وزارت علوم تحقیقات و فن‌آوری
دانشگاه پیام نور

زبان ماشین و برنامه سازی سیستم

(رشته مهندسی کامپیوتر)

مهندس داریوش نیک‌مهر

فهرست مطالب

عنوان **صفحه**

فصل سوم : برنامه‌نویسی

فصل چهارم : دستورالعملهای اساسی

فصل پنجم : انشعاب و تکرار

فصل ششم : عملیات بیتی

فصل هفتم: مکروها و روال‌ها و وقفه‌ها

فصل هشتم : عملیات پردازش رشته‌ها

پیشگفتار

با لطف و عنایت پرودگار متعال کتاب زبان ماشین و برنامه سازی سیستم با توجه به نیاز دانشجویان دانشگاه پیام نور در رشته مهندسی کامپیوتر بصورت خودآموز و با مثالهای زیاد و ساده و روان تهیه گردیده است. مطالب ارائه شده با توجه به تجربیات تدریس در درس برنامه‌نویسی زبان اسمبلی در سالیان متمادی در دانشگاه می‌باشد. این کتاب در ده فصل آماده شده که هر فصل دارای اهداف فصل، تمرین و مروری بر مطالب فصل می‌باشد. در انتهای کتاب سؤالات چهار گزینه‌ای و نهایتاً واژه نامه گنجانده شده است.

در نهایت از زحمات و همکاری آقای مهندس نوید نیک‌مهر در تهیه این کتاب قدردانی و سپاسگزاری نموده و این کتاب را به همسر مهربانم و فرزندان دلبندم که همواره مشوق اینجانب در کارهای علمی و پژوهشی می‌باشند تقدیم می‌نمایم.

داریوش نیک‌مهر

فصل اول

سیستم اعداد

هدف کلی

نمایش مقادیر در سیستم دودوئی و نحوه تبدیل آنها به سایر سیستمها.

اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می‌شوید.

۱-مقادیر دودوئی یا باینری.

۲-واحدهای مختلف اندازه‌گیری حافظه.

۳-نمایش اعداد منفی.

٤-تبدیل مقادیر باینری به سیستم دهدهی و برعکس.

۵-نمایش مقادیر در سیستم شانزده تائی.

٦-نمایش مقادیر در سیستم هشت تائی.

۷-تبدیل مقادیر از سیستم دهدهی به سیستم هشت تائی و برعکس.

۸-تبدیل مقادیر از سیستم شانزدهدهی به سیستم دهدهی و برعکس.

۹-تبدیل مقادیر از سیستم شانزدهدهی به سیستم مبنای هشت و برعکس.

۱-۱- مقادیر دودوئی (Binary)

بشر با توجه به تعداد انگشت‌هایش از ده رقم 0,1,2,3,4,5,6,7,8,9 برای ایجاد مقادیر و اعداد و انجام محاسبات روی آنها استفاده می‌نماید. به بیانی دیگر بشر در یک سیستم دهدهی یا Decimal کار می‌کند. از طرف دیگر کامپیوتر در یک سیستم دودویی یا Binary کار می‌کند و فقط دو رقم 1 و 0 را می‌شناسد. در نتیجه هر مقداری که به کامپیوتر داده شود بایستی تبدیل به یک سری 0 و 1 گردد تا بتواند در کامپیوتر ذخیره و مورد استفاده در محاسبات قرار گیرد. برای تبدیل مقادیر از سیستم دهدهی به سیستم دودویی بایستی آن مقدار بطور متوالی بر 2 تقسیم نمائیم. بعنوان مثال عدد 50 را در نظر بگیرید.

مثال ۱-۱

مقدار	تقسیم بر	نتیجه	باقیمانده
50	2	25	0
25	2	12	1
12	2	6	0
6	2	3	0
3	2	1	1
1	2	0	1

عدد 50 معادل 110010 در سیستم دودویی می‌باشد.

به منظور تبدیل مقداری از سیستم باینری به سیستم دهـدهی، ارقـام عـدد را می‌بایستی بترتیب از راست به چپ در 1، 2، 8، 16 ... ضرب نموده بـا هـم جمـع نمائیم. به عنوان مثال عدد 11010 در سیستم دودویی را در نظر بگیرید.

مثال ۲-۱

1	1	0	1	0
16	8	4	2	1
16*1+			16+	
8*1			8	
4*0			0	
2*1			2	
1*0			0	
			26	

بعبارت دیگر ارقام را بایستی بترتیب در 2^0 ، 2^1، 2^2، 2^3، 2^4 ، ... ضرب نمود.

1	1	0	1	0
2^4	2^3	2^2	2^1	2^0

مثال ۳-۱

عدد 37 را به سیستم دودویی تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
37	2	18	1
18	2	9	0
9	2	4	1
4	2	2	0
2	2	1	0
1	2	0	1

بنابراین مقدار 37 برابر با 100101 در سیستم دودویی می‌باشد.

مثال ٤-١

عدد 1101101 را به سیستم دهدهی تبدیل نمائید.

1	1	0	1	1	0	1
64	32	16	8	4	2	1

64+
32
8
4
1

109

نتیجه میشود که عدد 1101101 در سیستم دودویی معادل 109 در سیستم دهدهی می‌باشد.

۲-۱- جمع و تفریق در سیستم دودویی

جمع و تفریق در سیستم دودویی شبیه جمع و تفریق در سیستم دهدهی می‌باشد با این تفاوت که به جای ده بر یک، دو بر یک (Carry) ایجاد می‌شود. فرض کنید دو مقدار 3 و 10 در سیستم دودویی با هم جمع نمائیم. ابتدا بایستی هر کدام از این مقادیر را به سیستم دودویی تبدیل نموده سپس آنها را با هم جمع نمائیم.

10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

ملاحظه می‌شود که 10 در سیستم دودویی برابر است با 1010.

از طرف دیگر مقدار 3 در سیستم دودویی را بدست می‌آوریم.

3	2	1	1
1	2	0	1

حال دو مقدار 11 و 1010 با هم جمع می‌نمائیم.

1 Carry

1010+

11

1101

در مورد 1+1 بایستی در نظر داشت که نتیجه میشود 10. که یک carry یک به ستون بعدی منتقل می‌گردد.

مثال ۵-۱

مجموع دو مقدار 20 و 17 را بدست آورید.

ابتدا مقادیر 17 و 20 را به سیستم دودویی تبدیل می‌نمائیم.

20	2	10	0
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

مقدار 20 میشود 10100 در سیستم دودویی.

17	2	8	1
8	2	4	0
4	2	2	0
2	2	1	0
1	2	0	1

این نشان می‌دهد که 17 معادل 10001 در سیستم دودویی می‌باشد.

حال

1 Carry

10001+

10100

100101

که این مقدار یعنی 100101 اگر به سیستم دهدهی تبدیل شود برابر است با

1	0	0	1	0	1
32	16	8	4	2	1

32+

4

1

37

در مورد تفریق در سیستم دهدهی همانطوریکه ملاحظه می‌گردد در صورت لزوم یک 1 در سیستم دهدهی قرض گرفته می‌شود.

مثال ٦-١

534 -

281

253

ولی در سیستم دودویی در صورت لزوم یک 1 در سیستم دودویی قرض گرفته که borrow نامیده می‌شود. مثال

```
 1011-
 0110
------
 0101
```

۳-۱- بایت (Byte)

در حافظه کامپیوتر فقط مقادیر 0 و 1 ذخیره میشود. به ارقام 0 و 1 بیت گفته میشود. بیت مخفف کلمات binary digit می‌باشد. به هر هشت بیت کنار هم در حافظه کامپیوتر بایت گفته میشود. بیت‌های یک بایت از 0 تا 7 شماره گذاری شده و بیت شماره 0 بیت کم ارزش‌ترین یا LSB و بیت شماره 7 بیت با بیشترین ارزش یا MSB می‌باشد.

7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	1

هر بایت 256 وضعیت مختلفه از 0 و 1 را ایجاد می‌نماید. بنابراین اعداد صحیح بین 0 تا 255 را می‌توان در یک بایت قرار داد. از طرف دیگر در کامپیوتر از 256 کارکتر مختلف می‌توان استفاده نمود. با استفاده از جدول کد ASCII می‌توان به هر کاراکتر یک کد منحصر بفرد بین 0 تا 255 تخصیص داد. بنابراین هر کاراکتر عملاً یک بایت اشغال می‌نماید.

٤-۱- مقادیر منفی

اعداد و مقادیر منفی در کامپیوتر با استفاده از روش مکمل 2 نمایش داده می‌شوند. برای نمایش یک مقدار منفی در کامپیوتر بایستی مراحل زیر را طی نمود.

۱–ابتدا عدد را بدون علامت تصور نموده آنرا به سیستم دودویی تبدیل نمائید.

۲–سپس آنقدر رقم 0 در سمت چپ نتیجه مرحله 1 قرار می‌دهیم تا تعداد ارقام آن مضربی از هشت گردد. چنانچه نتیجه مرحله 1 از هشت رقم بیشتر باشد بایستی آنقدر 0 در سمت چپ قرار دهیم تا شانزده رقمی گردد.

۳–سپس ارقام نتیجه مرحله 2 را مکمل می‌نمائیم یعنی 0 به 1 و 1 به 0 تبدیل می‌کنیم.

٤–نتیجه بدست آمده را در سیستم دودویی با 1 جمع می‌نمائیم.

مثال ۷–۱

عدد 26– را در نظر بگیرید. ابتدا عدد 26 را به سیستم دودویی تبدیل می‌نمائیم.

26	2	13	0
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

که میشود 11010.

حال نتیجه بدست آمده را هشت رقمی می‌نمائیم.

00011010

سپس 0 ها را به 1 و 1 ها را به 0 تبدیل می‌کنیم.

11100101

حال نتیجه بدست آمده را با 1 جمع می‌نمائیم.

$$\begin{array}{r} 11100101+ \\ \qquad\quad 1 \\ \hline 11100110 \end{array}$$

عدد 11100110 در سیستم دودویی نمایش 26– می‌باشد که یک بایت اشغال می‌نماید. نکته مهمی که بایستی در نظر داشت این است که MSB اعداد منفی در روش مکمل 2 همیشه 1 می‌باشد.

مثال ۸–۱

عدد 35– را به سیستم دودویی تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
35	2	17	1
17	2	8	1
8	2	4	0
4	2	2	0
2	2	1	0
1	2	0	1

که نتیجه می‌شود 35 معادل 100011 در سیستم دودویی می‌باشد. حال نتیجه بدست آمده را هشت رقمی می‌نمائیم.

00100011

سپس 0ها را به ۱ و 1 ها را به 0 تبدیل می‌کنیم.

11011100

حال نتیجه بدست آمده را با 1 جمع می‌کنیم

$$\begin{array}{r} 1101100 + \\ \quad\quad 1 \\ \hline 11011101 \end{array}$$

مقدار 11011101 در سیستم دودویی معادل 35– می‌باشد. که همانطوریکه ملاحظه میگردد بیت MSB آن برابر با یک می‌باشد.

مثال ۹-۱

عمل زیر را با استفاده از روش مکمل 2 انجام دهید.

```
 27-
 20
 ---
```

این عمل تفریق در حقیقت بمنزله جمع دو مقدار زیر می‌باشد.

27+(-20)

حال مقادیر 20– و 27 را به سیستم دودویی تبدیل نموده.

27	2	13	1
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

مقدار 27 معادل 11011 در سیستم دودویی می‌باشد. حال ابتدا مقـدار 20 را به سیستم دودویی تبدیل نمود.

20	2	10	0
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

مقدار 20 برابر است با 10100 در سیستم دودویی. حال 20- را در سیـستم دودویی بدست می‌آوریم. برای این کار ابتدا عدد را هشت رقمی نموده

00010100

سپس صفرها را به 1 و یکها را به صفر تبدیل می‌نمائیم.

11101011

آنگاه مقدار 1 به آن اضافه می‌نمائیم.

```
11101011+
       1
--------
11101100
```

نتیجه می‌شود که مقدار 20– برابر است با 11101100 در سیستم دودویی.

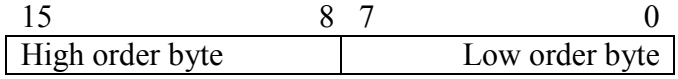
حال دو مقدار 20– و 27 را در سیستم دودویی با هم جمع می‌نمائیم.

```
11101100 +
   11011
---------
100000111
```

با توجه به آنکه نتیجهٔ جمع دو بایت بصورت یک بایت می‌باشد بیت 1 سمت چپ بایستی حذف گردد، نتیجه می‌شود 111 که برابر با 7 می‌باشد.

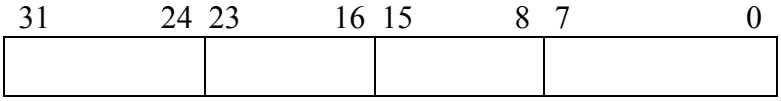
۵-۱- گروه‌بندی بیت‌ها

به هر هشت بیت کنار هم بایت گفته می‌شود. دو بایت کنار هم یعنی شانزده بیت متوالی را word می‌نامند(البته در بعضی از کامپیوتر ها هر کلمه می تواند شامل ٤ بایت باشد). بیت‌های یک word از 0 تا 15 شماره‌گذاری می گردد. در یک word بایت سمت راست را بایت مرتبه پائین (Low order byte) و بایت سمت چپ را بایت مرتبه بالا (High order byte)گفته می‌شود.



در یک Word بیت شماره 0 را LSB و بیت شماره 15 را MSB می‌نامند.

از طرف دیگر چهار بایت متوالی تشکیل یک Double word میدهند.



هر هشت بایت متوالی تشکیل یک Quadword میدهد و نهایتاً هر هشتاد بیت متوالی یا ده بایت متوالی تشکیل یک Tenbyte می‌دهد. جدول ذیل مقادیری که در یک byte ، word، double word قرار می‌گیرند را نشان می‌دهد.

جدول ۱-۱

نوع	مقادیر بدون علامت	مقادیر علامت دار
Byte	0 تا 255	–128 تا 127
Word	0 تا 65535	–32768 تا 32767
Double word	0 تا 2^{32}-1	-2^{31} تا 2^{31}-1

بایستی توجه داشت که عملیات باینری روی بیت‌ها انجام می‌شود. جدول عملگر جمع بصورت زیر می‌باشد.

جدول ۱-۲

بیت 1	بیت 2	نتیجه	دوبریک (carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

جدول عملگر تفریق نیز بصورت زیر می‌باشد.

جدول ۳-۱

بیت 1	بیت 2	نتیجه	یک قرضی (borrow)
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

٦-١- عملیات در سیستم مبنای شانزده

ارقام در سیستم مبنای شانزده یا Hexadecimal عبارتند از 0 تا 15. بمنظور جلوگیری از ابهام، ارقام 10 تا 15 را بترتیب با حروف A تا F نشان داده میشوند.

A	10
B	11
C	12
D	13
E	14
F	15

برای تبدیل مقداری از سیستم دهدهی به سیستم مبنای شانزده آن عدد را بطور متوالی بر 16 تقسیم می‌نمائیم. بعنوان مثال عدد 174 را در نظر بگیرید.

مثال ۱۰-۱

مقدار	تقسیم بر	نتیجه	باقیمانده	
174	16	10	14	E
10	16	0	10	A

که نتیجه می‌شود AE.

مثال ۱۱-۱

عدد 3740 را از سیستم دهدهی به سیستم مبنای شانزده تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
3740	16	233	12 C
233	16	14	9
14	16	0	14 E

چون 14 معادل E می‌باشد و C معادل 12 می‌باشد بنابراین جواب مـی‌شـود E9C در سیستم مبنای شانزده.

مثال ۱۲-۱

مقدار 27845 را به سیستم مبنای شانزده تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
27845	16	1740	5
1740	16	108	12 C
108	16	6	11 B
6	16	0	6

مقدار 27845 برابر با 6CC5 در سیستم شانزدهی می‌باشد.

برای تبدیل مقداری از سیستم شانزدهی به سیستم دهدهی ارقام عدد را از سمت راست بترتیب در 16^3،16^2،16،1 ... ضرب نموده با هم جمع می‌نمائیم.

مثال ۱۳-۱

عدد 2AF5 را در نظر بگیرید.

2	A	F	5
16^3	16^2	16	1

5*1+	5 +
F*16	15*16
A*16^2	10*256
2*16^3	2*4096
	10997

که نتیجه منجر میشود به 2AF5 که برابر با 10997 می‌باشد.

مثال ۱۴-۱

مقدار 4F2 در سیستم مبنای شانزده چه مقدار در سیستم دهدهی می‌باشد؟ برای اینکار ابتدا رقم 2 را در 1، رقم F را در 16 و رقم 4 را در 16^2 ضرب می‌نمائیم. سپس مقادیر بدست آمده را با هم جمع می‌کنیم.

4	F	2
16^2	16	1

$$4*16^2+$$
$$F*16$$
$$2*1$$

که منجر میشود به

$$4*256+$$
$$15*16$$
$$2*1$$

که نتیجه می‌شود

$$1024+$$
$$240$$
$$2$$
$$1266$$

مقدار 4F2 در سیستم شانزدهدهی برابر با 1266 در سیستم دهدهی می‌باشد.

از طرف دیگر هر رقم در سیستم مبنای شانزده را می‌توان بوسیله چهار رقم در سیستم باینری نمایش داد.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

حال برای تبدیل یک مقدار در سیستم مبنای شانزده به سیستم دودویی می‌توان از جدول مذکور استفاده نموده و ارقام را با مقدار معادل آن جایگزین نمود. به عنوان مثال عدد 2FA5B را در نظر بگیرید. با جایگزینی هر رقم با چهار رقم معادل آن در سیستم دودویی نتیجه زیر حاصل می‌گردد.

0010111111010010111011

به منظور تبدیل یک مقدار از سیستم دودویی به سیستم مبنای شانزده ابتدا ارقام را از سمت راست چهار تا چهار تا جدا نموده سپس با استفاده از جدول فوق مقادیر معادل را قرار می‌دهیم.

مثال ۱۵-۱

1110110010111101

که ابتدا بصورت زیر در می‌آوریم.

0111 0110 0101 1101

که معادل 765D می‌باشد.

۷-۱- عملیات در سیستم مبنای هشت (Octal)

ارقام در سیستم مبنای هشت عبارتند از 0 تا 7 . برای تبدیل مقداری از سیستم دهدهی به سیستم مبنای هشت بایستی آن مقدار را بطور متوالی بر هشت تقسیم نمود. بعنوان مثال عدد 125 را در نظر بگیرید.

مثال ۱۶-۱

مقدار	تقسیم بر	نتیجه	باقیمانده
125	8	15	5
15	8	1	7
1	8	0	1

که نتیجه می‌شود 175 در سیستم مبنای هشت.

مثال ۱۷-۱

بمنظور تبدیل مقداری از سیستم مبنای هشت به سیستم دهدهی، ارقام عدد را از سمت راست بترتیب در 1، 8، 8^2، 8^3، ... ضرب نموده نتایج حاصله را با هم جمع می‌نمائیم. بعنوان مثال عدد 237 در سیستم مبنای هشت را در نظر بگیرید.

2 3 7

8^2 8 1

7*1+	7
3*8	24
$2*8^2$	128
	159

که نتیجه می‌شود 159 در سیستم دهدهی.

مثال ۱۸-۱

عدد 4260 را از سیستم دهدهی به سیستم مبنای هشت تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
4260	8	532	4
532	8	66	4
66	8	8	2
8	8	1	0
1	8	0	1

نتیجه می‌شود که 4260 در سیستم دهدهی معادل 10244 در سیستم مبنای هشت می‌باشد.

مثال ۱۹-۱

عدد 382 را به سیستم مبنای هشت تبدیل نمائید.

مقدار	تقسیم بر	نتیجه	باقیمانده
382	8	47	6
47	8	5	7
5	8	0	5

که نتیجه می‌شود 576 در سیستم مبنای هشت.

مثال ۲۰-۱

مقدار 4327 را از سیستم مبنای هشت به سیستم دهدهی تبدیل نمائید. برای اینکار ابتدا رقم 7 را در 1، رقم 2 را در 8، رقم 3 را در 8^2 و نهایتاً رقم 4 را در 8^3 ضرب می‌نمائیم سپس مجموع مقادیر بدست آمده را محاسبه می‌نمائیم.

4 3 2 7

8^3 8^2 8 1

که نتیجه می‌شود

$4*8^3$ +

$3*8^2$

2*8

7*1

که معادل است با

4*512+

3*64

2*8

7*1

که نهایتاًبرابر است با

2048+

192

16

7

2263

بنابراین مقدار 4327 در سیستم مبنای هشت برابر است با 2263 در سیستم دهدهی.

بایستی توجه نمود که هر رقم در سیستم مبنای هشت را می‌توان بوسیله سه رقم در سیستم دودویی نمایش داد.

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

برای تبدیل مقداری از سیستم هشت تائی به سیستم دودویی کافی است که به جای هر رقم در سیستم مبنای هشت سه رقم معادل آنرا قرار داد. بعنوان مثال عدد 417 در سیستم مبنای هشت معادل 100001111 در سیستم دودویی می‌باشد.

بمنظور تبدیل مقداری از سیستم دودویی به سیستم مبنای هشت کافی است که ارقام عدد از طرف راست سه تا سه تا جدا نموده و به جای آنها مقدار معادل در سیستم مبنای هشت قرار دهیم.

مثال ۲۱-۱

عدد 10110111010111 در سیستم دودویی در نظر بگیرید که می‌توان بصورت زیر جدا نمود.

010 110 111 010 111

که جواب نهائی میشود 26727 در سیستم مبنای هشت.

از طرف دیگر برای تبدیل مقداری از سیستم مبنای هشت به سیستم مبنای شانزده و برعکس می‌بایستی ابتدا مقدار را به سیستم دودویی تبدیل نموده سپس به سیستم مبنای هشت یا مبنای شانزده تبدیل نمود.

مثال ۲۲-۱

عدد 2AFB5 را در نظر بگیرید.

2AFB5

2	A	F	B	5
0010	1010	1111	1011	0101

حال سه رقم سه رقم از سمت راست جدا نموده.

000 101 010 111 110 110 101

که نهایتاً برابر با 527665 در سیستم مبنای هشت می‌باشد.

۸-۱- مقادیر اعشاری

به منظور تبدیل یک مقدار اعشاری به سیستم دودویی ابتدا قسمت صحیح آنرا به طریق گفته شده به سیستم دودویی تبدیل نموده، سپس قسمت اعشاری آنرا جدا نموده بطور مکرر در 2 ضرب می‌نمائیم. بعنوان مثال عدد 14.725 را در نظر بگیرید. عدد 14 بصورت 1110 در سیستم دودویی می‌باشد. برای تبدیل قسمت اعشاری یعنی 0.725 به سیستم دودویی آنرا در 2 ضرب می‌نمائیم.

مثال ۲۳-۱

$$\begin{array}{r} 0.725* \\ \quad 2 \\ \hline 1.450 \end{array}$$

قسمت صحیح یعنی 1 را جدا نموده، قسمت اعشار را در 2 ضرب می‌نمائیم.

$$\begin{array}{r} 0.45* \\ \quad 2 \\ \hline 0.90 \end{array}$$

قسمت صحیح یعنی 0 را جدا نموده، قسمت اعشار را در 2 ضرب می‌کنیم.

$$\begin{array}{r} 0.8* \\ 2 \\ \hline 1.6 \end{array}$$

و به همین روال کار را ادامه می‌دهیم.

$$\begin{array}{r} 0.6* \\ 2 \\ \hline 1.2 \end{array}$$

جواب می‌شود 1110.10111

برای تبدیل یک مقدار اعشاری از سیستم دودویی به سیستم دهدهی قسمت صحیح آنرا از سمت راست بترتیب در 2^0 ، 2^1 ، 2^2 ، 2^3 ، .... ضرب نموده با هم جمع می‌کنیم سپس قسمت اعشار آنرا بترتیب از سمت چپ در 2^{-1}، 2^{-2} ، 2^{-3} ، 2^{-4}،... ضرب نموده با هم جمع می‌نمائیم. بعنوان مثال عدد 1101.01011 در سیستم دودویی را در نظر بگیرید.

