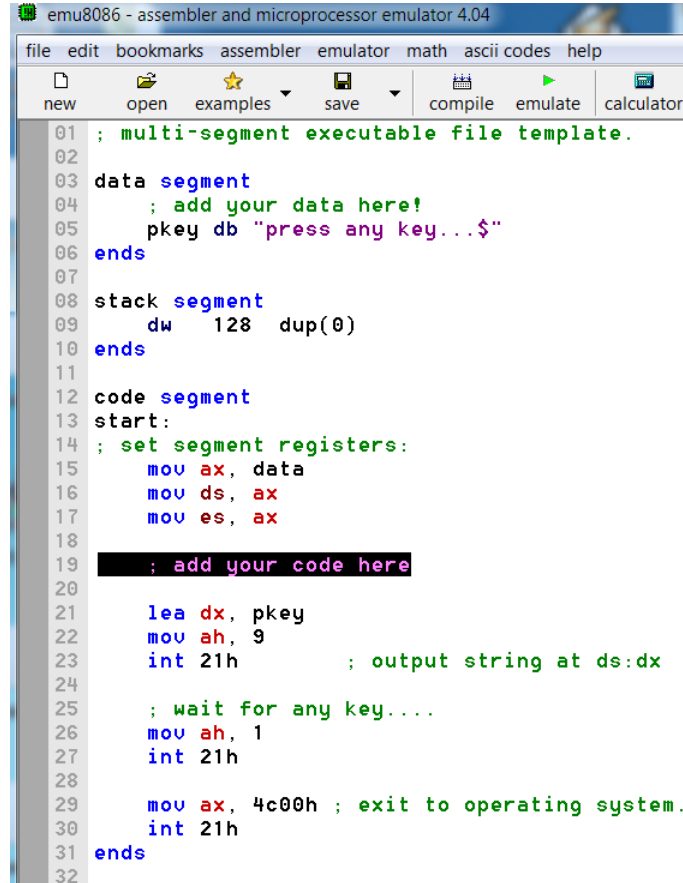
### Регистри за специјална намена

- IP – Инструкциски покажувач
- Flags register – Ја разрешува моменталната состојба на микропроцесорот

Во оваа лабораториска вежба ќе ги користиме 4-те основни регистри за операции – AX, BX, CX, DX. За таа цел, ќе го користиме симулаторот EMU8086. Отворете го од работната површина симулаторот EMU8086 каде што ќе можете да ги следите промените не само на AX, BX, CX, DX регистрите, туку и на останатите регистри и меморијата. Исто така, со симулаторот ќе може и да се креираат и изврши датотеки - ехе од напишаните програми.

## Креирање на нова програма

За да може да се креира некаков асемблерски код, претходно мора да се избере / креира некаков шаблон на кој ќе се работи. Тоа се прави со File -> New по што се отвара прозорец во кој се дадени 4 опции. Од опциите изберете ја опцијата *exe*. Со избирање на опцијата *exe* се појавува следниот прозорец.



```
01 ; multi-segment executable file template.
02
03 data segment
04 ; add your data here!
05 pkey db "press any key...$"
06 ends
07
08 stack segment
09 dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15 mov ax, data
16 mov ds, ax
17 mov es, ax
18
19 ; add your code here
20
21 lea dx, pkey
22 mov ah, 9
23 int 21h ; output string at ds:dx
24
25 ; wait for any key....
26 mov ah, 1
27 int 21h
28
29 mov ax, 4c00h ; exit to operating system.
30 int 21h
31 ends
32
```

Дадена е една почетна програма напишана во асембли јазикот (датотека со екстензија .asm). Во почетниот код во програмата има дефинирано 3 сегменти. Првиот сегмент е податочен (со име data), вториот е стек сегментот (со име stack) и третиот е кодниот сегмент (со име code).

Во податочниот сегмент е дефиниран стринг. За овие податоци ќе зборуваме повеќе во следните лабораториски вежби. Покрај дадената променлива, во овој сегмент може да се дефинираат и други променливи. Подолу ќе бидат дадени примери.

Со стек сегментот е дефиниран стек со големина 128 податоци од тип dw (data word) кои се големина од 2 бајти. Сите податоци во стекот се иницијализирани на 0.

Во последниот коден сегмент се пишува кодот на програмата кој ќе се изврши.

