

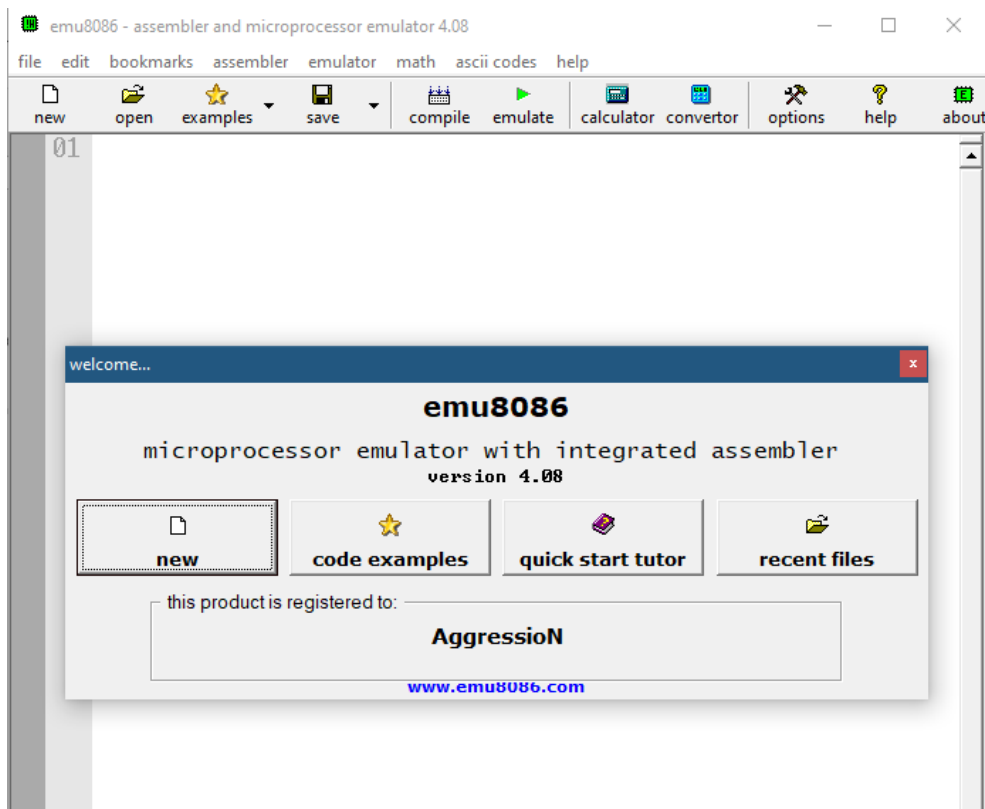
به نام خدا

## تمرین دوم درس ریزپردازنده

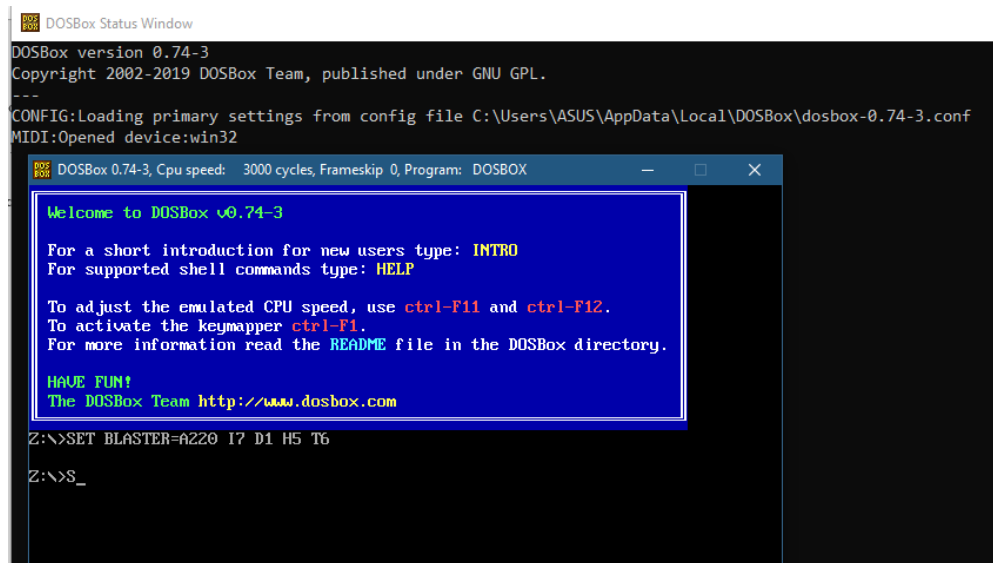
غزل زمانی نژاد

۹۷۵۲۲۱۶۶

مورد اول: ابتدا نرم افزار emu8086 را نصب میکنیم.



سپس با استفاده از [این لینک](#) نرم افزار DOSBox را نصب کرده و پس از دانلود 8086 فولدر آن را در درایو C کپی میکنیم.



مورد سوم و چهارم:

Emu8086

- Simplified segment definition: ابتدا مدل را انتخاب میکنیم. بعد stack segment را مشخص میکنیم. در data segment متغیرهای مورد نیاز یعنی دو عدد و حاصل ضرب شان را تعریف میکنیم. در code segment، برای انجام عملیات ضرب بدون استفاده از mul، از یک حلقه استفاده میکنیم و در آن به تعداد عدد دوم، عدد اول را با خودش جمع میکنیم. برای اینکه متغیر دوم نقش متغیر حلقه را داشته باشد، آن را در CX قرار میدهیم. که به صورت خودکار در هر loop از مقدار آن کاسته شود.

```

01 ; simplified segment definition
02
03 .MODEL SMALL
04 .STACK 64
05 .DATA
06 ;
07 ;place data definitions here
08 DATA1 DW 32H
09 DATA2 DW 23H
10 SUM DW ?
11 ;
12
13 .CODE
14 MAIN PROC FAR
15     MOV     AX,@DATA
16     MOV     DS,AX
17     ;
18     ;place code here
19     MOV     AX,DATA1
20     MOV     CX,DATA2