مثال ۲٤-۱

1	1	0	1	.	0	1	0	1	1
8	4	2	1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

$$1*8+4*1+2*0+1*1+0*\frac{1}{2}+1*\frac{1}{4}+0*\frac{1}{8}+1*\frac{1}{16}+1*\frac{1}{32}$$

که خلاصه می‌شود

$$8+4+1+\frac{1}{4}+\frac{1}{16}+\frac{1}{32}=13.34375$$

مروری بر مطالب فصل

در حافظه کامپیوتر فقط ارقام 0 و 1 ذخیره می‌گردد. به این ارقام 0 و 1 بیت گفته می‌شود. به هر هشت کنار هم بایت و به هر شانزده بیت کنار هم word گفته می‌شود. برای ذخیره یک مقدار در حافظه بایستی ابتدا آنرا بصورت یک‌سری بیت درآورد. و برای نمایش مقادیر روی صفحه مانیتور بایستی آنها را به سیستم ده‌تایی تبدیل نمود. در سیستم دهدهی فقط از ارقام 0 تا 9 استفاده می‌گردد. در سیستم دودویی فقط از ارقام 0 و 1 استفاده می‌گردد. اعداد منفی را می‌توان در حافظه کامپیوتر با استفاده از روش مکمل 2 نشان داد. استفاده از سیستم‌هایی مبنای شانزده و مبنای هشت نیز امکان پذیر می‌باشد. در سیستم مبنای شانزده از ارقام 0 تا 9 و A تا F استفاده میگردد. در سیستم مبنای هشت فقط از ارقام 0 تا 7 می‌توان استفاده نمود. مقادیر اعشاری را نیز می‌توان در حافظه کامپیوتر قرار داد.

⌘ تمرین

۱–هر بایت از ..... بیت تشکیل شده است.

۲–هر کلیو بایت معادل ..... بایت می‌باشد.

۳–در هر Word اعداد 0 تا ..... را می‌توان جا داد.

٤–در هر بایت اعداد ..... – تا .....+ را می‌توان‌قرار داد.

۵–در کامپیوتر برای نمایش اعداد منفی از ..... استفاده می‌شود.

٦–در کامپیوتر به جای عمل تفریق از ..... استفاده می‌گردد.

۷–در هر Word به تعداد ..... وضعیت مختلفه 1 و 0 وجود دارد.

۸–عدد 20– را به سیستم دودویی تبدیل نمائید.

۹–عمل 18-25 را با استفاده از روش مکمل دو انجام دهید.

۱۰–عدد 1101101 را به سیستم دهدهی تبدیل نمائید.

۱۱–عدد 101101.11001 را به سیستم ده تائی تبدیل کنید.

۱۲–عدد 2FABC را به سیستم هشت تائی تبدیل نمائید.

۱۳–عدد 43271 در سیستم مبنای هشت را به سیستم دودویی تبدیل نمائید.

١٤–مشخص نمائید که اگر MSB یک مقداری یک باشد آیا آن مقدار منفی است؟

۱۵–مشخص نمائید که آیا می‌توان عمل ضرب و تفریق و تقسیم را به عمل جمع تبدیل نمود؟

١٦–اگر مقداری منفی باشد.

الف–MSB آن صفر است.

ب–MSB آن یک است.

ج–LSB آن یک است.

د.–هیچکدام.

۱۷-اگر مقداری از نوع double word داشته باشیم به چند بایت حافظه نیاز است؟

الف-2 ب-3

ج-4 د-هیچکدام

۱۸-مقدار 7AB در سیستم شانزدهدهی چه مقداری در سیستم دودویی می‌باشد؟

الف-11110101011 ب-101110100111

ج-11011011111 د-هیچکدام

۱۹-مقدار 6- در سیستم دهدهی معادل چه مقداری در سیستم دودویی می‌باشد؟

الف-11111001 ب-11111010

ج-11110111 د-هیچکدام

۲۰-تفاضل کد اسکی 'a' و 'A' چیست؟

الف-30 ب-32

ج-40 د-هیچکدام

فصل دوم

معماری ریزپردازنده 80286

هدف کلی

معرفی ریزپردازنده 80286

اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می‌شوید

۱-یزپردازنده 80286 و عملیات آن.

۲-معماری ریزپردازنده ۸۰۲۸٦ و اجزاء تشکیل دهنده.

۳- ثبات‌ها

٤- فلگ‌ها

۱-۲- ریز پردازنده 80286

این ریزپردازنده دارای ویژگیهای پیشرفته‌ای برای عملکرد در سطح بالائی را دارد. در 80286، همچنین استفاده از ویژگی‌های ذیل امکان پذیر می‌باشد.

۱- Multitasking

۲- Multiuser systems

این ریزپردازنده دارای هشت ثبات (Register) شانزده بیتی بنامهای AX، BX، CX، DX، SP، BP، SI، DI می‌باشد. چهار ثبات AX، BX، CX، DX را می‌توان بعنوان ثباتهای شانزده بیتی در نظر گرفت یا هر کدام را بعنوان دو ثبات هشت بیتی در نظر گرفت و استفاده نمود.

15 8	7 0	
AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

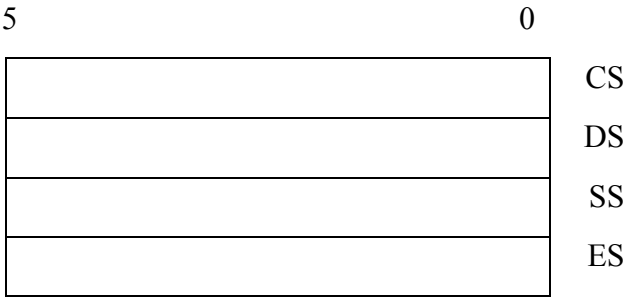
AX	Accumulator
BX	Base
CX	Count
DX	Data

15 0	
	SP
	BP
	SI
	DI

SP	Stack pointer
BP	Base pointer
SI	Source index

DI Destination index

معمولاً از ثباتهای SP و BP در مورد عملیات روی پشته‌ها و از ثباتهای SI و DI بعنوان شاخص در ساختارهای پیچیده‌تر استفاده می گردد. ریزپردازنده همچنین دارای چهار ثبات شانزده‌بیتی معروف به ثباتهای سگمنت می‌باشد که بمنظور آدرس دهی از آنها استفاده می‌نماید. این ثبات‌ها بنامهای CS،DS ، SS، ES می‌باشند.



CS	Code	Segment
DS	Data	Segment
SS	Stack	Segment
ES	Extra	Segment

سگمنت ها ناحیه‌های پیوسته در حافظه می باشند. اندازه هر سگمنت می‌تواند تا 64K بایت باشد.

بایستی توجه داشت که در ثبات CS آدرس شروع ناحیه‌ای از حافظه که کدهای دستورالعمل در آن قرار دارد گذاشته میشود. در ثبات DS آدرس شروع ناحیه‌ای از حافظه که داده‌ها در آن قرار دارد گذاشته می‌شود. در ثبات SS آدرس شروع ناحیه‌ای از حافظه که پشته در آن ایجاد میشود قرار داده میشود. و در صورتیکه در برنامه‌ها از دستورالعملهای پردازش

رشته‌ها استفاده شود آدرس ابتدای آن ناحیه در حافظه در ثبات ES قرار داده می‌شود.

۱-۱-۲- ثبات فلگ (Flag register)

ریزپردازنده 80286 دارای یک ثبات شانزده بیتی بنام ثبات فلگ می‌باشد. دوازده بیت از این ثبات فقط می‌تواند مورد استفاده برنامه نویس قرار بگیرد. هر کدام از این بیت‌ها نام خاصی دارد و مشخص کننده وضعیت خاصی می‌باشد.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
//	NT	IO	PL	OF	DF	IF	TF	SF	ZF	//	AF	//	PF	//	CF

CF	Carry Flag
PF	Parity Flag
AF	Auxiliary Flag
ZF	Zero Flag
SF	Sign Flag
OF	Overflow Flag
TF	Trap Flag
IF	Interrupt Enable Flag
IOPL	Input / output privilege Level Flag
NT	Nested Task Flag

زمانی که در عملیات جمع و تفریق (هشت یا شانزده‌بیتی) یک carry یا borrow ایجاد می‌گردد مقدار CF برابر با یک میشود. در غیر اینصورت مقدار فلگ CF برابر با صفر می‌شود. از فلگ CF در دستورالعملهای شیفت و چرخش نیز استفاده می‌گردد.

از فلگ PF اساساً در کاربردهای ارتباطات داده‌ای استفاده می‌شود. زمانیکه توازن فرد ایجاد شود مقدار آن یک میشود و در صورت ایجاد توازن زوج مقدار آن

صفر می‌گردد. بعبارت دیگر اگر تعداد بیت‌های یک در بایت مرتبه پائین word زوج باشد مقدار PF برابر یک می‌گردد. در غیر اینصورت مقدار PF برابر با صفر می‌گردد.

در عمل جمع دو مقدار اگر کلیه بیت‌های نتیجه صفر گردد مقدار ZF برابر با یک می‌گردد. در غیر اینصورت مقدار ZF برابر با صفر می‌شود.

در عمل جمع دو مقدار باینری اگر از MSB نتیجه یک carry خارج گردد مقدار CF برابر با یک می‌شود. در غیر اینصورت مقدار CF برابر با صفر می‌شود.

در عمل جمع دو مقدار باینری اگر MSB نتیجه برابر با یک باشد مقدار فلگ SF برابر با یک میشود. در غیر اینصورت مقدار SF برابر با صفر می‌باشد.

در عمل جمع دو مقدار باینری اگر یک carry در بیت قبل از MSB نتیجه داشته باشیم ولی در بیت MSB carry نداشته باشیم و یا بالعکس مقدار فلگ OF 1 می‌شود. در غیر اینصورت مقدار فلگ OF صفر می‌گردد. زمانیکه OF برابر با 1 می‌شود یعنی نتیجه محاسبه در محل تعیین شده در حافظه جا نمی‌گیرد و باعث می‌شود که تعدادی از بیت‌های نتیجه حذف شود.

فلگ AF در مورد عملیات BCD کاربرد دارد که در بخشهای بعدی بحث میشود. در عمل جمع دو مقدار باینری مقدار فلگ AF یک میشود اگر از بیت شماره 3 یک carry خارج گردد.

مثال ۱-۲

دو مقدار 2345 و 3219 را در سیستم مبنای شانزده در نظر بگیرید و با هم جمع نمائید و مقدار فلگ‌ها را مشخص نمائید.

```
 00110010 00011001+
 0010001101000101
------------------
 0101010101011110
```

MSB برابر با 0 می‌باشد.	SF=0
تمام بیت‌ها 0 نمی‌باشند.	ZF=0
تعداد بیت‌های 1 در هشت بیت اول فرد می‌باشد.	PF=0
در بیت 15 ، carry نداریم.	CF=0
در بیت شماره 3 ، carry نداریم.	AF=0
در بیت‌های 14 و 15 ، carry نداریم.	OF=0

مثال ۲-۲

حال مجموع دو مقدار 5439 و 456A در سیستم مبنای شانزده را محاسبه نموده سپس مقدار فلگ‌ها را مشخص می‌نمائیم.

$(5439)_{16}=(0101010000111001)_2$
$(456A)_{16}=(0100010101101010)_2$

```
 0101010000111001 +
 0100010101101010
------------------
 1001100110100011
```

که فلگ‌ها بصورت زیر تغییر می‌یابند.

تمام بیت‌های نتیجه 0 نیستند.	ZF=0
carry در بیت شماره 15 وجود ندارد.	CF=0
MSB نتیجه برابر با یک می‌باشد.	SF=1
در بیت شماره 14 carry وجود دارد ولی در بیت شماره 15 وجود ندارد.	OF=1
در بیت شماره 3 یک carry وجود دارد.	AF=1
تعداد بیت‌های یک در بایت مرتبه پائین نتیجه زوج است.	PF=1

نکته‌ای که بایستی توجه داشت این است که تمام دستورالعملها محتوی فلگ‌ها را تغییر نمی‌دهند. بعنوان مثال دستورالعمل MOV که برای انتقال داده‌ها استفاده می‌شود روی هیچ فلگی اثر ندارد.

۲-۱-۲- ثبات IP

ثبات IP یک ثبات شانزده‌بیتی می‌باشد که آدرس دستورالعمل بعدی که بایستی اجرا گردد در این ثبات قرار داده میشود. ریزپردازنده با استفاده از این آدرس دستورالعمل بعدی را مورد پردازش قرار می‌دهد. برنامه نویس به این ثبات دسترسی نداشته و محتوی آنرا نمی‌تواند تغییر دهد.

۳-۱-۲- صف دستورالعمل (Instruction Queue)

در ریزپردازنده 80286 یک صف دستورالعمل وجود دارد که طول آن 6 بایت می‌باشد و از این صف برای قراردادن تعدادی دستورالعمل که از حافظه به ریزپردازنده منتقل می‌شود استفاده می‌گردد. اینکار سرعت اجرای دستورالعملها را بیشتر می‌نماید. ریزپردازنده از طریق BUS به حافظه متصل می‌گردد. BUS در هر لحظه شانزده‌بیت را از حافظه به ریزپردازنده و یا برعکس منتقل می‌نماید. داده‌ها، داده‌های کنترلی وآدرس می‌توانند توسط BUS بین حافظه و ریزپردازنده جابه‌جا گردند.

همانطوریکه میدانید بایت‌های حافظه بصورت منحصر بفرد شماره‌گذاری شده و بعنوان آدرس مورد استفاده کامپیوتر قرار می‌گیرد. بایستی توجه داشت که انتقال داده‌ها فقط بصورتهای ذیل میسر می‌باشد.

۱-انتقال از یک ثبات به محلی در حافظه.

۲-انتقال از محلی از حافظه به یک ثبات.

۳-انتقال یک مقدار ثابت به یک ثبات.

٤-انتقال یک مقدار ثابت به یک محل در حافظه.

٥-انتقال از یک ثبات به ثبات دیگر.

انتقال داده‌ها بوسیلهٔ دستور MOV انجام می‌شود که در فصل‌های بعدی تشریح خواهد شد.

مروری بر مطالب فصل

ریزپردازنده 80286 دارای ویژگیهای خاصی می‌باشد. این ریزپردازنده دارای هشت ثبات شانزده‌بیتی بنامها AX، BX ، CX، DX، SP ، BP، SI، DI می‌باشد. چهار ثبات AX ، BX، CX، DX هر کدام بعنوان دوثبات هشت بیتی می‌توانند مورد استفاده قرار گیرند بنامهای AL ، AH، BL ، BH، CL، CH، DL، DH. ثبات فلگ نیز شانزده بیتی بوده که هر بیت مشخص کننده وضعیت خاصی می‌باشد که توسط دستورالعملها تغییر می‌کنند. برخی از این فلگ‌ها عبارتند از OF، PF، CF، AF، ZF، DF که در برنامه‌ها کاربرد زیادی دارند. انتقال اطلاعات از حافظه به ثبات، از ثبات به حافظه، مقدار ثابت به حافظه، مقدار ثابت به ثبات، ثبات به ثبات امکان پذیر می‌باشد. ولی امکان انتقال اطلاعات بین قسمتی از حافظه و قسمت دیگری از حافظه نمی‌باشد.

⌘ تمرین

۱-تعداد ثباتهای هشت بیتی را مشخص نمائید.

۲-کار Flag Register را شرح دهید.

۳-مکانیزمهای انتقال داده‌ها را بنویسید.

٤-دو مقدار A2FB و CD1A را در مبنای شانزده با هم جمع نموده سپس مقدار فلگ‌هایAF، OF، SF، ZF، PF، F و را مشخص نمائید.

٥-دو مقدار C2BF و A5B2 را در مبنای شانزده با هم جمع نموده سپس مقدار فلگ‌های OF، SF، PF، ZF، CF و AF را مشخص نمائید.

٦-مورد استفاده ثباتهایCS، DS، ES ، SS را بنویسید.

۷-صف دستورالعمل و مورد استفاده آنرا بنویسید.

۸-دو مقدار 54C2 و 3271 در سیستم مبنای شانزده را از هم کم نموده سپس مقدار فلگ‌هایAF ، OF ، SF ، ZF ، PF ، CF را مشخص نمائید.

۹- آیا تمام دستور العمل‌ها روی فلگ‌ها اثردارند؟

۱۰- کار BUS را بنویسید.

فصل سوم
برنامه‌نویسی

هدف کلی

آشنائی با شکل کلی دستور العملها و تکنیکهای انتقال داده .

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

۱-برنامه و شکل کلی دستورالعملها.

۲-نامگذاری فیلدها.

۳-انواع فیلدها و کاربرد آنها.

٤-تکنیکهای آدرس دهی.

۳-۱- برنامه و دستورالعملها

در زبان اسمبلی برنامه تشکیل شده است از تعدادی دستورالعملهای اجرائی که بیانگر عملیاتی است که بایستی انجام شود. این سری دستورالعملها همانطوریکه میدانیم SOURCE CODE یا کد منبع نامیده می‌شود. مانند هر زبان برنامه‌نویسی دیگر زبان اسمبلی شکل و قالب از پیش تعریف شده‌ای برای کد منبع دارد. هر دستورالمل اسمبلی شامل چهار فیلد می‌باشد.

فیلد ملاحظات فیلد عملوند فیلدعملیات فیلداسم

البته بایستی توجه داشت که در بعضی از دستورالعملها از تمام فیلدها استفاده نمی‌گردد.

۳-۲- قانون نامگذاری

نام در زبان اسمبلی حداکثر می‌تواند شامل 31 کاراکتر باشد. کاراکترها شامل حروف Z تا A و ارقام 9 تا 0 و سیمبلهای مخصوص @ ؟ . $ _ می‌باشد.

موارد ذیل بایستی در نامگذاری رعایت گردد.

۱-اسم نمی‌تواند با یک رقم شروع گردد.

۲-اسم نبایستی یکی از کلمات ذخیره شده در اسمبلی باشد.

۳-در صورتیکه از ۰ در نام استفاده گردد، بایستی اولین کاراکتر نام باشد. کلمات زیر اسامی مجاز در اسمبلی می‌باشند.

LOOP1	B@A2
X	. XY2
Y2A	SUM2
A_5B	ADDX
COUNT	

کلمات زیر مجاز نمی‌باشند.

LOOP	NEAR
LABEL	ADD
2AB	(5AX
FAR	A2.B

۳-۳- متغیرها (Variables)

نام متغیر مشخص کننده محلی از حافظه می‌باشد که بوسیله برنامه قابل دسترسی می‌باشد و محتوی آنرا در حین اجرای برنامه می‌توان تغییر داد. تعریف متغیر شامل آدرس، نوع داده و اندازه آن می‌باشد. از متغیرها می‌توان بعنوان عملوند در دستورالعملها استفاده نمود. برای متغیرها از نوع بایت از DB، متغیرهای از نوع word از DW و متغیرهای از نوع double word از DD استفاده می‌گردد.

٤-۳- برچسب‌ها (Labels)

از برچسب‌ها بعنوان آدرس دستورالعمل در برنامه‌های کاربردی استفاده می‌شود. از برچسب‌ها به دو صورت استفاده می‌گردد. اگر برچسب در همان سگمنت کد باشد نوع آن NEAR در غیر اینصورت از نوع FAR می‌باشد. در صورتیکه نوع آن NEAR باشد می‌توان بعد از نام برچسب از : استفاده نمود و دیگر نیازی به کلمه NEAR نمی‌باشد.

مثال ۳-۱

```
LOOP1:
```

یا

```
LOOP1    LABEL    NEAR
```

در صورتیکه برچسب از نوع FAR باشد استفاده از کلمه FAR الزامی می‌باشد.

```
MYCODE    LABEL    FAR
```

۵-۳- ثابت‌ها (Constants)

ثابت‌ها مقادیری هستند که در دستورالعملهای برنامه‌ها مورد استفاده قرار می‌گیرند. ثابت‌ها از انواع ذیل می‌باشند.

۱- **Binary** : شامل یک سری 0 و 1 می‌باشد که در انتهای آنها حرف B قرار داده می‌شود.

مثال ۲-۳

110111 B
1000 B

۲- **Decimal** : شامل ارقام 0 تا 9 می‌باشد و بطور اختیاری می‌توان حرف D را به آخر آن اضافه نمود.

40
یا
40D

۳- **Hexadecimal** : شامل ارقام 0 تا 9 و حروف A تا F می‌باشد که در انتهای آنها حرف H اضافه می‌گردد.

مثال ٤-۳

32H
0FFH

اگر مقداری در سیستم مبنای شانزده با یکی از حروف A تا F شروع گردد بایستی 0 به ابتدای آن اضافه گردد. در این صورت کامپیوتر آنرا با نام یک برچسب یا متغیر اشتباه نمی‌گیرد.

٤- **Octal**: شامل ارقام 0 تا 7 می‌باشد کـه در انتهـای آنهـا حـرف O قـرار می‌گیرد. می‌توان به جای O از حرف Q نیز استفاده نمود.

مثال ۵-۳

```
6O
24O
12Q
```

۵- **Character** : ثابت‌های کاراکتری شامل هر کاراکتر از کـدهای ASCII می‌باشد که بین علامت نقل قول ′ یا ″ قرار می‌گیرند.

مثال ۵-۳

```
'B'
"JOHN"
'BOB'
```

٦- **Floating point** : این نـوع data نمـایش مقـادیر اعـشاری بـصورت نمائی می‌باشد.

مثال ٦-۳

```
SINE    DD    0.332E-1
```

که بوسیله اکثر کامپیوترها حمایت نمی‌گردد.

۳-۶- فیلد عملیات

در فیلد عملیات نام دستورالعمل واقعی ریزپردازنده یا عملی که بایستی انجام شود ذکر می‌گردد. نام دستورالعمل بین 2 تا 6 کاراکتر می‌باشد.

مثال ۳-۸

MOV	REP
CMP	LOOP
REPNE	
LEA	

۳-۷- فیلد عملوند

این فیلد شامل آدرس dataهائی که بایستی بوسیله فیلد عملیات پردازش گردد می‌باشد. فیلد عملوند با حداقل یک فاصله از فیلد عملیات جدا میشود. بعضی از دستورالعملها فاقد عملوند می‌باشند. سایر دستورالعملها یک یا دو عملوند دارند که با کاما از هم جدا می‌شوند. مانند

CBW		عملوند ندارد
NOP		عملوند ندارد
CLC		عملوند ندارد
NOT	AL	یک عملوند دارد
MOV	AX , Y	دو عملوند دارد

در مواردی که فیلد عملوند دارای دو عملوند می‌باشد عملوند اول را عملوند مقصد و عملوند دوم را عملوند مبداء می‌نامند.

مثال ۱۰-۳

```
AND   AX , X
```

که AX را عملوند مقصد و X را عملوند مبداء می‌نامند.

۸-۳- فیلد ملاحظات (Comment)

این فیلد آخرین فیلد دستورالعمل می‌باشد که شامل توضیحات در مورد دستورالعمل یا برنامه می‌باشد. این فیلد از سایر فیلدها توسط ; جدا می‌گردد.

مثال ۱۱-۳

```
MOV   AH, 45H; Parameter for reading a character
```

دستورالعملها می‌توانند فقط شامل فیلد Comment باشند. در اینصورت دستورالعمل با ; شروع می‌شود.

مثال ۱۲-۳

```
; This is an assembly Program
; For calculating the n factorial.
```

۹-۳- تکنیکهای آدرس‌دهی

ریزپردازنده 80286 از هفت روش آدرس‌دهی استفاده می‌نماید که عبارتند از

۱- آدرس‌دهی بدون واسطه

۲- آدرس‌دهی ثبات

۳- آدرس‌دهی مستقیم

٤- آدرس‌دهی غیرمستقیم ثبات

۵- آدرس‌دهی مبنا

٦- آدرس‌دهی اندیس مستقیم

۷- آدرس‌دهی اندیس مبنا

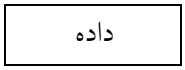
از این امکانات متنوع آدرس‌دهی برای عملوندها استفاده می‌شود.

۱-۹-۳- آدرس دهی بدون واسطه

در این مد آدرس‌دهی داده می‌تواند 8 بیت یا 16 بیت طول داشته باشد و بعنوان عملوند در دستورالعمل استفاده می‌گردد.

مثال ۱۳-۳

MOV BL , 10

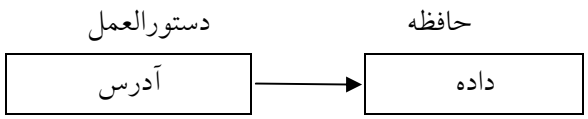


۲-۹-۳- آدرس‌دهی مستقیم

در این روش آدرس داده که شانزده‌بیت می‌باشد جزء دستورالعمل می‌باشد.

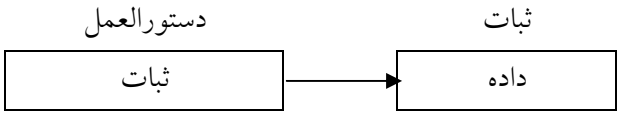
مثال

MOV AX,TABLE



۳-۹-۳- آدرس‌دهی ثبات

در این مد آدرس‌دهی داده در ثباتی قرار دارد که بوسیله دستورالعمل مشخص می‌شود.