Со првите три инструкции се поставуваат вредностите за регистрите ds и es кои ќе го означуваат податочниот сегмент дефиниран погоре. Овие инструкции се:


```
MOV AX, data
MOV DS, AX
MOV ES, AX
```

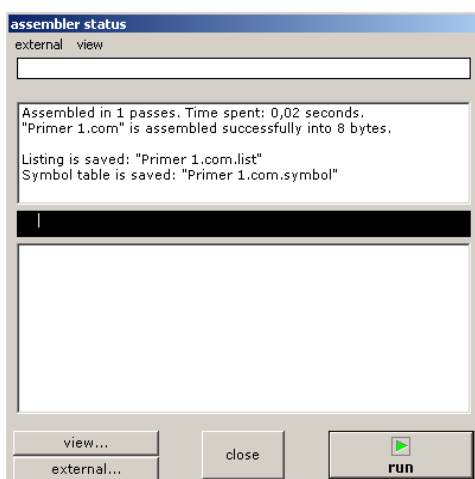
Со првата инструкција се поставува точната адреса за податочниот сегмент означен со data во регистрот AX. Потоа, оваа адреса се копира во DS и ES. Да забележиме дека не може директно да се копира data во DS.

По овие инструкции има коментар со кој се означува каде треба да го пишувате кодот. Коментарот се пишува после симболот точка - запирка.

Кодот кој е напишан после коментарот е за печатење на порака на екранот и за излез од програмата. Овие инструкции ќе ги изучиме понатака. За сега ќе останат така во програмата.

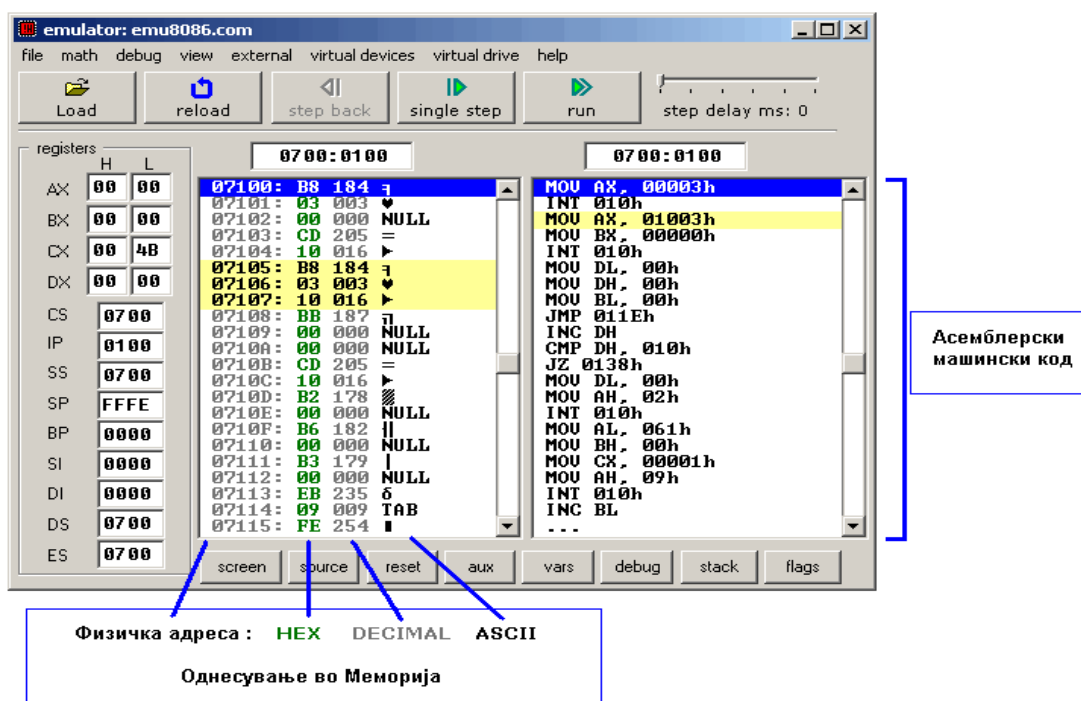
За да видиме што сме напишале и дали е се во ред со програмата, треба програмата да се компајлира. Со компајлирањето се проверува дали програмата има грешки. Тоа се прави со притискање

на копчето  **compile**. Потоа се бара да се зачува оваа програма некаде на вашиот диск. Откако ќе се зачува се појавува следниот прозорец.



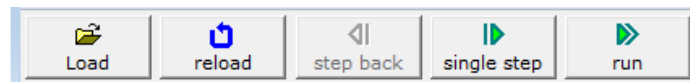
Тука се појавуваат информации за програмата т.е за колку време била компајлирана. Ако имате грешки ќе добиете информација за грешките. Ако нема грешки, ќе се генерира .exe верзија од програмата.