21     MULTIPLY:ADD SUM,AX
22     LOOP    MULTIPLY
23     ;
24
25     MOV     AH,4CH
26     INT     21H
27 MAIN ENDP
28     END     MAIN

```

سپس بر روی emulate و بعد run میزنیم و مقدار خروجی را چک میکنیم. همچنین کد را compile میکنیم.

emulator: multiply-simplified.exe\_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	4C	32
BX	00	00
CX	00	00
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	003A	
BP	0000	
SI	0000	

F400:0204

F4200:	FF	255	RES
F4201:	FF	255	RES
F4202:	CD	205	=
F4203:	21	033	!
F4204:	CF	207	±
F4205:	00	000	NULL
F4206:	00	000	NULL
F4207:	00	000	NULL
F4208:	00	000	NULL
F4209:	00	000	NULL
F420A:	00	000	NULL
F420B:	00	000	NULL
F420C:	00	000	NULL
F420D:	00	000	NULL
F420E:	00	000	NULL
F420F:	00	000	NULL
F4210:	00	000	NULL
F4211:	00	000	NULL
F4212:	00	000	NULL

BIOS DI  
INT 021h  
IRET  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL  
ADD [BX + SI], AL

original source code

```

15 MOV     AX,@DATA
16 MOV     DS,AX
17 ;
18 ;place code here
19 MOV     AX,DATA1
20 MOV     CX,DATA2
21 MULTIPLY:ADD SUM,AX
22 LOOP    MULTIPLY
23 ;
24
25 MOV     AH,4CH
26 INT     21H