بـرای عملونـد شـانزده‌بیتـی از ثباتهـای AX، BX، CX، DX، BP، SI، DI استفاده می‌گردد.

مثال ۱۵-۳

MOV AX , CX

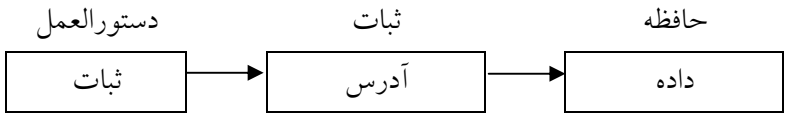
و در مورد عملوند هشت بیتی از ثباتهـای AH ، AL، BL، BH، CL، CH، DH ، DL استفاده می‌گردد.

مثال ۱۶-۳

MOV DL , AL

۴-۹-۳- آدرس‌دهی غیرمستقیم ثبات

در این روش آدرس داده در یکی از ثباتهایBX ، DI، SI قرار داده می‌شود.



MOV AX , [BX]

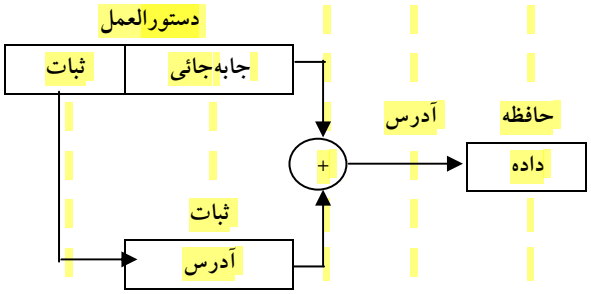
۵-۹-۳- آدرس‌دهی مبنا

در این روش آدرس داده در یکی از ثباتهـای BX، BP، SI ، DI قـرار داده می‌شود. در این روش یک جابه‌جائی باندازه 8 بیت یا 16 بیت دارد.

```
MOV    AX,[BX]+4
```

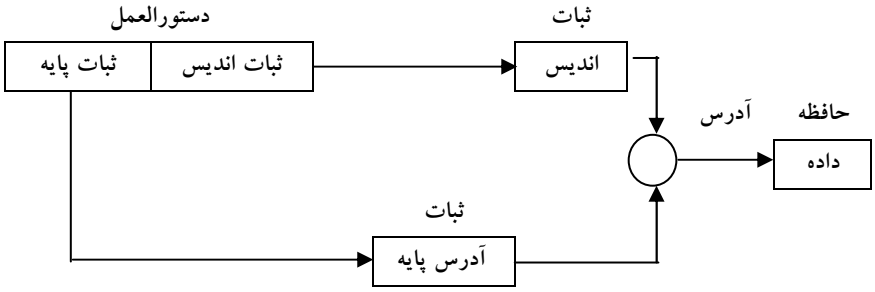
که 4، مقدار جابه‌جائی و آدرس داده در BX قرار داده شده است. دو دستور ذیل معادل دستور فوق می‌باشد.

```
MOV    AX, 4[BX]
MOV    AX, [BX+4]
```



٦-٩-٣- آدرس‌دهی اندیس مستقیم

دراین روش آدرس داده در یکی از ثباتهای BX و یا BP قرار داده می‌شود. و از ثباتهای SI و یا DI بعنوان اندیس استفاده می‌گردد. در حقیقت آدرس عبارتست از مجموع BX و یا BP با SI یا DI .



مثال ۱۷-۳

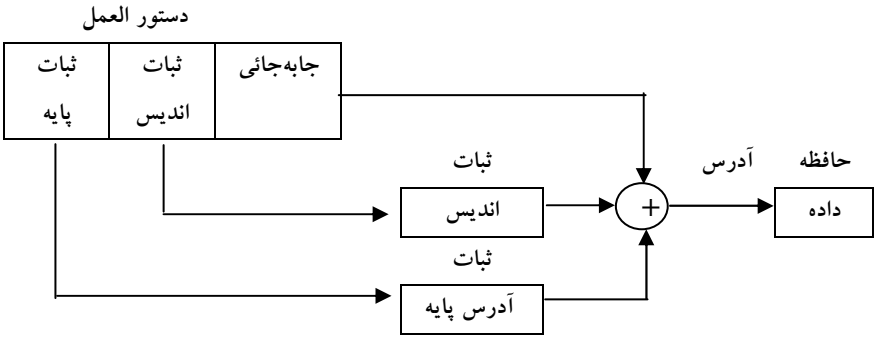
```
MOV   AX,[BX][DI]
```

۷-۹-۳- آدرس‌دهی اندیس مبنا

در این روش آدرس داده شبیه قبل بوده با این تفاوت که یک جـا بـه جـائی هشت بیتی یا شانزده‌بیتی نیز وجود دارد.

مثال ۱۸-۳

```
MOV   AX, VALUE [BX][DI]
MOV   AX, [BX+2][DI]
MOV   AX,[BX] [DI+2]
```



مروری بر مطالب فصل

هـر دسـتورالعمل مـی‌توانـد شـامل چهـار فیلـد اسـم، عملیـات، عملونـد و ملاحظات باشد. نام مشخـصه‌هـا در زبـان اسـمبلی شـامل 31 کـاراکتر مـی‌باشـد و می‌بایستی با یکی از حروف شروع شد و نمی‌توانـد یکـی از کلمـات ذخیـره شـده بوسیله زبان اسـمبلی باشـد. ثابتهـا را مـی‌تـوان در مبنـای 2، 8، 10، 16 در برنامـه استفاده نمود که بترتیب برای آنها از پـسوندهای B، O، D، H اسـتفاده مـی‌نمـائیم. دستورالعملها می‌توانند بدون عملوند، یک عملوند، یـا دو عملونـد مـی‌باشـند. فیلـد ملاحظات با ; شروع مـی‌شـود. ریزپردازنـده 80286 از هفـت تکنیـک آدرس‌دهـی استفاده می‌نماید که بترتیب عبارتند از :

۱- آدرس دهی بدون واسطه

۲- آدرس‌دهی ثبات

۳- آدرس‌دهی مستقیم

٤- آدرس‌دهی غیرمستقیم ثبات

۵- آدرس‌دهی مبنا

٦- آدرس‌دهی اندیس مستقیم

۷- آدرس‌دهی اندیس مبنا

یک دستور العمل می‌تواند با ; شروع شده که جنبـه ملاحظـات و توضـیحات بـرای برنامه دارد.

⌘ تمرین

۱-کدامیک از کلمات زیر نام مجاز در اسمبلی می‌باشد؟

BOOK	VARIABLE	$ 5AC
2 NAME	FOR	. 2_A7
+ PLUS	NAME3	BOOK2

۲-فیلدهای یک دستورالعمل را نام ببرید.

۳-انواع ثابت‌ها را بنویسید و در مورد هر یک مثالی بدهید.

٤-تعداد حداقل و حداکثر عملوندها در یک دستورالعمل را بدهید.

۵-مدهای آدرس‌دهی را نام ببرید. و در مورد هر کدام مثالی بدهید.

٦- قانون نامگذاری مشخصه‌ها را بیان نمائید.

۷- در مورد ثابت‌ها در مبنای 16 از چه پسوندی استفاده می‌گردد.

۸- آیا استفاده از پسوند در مورد ثابتهای مبنای 10 اختیاری است؟

فصل چهارم

دستورالعملهای اساسی

هدف کلی

معرفی دستورالعملهای اساسی در زبان اسمبلی.

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

۱-انتقال داده‌ها در حافظه

۲-دستورالعملهای جمع و تفریق

۳-دستورالعملهای جمع و تفریق مقادیر بزرگ

٤- ضرب و تقسیم

۵- دستورالعملهای ضرب مقادیر بزرگ

٦- دستورالعملهای کاهش و افزایش

۷- دستورالعمل محاسبه مکمل 2 یک مقدار.

١-٤- انتقال داده‌ها در حافظه

انتقال داده‌ها بین مکانهای مختلف حافظه اصلی و ثباتها بوسیله دستورالعمل MOV انجام می‌شود. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

```
MOV     dst , src
```

این دستورالعمل محتوی src را در dst قرار داده و محتوی src بدون تغییر باقی می‌ماند.

```
MOV     CL, -30
```

-30 را در ثبات CL قرار می‌دهد.

```
MOV     X, 25H
```

مقدار 37 را در مکان X در حافظه قرار می‌دهد.

```
MOV     AX, BX
```

محتوی ثبات BX را در AX قرار می‌دهد و محتوی ثبات BX بدون تغییر باقی می‌ماند.

```
MOV     DS, AX
```

محتوی ثبات AX را در DS قرار می‌دهد.

```
MOV     AX, TABLE
```

محتوی حافظه TABLE را در ثبات AX قرار می‌دهد.

```
MOV     TABLE, AX
```

محتوی ثبات AX را در مکان TABLE از حافظه اصلی قرار می‌دهد.

درمورد دستورالعمل MOV بایستی در نظر داشت که :

۱–هر دو عملوند یعنی sdt و src بایستی از نوع بایت یا هر دو از نوع word باشند.

۲–هر دو عملوند نمی‌توانند متغیر باشند. یعنی دستورالعمل زیر غلط می‌باشد.

```
MOV    X, Y
```

۳–هیچکدام از عملوندها نمی‌توانند ثبات IP باشند.

٤–هیچکدام از عملوندها نمی‌توانند ثبات فلگ باشند.

۵–محتوی یک ثبات سگمنت را نمی‌توان مستقیماً بیک ثبات سگمنت دیگر منتقل نمود. این کار را بایستی بصورت غیر مستقیم انجام داد.

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

٦–عملوند dst نمی‌تواند ثبات CS باشد.

۷–در دستورالعمل MOV بجزء در مواردیکه src ثابت باشد حتماًیکی از عملوندها بایستی ثبات باشد.

۸–یک ثابت را نمی‌توان مستقیماً به یک ثبات سگمنت منتقل نمود.

این کار را بایستی بصورت زیر انجام داد.

```
MOV    AX, DATA_SEG
MOV    DS , AX
```

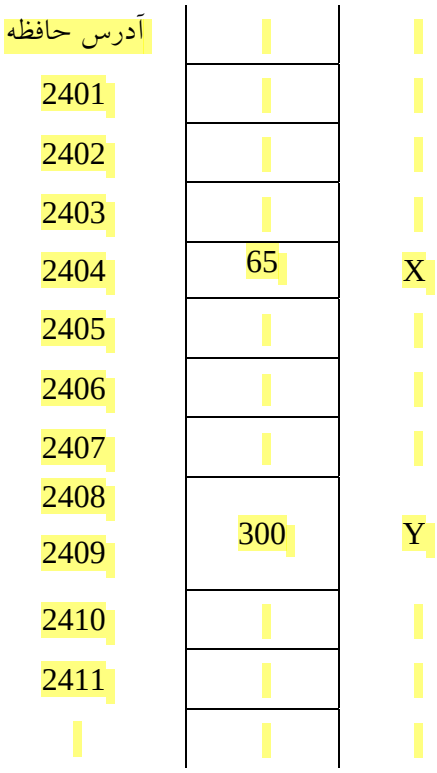
۹–دستورالعمل MOV روی هیچ فلگی اثر ندارد.

همانطور که میدانید هر مکان از حافظه یا متغیر دارای مشخصات زیر می‌باشد.

۱–نام، که از قانون نامگذاری تبعیت می‌نماید.

۲–آدرس، که نشان دهنده مکان متغیر در حافظه می‌باشد.

۳–مقدار، که محتوی آن مکان را نشان می‌دهد.



در بالا حافظه اصلی را نشان می‌دهد که X متغیری از نوع بایت بـا محتـوی 65 و آدرس آن 2404 می‌باشد. همچنین Y متغیری است از نوع word با محتـوی 300 که آدرس آن 2408 می‌باشد. بایستی توجه داشت که آدرس هـر data آدرس بایت شروع آن data می‌باشد. متغیر y چون بایتهـای 2408 و 2409 را در حافظـه اشغال می‌کند آدرس آن 2408 می‌باشد. حال چنانچه بخواهیم آدرس متغیـری را در ثبات BX قرار دهیم از دستورالعمل زیر استفاده می‌گردد.

```
MOV    BX , OFFSET    Y
```

با اجرای این دستورالعمل آدرس 2408 در ثبات BX قرار می‌گیرد.

BX	Y
2408	300

با اجرای دستورالعمل

```
MOV   BX, Y
```

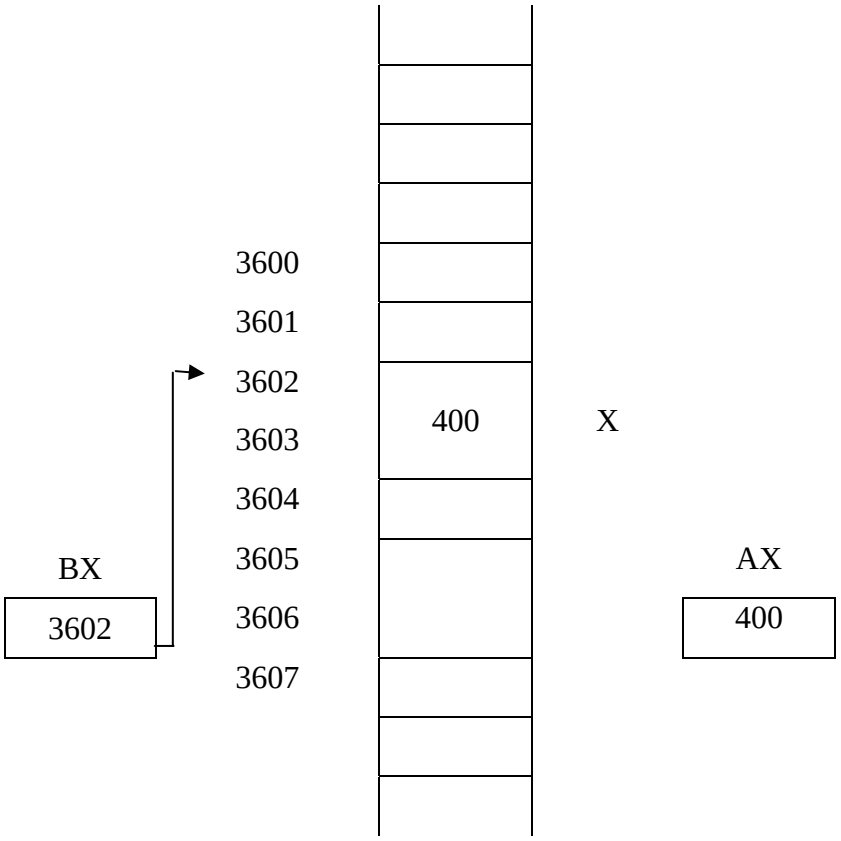
محتوی Y یعنی 300 در ثبات BX قرار داده می‌شود.

BX	Y
300	300

معمولاً آدرس متغیرها را در یکی از ثباتهای SI, DI, BP, BX قرار داده می‌شود. حال سه دستورالعمل زیر را در نظر بگیرید.

```
X     DW  400
MOV   BX, OFFSET   X
MOV   AX, [BX]
```

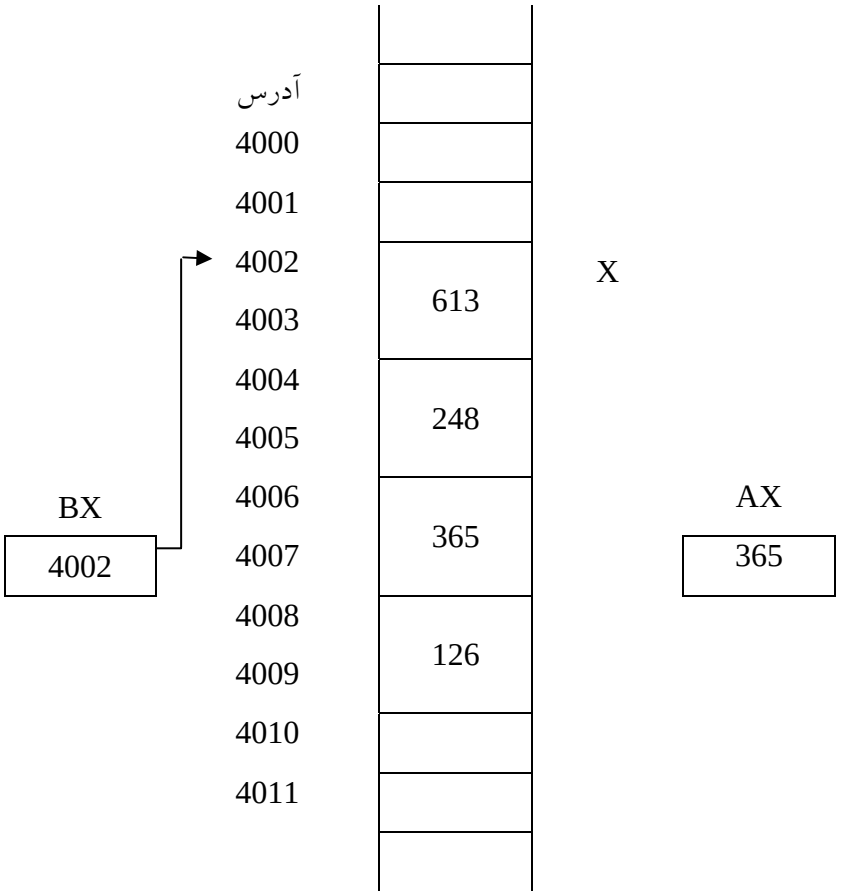
همانطور که در شکل زیر نشان داده شده است. X مکانی از حافظه است که یک word را اشغال نموده است. دستورالعمل دوم آدرس متغیر X را در ثبات BX قرار می‌دهد. دستورالعمل آخر محتوی مکانی از حافظه که بوسیله BX اشاره می‌شود را به ثبات AX منتقل می‌نماید.



مثال ١-٤

دستورالعملهای ذیل را در نظر بگیرید.

```
X   DW 613,248,365,126
MOV BX, OFFSET  X
MOV AX, [BX] +4
```



اولین دستورالعمل ایجاد یک آرایه چهار عنصری از نوع word می‌نماید با مقادیر 613 ، 248 ، 365 ، 126 بترتیب در آدرسهای4002، 4004، 4006، 4008. دستورالعمل دوم آدرس متغیر X را در ثبات BX قرار می‌دهد. آخرین دستورالعمل 4 واحد به آدرس درون BX اضافه نموده به آدرس 4006 می‌رسد، سپس مقداری که در آدرس 406 حافظه قرار دارد یعنی 365 به ثبات AX منتقل می‌گردد. بایستی توجه داشت که محتوی ثبات BX پس از اجرای دستورالعمل‌های فوق 4002 می‌باشد.

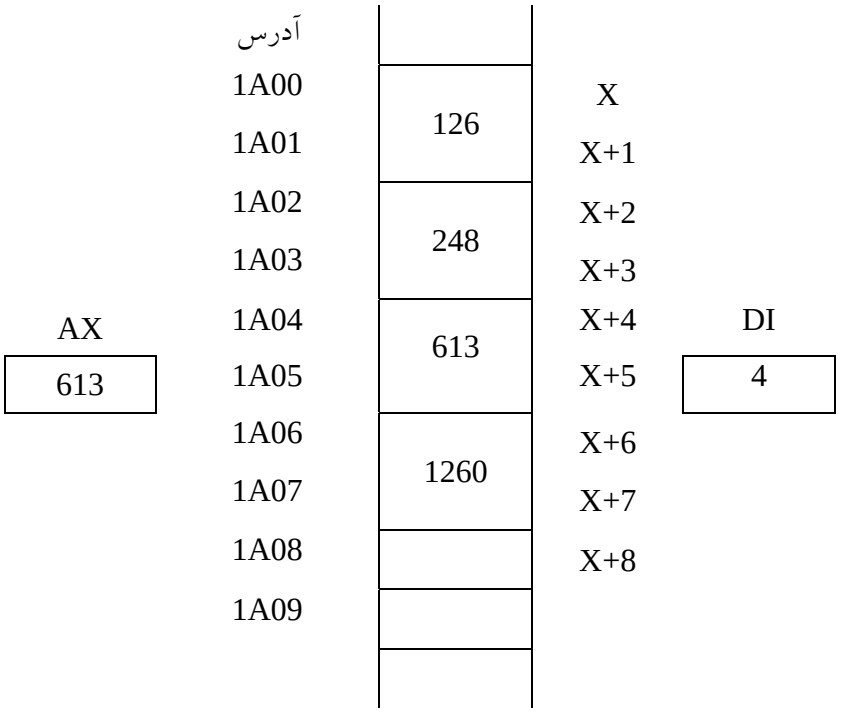
بایستی توجه داشت که سه دستورالعمل ذیل معادلند

```
MOV   AX, [BX]+4
MOV   AX, 4[BX]
MOV   AX, [BX+4]
```

مثال ۲-٤

دستورالعمل‌های ذیل را در نظر بگیرید.

```
X     DW   126,248,613,1260
MOV   DI, 4
MOV   AX, X[DI]
```



دستورالعمل اول ایجاد یک آرایه می‌نماید بنام X از نوع word با مقادیر 126، 248،613،1260بترتیب در آدرس‌های1A00، 1A02، 1A04، 1A06 از

حافظه. دستورالعمل دوم مقدار 4 را در ثبات DI قرار می‌دهد. آخرین دستورالعمل محتوی X+4 یعنی مقدار 613 را در ثبات AX قرار می‌دهد.

مثال ۳-٤

دستورالعمل ذیل را در نظر بگیرید.

```
MOV   AX, X[BX][DI]
```

این دستورالعمل محتوی آدرسی که از مجموع مقادیر X ، BX ، DI بدست می‌آید را در ثبات AX قرار می‌دهد. در این دستور می‌توان بجای BX از BP و به جای DI از SI استفاده نمود.

مثال ٤-٤

```
MOV   AX, X[BP][DI]
MOV   AX, X[BX][SI]
MOV   AX, X[BP][SI]
```

نهایتاً دستورالعمل MOV نیز می‌تواند به یکی از شکل ذیل باشد.

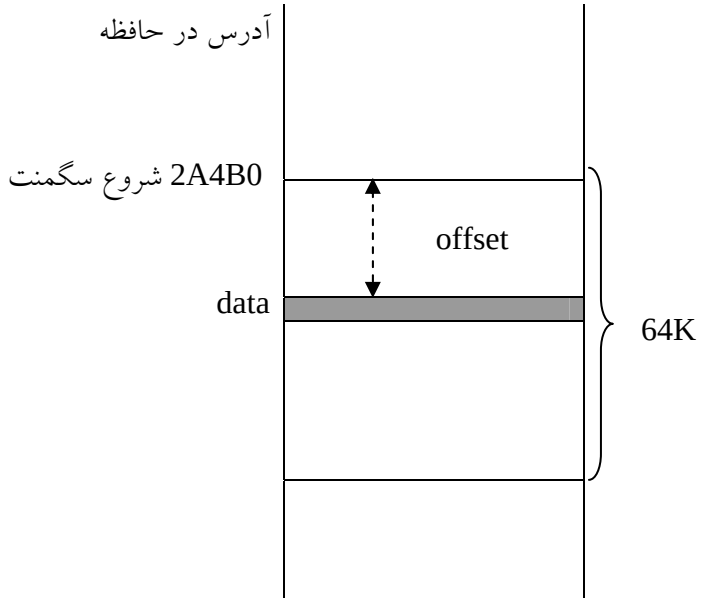
```
MOV   AX, [BX][DI+2]
MOV   AX, [BX+2][DI]
MOV   AX,[BX][DI+2]
MOV   AX,[BX+DI+2]
```

از این فرم‌های MOV برای استخراج داده‌ها از یک آرایه دو بعدی استفاده می‌گردد.

همانطور که قبلاً متذکر شدیم در برنامه از چهار سگمنت بنامهای زیر می‌توان استفاده نمود.

```
code   segment   CS
data   segment   DS
stack  segment   SS
extra  segment   ES
```

کامپیوتر آدرس شروع این سگمنت‌ها را طوری انتخاب می‌نماید که قابل تقسیم بر 16 باشند یعنی چهار بیت سمت راست آدرس شروع سگمنت‌ها 0000 باشد. در حقیقت آدرس شروع سگمنت‌ها بیست بیتی می‌باشد که چون چهار بیت سمت راست آنها 0000 می‌باشد به شانزده بیت تقلیل یافته و در ثباتهای CS، DS، SS، ES قرار داده میشوند. برا یافتن آدرس واقعی یا فیزیکی سگمنت data در حافظه بایستی محتوی DS را در 16 ضرب نمود. همانطوریکه نیز قبلاً بیان شد اندازه سگمنت‌ها تا 64K بایت می‌تواند باشد. آدرس هر data در حافظه چون این data در data segment قرار دارد می‌توان نسبت به ابتدای آن segment مشخص نمود که به این آدرس offset گفته میشود.