Следно, за да се изврши програмата, се притиска на копчето **EMULATE**. Со тоа сме ја повикале програмата за извршување. Веднаш се појавуваат 2 нови прозорци:



**Емулатор** – Ова е прозорец дава визуелен приказ за извршувањето на програмата со процесор 8086. Од левата страна се прикажани сите регистри со нивните вредности. Во прозорецот на средина е дадено како во меморија се напишани инструкциите кои следно треба да извршат (следната инструкција е со жолта позадина на буквите). Јасно се гледа дека следната инструкција е запишана во 3 бајти во меморија со физички адреси (07105, 07106 и 07107). За секој бајт е дадена неговата хексидекадна вредност (во зелено), декадна вредност (сиво) и вредноста на ASCII (црно). Во прозорецот од десната страна паралелно се прикажува следната инструкција во машинскиот код.

Во горниот дел од прозорецот има повеќе копчиња како на сликата. Со RUN програмата се извршува, а со SINGLE STEP се извршува само следната инструкција (**single step**).



**Оригинален изворен код** – Тоа е целиот испишан код по кој ќе се движи емулаторот т.е секоја линија код ќе биде обработена од емулаторот и ќе биде испишана во неговиот прозорец. Со жолто е означена следна инструкција која ќе се изврши напишана во јазикот асемблер.

```
original source code
01 ; multi-segment executable file template.
02
03 data segment
04 ; add your data here!
05 pkey db "press any key...$"
06 ends
07
08 stack segment
09 dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15 mov ax, data
16 mov ds, ax
17 mov es, ax
18
19 ; add your code here
20
21 lea dx, pkey
22 mov ah, 9
23 int 21h ; output string at ds:dx
24
25 ; wait for any key....
26 mov ah, 1
27 int 21h
28
29 mov ax, 4c00h ; exit to operating system.
30 int 21h
31 ends
```

## Извршување инструкции

Во овој дел ќе се запознаеме со некои од позначајните инструкции. Ќе започнеме со еден основен асемблерски код со цел подобро објаснување на асемблерот како јазик и неговите основни инструкции. Искуцајте го следниот код во обележаното место:

```
MOV AX, 1010H
MOV AH, 25D
ADD AL, 101B
```

Да ги појасниме сите редови еден по еден:


**MOV AX, 1010H** - Во регистарот AX се сместува хексадецимална вредност 1010h. Запаметете дека сите вредности кои имаат “h” се хексадецимални броеви, ако има “b” после вредноста се бинарни, и ако има “d” се декадни вредности.

Со оваа наредба, во истовреме, AH регистарот добива вредност 10h, и тоа од горниот бајт од **1010** (означен со црвено). Исто така, и AL регистарот добива вредност 10h, и тоа е долниот бајт од **1010** (означен со црвено).


**MOV AH, 25D** - На регистарот AH му се доделува декадна вредност 25 што е исто со 19 во хексидекаден формат.

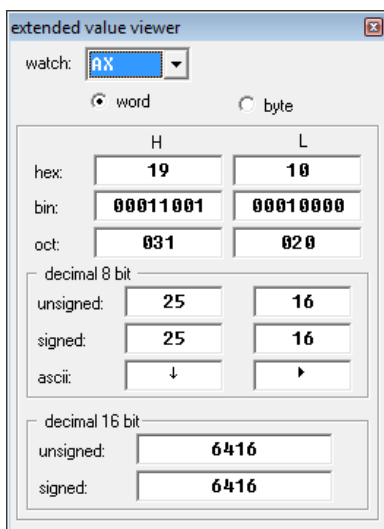
**ADD AL, 101B** - На регистарот AL (на веќе запаметената вредност од погоре **10h**) му се додава вредност 101B.

Се вчитува инструкцијата **MOV AX, 1010H** и се извршува. Тоа значи дека во регистарот AX


сместуваме (му доделуваме) вредност 1010H. По кликување на копчето  се преминува на следната инструкција, но се врши промена на вредноста во регистарот AX. Следете го регистарот AX при извршување на секоја инструкција чекор по чекор.

Се вчитува инструкцијата **MOV AH, 25D** и се извршува. Тоа значи на регистарот AH му поставуваме

(доделуваме) вредност 19H. По кликување на копчето  се преминува на следната инструкција, но се врши промена на вредноста во регистарот AH. Во емулаторот вредностите на регистрите се покажуваат во хексидекаден формат, а за да ги видите вредностите во другите формати со двоен клик притиснете на соодветниот регистар.



Потоа се вчитува инструкцијата `ADD AL 101B` и се извршува. Тоа значи дека на веќе внесената

вредност на регистарот AL му додаваме 101B. По кликување на копчето  се преминува на следна инструкција, но се врши промена на вредноста во регистарот AL т.е на претходната вредност и се додава вредност 101B. Вредноста која ќе се зачува во AL е 15h. Зошто?

Во дадениот код се користи инструкцијата MOV за податочен трансфер и аритметичката инструкција ADD за собирање. На зудиториските вежби поркај иснструкцијата ADD дадени се и други аритметички и логички инструкции (SUB, INC, MUL, OR, XOR,AND....).