```

variables

size: word elements: 1

edit show as: hex

DATA1	0032h
DATA2	0023h
SUM	06D6h

32 x 23 =  
6D6

MS M\*

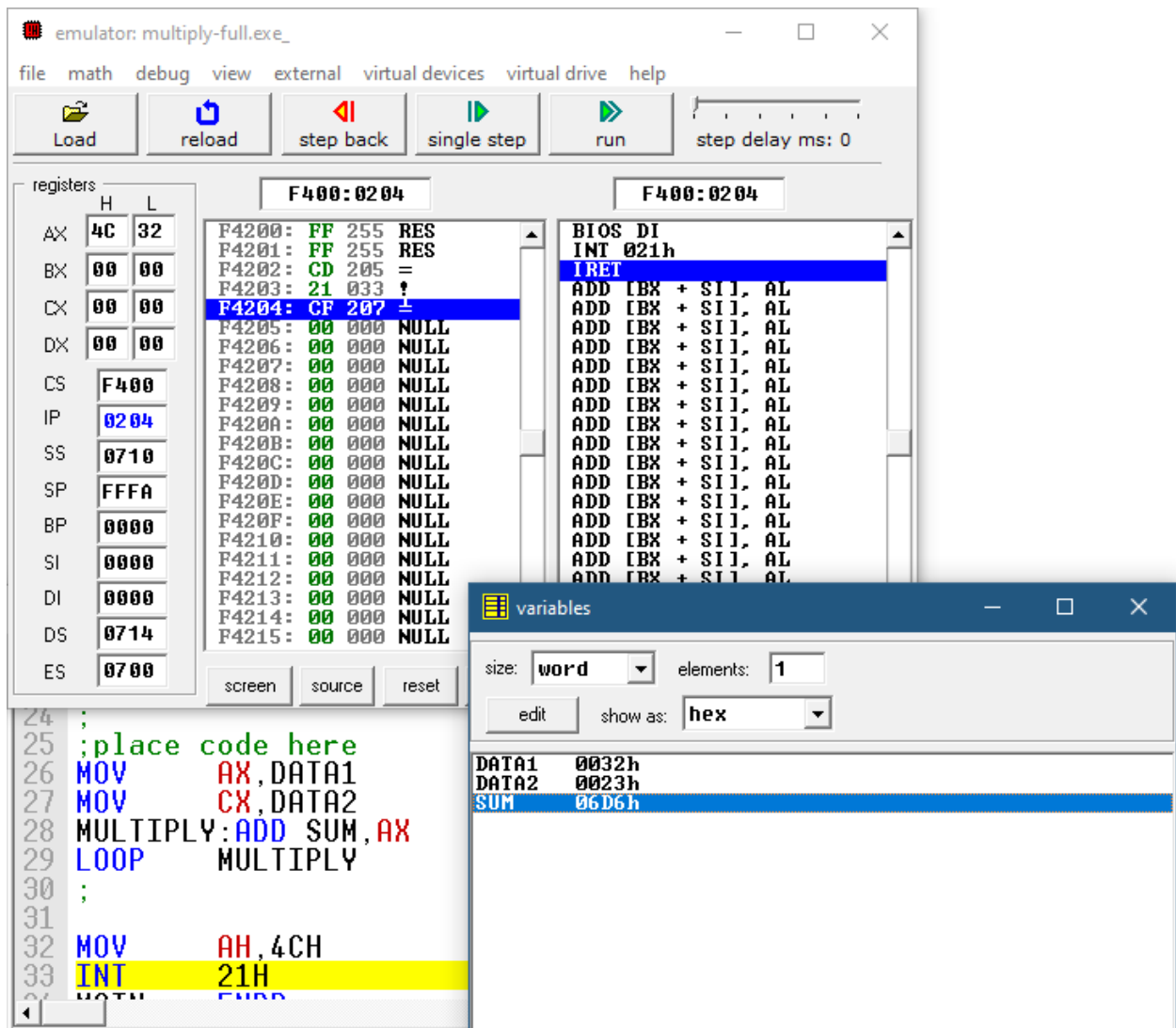
- Full segment definition: پیاده سازی آن مشابه simplified است. تنها تفاوت در آن است که باید segmentها را به صورت جداگانه تعریف کنیم و ابتدا و انتهای آن ها را مشخص کنیم. و بعد از دستور assume استفاده میکنیم تا نام هایی که برای segmentها مشخص کرده ایم به نام اصلی شان مپ کنیم.

```

01 ; full segment definition
02
03 ;-----STACK SEGMENT -----
04 STSEG    SEGMENT
05         DB      64 DUP(?)
06 STSEG    ENDS
07
08 ;-----DATA SEGMENT -----
09 DTSEG    SEGMENT
10 ;
11 ;place data definitions here
12 DATA1   DW      32H
13 DATA2   DW      23H
14 SUM      DW      ?
15 ;
16 DTSEG    ENDS
17
18 ;-----CODE SEGMENT -----
19 CDSEG    SEGMENT
20 MAIN     PROC     FAR
21         ASSUME   CS:CDSEG,DS:DTSEG,SS:STSEG
22         MOV      AX,DTSEG
23         MOV      DS,AX
24         ;
25         ;place code here
26         MOV      AX,DATA1
27         MOV      CX,DATA2
28         MULTIPLY:ADD SUM,AX
29         LOOP     MULTIPLY
30         ;
31
32         MOV      AH,4CH
33         INT      21H
34 MAIN     ENDP
35
36 CDSEG    ENDS
37 END      MAIN
38