در حقیقت آدرس سگمنت 2A4B در ثبات DS قرار داده میشود. حال برای بدست آوردن آدرس فیزیکی یا واقعی data بطریق ذیل عمل می‌نمائیم.

DS محتوی *16+offset

٢-٤- دستورالعمل LEA

دستورالعمل LEA نیز آدرس متغیر را در یکی از ثباتهای SI، DI، BX، BP می‌دهد. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

LEA dst , src

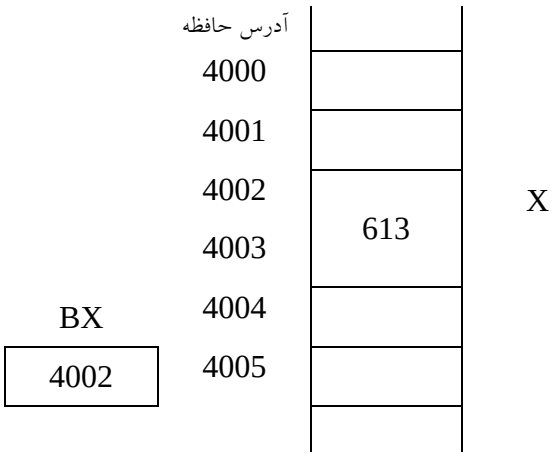
1- src یک متغیر از نوع بایت یا word می‌باشد.

2- dst یکی از چهار ثبات BX ، BP ، SI ، DI می‌باشد.

3- دستورالعمل LEA بر هیچ فلگی اثر ندارد.

مثال ٤-٥

```
X    DW  613
LEA  BX , X
```



در حقیقت عملکرد این دستور معادل دستور ذیل می‌باشد.

MOV BX,OFFSET X

مثال ٤-٦

```
LEA   SI, COL[BX]
```

این دستورالعمل آدرس متغیر COL را با محتوی BX جمع نموده آدرس بدست آمده را در SI قرار می‌دهد.

مثال ٤-٧

```
X     DB    ?
LEA   BX,   X
```

متغیر X از نوع بایت تعریف گردیده و آدرس آن در ثبات BX قرار داده شده است.

بایستی توجه داشت که دستورالعمل LEA بر هیچ فلگی اثر ندارد.

٤-٣- مبادله داده‌ها

برای مبادله محتوی دو آدرس داده از دستورالعمل XCHG استفاده می‌گردد. شکل کلی دستورالعمل بصورت زیر می‌باشد.

```
XCHG   dst, src
```

این دستورالعمل محتوی src با dst را مبادله می‌نماید یعنی محتوی src در dst قرار می‌گیرد و محتوی dstدر src.

در مورد دستورالعمل XCHG بایستی در نظر داشت که:

١- dst , src نمی‌تواند ثابت باشند.

٢- dst , src بایستی هر دو از نوع بایت یا هر دو از نوع word باشند.

٣- dst , src هر دو متغیر نمی‌توانند باشند.

٤- این دستورالعمل بر روی هیچ فلگی اثر ندارد.

مثال ٨-٤

```
MOV   AX, 1000
MOV   X, 3000
XCHG  X, AX
```

قبل از اجرای دستورالعمل XCHG

AX	X
1000	3000

بعد از اجرای دستورالعمل XCHG

AX	X
3000	1000

مثال ٩-٤

```
XCHG  AX , BX
```

محتوی BX را در AX و محتوی AX را در X قرار می‌دهد.

```
X  DB  65
Y  DB  48
MOV    AL , X
XCHG   AL, Y
MOV    X , AL
```

دستورالعملهای فوق باعث مبادله مقادیر X و Y که هر دو از نوع بایت می‌باشند می‌شود. قبل از اجرای دستورالعمل‌های فوق

X	Y
65	48

پس از اجرای دستورالعملهای فوق

AL	X	Y
48	48	65

بایستی توجه داشت که دستورالعمل XCHG روی هیچ فلگی اثر ندارد.

٤-٤- جمع و تفریق

جمع دو مقدار بوسیله دستورالعمل ADD انجام می‌شود. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

```
ADD     dst , src
```

این دستورالعمل محتوی src را با محتوی dst جمع نموده نتیجه را در dst قرار می‌دهد و مقدار src بدون تغییر می‌ماند.

dst ← dst + src

مثال ٤-١٠

```
MOV   AX , 613
MOV   BX, 248
ADD   AX , BX
```

دستورالعمل اول مقدار 613 را در AX قرار می‌دهد. دستورالعمل دوم مقدار 248 را در BX قرار می‌دهد. دستورالعمل سوم مجموع AX و BX را محاسبه و نتیجه را در AX قرار می‌دهد. که همانطور که ملاحظه می‌شود مقدار BX تغییر نمی‌کند.

قبل از اجرای دستورالعمل فوق

AX	BX
613	248

بعد از اجرای دستورالعملهای فوق مقادیر عبارتند از:

AX	BX
861	248

در مورد دستورالعمل ADD بایستی موارد زیر را در نظر داشت.

١- هر دو عملوند بایستی از نوع بایت یا هر دو از نوع word باشند.

٢- هر دو عملوند نمی توانند از نوع متغیر باشند.

۳- به جزء در مواردیکه یکی از عملوندها ثابت باشد حتماً یکی از عملوندها بایستی از نوع ثبات باشد.

٤- دستورالعمل ADD بر روی فلگ‌های AF, CF, OF, PF, SF, ZF اثر دارد.

مثال ١١-٤

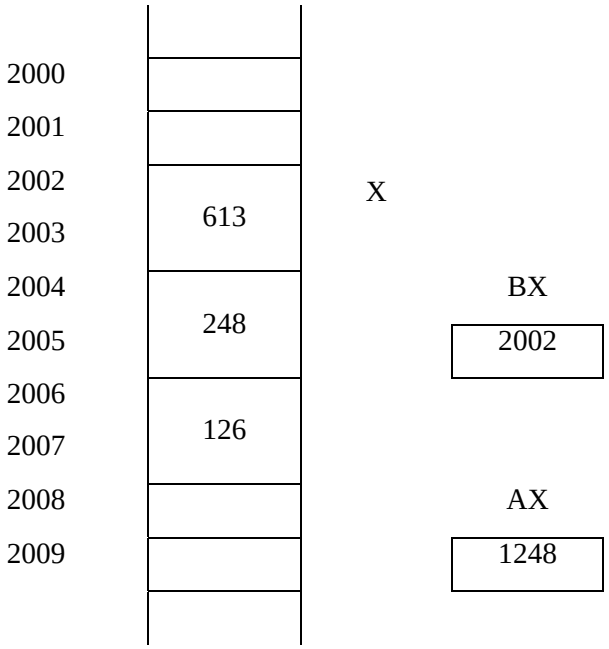
```
X       DB   13
MOV     AL , 10
ADD     X , AL
```

دستورالعمل اول X را از نوع بایت تعریف نموده و مقدار آنرا 13 قرار می‌دهد. دستورالعمل دوم مقدار 10 را در ثبات AL قرار می‌دهد. دستورالعمل سوم مقدار AL را با محتوی X جمع نموده نتیجه که می‌شود 23 را در X قرار می‌دهد. و محتوی AL نیز 10 می‌باشد.

مثال ١٢-٤

```
X       DW  613,248,126
MOV     BX, OFFSET  X
MOV     AX, 1000
ADD     AX, [BX + 2]
```

اولین دستورالعمل یک آرایه سه عنصری از نوع word ایجاد می‌نماید. مقادیر عناصر آرایه بترتیب عبارتند از613، 248، 128دستورالعمل دوم آدرس X را در BX قرار می‌دهد. دستورالعمل سوم مقدار 1000 را در AX قرار می‌دهد. آخرین دستورالعمل محتوی دومین عنصر آرایه را یعنی 248 با محتوی AX جمع نموده و نتیجه را در AX قرار می‌دهد.



مثال ١٣-٤

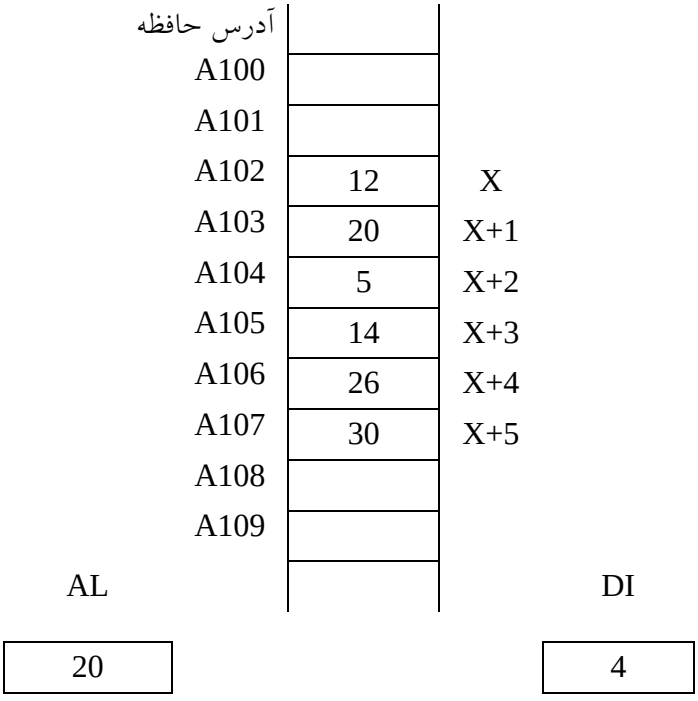
```
X       DB    18
MOV     AL  ,  -18
ADD     AL  ,  X
```

دستورالعمل اول مقدار 18 را در X قرار می‌دهد. دستورالعمل دوم مقدار 18- را در AL قرار می‌دهد. آخرین دستورالعمل محتوی X را با محتوی AL جمع نموده، نتیجه را که 0 می‌شود در AL قرار می‌دهد. بایستی توجه داشت که پس از انجام عمل جمع مقدار فلگ ZF برابر با یک می‌شود.

مثال ١٤-٤

```
X       DB  12,20,5,14,26,30
MOV     DI , 5
MOV     AL , 20
ADD     X [DI] , AL
```

دستورالعمل اول یک آرایه 6 عنصری از نوع بایت با مقادیر بترتیب 12، 20، 5، 14، 26، 30 تعریف می‌نماید. دستورالعمل دوم مقدار 5 را در DI قرار می‌دهد. دستورالعمل سوم مقدار 20 را در AL قرار می‌دهد. آخرین دستورالعمل محتوی مکانی از حافظه که بوسیله X+5 مشخص می‌شود را با AL جمع می‌نماید. یعنی مقدار 30 را با 20 جمع نموده و نتیجه را در محل X+5 از حافظه قرار می‌دهد. و مقادیر ثباتهای AL و DI بدون تغییر باقی‌می‌ماند.



پس از اجرای دستورالعملهای فوق شکل حافظه بصورت زیر در می‌آید.

A100		
A101		
A102	12	X
A103	20	X+1
A104	5	X+2
A105	14	X+3
A106	26	X+4
A107	50	X+5
A108		
A109		

از دستورالعمل ADC نیز برای جمع مقادیر می‌توان استفاده نمود. فرم کلی این دستورالعمل بصورت زیر است.

ADC dst , src

که محتوی src را با محتوی dst جمع نموده نتیجه را با مقدار فلگ CF جمع نموده و نهایتاً نتیجه بدست آمده را در dst قرار می‌دهد.

dst ⟵ dst + src + CF

در موقع استفاده از این دستورالعمل بایستی توجه داشت که :

۱- هر دو عملوند src , dst بایستی از نوع بایت یا هر دو از نوع word باشند.

۲- src , dst هر دو متغیر نمی‌توانند باشند.

۳- به جزء در مواردیکه یکی از عملوندهای src ، dst ثابت باشد یکی از عملوندها بایستی حتماً از نوع ثبات باشد.

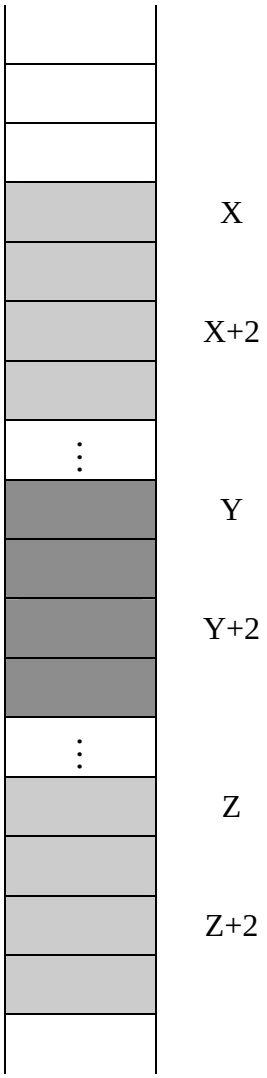
٤- دستورالعمل ADC روی فلگ‌هایZF ، SF، AF، PF، OF، CF اثر دارد.

مثال ١٥-٤

```
X   DW   ?
MOV      AX , 1000
MOV      X, 3000
ADC      AX , X
```

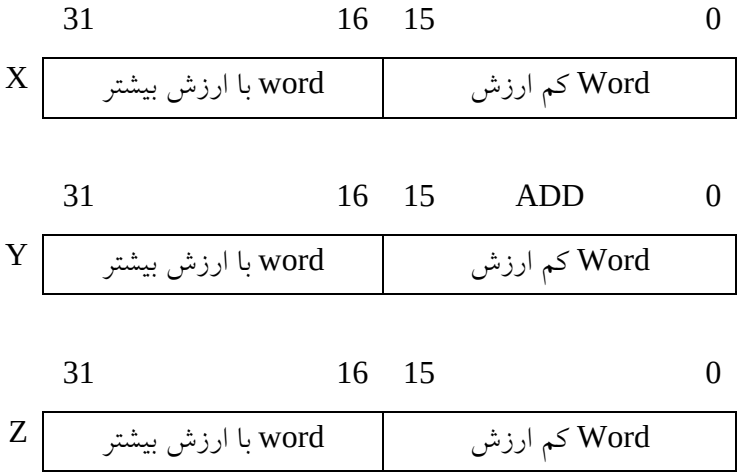
دستورالعمل اول X را از نوع word تعریف می‌نماید. دستورالعمل دوم مقدار 1000 را در ثبات AX قرار می‌دهد. دستورالعمل سوم مقدار 3000 را در متغیر X قرار می‌دهد. آخرین دستورالعمل محتوی X را با محتوی ثبات AX جمع نموده و چنانچه CF=1 باشد یک واحد به جمع بدست آمده اضافه نموده 4001 را در ثبات AX قرار می‌دهد. و چنانچه مقدار CF=0 باشد مقدار 4000 را در ثبات AX قرار می‌دهد.

یکی از کاربردهای مهم دستورالعمل ADC در محاسبه مجموع دو مقدار از نوع double word می‌باشد. فرض کنید می‌خواهیم مجموع دو متغیر Y و X از نوع double word را محاسبه نموده و نتیجه را در متغیر Z از نوع double word قرار دهیم.



همانطور که در شکل ملاحظه می‌گردد، متغیر X از نوع double word را می‌توان بصورت دو تا word بنامهای X و X+2 در نظر گرفت. بطور مشابه همین عمل را در موردY و Z می‌توان انجام داد. برای جمع دو متغیر از نوع double word کافی است که ابتدا دو تا word با ارزش کم‌تر را جمع نموده با

استفاده از دستورالعمل ADD سپس دو تا word با ارزش بیشتر را با هم جمع نموده با استفاده از دستورالعمل ADC.



قطعه برنامه‌ای که دو متغیر X و Y از نوع double word را با هم جمع نموده و نتیجه را در متغیر Z از نوع double word قرار می‌دهد بصورت ذیل می‌باشد:

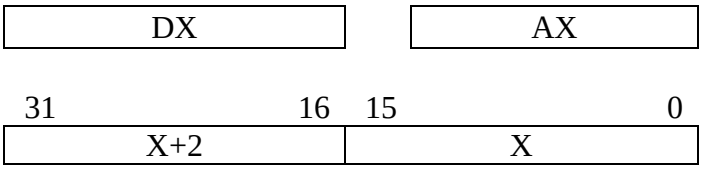
```
X       DD      60000
Y       DD      40000
Z       DD      ?
MOV     AX , X      .
ADD     AX , Y      .
MOV     Z , AX      .
MOV     AX, X+2     .
ADC     AX , Y+2    .
MOV     Z +2, AX    .
```

سه دستورالعمل اول متغیرهای X, Y , Z را از نوع double word تعریف نموده و مقدار X را برابر با 60000 و مقدار Y را برابر با 40000 قرار می‌دهد. سه دستورالعمل بعدی دو تا word با ارزش کم X و Y را با هم جمع نموده نتیجه را

در word با ارزش کم Z قرار می‌دهد. سه دستورالعمل بعدی دو تا word با ارزش بیشترX و Y را با هم جمع نموده با استفاده از ADC و نتیجه را در word با ارزش بیشتر Z قرار می‌دهد. بایستی توجه داشت که در دستورالعملهای MOV ، ADD، ADC هر دو عملوند نمی‌توانند متغیر باشند.

مثال ١٦-٤

قطعه برنامه زیر مقدار X از نوع double word را با مقدار ثباتهای DX و AX جمع نموده نتیجه را در X قرار می‌دهد.



ابتدا محتوی X را با AX جمع نموده با استفاده از دستورالعمل ADD سپس محتوی X+2 را با DX با استفاده از دستورالعمل ADC جمع می‌نمائیم.

```
X     DD  40000
MOV   AX , 300
MOV   DX, 400
ADD   X , AX
ADC   X+2 , DX
```

قطعه برنامه ذیل مجموع متغیرهای X و Y از نوع double word و عدد 24 را محاسبه و نتیجه را در متغیر W قرار می‌دهد. یعنی :

$$W \longleftarrow X + Y + 24$$

برای اینکـار ابتـدا X ، Y ، W را بـصورت double word تعریـف نمـوده سپس مجموع مقادیر X و Y را بدست می‌آوریم. آنگاه عـدد 24 را بـصورت یـک مقدار از نوع double word در نظر گرفته با مجموع قبلی بدست آمده جمع نموده و نهایتاًنتیجه نهائی را در W قرار می‌دهیم.

```
X       DD      ?
Y       DD      ?
Z       DD      ?
MOV     AX , X
MOV     DX,X+2
ADD     AX,Y
ADC     DX,Y+2
ADD     AX,24
ADC     DX ,0
MOV     W, AX
MOV     W+2 , DX
```

برای انجام عمل تفریق از دستورالعمل SUB اسـتفاده مـی‌گـردد. فـرم کلـی دستورالعمل SUB عبارتست از

```
SUB          dst , src
dst  <-----  dst – src
```

مقدار src از dst کم می‌شود نتیجه در dst قرار می‌گیرد. در بکـارگیری ایـن دستورالعمل بایستی موارد زیر را در نظر داشت.

۱- هر دو عملوند بایستی از یک نوع باشند هر دو از نوع بایـت یـا هـر دو از نـوع word باشند.

۲- هر دو عملوند نبایستی از نوع متغییر باشند.

۳- به جـزء در مواردیکـه یکـی از عملونـدها ثابـت باشـد حتماً بایـستی یکـی از عملوندها از نوع ثبات باشد.

٤- دستورالعمل SUB بر روی فلگ‌های ZF ، CF، OF، PF، AF، SF اثر دارد.

٥- محتوی عملوند dst در عمل SUB تغییر نمی‌کند.

مثال ١٧-٤

```
MOV    AL , 10
MOV    BL , 6
SUB    AL , BL
```

دستورالعمل اول مقدار 10 را در AL قرار می‌دهد. دستورالعمل دوم مقدار 6 را در BL قرار می‌دهد و دستورالعمل سوم محتوی BL را از محتوی AL کم نموده نتیجه را در AL قرار می‌دهد و مقدار BL تغییر نمی‌کند.

قبل از اجرای دستورالعمل SUB

AL	BL
10	6

بعد از اجرای دستورالعمل SUB

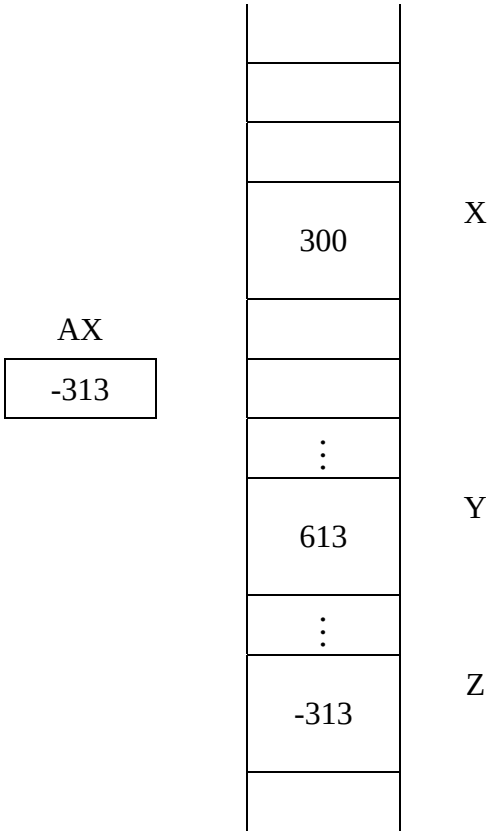
AL	BL
4	6

مثال ١٨-٤

```
X      DW     300
Y      DW     613
Z      DW     ?
MOV    AX, X
SUB    AX , Y
MOV    Z , AX
```

سه دستورالعمل اول سه متغیر X با مقدار 300 و Y با مقدار 613 و Z از نوع word تعریف می‌نماید. دستورالعمل چهارم محتوی متغیر X را به AX منتقل می‌نماید. دستورالعمل بعدی محتوی متغیر Y را از محتوی ثبات AX کم نموده نتیجه را در AX قرار می‌دهد. آخرین دستورالعمل محتوی ثبات AX را به متغیر Z

منتقل می‌نماید. در نتیجه اجرای دستورالعمل SUB مقدار فلگ SF=1 می‌شود، زیرا نتیجه تفریق مقداری منفی است.

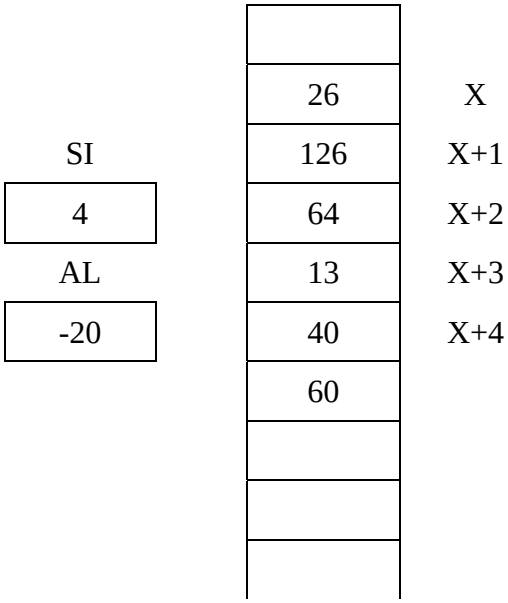


مثال ١٩-٤

```
ARR     DB   26,126,64,13,40,60
MOV     SI, 4
MOV     AL, 20
SUB     AL, ARR [SI]
```

اولین دستورالعمل یک آرایه شش عنصری از نوع بایت با مقادیر 6,126,64,13,40,60 تعریف می‌نماید. دستورالعمل دوم مقدار 4 را در ثبات SI

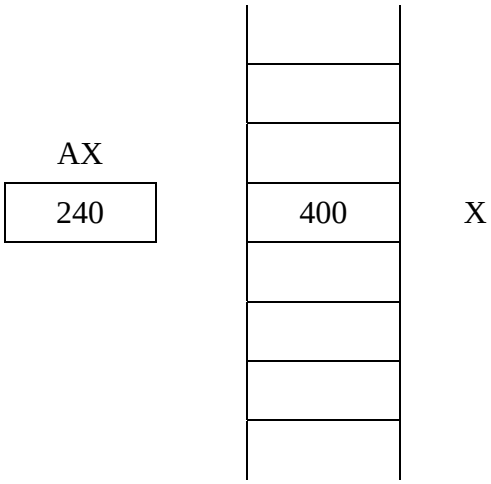
قرار می‌دهد. دستورالعمل سوم مقدار 20 را در ثبات AL قرار می‌دهد. آخرین دستورالعمل، محتوی ARR+4 را از AL کم نموده نتیجه را که می‌شود 20– در AL قرار می‌دهد. و مقدار فلگ SF برابر با یک می‌شود.