```

سپس بر روی emulate و بعد run میزنیم و مقدار خروجی را چک میکنیم. همچنین کد را compile میکنیم.



- Simplified segment definition: ابتدا مدل را انتخاب میکنیم. بعد stack segment را مشخص میکنیم. در data segment متغیرهای مورد نیاز یعنی آرایه 10 تایی از اعداد و حاصل جمع، میانگین، ماکسیمم و مینیمم را تعریف میکنیم. در code segment، برای تمیزتر بودن کد، هر یک از موارد خواسته شده را جداگانه تعریف میکنیم و در main تنها آنها را call میکنیم.
- توضیح sum: ابتدا source index pointer را به ابتدای آرایه اعداد اشاره میدهیم. متغیر حلقه را برابر 10 قرار میدهیم (طول آرایه). در یک loop مقدار آدرسی که پوینتر به آن اشاره میکند را به sum اضافه میکنیم. بعد پوینتر را یکی به جلو اشاره میدهیم.
- توضیح average: مجموع اعداد که قبلاً محاسبه کرده بودیم در ax میریزیم. در bl عدد 10 (تعداد اعداد) را میریزیم. با اجرای دستور div bl، مقدار خارج قسمت تقسیم انجام شده در ah ذخیره میشود. آن را در avg میریزیم.
- توضیح max: مقدار متغیر max را عدد کوچک (صفر) قرار میدهیم. در یک لوپ با دستور cmp چک میکنیم که آیا مقدار خانه آرایه از مقدار max بزرگتر است یا خیر. اگر بزرگتر باشد به label دیگری jump میکنیم تا مقدار max را آپدیت کنیم. بعد به همان نقطه return میکنیم و پوینتر را یکی به جلو اشاره میدهیم.
- توضیح min: مشابه max است. اما در متغیر min عددی بزرگ (FF) را ذخیره میکنیم.

```

01 ; simplified segment definition
02
03 .MODEL SMALL
04 .STACK 64
05 .DATA
06 ;
07 ;place data definitions here
08 NUMBERS DB 0H, 2H, 4H, 6H, 8H, 9H, 7H, 5H, 3H, 1H
09 SUM DB ?
10 MAX DB ?
11 MIN DB ?
12 AVG DB ?
13 ;
14
15 .CODE
16 MAIN PROC FAR
17     MOV AX,@DATA
18     MOV DS,AX
19     ;
20     ;place code here
21     MOV SI,OFFSET NUMBERS ;put offset of array in source index pointer
22     CALL CAL_SUM
23     CALL CAL_AVG
24     CALL CAL_MAX
25     CALL CAL_MIN
26     ;
27
28     MOV AH,4CH
29     INT 21H
30 MAIN ENDP
31
32
33 ;-----SUM-----
34 CAL_SUM PROC
35     MOV SI,0H
36     MOV CX,0AH
37
38 LOOP1: MOV AL,[SI]
39         ADD SUM,AL
40         INC SI ;increment source ptr by 1
41     LOOP LOOP1
42     RET
43 CAL_SUM ENDP
44
45
46 ;-----AVG-----
47 CAL_AVG PROC
48     MOV AL,SUM
49     MOV AH,0
50     MOV BL,0AH
51     DIV BL ;al / bl
52     MOV AVG,AX
53     RET
54 CAL_AVG ENDP
55
56 ;-----MAX-----
57 CAL_MAX PROC
58     MOV SI,0H ;set source pointer to head of the array
59     MOV CX,0AH ;loop condition checks cx
60
61     MOV MAX,0H
62
63 LOOP2: MOV AL,[SI]
64         CMP AL,MAX
65         JA GREATER
66     RTN_POINT: INC SI
67     LOOP LOOP2
68     RET
69
70 GREATER: MOV MAX,AL
71     JMP RTN_POINT
72
73 CAL_MAX ENDP
74
75 ;-----MIN-----
76 CAL_MIN PROC
77     MOV SI,0H ;set source pointer to head of the array
78     MOV CX,0AH ;loop condition checks cx
79
80     MOV MIN,0FFH
81
82 LOOP3: MOV AL,[SI]
83         CMP AL,MIN
84         JB LOWER
85     RTN_POINT2: INC SI
86     LOOP LOOP3
87
88 LOWER: MOV MIN,AL
89     JMP RTN_POINT2
90
91 CAL_MIN ENDP
92
93
94
95
96
97
98 END MAIN