مثال ۲۰–٤

```
X       DW      613
MOV     AX , 248
SUB     AX, 48
SUB     X , AX
```

اولین دستورالعمل متغیر X را از نوع word با مقدار 613 تعریف نموده، دستورالعمل دوم مقدار 248 را در AX قرار می‌دهد. دستورالعمل سوم مقدار 48 را از محتوی ثبات AX کم نموده و نتیجه را که می‌شود 200 در AX قرار می‌دهد. آخرین دستورالعمل محتوی AX از محتوی متغیر X کم نموده نتیجه که می‌شود 400 را در متغیر X قرار می‌دهد.



از دستورالعمل SBB نیز برای عمل تفریق استفاده میگردد. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

SBB dst , src

dst ← dst – src – CF

محتوی src از محتوی dst کم می‌شود سپس مقدار CF را از نتیجه کم نموده آنگاه نتیجه را در dst قرار می‌دهد.

مثال ٢١-٤

```
MOV   AX , 1000
SBB   AX , 800
```

دستورالعمل اول مقدار 1000 را در AX قرار می‌دهد. دستورالعمل دوم 800 را از محتوی AX کم نموده چنانچه مقدار فلگ CF برابر با یک باشد نتیجه می‌شود 199 در غیر اینصورت نتیجه می‌شود 200.

مواردیکه بایستی در استفاده از دستورالعمل SBB رعایت گردند عبارتند از:

١- هر دو عملوند بایستی از نوع بایت یا هر دو عملوند از نوع word باشند.

٢- هر دو عملوند نمی‌توانند از نوع متغیر باشند.

۳- بجز در مواردیکه یکی از عملوندها ثابت باشد حتماًبایستی یکی از عملوندها از نوع ثبات باشد.

٤- دستورالعمل SBB روی فلگ‌هایAF، PF، ZF، SF، OF، CF اثر دارد.

مثال ٤-۲۲

با در نظر گرفتن اینکه مقدار فلگ CF برابر با صفر می‌باشد قطعه برنامه زیر را اجرا نمائید.

```
MOV   AL , 100
SBB   AL , 60
```

دستورالعمل اول مقدار 100 را در ثبات AL قرار می‌دهد و چون دستورالعمل MOV روی هیچ فلگی اثر ندارد، مقدار فلگ CF بدون تغییر مقدار صفر می‌ماند. دستورالعمل دوم در اینجا چون مقدار CF برابر با صفر است دقیقاً مانند دستورالعمل SUB عمل نموده مقدار 60 را از محتوی AL کم نموده و نتیجه را که برابر با 40 می‌باشد در AL قرار می‌دهد.

قبل از اجرای دستور SBB

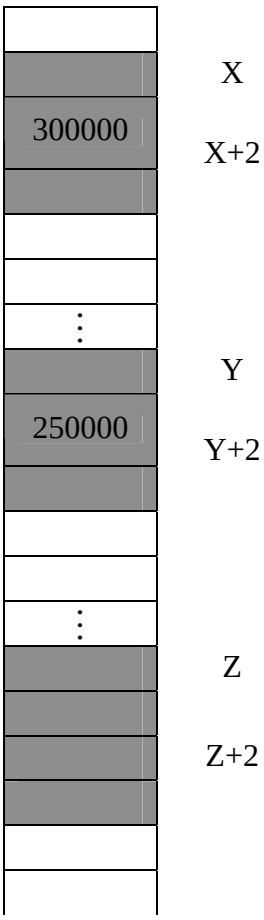
CF	AL
0	100

بعد از اجرای دستور SBB

CF	AL
0	40

از کاربردهای مهم دستورالعمل SBB در تفریق دو مقدار از نوع double word می‌باشد. بعنوان مثال دو متغیر Y و X را از نوع double word در نظر بگیرید قطعه برنامه‌زیر تفاضل آنها را محاسبه نموده نتیجه را در متغییر Z که از نوع double word می‌باشد قرار می‌دهد.

```
X       DD      300000
Y       DD      250000
Z       DD      ?
MOV     AX, X
SUB     AX, Y
MOV     Z , AX
MOV     AX , X+2
SBB     AX, Y+2
MOV     Z+2, AX
```

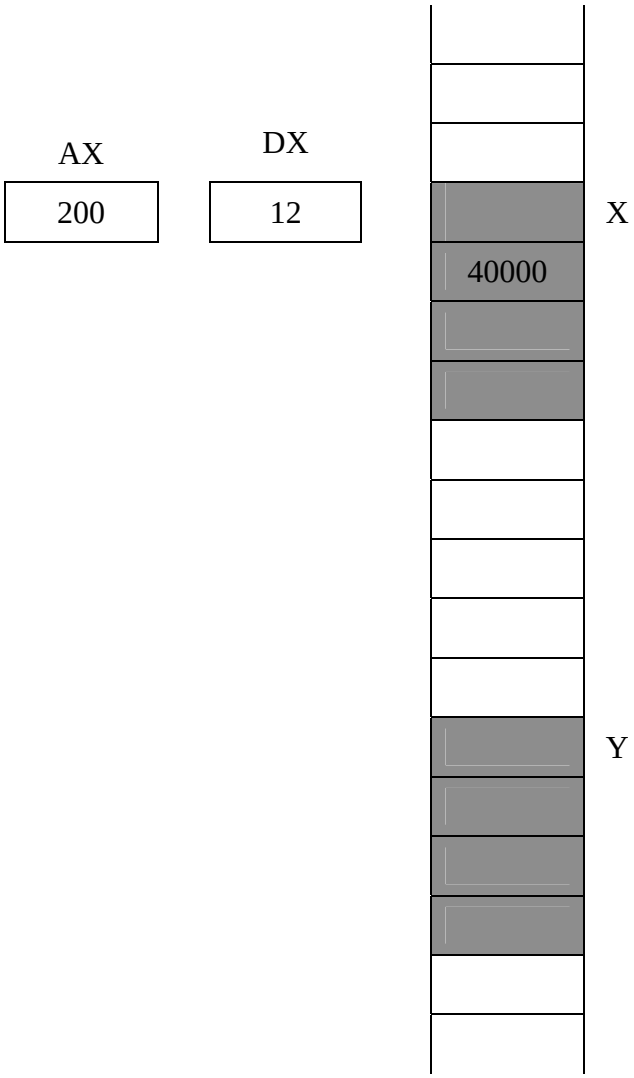


سه دستورالعمل اول متغیرهای X , Y, Z را از نوع double word تعریف می‌نماید سه دستورالعمل بعدی تفاضل دو Word بنامهای Y و X را محاسبه نموده و نتیجه را در دو بایت اول Z قرار می‌دهد. سه دستورالعمل آخر تفاضل دو بایت آخر X و دو بایت آخر Y را با استفاده از دستورالعمل SBB محاسبه نموده و نتیجه را در دو بایت آخر Z قرار می‌دهد.

مثال ۲۳-٤

قطعه برنامه‌ریز تفاضل مقدار متغیر X از نوع double word با محتوی ثباتهای AX:DX محاسبه می‌نماید و نتیجه را در متغیر Y قرار می‌دهد.

```
X       DD      40000
Y       DD      ?
MOV     AX,200
MOV     DX, 12
SUB     DX, X
SBB     AX, X+2
MOV     Y, DX
MOV     Y+2, AX
```



٥-٤- ضرب دو مقدار

دستورالعمل MUL و IMUL برای ضرب دو مقدار استفاده می‌گردد.

دستورالعمل MUL وقتی استفاده می‌گردد که عملوندها بصورت بدون علامت (unsigned) در نظر گرفته شوند. از دستور IMUL وقتی استفاده می‌گردد

که عملوندها بصورت علامت دار(signed) در نظر گرفته شوند. شکل کلی دستور MUL بصورت زیر می‌باشد.

MUL Opr

در مورد عملوند opr نکات زیر را بایستی رعایت نمود.

الف) عملوند opr بایستی از نوع بایت یا word باشد.

ب) عملوند opr می‌تواند متغییر یا ثبات باشد.

ج) عملوند نمی‌تواند ثابت باشد.

د) دستورالعمل MUL روی فلگ‌هایCF و OF اثر دارد.

هـ) در این دستورالعمل مقادیر فلگ‌هایSF، AF، ZF، PF تعریف نشده‌اند.

و) چنانچه عملوند opr از نوع بایت باشد. محتوی ثبات AL در محتوی opr ضرب شده نتیجه در AX قرار می‌گیرد.

ز) چنانچه عملوند opr از نوع word باشد محتوی ثبات AX در محتوی opr ضرب گردیده نتیجه در DX:AX قرار می‌گیرد.

مثال ٢٤-٤

```
MOV   BL, 11H
MOV   AL, 0B4H
MUL   BL
```

دستور العمل اول مقدار 11H یعنی 17 را در ثبات BL قرار می‌دهد.

BL

00010001

دستورالعمل دوم مقدار 0B4H را در ثبات AL قرار می‌دهد. همانطوریکه میدانیم B4 در مبنای شانزده معادل 180 در مبنای ده می‌باشد.

AL

10110100

در اینجا گرچه MSB ثبات AL یک مـی‌باشـد ولـی چـون از دسـتورالعمل MUL استفاده می‌شود محتوی AL را بصورت منفی در نظر نمی‌گیریم.

17 * 180 = 3060

مقدار 3060 در ثبات AX قرار می‌گیرد.

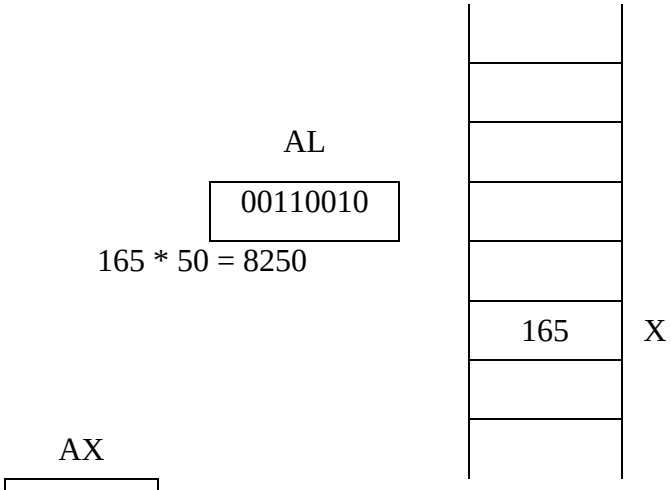
AX

3060

مثال ۲۵-٤

```
X       DB      ?
MOV     X, 0A5H
MOV     AL, 62O
MUL     X
```

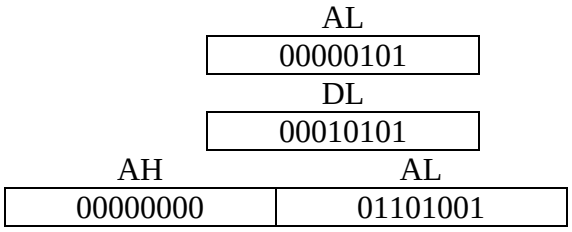
اولین دستورالعمل متغیر X را از نوع بایت تعریف نموده،دومین دستورالعمل 0A5H یعنی مقدار 165 را در متغیر X قرار داده و دستورالعمل سوم 62 در مبنـای هشت یعنی مقدار 50 را در ثبات AL قرار می‌دهد.



مثال ٤-٢٦

```
MOV   AL , 5
MOV   DL , 21
MUL   DL
```

اولین دستورالعمل مقدار 5 را در ثبات AL قرار داده. دومین دستورالعمل مقدار 21 را در ثبات DL قرار داده. سومین دستورالعمل محتوی ثبات AL را در محتوی ثبات DL ضرب نموده نتیجه را در AX قرار می‌دهد.



در اینجا مقدار OF و CF هر دو صفر می‌شود که نشان دهنده اینست که نتیجه حاصلضرب دو بایت در یک بایت جای می‌شود و نتیجه در AL قرار می‌گیرد و مقدار ثبات AH صفر می‌باشد.

دستورالعمل MUL نیز برای محاسبه حاصلضرب دو مقدار از نوع word نیز می‌توان استفاده نمود. برای این کار یکی از عملوندها بایستی حتماًدر ثبات AX قرار گیرد. بایستی توجه داشت که نتیجه حاصلضرب در ثباتهای DX:AX قرار می‌گیرد.

مثال ٤-٢٧

```
MOV   AX, 2000
MOV   BX, 15
MUL   BX
```

اولین دستورالعمل مقدار 2000 را در ثبات AX قرار می‌دهد. دومین دستورالعمل مقدار 15 را در ثبات BX قرار می‌دهد. سومین دستورالعمل حاصلضرب محتوی ثباتهای AX و BX را محاسبه نموده نتیجه حاصلضرب را در ثباتهای DX:AX قرار می‌دهد.

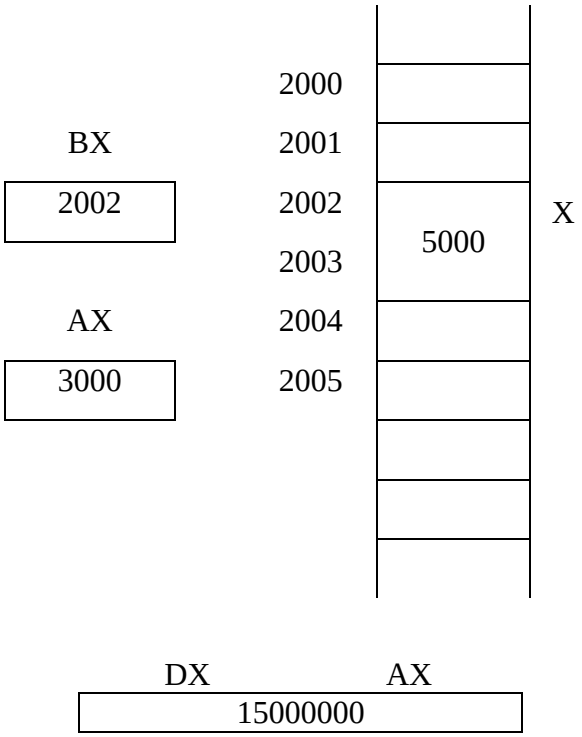
15 * 2000 = 30000

یعنی مقدار 30000 را در ثباتهای DX:AX قرار می‌دهد. در اینجا چون نتیجه در یک Word جا می‌شود مقدار فلگ‌های OF, CF برابر با صفر می‌شود.

مثال ٢٨-٤

```
X     DW    5000
MOV   AX, 3000
LEA   BX , X
MUL   [BX]
```

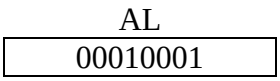
اولین دستورالعمل متغیر X را از نوع Word با مقدار 5000 تعریف نموده دومین دستورالعمل مقدار 3000 را در ثبات AX قرار می‌دهد. سومین دستورالعمل آدرس X را در ثبات BX قرار می‌دهد. آخرین دستورالعمل مقداری که توسط ثبات BX اشاره می‌شود یعنی 5000 را در محتوی AX یعنی 3000 ضرب نموده نتیجه در ثباتهای DX:AX قرار می‌دهد.



دستورالعمل IMUL نیز برای حاصلضرب دو مقدار استفاده می‌گردد. با این تفاوت که عملوندها را بصورت علامتدار (Signed) در نظر می‌گیرد. مثال

```
MOV   AL, 11H
MOV   BL, 0B4H
MUL   BL
```

دستورالعمل اول مقدار H 11 یعنی 17 را در ثبات AL قرار می‌دهد.



دستورالعمل دوم مقدار B4H یعنی 10110100 را در ثبات BL قرار می‌دهد.

BL
10110100

چون MSB محتوی BL برابر با یک می‌باشد محتوی BL را بصورت منفی در نظر می‌گیریم.

1	0	1	1	0	1	0	0
128	64	32	16	8	4	2	1

$128 + 32 + 16 + 4 = 180$

$180 - 2^8 = 180\text{-}256 = -76$

آخرین دستورالعمل مقدار 76– را در 17 ضرب نموده نتیجه یعنی 1292– در ثبات AX قرار می‌گیرد.

شکل کلی این دستورالعمل بصورت زیر می‌باشد.

IMUL opr

در مورد عملوند opr نکات زیر را بایستی رعایت نمود.

الف) عملوند بایستی از نوع بایت یا word باشد.

ب) عملوند بایستی از نوع متغییر یا ثبات باشد.

ج) عملوند نبایستی ثابت باشد.

د) دستورالعمل IMUL روی فلگ‌های OF, CF اثر دارد.

هـ) در مورداین دستورالعمل مقادیر فلگ‌هـای SF، ZF،AF و PF تعریـف نـشده می‌باشند.

و) چنانچه عملوند opr از نوع بایت باشد محتوی ثبات AL در محتوی opr ضرب شده نتیجه حاصلضرب در AX قرار داده می‌شود.

ز)چنانچه عملونـد opr از نـوع word باشـد محتـوی ثبـات AX در محتـوی opr ضرب شده نتیجه در DX:AX قرار داده می‌شود.

مثال ۲۹-٤

```
X       DW    ?
MOV     X, -300
MOV     AX, 20
IMUL    X
```

اولین دستورالعمل X را از نوع word تعریف نموده، دومین دستورالعمل مقدار 300– را در متغیر X قرار داده، سومین دستورالعمل مقدار 20 را در ثبات AX قرار داده حاصلضرب یعنی 6000- را در ثباتهای DX:AX قرار می‌دهد. در اینجا مقدار فلگ‌های CF و OF برابر با صفر می‌شود که نتیجه می‌شود مقدار DX برابر با صفر می‌باشد و مقدار AX برابر با 6000– می‌باشد.

```
X       DB    10110110B
MOV     AL, 10010001B
IMUL          X
```

اولین دستورالعمل مقدار X را از نوع بایت بصورت زیر تعریف می‌نماید.

متغیر X
10110110

دومین دستورالعمل مقدار AL را بصورت زیر تعریف می‌نماید.

AL
10010001

آخرین دستورالعمل محتوی ثبات AL را در مقدار متغیر X ضرب نموده نتیجه را در AX قرار می‌دهد. چون از دستورالعمل IMUL استفاده گردیده و MSB متغیر X و ثبات AL برابر با یک می‌باشد مقادیر متغیر X و ثبات AL بصورت منفی در نظر گرفته می‌شود.

1	0	1	1	0	1	1	0
128	64	32	16	8	4	2	1

128 + 32 + 16 + 4 + 2 = 192

192 – 256 = -64

حال

1	0	0	1	0	0	0	1
128	64	32	16	8	4	2	1

128 + 16 + 1 = 145

145 – 256 = -91

٦-٤- ضرب دو مقدار 32 بیتی بدون علامت

از دستور MUL وقتی استفاده می‌شود که عملوندها هشت یا شانزده بیتی باشند. اما برای ضرب دو مقدار بدون علامت 32 بیتی بایستی از الگوریتم زیر استفاده نمود. همانطوریکه وقتی دو مقدار را روی کاغذ می‌خواهیم ضرب نمائیم برای جمع حاصلضرب‌های جزئی، آنها را یک ستون بطرف چپ شیفت می‌دهیم از این روش بایستی استفاده نمود برای ضرب مقادیر بزرگ. بعنوان مثال دو مقدار 124 و 103 را در نظر بگیرید.

مثال ۳۰-٤

		1	2	4	*	
		1	0	3		
		3	7	2		حاصلضرب جزئی ۱
	0	0	0			حاصلضرب جزئی ۲
1	2	4				حاصلضرب جزئی ۳
1	2	7	7	2		حاصلضرب نهائی

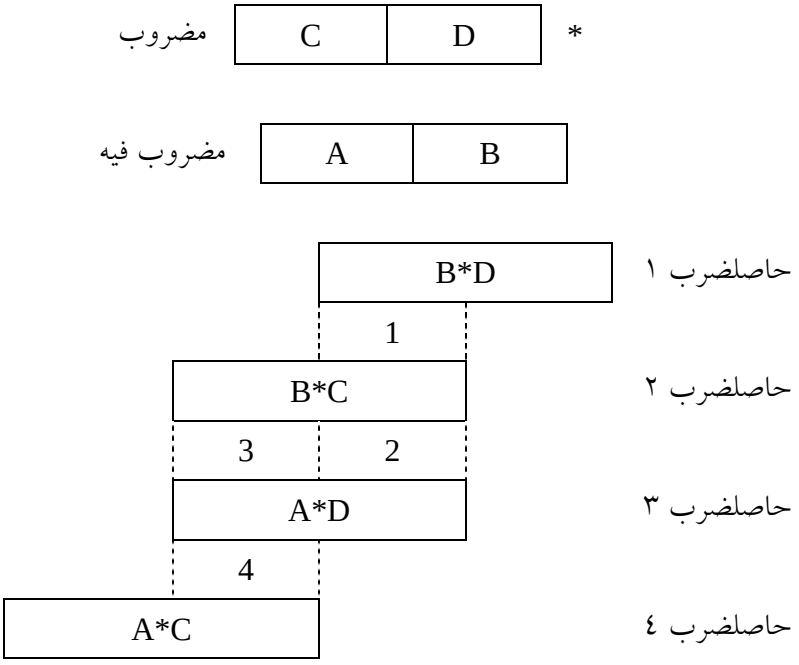
همانطور که دقت می‌کنید

$$103*124 = (3*124) + (0*124) + (100*124)$$

یا

$$103*124 = (3*1*124) + (0*10*124) + (1*100*124)$$

با این طریق می‌توان با استفاده از دستور MUL دو مقدار 32 بیتی را بدون علامت را در هم ضرب و به یک نتیجه 64 بیتی رسید. در شکل زیر D ، C ، B و A، هر کدام بصورت 2 بایت در نظر گرفته شده است.



حاصلضرب نهائی (64 بیتی) = مجموع

برنامه این الگوریتم در فصل نهم کتاب داده شده است.

۷-٤- تقسیم دو مقدار

بااستفاده از دستورالعمل DIV می‌توان دو مقدار را بر هم تقسیم نمود. شکل کلی دستورالعمل DIV بصورت زیر می‌باشد:

DIV Opr

در مورد عملوند opr بایستی نکات زیر را رعایت نمود:

الف) opr بایستی از نوع بایت یا word باشد.

ب) opr نمی‌تواند ثابت باشد.

ج) opr بایستی از نوع ثبات یا متغیر باشد.

د) چنانچه opr از نوع بایت باشد محتوی محتوی ثبات AX بر opr تقسیم شده، خارج قسمت را در ثبات AL و باقیمانده را در ثبات AH قرار می‌دهد.

هـ) چنانچه opr از نوع word باشد محتوی ثباتهای DX:AX را بر opr تقسیم نموده، نتیجه تقسیم را در AX و باقی مانده را در DX قرار می‌دهد.

ز) دستورالعمل DIV هر دو عملوند را بصورت باقی مانده unsigned (بدون علامت) در نظر می‌گیرد.

مثال ٤-۳۱

```
MOV   AX , 130
MOV   BL , 5
DIV   BL
```

در اولین دستورالعمل محتوی AX می‌شود 130، دومین دستورالعمل مقدار 5 را در ثبات BL قرار می‌دهد. آخرین دستورالعمل محتوی AX را بر محتوی BL تقسیم نموده نتیجه تقسیم را در AL و باقیمانده را در AH قرار می‌دهد.

AX

0000000010000010

BL
00000101

پس از اجرای دستورالعمل تقسیم داریم که

AL خارج قسمت
00011010

AH باقیمانده
00000000

BL
00000101

بایستی توجه داشت که دستورالعمل DIV بر روی هیچ فلگی اثر ندارد و مقدار فلگ‌هایAF، OF، PF، SF، ZF، CF تعریف نشده می‌باشند. در ضمن بایستی توجه داشت که مقدار عملوند opr بدون تغییر باقی می‌ماند.

مثال ۳۲-٤

```
X     DB   10110100B
MOV   AX, 0400H
DIV   X
```

اولین دستورالعمل مقدار 10110100B را در متغیر X قرار می‌دهد.

X
10110100

1	0	1	1	0	1	0	0
128	64	32	16	8	4	2	1

128 + 32 + 16 + 4 = 180

دومین دستورالعمل مقدار 0400H را در ثبات AX قرار می‌دهد.

AX

0000010000000000

محتوی ثبات AX برابر با 1024 می‌باشد. آخرین دستورالعمل مقدار 1024 را بر 180 تقسیم نموده مقدار خارج قسمت یعنی 5 را در ثبات AL و باقیمانده یعنی 124 را در ثبات AH قرار می‌دهد. پس از انجام عمل تقسیم داریم که

AL

00000101

AH

01111100

متغیر X

10110100

مثال ۳۳-٤

```
X     DW   2600
MOV   AX, 00A2H
MOV   DX, 0B1CH
DIV   X
```

DX	AX
0000101100001100	0000000010100010

اولین دستورالعمل متغیر X را از نوع Word با مقدار 2600 تعریف نموده، دومین دستورالعمل مقدار 00A2H را در ثبات AX قرار داده و دستورالعمل‌سوم مقدار BlC در سیستم 16 تائی را در ثبات AX قرار داده و نهایتاً آخرین دستورالعمل محتوی DX: AX یعنی 0B1C00A2 در سیستم 16 تائی را بر

2600 تقسیم نموده خارج قسمت را در AX و باقیمانده در ثبات DX قرار می‌دهد و مقدار X بدون تغییر یعنی مقدار 2600 باقی می‌ماند.

دستورالعمل IDIV مشابه دستورالعمل DIV می‌باشد با این تفاوت که عملوندها را بصورت علامتدار در نظر می‌گیرد. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

IDIV Opr

در مورد استفاده از دستورالعمل IDIV بایستی نکات زیر را در نظر داشت.

الف) عملوند opr بایستی از نوع بایت یا word باشد.

ب) عملوند opr نمی‌تواند ثابت باشد.

ج) عملوند opr بایستی از نوع ثبات یا متغیر باشد.

د) چنانچه opr از نوع بایت باشد محتوی ثبات AX بر مقدار opr تقسیم نموده خارج قسمت را در ثبات AL و باقیمانده را در ثبات AH قرار می‌دهد.

هـ) چنانچه opr از نوع word باشد محتوی ثبات DX:AX را بر opr تقسیم نموده و نتیجه تقسیم را در AX و باقیمانده را در ثبات DX قرار می‌دهد.

ز) دستورالعمل IDIV، هر دو عملوند را بصورت Signed (علامتدار) در نظر می‌گیرد.

مثال ٣٤-٤

```
MOV     BL, 0B4H
MOV     AX, 0400H
IDIV    BL
```

BL

10110100

AX

0000010000000000

اولین دستورالعمل مقدار B4 در سیستم مبنای 16 را در ثبات BL قرار می‌دهد چون از دستورالعمل IDIV استفاده شده است عملوندها را بصورت علامت دار در نظر می‌گیرد. یعنی اگر MSB عملوند برابر با یک باشد آن را منفی تلقی می‌نماید بنابراین مقدار ثبات BL را بصورت زیر در نظر می‌گیرد.

1	0	1	1	0	1	0	0
128	64	32	16	8	4	2	1

128 + 32 + 16 + 4 = 180
180 – 256 = -76

دومین دستورالعمل مقدار 400H را در ثبات AX قرار می‌دهد، که مقدار آن برابر با 1024 می‌باشد. آخرین دستورالعمل مقدار 1024 را بر 76– تقسیم نموده، نتیجه تقسیم برابر با 13– می‌باشد که در ثبات AL قرار داده می‌شود و باقیمانده راکه برابر با 36 می‌باشد در ثبات AH قرار می‌دهد و مقدار BL بدون تغییر باقی می‌ماند. مقادیر ثباتها پس از اجرای دستورالعملها عبارتند از :

AL
11110011
AH
00100100
BL
10110100

بایستی توجه داشت که دستورالعمل IDIV روی هیچ فلگی اثر ندارد و مقادیر فلگ‌هایAF، CF، OF، PF، ZF، SF تعریف نشده می‌باشند.

مثال ٤-٣٥

```
MOV     AX, 2ACH
MOV     DX, 0B2H
MOV     BX, 004AH
IDIV    BX
```

اولین دستورالعمل مقدار 2AC در سیستم مبنای 16 را در ثبات AX و مقدار B2 در سیستم مبنای 16 را در ثبات DX قرار داده محتوی ثباتهای DX:AX یعنی 00B202AC در سیستم 16 مبنای را بر 004A در سیستم مبنای 16 تقسیم نموده نتیجه تقسیم را در AX و خارج قسمت را در DX قرار می‌دهد.

مثال ٤-٣٦

```
X       DB    0A2H
MOV     AX, 0502H
IDIV    X
```

اولین دستورالعمل مقدار A2 در سیستم 16 تائی یعنی 10100010 در سیستم دودویی را در متغیر X قرار می‌دهد.

متغیر X

10100010

1	0	1	0	0	0	1	0
128	64	32	16	8	4	2	1

128 + 32 + 2 = 162
162 – 256 = -94

دستورالعمل دوّم مقدار 0502H را در ثبات AX قرار می‌دهد.

AX

0000010100000010

1	0	1	0	0	0	0	0	0	1	0
1024	512	256	128	64	32	16	8	4	2	1

$$1024 + 256 + 2 = 1282$$

آخرین دستورالعمل مقدار 1282 را بر 94– تقسیم نموده خارج قسمت که برابر با 13– می‌باشد را در ثبات AL و باقیمانده را که معادل 60 می‌باشد در ثبات AH قرار می‌دهد و مقدار X همان مقدار قبلی یعنی A2H را دارد.

متغیر X

10100010

AL

11110011

AH

00111100

٨-٤- دستورالعملهای کاهش و افزایش

با استفاده از دستورالعمل DEC می‌توان یک واحد از مقدار عملوند کم نمود. شکل کلی دستورالعمل بصورت زیر می‌باشد.

DEC opr

نکات ذیل را بایستی در موقع استفاده از این دستورالعمل در نظر داشت.

الف) opr بایستی از نوع word یا بایت باشد.

ب)opr نمی‌تواند ثابت باشد.

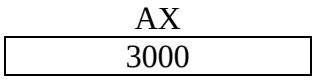
ج)این دستور العمل فقط روی فلگ‌های SF ، OF، ZF ، AF ، PF اثر دارد.

مثال٣٧-٤

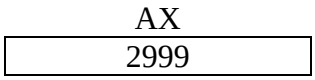
```
MOV     AX, 3000
DEC     AX
```

دستورالعمل اول مقدار 3000 را در ثبات AX قرار مـی‌دهـد. دسـتورالعمل دوم یک واحد از محتوی AX کم نموده نتیجه را در AX قرار می‌دهد.

قبل از اجرای DEC



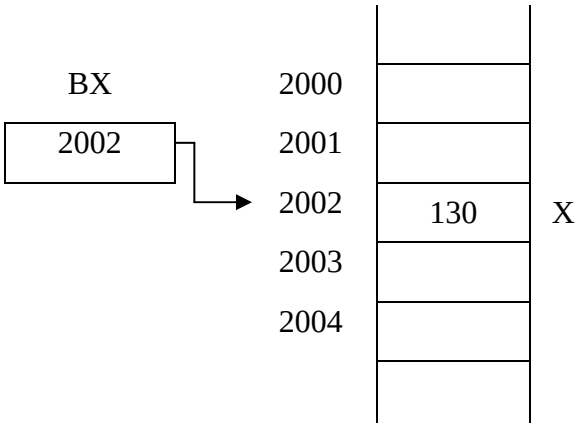
بعد از اجرای DEC



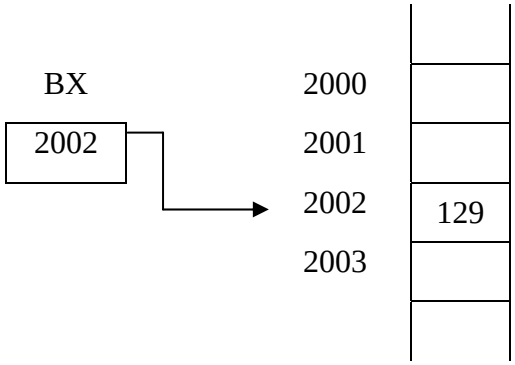
مثال ۳۸-٤

```
X      DB     130
LEA    BX , X
DEC       [BX]
```

اولین دستورالعمل مقدار 130 را در متغیر X قرار داده، دومین دسـتورالعمل آدرس متغیر X را در ثبات BX قرار می‌دهد. آنگاه محتوی محلی کـه بوسـیله BX اشاره می‌شود را یکی کاهش می‌دهد.



بعد از اجرای دستورالعمل DEC



دستورالعمل INC باعث می‌شود که یک واحد به عملوند اضافه گردد. شکل کل این دستورالعمل عبارتست از

INC Opr

در مورد استفاده از این دستورالعمل بایستی نکات ذیل را رعایت نمود.

الف) opr نمی‌تواند ثابت باشد.

ب) opr بایستی از نوع ثبات یا متغیر باشد.

ج) opr بایستی از نوع بایت یا word باشد.

د) این دستورالعمل روی فلگ‌های SF ، OF، ZF ، AF و PF اثر دارد.

مثال ۳۹-٤

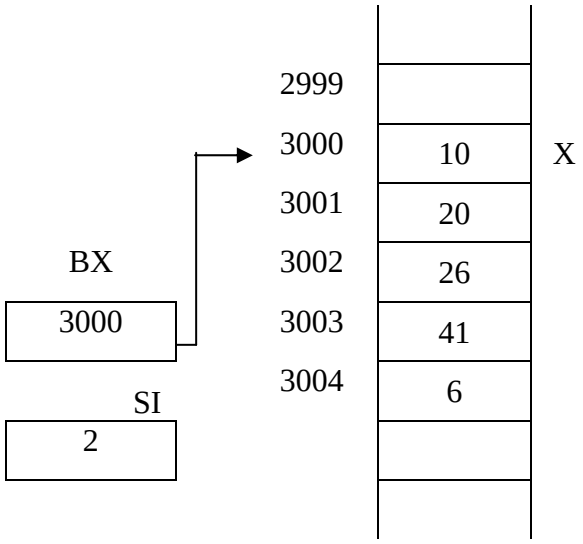
```
MOV     AL, 100
INC     AL
```

مقدار AL را به 101 افزایش می‌دهد.

مثال ٤٠-٤

```
X       DB   10,20,26,44,6
MOV          SI , 2
MOV      BX, OFFSET  X
INC          [BX] [SI]
```

اولین دستورالعمل یک آرایه 5 عنصری از نوع بایت بنام X ایجاد می‌نماید. مقادیر عناصر آرایه عبارتند از بترتیب6، 40، 26، 20، 10. دستورالعمل دوم مقدار 2 را در رجیسر SI قرار می‌دهد. دستورالعمل سوم آدرس متغیر X را در ثبات BX قرار می‌دهد. آخرین دستورالعمل یک واحد به محتوی خانه‌ای از حافظه که بوسیله محتوی BX+2 اشاره می‌کند اضافه می‌گرداند.



در حقیقت مقدار خانه حافظه با آدرس 3002 از 26 به 27 افزایش می‌یابد.

۹-٤- دستورالعمل محاسبه مکمل ۲

برای پیدا نمودن مکمل 2 یک مقدار، از دستورالعمل NEG استفاده می‌گردد. شکل کلی آن بصورت زیر می‌باشد.

NEG Opr

الف) مقدار مکمل 2 عملوند opr را محاسبه نموده در opr قرار می‌دهد.

ب) opr می‌تواند از نوع بایت یا word باشد.

ج) opr می‌تواند ثبات یا متغیر باشد.

د) opr ثابت نمی‌تواند باشد.

هـ) روی فلگ‌های SF ، OF، CF، ZF ، AF و PF اثر دارد.

مثال ٤١-٤

```
MOV     AX, -100
NEG       AX
```

مقدار AX را به 100 تغییر می‌دهد.

```
X       DB      ?
MOV       X, 26
NEG         X
```

مقدار X که از نوع بایت می‌باشد نهایتاً برابر با 26– می‌باشد.

در فصل نهم نحوه نوشتن برنامه و اجزای آن بیان گردیده است.

مروری بر مطالب فصل

در این فصل دستورالعمل ADD برای جمع نمودن دو مقدار از نوع بایت یا word و دستورالعمل SUB برای تفریق کردن دو مقدار از نوع بایت یا word داده شد. شکل کلی آنها بصورت زیر می باشد.

```
ADD    dst , src
SUB    dst , src
```

لازم به ذکر است که این دو دستورالعمل روی فلگ‌های محاسباتی اثر می‌گذارند و هر دو عملوند نمی‌تواند ثابت یا متغیر باشند. از دستورالعملهای ADC و SBB بترتیب برای جمع و تفریق دو مقدار از نوع double word استفاده می‌گردد. فرق این دستورالعملها با دو دستورالعمل بالا فقط در استفاده از مقدار CF می‌باشد. برای ضرب دو مقدار بدون علامت از نوع بایت یا word از دستورالعمل MUL و برای ضرب دو مقدار با علامت از نوع بایت یا word از دستورالعمل IMUL استفاده می‌گردد. دستورالعملهای ضرب یک عملوندی می‌باشند و این عملوند ثابت نمی‌تواند باشد. عملوند دیگر از ثبات‌های AL یا AX استفاده می‌گردد. شکل کلی عبارتند از :

```
MUL    Opr
IMUL   Opr
```

بهمین ترتیب برای تقسیم دو مقدار از نوع بایت یا word از دستورالعملهای DIV و IDIV بر حسب آنکه عملوندها بدون علامت یا با علامت در نظر گرفته شدند استفاده می‌گردد. دستورالعملهای DEC و INC یک عملوندی بوده و عملوند نمی‌تواند ثابت باشد و باعث بترتیب کاهش یا افزایش یک واحد به عملوند می‌باشد. دستورالعمل NEG نیز یک عملوندی بوده باعث تغییر علامت عملوند می‌گردد.

⌘ تمرین

۱-دستورالعمل ADD روی کدام فلگ اثر دارد؟

۲-دستورالعمل DEC روی کدام فلگ اثر ندارد؟

۳-در دستورالعمل IDIV مقادیر کدام فلگ تعریف نشده می‌باشد؟

٤-الگوریتمی را ارایه دهید که دو مقدار از نوع double word را در هم ضرب نماید.

۵-قطعه برنامه‌ای برای الگوریتم تمرین 4 ارایه کنید.

٦-دستورالعملهای لازم برای محاسبه مجموع ثباتهای AX، BX، CX و DX ارایه کنید.

۷-به چند طریق می‌توان مقدار ثبات AX را صفر نمود؟ دستورالعملهای لازم را ارایه کنید.

۸-دستورالعملها MUL بر چه فلگ‌هائی اثر ندارد؟

۹-قطعه برنامه‌ای بنویسید که مجموع عناصر آرایه 5 عنصری X از نوع word را محاسبه نماید.

۱۰-قطعه برنامه‌ای بنویسید که مجموع عناصر آرایه 5 عنصری X از نوع double word را محاسبه نماید.

۱۱-دستورالعمل معادل LEA مشخص کنید.

۱۲-دستورالعمل معادل INC BX مشخص کنید.

۱۳-قطعه برنامه زیر را به اسمبلی تبدیل نمائید.

```
long int x,y,z,w;
w=x+y-z+30;
```

١٤-قطعه برنامه‌ای بنویسید که حاصلضرب مقادیر ثباتهای AL، BL، CL و DL را محاسبه نماید.

۱۵-قطعه برنامه زیر را به اسمبلی تبدیل نمائید.

VAR

X, Y, Z, W : 1 ..20 ;
W : = X + Y * Z-W + 100;

۱۶–قطعه برنامه‌ای بنویسید که حاصلضرب عناصر آرایه چهار عنصری X از نوع بایت را مشخص نماید؟ در صورتیکه با مشکلی روبرو شدید ذکر نمائید.

۱۷– قطعه برنامه معادل زیر به اسمبلی بدهید.

int x, y, z, w;
w = x –y + z – 200;

۱۸–چنانچه بخواهیم محتوی AL را بیک word تبدیل نموده و نتیجه را در AX قرار دهیم محتوی AH چیست؟

۱۹–در مورد تمرین 18 چنانچه MSB ثبات AL برابر یک باشد محتوی AH چیست؟

۲۰–دستورالعمل لازم برای انجام محاسبه زیر را مشخص کنید.

int x,y,z,w;
w = x/y * z-10;

فصل پنجم

انشعاب و تکرار

هدف کلی

آشنائی با دستورالعملهای انشعاب و تکرار در زبان اسمبلی

اهداف رفتاری

پس از مطالعه این فصل با مفاهیم و مطالب زیر آشنا می‌شوید.

١- پرش غیر شرطی

٢- پرشهای شرطی

٣- مقایسه

٤- انواع دستورالعملهای تکرار

١-٥- دستورالعمل پرش غیر شرطی

دستورالعمل پرش غیر شرطی در زبان اسمبلی JMP می‌باشد. این دستورالعمل معادل دستورالعمل GOTO در سایر زبانهای برنامه‌نویسی می‌باشد.

شکل کلی این دستورالعمل بصورت زیر می‌باشد. این دستور روی هیچ فلگی اثر ندارد.

```
JMP        آدرس
```

مثال ۱-۵

```
JMP        LAB2
```

با اجرای این دستورالعمل کنترل به LAB2 منتقل می‌گردد. بایستی توجه داشت که کنترل بدون هیچ گونه قید و شرطی به LAB2 منتقل می‌گردد.

مثال ۲-۵

```
        MOV   AL, 5
        ADD   AL, BL
        JMP   LAB1
        MUL   BL
        INC   BL
LAB1:   SUB   CX , 2
        ⋮
```

در قطعه برنامه فوق ابتدا مقدار 5 در ثبات AL قرار می‌گیرد، سپس مقدار BL به آن اضافه گردید. آنگاه کنترل به LAB1 منتقل می‌گردد و دستورالعمل SUB به بعد اجرا می‌گردد. بایستی توجه داشت که دو دستور MUL و INC اجرا نمی‌شوند.

۲-۵- دستورالعملهای پرش شرطی

دستورالعملهای پرش شرطی وقتی اجرا می‌گردد که در برنامه شرطی برقرار گردد. شکل کلی این دستورالعملها بصورت زیر می‌باشد.

JX آدرس

که X یک رشته یک تا سه کارکتری می‌باشد.

مثال ٣-٥

JZ LAB2

در صورتیکه مقدار ZF برابر با یک باشد کنترل به LAB2 در برنامه منتقل می‌گردد.

مثال ٤-٥

JS LAB5

در صورتیکه مقدار SF برابر با یک باشد کنترل به LAB5 در برنامه منتقل می‌گردد.

مثال ٥-٥

JNO LAB20

چنانچه مقدار OF برابر با صفر باشد کنترل به LAB20 در برنامه منتقل می‌گردد. از این دستورالعمل معمولاً پس از اجرای عملیات ریاضی استفاده می‌شود.

مثال ٦-٥

```
        MOV   AX, -100
        ADD   AX, BX
        SUB   AX, CX
        JNZ   LABNEXT
              ⋮
LABNEXT:  MOV CX,10
              ⋮
```

در قطعه برنامه بالا ابتدا مقدار 100– در ثبات AX قرار داده می‌شود و سپس مقدار BX به آن افزوده می‌گردد و سپس مقدار CX از آن کسر می‌گردد. حال چنانچه نتیجه محاسبه یعنی مقدار AX مخالف صفر باشد کنترل به آدرس LABNEXT در برنامه منتقل می‌گردد.

جدول ذیل انواع دستورالعمل‌های پرش شرطی را نشان می‌دهد.

جدول ۱–۵

شرط تست	نام دیگر دستورالعمل	نام دستورالعمل	عمل
ZF=1	JE	JZ	انشعاب روی صفر
ZF=0	JNE	JNZ	انشعاب روی مخالف صفر
SF=1		JS	انشعاب روی علامت منفی
SF=0		JNS	انشعاب روی علامت غیر منفی
OF=1		JO	انشعاب روی سرریزی
OF=0		JNO	انشعاب روی عدم سرریزی
PF=1	JPE	JP	انشعاب روی ایجاد بیت توازن
PF=0	JPO	JNP	انشعاب روی عدم ایجاد بیت توازن
CF=1		JC	انشعاب روی ایجاد بیت نقلی
CF=0		JNC	انشعاب روی عدم ایجاد بیت نقلی

در جدول بالا حروف مخفف کلمات زیر می‌باشند.

Z	ZERO	
S	SIGN	
N	NOT	
P	PARITY	
O	OVERFLOW	
O	ODD	در JPO
E	EQUAL	
J	JUMP	
E	EVEN	در JPE
C	CARRY	

بایستی توجه داشت که دستورالعملهای پرش در حقیقت نقش دستورالعمل IF در سایر زبانهای برنامه‌نویسی را دارد.

مثال ۷-۵

```
TOT    DW  ?
MOV    TOT, 0
MOV    CX, 10
BEGIN: ADD   TOT, CX
DEC    CX
JNZ    BEGIN
```

در قطعه برنامه بالا متغیر TOT از نوع word تعریف گردیده و مقدار آن برابر با صفر قرار داده شده است. مقدار اولیه CX نیز برابر با 10 می‌باشد. قطعه برنامه نقش یک حلقه تکرار دارد که مقادیر 1 تا 10 را با هم جمع می‌نماید یعنی تا مادامیکه مقدار CX مخالف صفر می‌باشد، مقدار CX با TOT جمع می‌گردد و یک واحد از CX کم می‌گردد.

مثال ۸-۵

```
X     DW   ?
MOV   AX ,X
SUB   AX, 100
NEG   AX
JNS   ACT2
        ⋮
ACT2 :  ADD BX , AX
        ⋮
```

مقدار X در ثبات AX قرار داده شده آنگاه 100 واحد کاهش داده شده سپس مقدار AX در منفی یک ضرب شده حال چنانچه مقدار AX منفی نباشد کنترل به ACT2 منتقل می‌گردد. در غیر اینصورت اجرای دستورالعملهای بعدی ادامه می‌یابد.

۳-۵- دستورالعمل مقایسه

دستورالعمل مقایسه در زبان اسمبلی CMP می‌باشد. شکل کلی دستورالعمل بصورت زیر می‌باشد.

```
CMP   opr1 , opr2
```

الف) opr1 و opr2 هر دو از نوع بایت یا word می‌باشند. این دستورالعمل مقادیر عملوندها را تغییر نمی‌دهد.

ب) opr1 و opr2 می‌توانند هر دو ثبات باشند.

ج) opr1 و opr2 هر دو نمی‌توانند متغیر باشند.

د) opr1 و opr2 هر دو نمی‌توانند ثابت باشند.

هـ)دستورالعمل CMP مانند دستورالعمل SUB عمل می‌کند، با این تفاوت که نتیجه درجائی ذخیره نمی‌گردد بلکه مقادیر فلگ‌ها را تغییر می‌دهد.

ز) این دستورالعمل روی فلگ‌های AF ، OF، SF، PF، ZF، SF اثر دارد.

مثال ۹-۵

```
CMP     AX , BX
```

این دستورالعمل دو مقدار AX ، BX را با هم مقایسه می‌نماید. در حقیقت مقدار BX را از AX کم نموده و برحسب نتیجه بدست آمده مقادیر فلگها را تعیین می‌نماید.

مثال ۱۰-۵

```
CMP     AL , 10
JZ      LAB2
```

مقدار AL را با 10 مقایسه نموده در صورتیکه برابر باشند کنترل به LAB2 منتقل می‌گردد.

تعدادی دستورالعمل‌های پرش شرطی وجود دارند که با دستورالعمل CMP استفاده می‌گردند. دستورالعملهای پرش زیر وقتی استفاده می‌گردند که عملوندها بصورت بدون علامت (Unsigned) در نظر گرفته شوند.

جدول ۲-۵

نام	نامهای دیگر	شرط	فلگ‌ها
JB	JNAE , JC	Opr 1 < Opr2	CF = 1
JNB	JAE, JNC	Opr 1 > = Opr2	CF = 0
JBE	JNA	Opr 1 < = Opr2	CF ∨ ZF = 1
JNBE	JA	Opr 1 > Opr2	CF ∨ ZF = 0

دستورالعملهای پرش زیر وقتی استفاده می‌شوند که عملوندها بصورت علامتدار (Signed) در نظر گرفته شوند.

جدول ۳-۵

نام	نامهای دیگر	شرط	فلگ‌ها
JL	JNGE	Opr1 < Opr 2	SF ⊕ OF =1
JNL	JGE	Opr 1 > = Opr2	SF ⊕ OF =0
JLE	JNG	Opr1 < = Opr2	(SF ⊕ OF) ∨ ZF=1
JNLE	JG	Opr1 > Opr2	(SF⊕OF) ∨ ZF=0

حروف مخفف کلمات ذیل می‌باشند.

B	Below
A	Above
G	Greater than
E	Equal to
L	Less than
C	Carry
N	Not

مقصود از علامت V عملگر OR و مقصود از علامت ⊕ عملگر Exclusive OR می‌باشد.

```
        MOV   AX, [BX]

مقدار اول در AX با مقدار دوم مقایسه می‌شود :  CMP   AX, [DI]

آیا مقدار اول کمتر یا مساوی مقدار دوم می‌باشد؟ :  JBE   DONE
در غیر اینصورت مبادلهٔ مقادیر :  XCHG  AX , [DI]
        MOV   [BX] , AX
DONE:
        ⋮
```

در قطعه برنامه بالا دو مقدار از حافظه که بوسیله ثباتهای DI و BX مشخص می‌شوند را بترتیب صعودی مرتب می‌نماید.