```

سپس بر روی emulate و بعد run میزنیم و مقدار خروجی را چک میکنیم. همچنین کد را compile میکنیم.



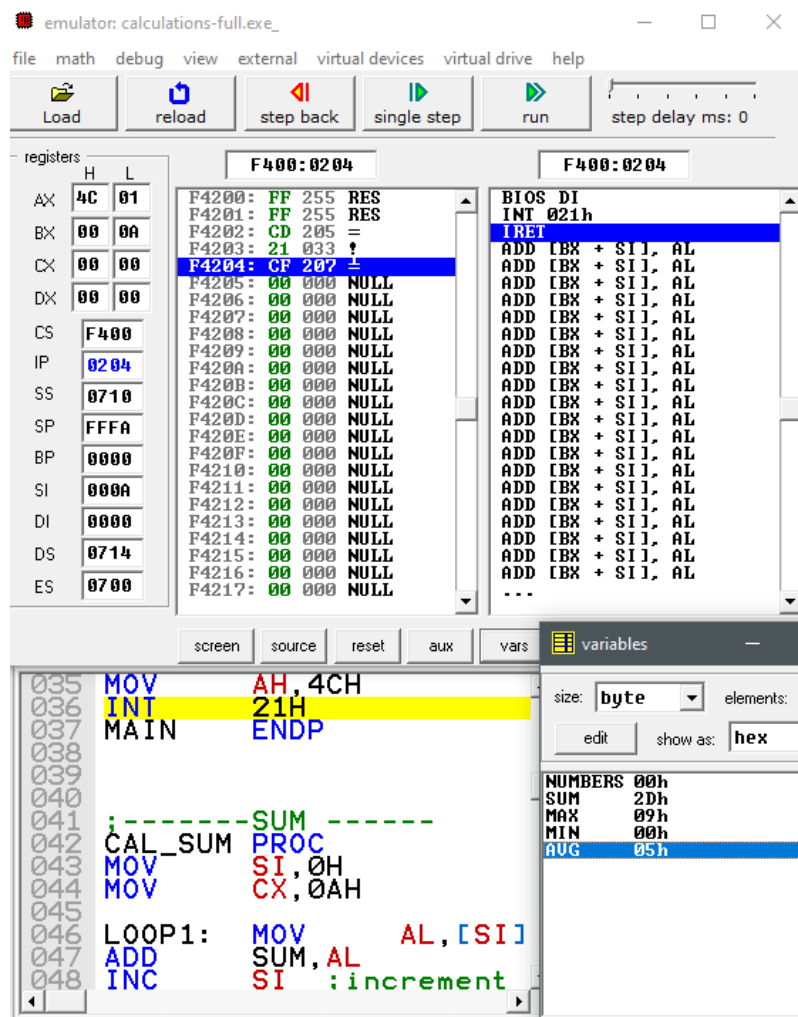


```

0001 ; full segment definition
0002
0003 ;-----STACK SEGMENT -----
0004 STSEG SEGMENT
0005 DB 64 DUP(?)
0006 STSEG ENDS
0007
0008 ;-----DATA SEGMENT -----
0009 DTSEG SEGMENT
0010 ;
0011 ;place data definitions here
0012 NUMBERS DB 0H, 2H, 4H, 6H, 8H, 9H, 7H, 5H, 3H, 1H
0013 SUM DB ?
0014 MAX DB ?
0015 MIN DB ?
0016 AVG DB ?
0017 ;
0018 DTSEG ENDS
0019
0020 ;-----CODE SEGMENT -----
0021 CDSEG SEGMENT
0022 MAIN PROC FAR
0023 ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
0024 MOV AX,DTSEG
0025 MOV DS,AX
0026 ;
0027 ;place code here
0028 MOV SI,OFFSET NUMBERS ;put offset of array in source index pointer
0029 CALL CAL_SUM
0030 CALL CAL_AVG
0031 CALL CAL_MAX
0032 CALL CAL_MIN
0033 ;
0034
0035 MOV AH,4CH
0036 INT 21H
0037 MAIN ENDP
0038
0039 ;
0040
0041 ;-----SUM-----
0042 CAL_SUM PROC
0043 MOV SI,0H
0044 MOV CX,0AH
0045
0046 LOOP1: MOV AL,[SI]
0047 ADD SUM,AL
0048 INC SI ;increment source ptr by 1
0049 LOOP LOOP1
0050 RET
0051 CAL_SUM ENDP
0052
0053 ;
0054
0055 ;-----AVG-----
0056 CAL_AVG PROC
0057 MOV AL,SUM
0058 MOV AH,0
0059 MOV BL,0AH
0060 DIV BL ;a1 / b1
0061 MOV AVG,AX
0062 RET
0063 CAL_AVG ENDP
0064
0065 ;
0066
0067 ;-----MAX-----
0068 CAL_MAX PROC
0069 MOV SI,0H ;set source pointer to head of the array
0070 MOV CX,0AH ;loop condition checks cx
0071 MOV MAX,0H
0072
0073 LOOP2: MOV AL,[SI]
0074 CMP AL,MAX
0075 JA GREATER
0076 RTN_POINT: INC SI
0077 LOOP LOOP2
0078
0079 GREATER: MOV MAX,AL
0080 JMP RTN_POINT
0081
0082 CAL_MAX ENDP
0083
0084 ;
0085
0086 ;-----MIN-----
0087 CAL_MIN PROC
0088 MOV SI,0H ;set source pointer to head of the array
0089 MOV CX,0AH ;loop condition checks cx
0090 MOV MIN,0FFH
0091
0092 LOOP3: MOV AL,[SI]
0093 CMP AL,MIN
0094 JB LOWER
0095 RTN_POINT2: INC SI
0096 LOOP LOOP3
0097
0098 LOWER: MOV MIN,AL
0099 JMP RTN_POINT2
0100
0101 CAL_MIN ENDP
0102
0103
0104
0105 CDSEG ENDS
0106 END MAIN
0107

```

سپس بر روی emulate و بعد run میزنیم و مقدار خروجی را چک میکنیم. همچنین کد را compile میکنیم.