مثال ۱۱-۵

```
        CMP     AL , 10 ;
        JAE     LABI
        ⋮
LAB1: JA        LAB 2
        ⋮
LAB2:
        ⋮
```

اگر محتوی AL بزرگتر از 10 باشد کنترل به LAB2 ، اگر محتوی AL مساوی 10 باشد کنترل به LAB1 در غیر اینصورت کنترل به دستورالعمل بعد از دستورالعمل JAE منتقل می‌گردد.

```
CMP   AL , BL
JE    ZERO
```

کنترل به آدرس ZERO منتقل می‌گردد اگر مقادیر BL و AL مساوی می‌باشند.

مثال ۱۲-۵

```
MOV   AX , -100
CMP   BX , AX
JG    LAB2
```

عملوندهای CMP علامت دار در نظر گرفته می‌شوند.

مثال ۱۳-۵

```
MOV     AX , 100
CMP     BX, AX
JA      LAB2
```

عملوندهای CMP بدون علامت در نظر گرفته می‌شوند.

٤-۵- دستورالعملهای تکرار

هر وقت بخواهیم تعدادی دستورالعمل بصورت مکرر اجرا گردد از دستورالعملهای تکرار بایستی استفاده نمائیم. دستورالعمل تکرار در زبان اسمبلی LOOP می‌باشد که شکل کلی آن بصورت زیر می‌باشد.

LOOP آدرس

هر وقت کنترل بدستور LOOP میرسد ابتدا مقدار ثبات CX یک واحد کاهش یافته سپس محتوی ثبات CX با صفر مقایسه می‌گردد و در صورتیکه محتوی ثبات CX مخالف صفر باشد کنترل به آدرس داده شده منتقل می‌گردد. تعداد دفعات تکرار عملاً بایستی در ثبات CX قرار داد. دستورالعمل LOOP روی هیچ فلگی اثر ندارد.

مثال ١٤-۵

```
        MOV     CX , 10
LABI:
        ⋮
        LOOP    LABI
```

این قطعه برنامه معادل برنامه پاسکال زیر می‌باشد یعنی دامنه تکرار ده بار اجرا می‌گردد.

```
FOR   I: =1   TO  10 DO
      BEGIN
            :
      END ;
```

قطعه برنامه زیر مجموع عناصر آرایـه X کـه از نـوع Word و N عنـصری می‌باشد را محاسبه نموده نتیجه را در متغیر TOTAL قرار می‌دهد.

```
N          DW       ?
TOTAL      DW       ?
X          DW            مقادیر عناصر آرایه
MOV        CX , N
MOV        AX, 0  ;      مجموع برابر با صفر
MOV        SI , AX   ;   برابر با صفر SI
START_LOOP:   ADD  AX,X [SI];    جمع عناصر
              ADD   SI, 2
              LOOP   START _ LOOP
              MOV    TOTAL, AX
```

قطعه برنامه زیر آرایـه N عنـصری A از نـوع word را بـصورت صـعودی بروش حبابی مرتب می‌نماید.

```
                Bubble  ; Sort
                N       DW      ?
                MOV     CX , N
                DEC     CX
LOOP 1:         MOV     DI , CX
                MOV     BX , 0
LOOP2:          MOV     AX , A[BX]
                CMP     AX, A[BX+2]
                JGE     CONTINUE
                XCHG    AX, A[BX+2]
                MOV     A [BX] , AX
CONTINUE:       ADD     BX,2
                LOOP    LOOP2
                MOV     CX , DI
                LOOP    LOOP1
```

شکل دیگر دستور تکرار بصورت زیر می‌باشد.

LOOPNE آدرس

یا

LOOPNZ آدرس

کار دستورالعمل LOOPNE یا LOOPNZ مانند دستورالعمل LOOP می‌باشد با این تفاوت که شرط تکرار آن است که مقدار CX مخالف صفر و مقدار ZF برابر با صفر باشد. این دستورالعمل روی هیچ فلگی اثر ندارد.

مثال ١٥-٥

```
        ARR     DB
        N       DW
        MOV     CX , N
        MOV     SI , -1
        MOV     AL, 20H; ASCII code for blank
NEXT:   INC     SI
        CMP     AL, ARR[SI]; test for blank
      LOOPNE    NEXT
        JNZ     NOT_FOUND
```

قطعه برنامه بالا رشتهٔ داده شده N عنصری ARR از نوع بایت را جستجو می‌نماید که آیا کارکتر blank یا فاصله در رشته وجود دارد یا خیر؟

توجه داشته باشید که دستور تکرار بالا وقتی متوقف می‌شود که عناصر رشته همه مورد بررسی قرار گرفته باشند یا به کارکتر فاصله رسیده باشیم. شکل دیگر دستورالعمل تکرار بصورت زیر می‌باشد.

LOOPE آدرس

یا

LOOPZ آدرس

دستورالعمل LOOPE یا LOOPZ مانند دستورالعمل LOOP عمل می‌نماید با این تفاوت که شرط تکرار آن است که مقدار CX مخالف صفر و مقدار ZF برابر با یک باشد. این دستورالعمل روی هیچ فلگی اثر ندارد.

```
; BX = offset  of   the   starting address
; DX = offset  of   the ending address
; BX = offset  of   nonzero (if found)
; BX = DI (if not found)
```

```
                SUB     DI , BX
                INC     DI ; تعداد بایت = (DI)-(BX)+1
                MOV     CX , DI
                DEC     BX
NEXT:           INC     BX ; point to next location
                CMP     BYTE  PTR [BX], 0 ; Compare it to zero
                LOOPE   NEXT ; go compare next byte
                JNZ     NZ _ FOUND; Nonzero byte found?
                 ⋮      ; NO.
NZ_FOUND:
                 ⋮      ; Yes.
```

برنامه فوق یک بلوک از حافظه که آدرس شروع ان توسط ثبات BX و آدرس انتهای آن توسط ثبات DI مشخص شده را بایت به بایت جستجو نموده برای یافتن عنصری که مخالف صفر باشد.

شکل دیگر دستور تکرار بصورت زیر می‌باشد.

JCXZ آدرس

در حقیقت این دستورالعمل یک نوع دستورالعمل پرش می‌باشد که براساس فلگها عمل نمی‌کند بلکه براساس محتوی ثبات CX عمل می‌کند. چنانچه محتوی CX مساوی صفر باشد کنترل به آدرس داده شده منتقل می‌شود. این دستورالعمل روی هیچ فلگی اثر ندارد. نهایتاً جدول ذیل را داریم.

جدول ٤-٥

عمل	نام	نام دیگر	شرط تکرار
LOOP	**LOOP**		**CX < > 0**
LOOP While equal or zero	LOOPE	LOOPZ	CX < > 0 and ZF=1
LOOP while not equal or nonzero	LOOPNE	LOOPNZ	CX < > 0 and ZF=0
Branch on CX	JCXZ		CX=0

مروری بر مطالب فصل

در این فصل دستورالعملهای پرسش تشریح گردیده این دستورالعملها بر اساس فلگ‌ها عمل می‌نمایند. ضمناً دستورالعمل‌های تکرار نیز مطرح گردیده است که در دستورالعملها تکرار تعداد دفعات تکرار در ثبات CX قرار می‌گیرد. دستورالعمل مقایسه CMP می‌باشد که مانند دستورالعمل SUB عمل نموده ولی نتیجه در جائی ذخیره نمی‌گردد بلکه روی فلگها اثر می‌گذارد.

⌘ تمرین

۱-دستورالعمل CMP مشابه کدام دستورالعمل می‌باشد؟

۲-آیا استفاده از دستورالعمل CMP X , Y مجاز می‌باشد؟

۳-دستورالعمل JNA چه موقعی استفاده می‌شود؟

٤-دستورالعمل JLE چه موقعی مورد استفاده قرار می‌گیرد؟

۵-تفاوت دستورالعمل LOOP با LOOPZ چیست؟

٦-اگر در ابتدا محتوی CX را برابر با صفر قرار دهیم. حلقه تکرار چند بار اجرا می گردد؟

۷-دستورالعملهای اسمبلی معادل قطعه برنامه پاسکال زیر بدهید.

```
S : = 0 ;
FOR  I:=1    TO   N  DO
        S : = S+I ;
```

۸-قطعه برنامه‌ای به زبان اسمبلی معادل قطعه برنامه زیر بدهید.

```
X: ARRAY [1..10]    OF INTEGER;
S, I: I NTEGER;
      S:= 0 ;
      I := 1 ;
      WHILE          I < = 10   DO
      BEGIN
                          S:=S +X [I];
                           I :=I + 1 ;
      END ;
```

۹–قطعه برنامه‌ای بنویسید که یک آرایه N عنصری از نوع بایت را در نظر بگیرد، تعداد عناصر مثبت آنرا مشخص نماید در متغیر TED قرار دهد.

۱۰–آیا می‌توان قطعه برنامه‌ای نوشت که N مقدار را بروش مرتب سازی درجی بصورت صعودی مرتب نماید؟

۱۱–یک قطعه برنامه بدهید که آرایه N عنصری X از نوع بایت را از نظر مکانی وارون نماید.

۱۲–قطعه برنامه‌ای بدهید که آرایه N عنصری X از نوع word را در نظر گرفته عناصر مخالف صفر آنرا در آرایه دیگری بنام Y قرار دهد (در صورت لزوم انتهای آرایه Y خالی بماند.)

۱۳–قطعه برنامه‌ای بنویسید که مقدار صحیح و مثبت N را در نظر گرفته فاکتوریل آنرا مشخص نماید.

۱٤-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را در نظر گرفته کوچکترین عنصر آرایه را مشخص نماید.

۱۵-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع double word را در نظر گرفته کوچکترین عنصر آرایه را مشخص نماید.

۱٦-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را در نظر گرفته اندیس بزرگترین عنصر آرایه را مشخص نماید.

۱۷-قطعه برنامه‌ای بنویسید که مجموع زیر را محاسبه نماید.

$$1+ 2^2+3^2+4^2+. . . + N^2$$

۱۸-قطعه برنامه‌ای بنویسید که مشخص نماید آیا عدد N اول می‌باشد یا خیر؟

۱۹-قطعه برنامه‌ای بنویسید که مشخص نماید آیا عدد N کامل می‌باشد یا خیر؟

۲۰-قطعه برنامه‌ای بنویسید که مقدار N از نوع بایت را گرفته N^X را محاسبه نماید.

۲۱-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را گرفته میانه آنرا مشخص نمود در MID قرار دهد.

فصل ششم

عملیات بیتی

هدف کلی

آشنائی با عملیات روی بیت‌ها.

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

۱- عملیات منطقی و عملگرهای وابسته

۲- عملیات شیفت بیت‌ها

۳- عملیات چرخش بیت‌ها

٤- عملیات مربوط به فلگ‌ها

۵- تبدیل حروف کوچک به بزرگ و بالعکس.

۱-٦- عملیات منطقی

از دستورالعملهای منطقی برای انجام عملیات منطقی استفاده می‌شود. این دستورالعملها بصورت بیتی روی عملوندها عمل می‌نماید. عملیات منطقی عبارتند از NOT، AND، OR، XOR و TEST .

۱-۱-٦- دستورالعمل NOT

شکل کلی دستورالعمل NOT بصورت زیر می‌باشد.

NOT opr

الف) opr می‌تواند از نوع wordیا بایت باشد.

ب) opr می‌تواند متغیر یا ثبات باشد.

ج) این دستورالعمل بیت‌های opr را از 0 به 1 و از 1 به 0 تبدیل می‌نماید. بعبارت دیگر مکمل 1 عملوند را می‌دهد.

د) دستورالعمل NOT روی هیچ فلگی اثر ندارد.

مثال ۱-٦

```
MOV     DL , 8AH
NOT     DL
```

قبل از اجرای دستور NOT

DL
10001010

بعد از اجرای دستور NOT

DL
01110101

بنابراین محتوی ثبات DL به 75H تغییر می‌نماید.

٢-١-٦- دستورالعمل AND

شکل کلی دستورالعمل AND بصورت زیر می‌باشد.

```
AND   dst , src
```

الف) عملوندهای src و dst هر دو از نوع بایت یا word می‌باشند.

ب) عملوندهای src و dst هر دو متغیر یاهر دو ثابت نمی‌توانند باشند.

ج) وقتی عملوند src ثابت باشد عملوند dst بایستی از نوع ثبات یا متغیر باشد.

د) بیت‌های dst و src نظیر به نظیر مطابق جدول ذیل and می‌شوند و نتیجه در dst قرار می‌گیرد.

جدول ١-٦

بیت اول	بیت دوم	بیت دوم **and** بیت اول
0	0	0
0	1	0
1	0	0
1	1	1

مثال ٢-٦

```
MOV   AL , 5BH
MOV   DH, 4DH
AND   AL, DH
```

مقادیر ثباتها قبل از اجرای دستورالعمل AND عبارتست از:

AL	01011011
DH	01001101

مقادیر ثباتها پس از اجرای دستورالعمل AND عبارتند از:

DH	01001101	بدون تغییر
AL	01001001	نتیجه

۳-۱-٦- دستورالعمل OR

شکل کلی دستورالعمل OR بصورت زیر می‌باشد.

OR dst , src

الف) عملوندهای dst و src از نوع بایت یا word می‌باشند.

ب) عملوندهای dst و src هر دو متغیر یا هر دو ثابت نمی‌توانند باشند.

ج) وقتی عملوند src ثابت باشد عملوند dst بایستی از نوع متغیر یا ثبات باشد.

د) بیت‌های dst و src نظیر به نظیر مطابق جدول ذیل or می‌شود ونتیجه در dst قرار می‌گیرد.

جدول ۲-٦

بیت اول	بیت دوم	بیت دوم or بیت اول
0	0	0
0	1	1
1	0	1
1	1	1

مثال۳-٦

```
MOV   BL , 0A5H
MOV   AL, 2AH
OR    AL , BL
```

مقادیر ثبات ها قبل از اجرای دستورالعمل OR

BL	10100101
AL	00101010

مقادیر ثباتها بعد از اجرای دستورالعمل OR

BL	10100101
AL	10101111

٤-١-٦- دستورالعمل XOR

شکل کلی دستورالعمل XOR بصورت زیر می‌باشد.

XOR dst , src

الف) src و dst هر دو از نوع بایت یا word می‌باشند.

ب) src و dst هر دو متغیر یا ثابت نمی‌تواند باشند.

ج) بیت‌های src و dst نظیر به نظیر با استفاده از جدول ذیل xor گردیده نتیجه در dst قرار می‌گیرد و مقدار src بدون تغییر باقی می‌ماند.

جدول ٣-٦

بیت اول	بیت دوم	بیت دوم xor بیت اول
0	0	0
0	1	1
1	0	1
1	1	0

مثال ٤-٦

```
MOV   CL, 2DH
MOV   AL , 0C2H
XOR   AL , CL
```

مقادیر ثباتها قبل از اجرای دستورالعمل XOR

AL	11000010
CL	00111101

مقادیر ثباتها بعد از اجرای دستورالعمل XOR

CL	0011 1101
AL	1111 1111

٥-١-٦- دستورالعمل TEST

شکل کلی دستورالعمل TEST بصورت زیر می‌باشد:

```
TEST   opr1, opr2
```

الف) opr1 و opr2 هر دو از نوع بایت یا word می‌باشد.

ب) opr1 و opr2 هر دو ثابت یا هر دو متغیر نمی‌توانند باشند.

ج) این دستورالعمل مانند دستورالعمل AND عمل می‌نماید ولی نتیجه را در جائی ذخیره نمی‌کند یعنی دو عملوند بدون تغییر باقی می‌مانند و فقط مقادیر فلگ‌ها را تغییر می‌دهد.

مثال ٥-٦

```
MOV   AL , 25
MOV   DH , 0E4H
TEST  AL , DH
```

مقادیر ثباتها قبل از اجرای دستورالعمل TEST

AL	00011001
DH	11100100

مقادیر ثباتها پس از اجرای دستورالعمل TEST

AL	00011001
DH	11100100

بایستی توجه کرد که دستورالعمل TEST باعث می‌شود که مقدار ZF برابر با یک گردد.

مثال ٦-٦

```
      MOV   AL , 0ABH
      NOT   AL
      TEST  AL, 10100101B
      JZ    YES
      ⋮
YES:
      ⋮
```

قطعه برنامه فوق مشخص می‌نماید که آیا مقادیر بیت‌های 7، 5، 2 و 0 ثبات AL برابر با یک می‌باشد یا خیر؟ مقدار 10100101B عملاً MASK می‌باشد که

در بیت‌هائی که می‌خواهیم برای یک بودن تست شود مقدار یک و در سایر بیت‌ها مقدار صفر را قرار می‌دهیم.

AL	10101011

پس از اجرای دستور NOT

AL	01010100

MASK	10100101

در این مثال پس از اجرای دستورالعمل TEST مقدار ثبات AL بدون تغییر باقی ماند و فقط مقدار فلگ ZF برابر با یک می‌شود.

مثال ۷-٦

```
        OR      DL, 00000101B
        XOR     DL, 01000010B
        AND     DL, 11100111B
        MOV     AL, DL
        NOT     AL
        TEST    AL, 10000010B
        JZ      EXIT
        .
        .
EXIT:
        .
        .
```

قطعه برنامه فوق ابتدا بیت‌های شماره 0 و 2 ثبات DL را یک می‌کند و بیت های شماره 4 و 3 را به صفر تبدیل می‌کند و بیت‌های شماره 1 و 6 را مکمل می‌نماید در ضمن چنانچه بیت‌های شماره 7و 1 برابر با یک باشند کنترل به EXIT منتقل می‌نماید.

مثال ۸-٦

قطعه برنامه زیر بیت‌های شماره فرد ثبات AL را مکمل می‌نماید. یعنی 1 به 0 و 0 به 1 تبدیل می‌نماید.

```
MOV   AL, 0C7H
MOV   MASK, 10101010B
XOR   AL, MASK
```

AL	11000111
MASK	10101010

پس از اجرای دستورالعمل XOR مقادیر AL و MASK عبارتند از:

AL	01101101
MASK	10101010

مثال ۹-٦

قطعه برنامه زیر بیت‌های شماره زوج ثبات AL را به یک تبدیل می‌نماید.

```
MOV   AL, 0A6H
MOV   MASK, 55H
OR    AL, MASK
```

AL	10100110
MASK	01010101

پس از اجرای دستورالعمل OR

AL	11110111
MASK	01010101

مثال ۱۰-٦

قطعه برنامه زیر بیت‌های شماره فرد AL را به صفر تبدیل می‌نماید.

```
MOV   AL, 0C7H
MOV   MASK, 55H
AND   AL, MASK
```

AL	11000111
MASK	01010101

پس از اجرای دستورالعمل AND

AL	01000101
MASK	01010101

در قطعه برنامه زیر اگر بیت‌های شماره 1 و 14 یا بیت‌های شماره 9 و 7 ثبات AX برابر با یک باشند کنترل به TASK 1 و اگر بیت 3 یا 4 برابر با یک باشند کنترل به TASK2 در غیر اینصورت کنترل به TASK3 منتقل می‌گردد.

```
        NOT     AX
        TEST    AX, 4002H
        JZ      TASK1
        TEST    AX, 280H
        JZ      TASK1
        NOT     AX
        TEST    AX, 18H
        JNZ     TASK2
TASK3:
        ⋮
TASK1:
        ⋮
TASK2:
        ⋮
```

۲-٦- عملیات شیفت

عملیات شیفت باعث تغییر مکان بیت‌های یک بایت یا یک word بطرف چپ یا راست می‌شود. دستورالعملهای متعددی برای این کار مورد استفاده قرار می‌گیرند که عبارتند از:

Shift	Logical Left	SHL
Shift	Logical Right	SHR
Shift	Arithmetic Left	SAL
Shift	Arithmetic Right	SAR

۱-۲-٦- دستورالعمل SHL

شکل کلی دستورالعمل SHL بصورت زیر می‌باشد.

```
SHL     opr , cnt
```

الف) cnt تعداد بیت‌هائی می‌باشد که بطرف چپ شیفت داده می‌شود. در صورتیکه cnt مخالف یک باشد از ثبات CL استفاده می‌نمائیم.

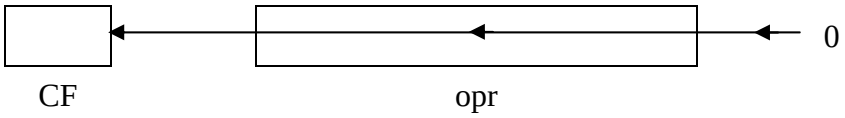
ب) opr می‌تواند از نوع بایت یا word باشد.

ج) oprمی‌تواند متغیر یا ثابت باشد.

د) opr ثابت نمی‌تواند باشد.

هـ) روی فلگهایOF ، SF ،ZF، PF و CF اثر دارد.

ز) بیت‌های oprرا با اندازه cnt بیت بطرف چپ شیفت می‌دهد و از طـرف راسـت با صفر پر می‌شود.



مثال ۱۱-٦

```
SHL   AX, CL
SHL   BL, CL
SHL   AL, 1

STC
MOV   CL, 3
MOV   DL, 8DH
SHL   DL , CL
```

دستورالعمل STC مقدار فلگ CF را به یک تبدیل می‌نماید.

قبل از اجرای دستورالعمل SHL

CL	00000011	CF	1
DL	10001101		

بعد از اجرای دستورالعمل SHL

CL	00000011
DL	01101000
CF	0

۲-۲-٦- دستورالعمل SHR

شکل کلی دستورالعمل SHR بصورت زیر می‌باشد.

SHR opr, cnt

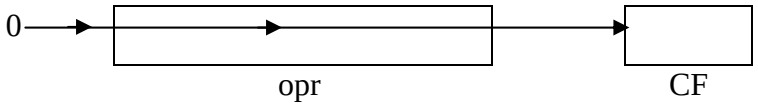
الف) opr می‌تواند از نوع بایت یا word باشد.

ب) opr می‌تواند متغیر یا ثابت باشد. ولی ثابت نمی‌تواند باشد.

ج) روی فلگهای CF ،ZF، PF اثر دارد.

د) اگر مقدار cnt برابر با یک باشد خودش را می‌نویسم در غیر اینصورت از ثبات CL استفاده می‌نمائیم.

هـ) بیت‌های opr را با اندازه cnt بیت بطرف راست شیفت داده و از طرف چپ با صفر پر می‌نمائیم.



مثال ۱۲-٦

```
SHR   DL, 1
SHR   AL, CL
```

مثال ۱۳-٦

```
STC
MOV   CL , 3
MOV   DL, 8DH
SHR   DL, CL
```

محتوی ثبات DL را سه بیت بطرف راست شیفت می‌دهد.

CL	00000011
DL	10001101
CF	1

مقادیر پس از اجرای دستورالعمل SHR عبارتند از

CL	00000011
DL	00010001
CF	1

۳-۲-٦- دستورالعمل SAL

شکل کلی دستورالعمل SAL بصورت زیر می‌باشد.

```
SAL   opr , cnt
```

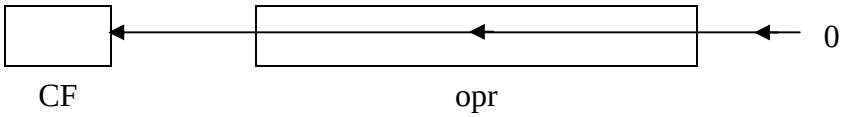
الف) opr می‌تواند از نوع بایت یا word باشد.

ب) cnt اگر یک باشد مقدار یک را می‌نویسم در غیر اینصورت از ثبات CL استفاده می‌نمائیم.

ج) opr ثابت نمی‌تواند باشد.

د) روی فلگهایOF ، SF ،ZF ، PF و CF اثر دارد.

هـ) بیت‌های opr را باندازه cnt بیت بطرف چپ شیفت می‌دهد و از طرف راست با صفر پر می‌شود.



مثال ١٤-٦

```
SAL   DL, 1
SAL   AL, CL
SAL   [BX] , CL
SAL   ARR [SI], CL
```

مثال ١٥-٦

```
 STC
MOV   CL , 3
MOV   DL, 8DH
 SAL   DL, CL
```

قبل از اجرای دستورالعمل SAL

CL	00000011
DL	10001101
CF	1

بعد از اجرای دستورالعمل SAL

CL	00000011
DL	01101000
CF	0

٤-٢-٦- دستورالعمل SAR

شکل کلی دستورالعمل SAR بصورت زیر می‌باشد.

SAR opr , cnt

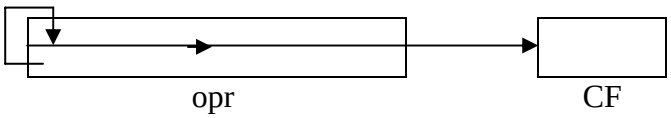
الف) opr می‌تواند از نوع ثبات یا متغیر باشد.