## :MASM+LINK

کدهای استفاده شده در این بخش، همان کدهای قبلی هستند. تنها نحوه کامپایل کردن آن متفاوت است. برای مثال، برای کامپایل کردن اولین سوال ابتدا وارد DOS می‌شویم. سپس با دستور mount درایو c را mount می‌کنیم. به درایو c می‌رویم. دستور masm q1simple.asm را اجرا می‌کنیم تا برای ما فایل obj را تشکیل دهد.

```
Z:\>mount c c:\8086
Drive C is mounted as local directory c:\8086\

Z:\>c:

C:\>masm q1simple.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

Object filename [q1simple.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

51688 + 464856 Bytes symbol space free

0 Warning Errors
0 Severe Errors
```

بعد از دستور link q1simple.obj استفاده می‌کنیم و فایل exe را می‌سازیم. سپس فایل exe ساخته شده را اجرا می‌کنیم.

```
C:\>link q1simple.obj

Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Run File [Q1SIMPLE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:








































C:\>q1simple.exe
```

تمامی این دستورها را برای 3 فایل دیگر نیز اجرا می‌کنیم.

## مورد پنجم:

با کامپایل کردن کدها توسط هر دو نرم افزار، چند نوع فایل ایجاد میشود.

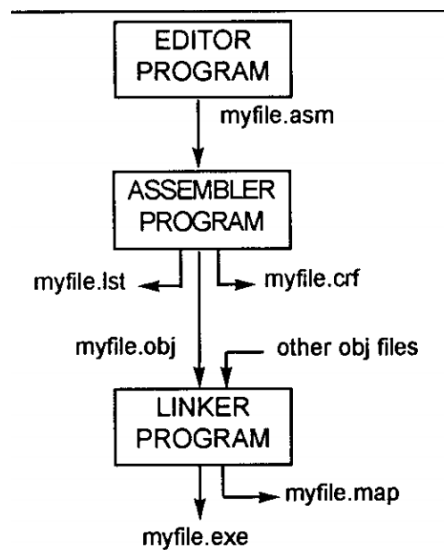
فایل های ایجاد شده با EMU8086:

Name	Date modified	Type	Size
 calculations-full.exe	11/23/2021 6:32 PM	Application	1 KB
 calculations-full.exe.asm	11/23/2021 6:32 PM	~ASM File	2 KB
 calculations-full.exe.debug	11/23/2021 6:32 PM	DEBUG File	3 KB
 calculations-full.exe.list	11/23/2021 6:32 PM	LIST File	10 KB
 calculations-full.exe.symbol	11/23/2021 6:32 PM	SYMBOL File	2 KB
 calculations-full.exe_	11/24/2021 6:53 PM	assembly source c...	1 KB
 calculations-full.exe_.asm	11/24/2021 6:53 PM	~ASM File	2 KB
 calculations-full.exe_.debug	11/24/2021 6:53 PM	DEBUG File	3 KB
 calculations-full.exe_.list	11/24/2021 6:53 PM	LIST File	10 KB
 calculations-full.exe_.symbol	11/24/2021 6:53 PM	SYMBOL File	2 KB
 calculations-simplified.exe	11/23/2021 6:38 PM	Application	1 KB
 calculations-simplified.exe.asm	11/23/2021 6:38 PM	~ASM File	2 KB
 calculations-simplified.exe.debug	11/23/2021 6:38 PM	DEBUG File	3 KB
 calculations-simplified.exe.list	11/23/2021 6:38 PM	LIST File	9 KB
 calculations-simplified.exe.symbol	11/23/2021 6:38 PM	SYMBOL File	2 KB
 calculations-simplified.exe_	11/24/2021 6:49 PM	assembly source c...	1 KB
 calculations-simplified.exe_.asm	11/24/2021 6:49 PM	~ASM File	2 KB
 calculations-simplified.exe_.debug	11/24/2021 6:49 PM	DEBUG File	3 KB
 calculations-simplified.exe_.list	11/24/2021 6:49 PM	LIST File	9 KB
 calculations-simplified.exe_.symbol	11/24/2021 6:49 PM	SYMBOL File	2 KB
 multiply-full.exe	11/23/2021 4:09 PM	Application	1 KB
 multiply-full.exe.asm	11/23/2021 4:09 PM	~ASM File	1 KB
 multiply-full.exe.debug	11/23/2021 4:09 PM	DEBUG File	1 KB
 multiply-full.exe.list	11/23/2021 4:09 PM	LIST File	6 KB
 multiply-full.exe.symbol	11/23/2021 4:09 PM	SYMBOL File	2 KB
 multiply-full.exe_	11/24/2021 6:32 PM	assembly source c...	1 KB
 multiply-full.exe_.asm	11/24/2021 6:32 PM	~ASM File	1 KB
 multiply-full.exe_.debug	11/24/2021 6:32 PM	DEBUG File	1 KB
 multiply-full.exe_.list	11/24/2021 6:32 PM	LIST File	6 KB
 multiply-full.exe_.symbol	11/24/2021 6:32 PM	SYMBOL File	2 KB
 multiply-simplified.exe	11/23/2021 3:50 PM	Application	1 KB
 multiply-simplified.exe.asm	11/23/2021 3:50 PM	~ASM File	1 KB
 multiply-simplified.exe.debug	11/23/2021 3:50 PM	DEBUG File	1 KB
 multiply-simplified.exe.list	11/23/2021 3:50 PM	LIST File	5 KB
 multiply-simplified.exe.symbol	11/23/2021 3:50 PM	SYMBOL File	1 KB
 multiply-simplified.exe_	11/24/2021 6:22 PM	assembly source c...	1 KB
 multiply-simplified.exe_.asm	11/24/2021 6:22 PM	~ASM File	1 KB
 multiply-simplified.exe_.debug	11/24/2021 6:22 PM	DEBUG File	1 KB
 multiply-simplified.exe_.list	11/24/2021 6:22 PM	LIST File	5 KB

## فایل های ایجاد شده با MASM+LINK:

Q2FULL.EXE	11/24/2021 6:02 PM	Application	1 KB
Q2FULL.OBJ	11/24/2021 6:02 PM	Object File	1 KB
Q2SIMPLE.EXE	11/24/2021 6:01 PM	Application	1 KB
Q2SIMPLE.OBJ	11/24/2021 6:01 PM	Object File	1 KB
q2simple.asm	11/24/2021 6:00 PM	Assembler Source	2 KB
Q1FULL.EXE	11/24/2021 5:59 PM	Application	1 KB
Q1FULL.OBJ	11/24/2021 5:59 PM	Object File	1 KB
q1full.asm	11/24/2021 5:59 PM	Assembler Source	1 KB
Q1SIMPLE.EXE	11/24/2021 5:52 PM	Application	1 KB
Q1SIMPLE.OBJ	11/24/2021 5:50 PM	Object File	1 KB
q1simple.asm	11/24/2021 5:44 PM	Assembler Source	1 KB
q2full.asm	11/23/2021 6:31 PM	Assembler Source	3 KB

مراتب اصلی تولید فایل ها به صورت زیر است:



فایل با پسوند .lst: این فایل برای برنامه نویس ها بسیار مفید است چون در آن لیستی از تمامی opcodeها، آدرس های آفست و همچنین ارورها وجود دارد. MASM به صورت خودکار فرض میکند که کاربر را نمیخواهد. اما اگر در prompt درخواست دهیم آن را برای ما ایجاد میکند. برنامه نویس ها از این فایل برای دیباگ کردن استفاده میکنند. بعد از برطرف کردن تمامی ارورهایی که در این فایل مشخص شده میتوانیم فایل obj را به عنوان ورودی به linker بدهیم. برای پرینت کردن محتویات این فایل از دستور زیر استفاده میشود:

**C>type myfile.lst | more**

برای اینکه فایل `lst` خواناتر باشد میتوانیم در کد اسمبلی از `PAGE` و `TITLE` استفاده کنیم.

فایل با پسوند `crf`: فایل `cross-reference` شامل یک لیست الفبایی از تمامی `symbol`ها و برچسب های استفاده شده در کد و همچنین شماره خطی که به آنها رجوع شده هست. این فایل برای برنامه های بزرگی که شامل تعداد زیادی `data segment` و `code segment` هستند بسیار مورد استفاده قرار میگیرد.

فایل با پسوند `map`: این فایل در برنامه ای که شامل `segment`های زیادی میشود میتواند مورد استفاده قرار گیرد. شامل نام هر `segment`، نقطه شروع آن، نقطه پایان آن و سایز آن به بایت میشود.