ب) opr نمی تواندثابت باشد.

ج) opr می‌تواند از نوع بایت یا word باشد.

د) cnt اگر معادل یک باشد مقدار 1 در غیر اینصورت از ثبات CL استفاده می‌نمائیم.

هـ) بیت‌های opr را باندازهٔ cnt بیت بطرف راست شیفت داده و از سمت چپ با MSB پر می‌نماید.



مثال ٦-١٦

```
SAR   DL, 1
SAR   AX , CL
SAR   [BX], CL
```

مثال ٦-١٧

```
MOV   CL,3
STC
MOV   DL, 8DH
SAR   DL, CL
```

مقادیر قبل از اجرای دستورالعمل SAR

CL	00000011
DL	10001101
CF	1

مقادیر بعد از اجرای دستورالعمل SAR

CL	00000011
DL	11110001
CF	1

۳-٦- عملیات چرخش (Rotate)

عملیات چرخش باعث دور زدن بیت‌های یک بایت یا word می‌شوند. بعبارت دیگر مانند دستورالعمل‌های شیفت باعث خارج شدن بیت‌ها از بایت یا word نمی‌شود. دستورالعملهای چرخش عبارتند از:

Rotate	Left	ROL
Rotate	Right	ROR
Rotate	Left through Carry	RCL
Rotate	Right through Carry	RCR

۱-۳-٦- دستورالعمل ROL

شکل کلی دستورالعمل ROL بصورت زیر می‌باشد.

```
ROL   opr , cnt
```

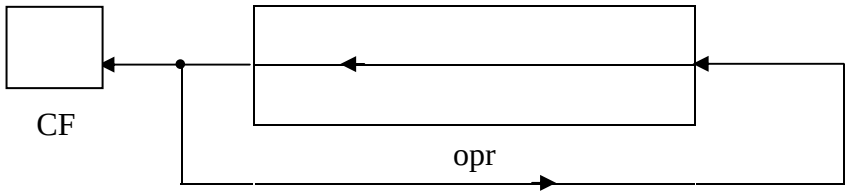
الف) opr می‌بایستی از نوع بایت یا word باشد.

ب) opr می‌بایستی متغیر یا ثبات باشد.

ج) اگر مقدار cnt برابر با یک باشد عدد 1 را می‌نویسم در غیر اینصورت از ثبات CL استفاده می‌کنیم.

د) روی فلگ CF اثر دارد.

هـ) این دستورالعمل باندازه cnt بیت از سمت چپ چرخش می‌دهد.



مثال ۱۸-٦

```
ROL   DL, CL
ROL   BX , 1
ROL   [BX] , CL
```

مثال ۱۹-٦

```
MOV   CL,3
MOV   DL, 8DH
 STC
ROL   DL, CL
```

مقادیر قبل از اجرای دستورالعمل ROL

CL	00000011
DL	10001101
CF	1

مقادیر بعد از اجرای دستورالعمل ROL

CL	00000011
DL	01101100
CF	0

۲-۳-٦- دستورالعمل ROR

شکل کلی دستورالعمل ROR بصورت زیر می‌باشد:

ROR opr , cnt

الف) opr از نوع بایت یا word می‌باشد.

ب) opr می‌بایستی از نوع متغیر یا ثبات باشد. opr ثابت نمی‌تواند باشد.

ج) cnt اگر معادل یک باشد عدد 1 را می‌نویسم در غیر اینصورت از ثبات CL استفاده می‌نمائیم.

د) روی فلگ CF اثر دارد.

هـ) باندازهٔ cnt بیت بطرف راست چرخش می‌دهد.

مثال ٢٠-٦

```
ROR   DL, 1
ROR   BX, CL
ROR   AL, CL
ROR   X[DI],CL
```

مثال ٢١-٦

```
MOV   CL, 3
STC
MOV   DL, 8DH
ROR   DL, CL
```

مقادیر قبل از اجرای دستورالعمل ROR

CL	00000011
DL	10001101
CF	1

مقادیر بعد از اجرای دستورالعمل ROR

CL	00000011
DL	10110001
CF	1

٦-٣-٣- دستورالعمل RCL

شکل کلی دستورالعمل RCL بصورت زیر می‌باشد.

```
RCL   opr , cnt
```

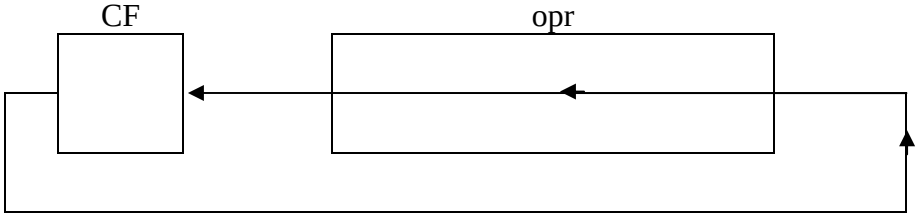
الف) چنانچه مقدار cnt یک باشد مقدار 1 نوشته می‌شود در غیر اینصورت از ثبات CL استفاده می‌گردد.

ب) opr بایستی از نوع بایت یا word باشد.

ج) opr بایستی متغیر یا ثبات باشد. opr مقدار ثابت نمی‌تواند باشد.

د) روی فلگ CF اثر دارد.

هـ) دستورالعمل RCL باندازه cnt بیت از بیت‌های opr را از طرف چپ و از طریق بیت CF چرخش می‌دهد.



مثال ۲۲-۶

```
RCL   DL, 1
RCL   BX, CL
RCL   AL, CL
RCL   BL, 1
RCL   [BX],CL
```

مثال ۲۳-۶

```
MOV   CL, 3
MOV   DL, 8DH
 STC
RCL   DL, CL
```

مقادیر قبل از اجرای دستورالعمل RCL

DL	10001101
CL	00000011
CF	1

مقادیر بعد از اجرای دستورالعمل RCL

CL	00000011
DL	01101110
CF	0

٤-٣-٦- دستورالعمل RCR

شکل کلی دستورالعمل RCR بصورت ذیل می‌باشد:

RCR opr , cnt

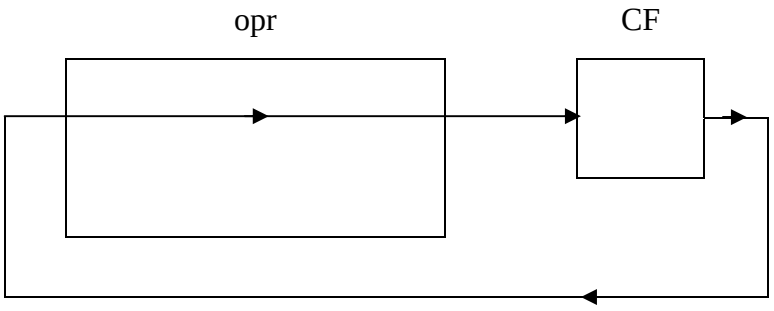
الف) opr بایستی از نوع word یا بایت باشد.

ب) opr بایستی ثبات یا متغیر باشد و ثابت نمی‌تواند باشد.

ج) اگر مقدار cnt برابر با یک باشد بایستی مقدار 1 را نوشت در غیر اینصورت از ثبات CL استفاده نمود.

د) روی فلگ CF اثر دارد.

هـ) دستورالعمل RCR باندازه cnt بیت بطرف راست از طریق فلگ CF چرخش می‌دهد.



مثال ٢٤-٦

```
RCR   DL, 1
RCR   [BX], CL
RCR   AL, CL
RCR   X[BX][DI],CL
RCR   [BX], 1
```

مثال ٢٥-٦

```
MOV   CL, 3
MOV   DL, 8DH
 STC
 RCR   DL, CL
```

مقادیر قبل از اجرای دستورالعمل RCR

CL	00000011
DL	10001101
CF	1

مقادیر بعد از اجرای دستورالعمل RCR

CL	00000011
DL	01110001
CF	1

٤-٦- عملیات فلگ‌ها

دستورالعملهای مربوط به عملیات روی فلگها در ذیل داده شده‌اند. این دستورالعملها عبارتند از:

جدول ٤-٦

Clear Carry	CLC	CF صفر می‌شود
Complement Carry	CMC	CF مکمل می‌شود
Set Carry	STC	CF یک می‌شود
Clear Direction	CLD	DF صفر می‌شود
Set Direction	STD	DF یک می‌شود
Clear Interrupt	CLI	IF صفر می‌شود
Set Interrupt	STI	IF یک می‌شود

همانطوریکه ملاحظه می‌شود این دستورالعملها فاقد عملوند می‌باشند. این دستورالعمل‌ها فقط روی فلگ مربوطه اثر دارند. بعنوان مثال CLC فقط روی فلگ CF اثر می‌کند و مقدار قبلی آنرا تغییر می‌دهد.

از طرف دیگر دو دستورالعمل LAHF و SAHF نیز مربوط به عملیات فلگ‌ها می‌باشند که در ذیل توصیف می‌گردند.

شکل کلی دستورالعمل LAHF بصورت زیر می‌باشد.

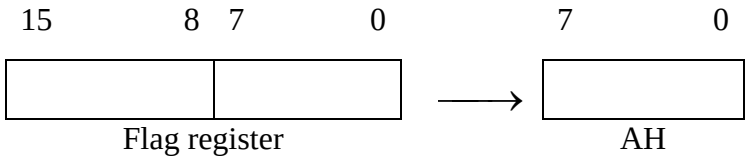
LAHF

الف) این دستورالعمل فاقد عملوند می‌باشد.

ب) بایت کم ارزش (بیت‌های 0 تا 7) ثبات فلگ را به AH منتقل می‌نماید.

ج) روی هیچ فلگی اثر ندارد.

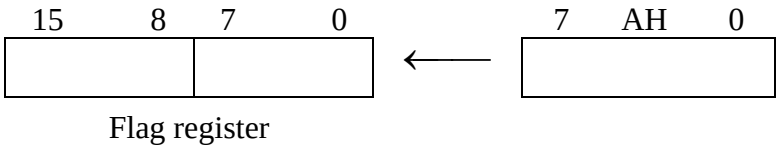
د) فقط مقادیر بیت‌های شماره 7، 6، 4، 2، 0 منتقل می‌گردد.



شکل کلی دستورالعمل SAHF بصورت زیر می‌باشد.

SAHF

الف) این دستورالعمل فاقد عملوند می‌باشد.
ب) روی هیچ فلگی اثر ندارد.
ج) محتوی ثبات AH را به بایت کم ارزش (بیت‌های0 تا7) ثبات فلگ منتقل می‌کند.



٥-٦- تبدیل حروف

همانطوریکه میدانید به هر کارکتر یک کد اسکی بین 0 تا 255 تخصیص می‌یابد. کداسکی برای حروف بزرگ A تا Z بترتیب از 65 تا 90 یا از 41H تا 5AH می‌باشد. حروف کوچک نیز دارای کد اسکی بین 97 تا 122 یا 61H تا 7AH می‌باشند.

A	65	01000001
A	97	01100001
Z	90	01011010
Z	122	01111010

همانطوریکه ملاحظه می‌کنید تفاوت نمایش بین A و a یا Z و z در بیت شماره 5 آنها می‌باشد. بنابراین نتیجه می‌گیریم برای تبدیل حروف کوچک به بزرگ کافی است که بصورت زیر عمل نمائیم.

AND حرف کوچک , 00100000B

و برای تبدیل حروف بزرگ به حروف کوچک کافی است که بصورت زیر عمل شود

OR حرف بزرگ, 00100000B

مثال ٢٦-٦

در رشتهٔ STR حروف کوچک را به بزرگ تبدیل نمائید. بایستی کنترل شود که کارکتر از نوع حروف کوچک می‌باشد سپس به حروف بزرگ متناظر تبدیل گردد.

```
        STR     DB   'Change to uppercase letters'
        LEA     BX , STR+1
        MOV     CX, 26
LAB1:   MOV     AX, [BX]
        CMP     AH, 61H
        JB      LAB2
        CMP     AH, 7AH
        JA      LB2
        AND     AH, 0DFH
        MOV     [BX], AH
LAB2:   INC     BX
        LOOP    LAB1
```

مروری بر مطالب فصل

در این فصل عملیات بیتی مورد بحث قرار گرفت. ابتدا عملیات منطقی مانند AND، OR، NOT ، XOR ، TEST بحث گردید. این دستورالعملها روی بیت‌ها عمل می‌نمایند و همانطوریکه گفته شد به جزء دستورالعمل NOT سایر دستورالعملها دارای دو عملوند می‌باشند. TEST شبیه AND می‌باشد ولی نتیجه در جائی ذخیره نمی‌گردد. عملیات بعدی دستورالعمل‌های شیفت و چرخش می‌باشند که این دستورالعملها نیز روی بیت‌ها عمل می‌کنند و چنانچه تعداد بیت‌های مورد عمل بیش از یک باشد بایستی تعداد بیت‌ها را در ثبات CL قرار دهیم. دستورالعملهای شیفت عبارتند از SHL، SHR، SAL، SAR و دستورالعملهای چرخش عبارتند از ROL، ROR، RCL، RCR. در انتهای فصل عملیات مربوط به تغییر فلگ‌ها مورد بحث قرار گرفته و در نهایت با استفاده از چندین مثال کاربرد مطالب فوق نشان داده شده است که از جمله تبدیل حروف کوچک در یک متن به حروف بزرگ می‌باشد.

⌘ تمرین

١–اشکال دستورالعملهای زیر را مشخص نمائید.

```
ROL   DL, 2
SHL   BX , CX
MOV   X , [BX]
```

٢–مقدار CF و BL پس از اجرای دستورالعملهای ذیل چیست؟

```
MOV   CL, 4
MOV   BL, 0BBH
CLC
SAL   BL, CL
XOR   BL, CL
SAR   BL, CL
```

٣–مقدار CF و BX پس از اجرای دستورالعملهای ذیل چیست؟

```
MOV   CL, 3
MOV   BX, 2ACH
STC
ROL   BX, CL
INC   CL
RCL   BX, CL
```

٤–دستورالعملهای ذیل روی چه فلگ‌هائی اثر دارند؟

RCR , SHL, MOV, TEST, AND, XOR

٥–برنامه‌ای بنویسید که یک رشته را گرفته تعداد حروف کوچک آنرا مشخص نماید.

٦-برنامه‌ای بنویسید که بیت‌های شماره فرد ثبات AX را به یک و بیت‌های زوج ثبات AX را به صفر تبدیل نماید.

۷-برنامه‌ای بنویسید که مشخص نماید آیا بیت‌های شماره 7 ، 5 ، 2 ثبات BL برابر با صفر می‌باشد یا خیر؟

۸-برنامه‌ای بنویسید که مشخص نماید آیا بیت‌های شماره زوج ثبات AX برابر با یک می‌باشد یا خیر؟

۹- برنامه‌ای بنویسید که بیت‌های ثبات AX را آنقدر بطرف چپ شیفت دهد تا MSB آن برابر با یک گردد. در صورتیکه در ابتدا MSB برابر با یک می‌باشد کنترل به EXIT منتقل گردد.

۱۰-پس از اجرای دستورالعملهای ذیل مقادیر ثباتها چیست؟

```
MOV   AX, 4BCH
MOV   DX, 0F2BCH
XOR   AX, DX
SUB   DX, 8
NOT   DX
ADD   AX, 16
AND   DX, AX
```

۱۱-با چه دستوری مقادیر IF, DF, CF برابر با یک می‌شود.

۱۲-به چند طریق می‌توان مقدار ثبات AX را برابر با صفر قرار داد؟

۱۳-برنامه‌ای بنویسید که مقدار بیت شماره n ام ثبات AX را مشخص نماید.

فصل هفتم

ماکروها و روال‌ها و وقفه‌ها

هدف کلی

معرفی پشته، مکرو، روال، وقفه‌ها.

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا می‌شوید.

۱-تعریف پشته، کاربرد و دستورالعملهای مربوطه.

۲-تعریف روال، کاربرد، نحوه ایجاد و فراخوانی آنها.

۳-تعریف ماکرو، نحوه ایجاد، کاربرد و فراخوانی آنها.

٤-عملگرهای ماکرو.

۵-وقفه‌ها و دستورالعملهای مربوطه.

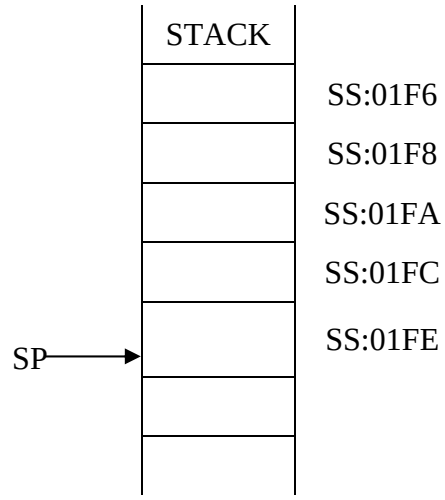
٦-روالهای تبدیل باینری به اسکی و بالعکس.

۷- اندازه‌گیری زمان اجرای برنامه‌ها.

۸- ایجاد تأخیر.

۱-۷- پشته (Stack)

پشته قسمتی از حافظه اصلی یا RAM می‌باشد. پشته خاصیت LIFO دارد یعنی عنصری که آخر وارد پشته می‌شود اولین عنصری است که از پشته خارج می‌شود. مقادیری که پشته در خود ذخیره می‌نماید از نوع word می‌باشد. ثبات SP به عنصر top پشته اشاره می‌نماید.



ثبات SP همیشه به آخرین word ای که وارد پشته شده اشاره می‌نماید. عملیاتی که روی پشته انجام می‌شوند عبارتند از PUSH و POP که در ذیل تشریح می‌گردد. همانطوریکه قبلاً گفته شد ثبات SS به ابتدای سگمنت پشته اشاره می‌نماید.

۱-۱-۷- دستورالعمل PUSH

شکل کلی دستورالعمل PUSH بصورت زیر می‌باشد.

PUSH opr

الف) opr بایستی از نوع word باشد.

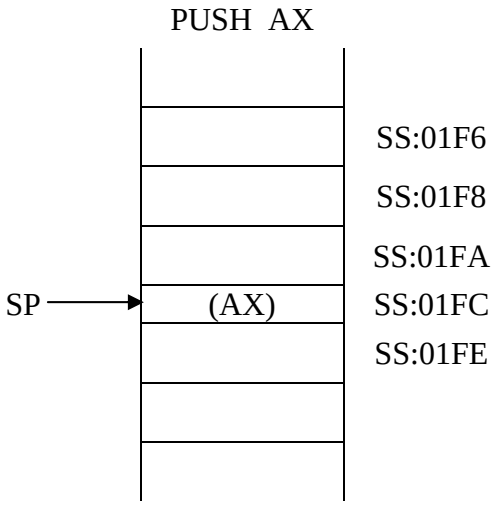
ب) opr بایستی متغیر یا ثبات باشد. ثابت نمی‌تواند باشد.

ج) opr می‌تواند یکی از ثباتهای DS، SS، ES باشد.

د) دستورالعمل PUSH روی هیچ فلگی اثر ندارد.

ه)این دستورالعمل opr را در پشته قرار می‌دهد و ثبات SP را باندازه دو واحد کاهش می‌دهد.

مثال ۱-۷



۲-۱-۷- دستورالعمل POP

شکل کلی دستورالعمل POP بصورت زیر می‌باشد.

POP dst

الف) dst بایستی از نوع word باشد.

ب) dst بایستی متغیر یا ثبات باشد.

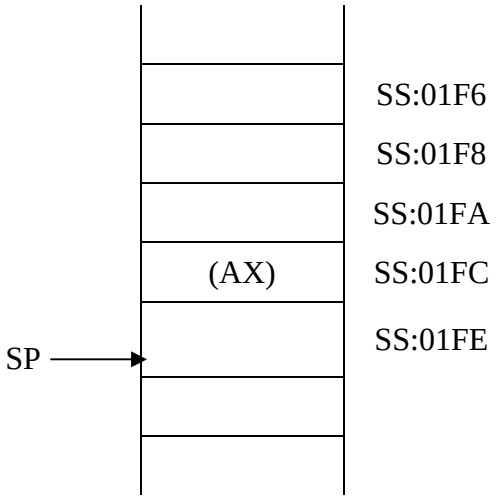
ج) dst می‌تواند یکی از ثباتها DS، SS، ES باشد.

د) دستورالعمل POP روی هیچ فلگی اثر ندارد.

ه) این دستورالعمل عنصری را که بوسیله SP مشخص می‌شود در dst قـرار داده و ثبات SP را باندازه دو واحد افزایش می‌دهد.

مثال ۲-۷

```
POP    AX
```



مثال ۳-۷

```
PUSH    AX
PUSH    BX
PUSH    ES
PUSH    DI
        ⋮
POP     DI
POP     ES
POP     BX
POP     AX
```

۳-۱-۷- دستورالعمل PUSHF

شکل کلی دستورالعمل PUSHF بصورت **زیر** می‌باشد.

PUSHF

الف) این دستورالعمل فاقد عملوند می‌باشد.
ب) این دستورالعمل روی هیچ فلگی اثر ندارد.
ج) این دستورالعمل محتوی ثبات فلگ را در پشته قرار می‌دهد. و ثبات SP را به اندازه دو واحد کاهش می‌دهد.

مثال ٤-٧

PUSHF

٤-١-٧- دستورالعمل POPF

شکل کلی دستورالعمل POPF بصورت زیر می‌باشد.

POPF

الف) این دستورالعمل فاقد عملوند می‌باشد.
ب) این دستورالعمل مقداری که در پشته بوسیله ثبات SP اشاره می‌شود را به ثبات فلگ منتقل نموده و محتوی ثبات SP را دو واحد افزایش می‌دهد.
ج) روی کلیه فلگ‌ها اثر دارد.

مثال ٥-٧

POPF

٢-٧- روال (Procedures)

برای راحتی نوشتن برنامه‌ها، برنامه‌ها را می‌توان به تعدادی روال تقسیم نمود و نوشتن هر روال را به یک برنامه‌نویس واگذار کرد. شکل کلی روال‌ها بصورت زیر می‌باشد.

```
name    PROC    NEAR
          ⋮
        RET
name    ENDP
```

الف) PROC مخفف کلمه Procedure می‌باشد.

ب) name نام روال می‌باشد.

ج) RET مخفف کلمه return می‌باشد که کنترل را به برنامه فراخوان این روال برمی‌گرداند.

د) ENDP مخفف end of procedure می‌باشد که انتهای روال را مشخص می‌کند.

ه) دستورالعمل RET روی هیچ فلگی اثر ندارد.

ز) NEAR مشخصه روال می‌باشد. کلیه روال‌های یک برنامه بایستی دارای مشخصه NEAR باشند.

مثال ٦-٧

```
SAVEREG   PROC    NEAR
 PUSH     AX
 PUSH     BX
 PUSH     CX
 PUSH     DX
 PUSH     SI
 PUSH     DI
   RET
SAVEREG   ENDP
```

روال فوق برای ذخیره کردن مقادیر ثباتهای مشخص شده استفاده می‌شود. بمنظور فراخوانی یک روال از دستور CALL استفاده می‌گردد. شکل کلی دستور CALL بصورت زیر می‌باشد.

```
CALL    Name
```

الف) دستور CALL روی هیچ فلگی اثر ندارد.
ب) name اسم روالی می‌باشد که توسط دستورالعمل CALL فراخوانی می‌شود. پس از فراخوانی روال به محض رسیدن به دستور RET کنترل به دستورالعمل بعد از دستور CALL برمی‌گردد.

مثال ۷-۷

```
CALL    SAVEREG
```

بایستی توجه داشت که زبان اسمبلی برنامه اصلی را بعنوان یک روال با مشخصه FAR در نظر می‌گیرد که به محض رسیدن به RET برنامه اصلی، کنترل به سیستم عامل بر میگردد.

بنابراین در یک برنامه فقط یک روال اصلی با مشخصه FAR وجود دارد و برنامه می‌تواند شامل تعدادی روال با مشخصه NEAR باشد که در روال اصلی توسط دستورهای CALL فراخوانی گردند. متذکر میشویم که آدرسهای برگشت روال‌ها در پشته ذخیره می‌گردند. یک روال را می‌توان بارها فراخوانی نمود. در فصل دهم استفاده از روال‌ها نشان داده شده است. یک روال را می‌توان بصورت بازگشتی یا Recursive فراخوانی نمود.

۳-۷- ماکروها (Macros)

ماکروها عملاً قسمت‌هائی از برنامه مبداء (source program) می‌باشند که با استفاده از نام آنها می‌توان در هر جای برنامه مبداء درج نمود. در حقیقت ماکروها دنباله‌ای از دستورالعملهای اسمبلی می‌باشند که ممکن است چندین بار در برنامه ظاهر شوند. به جای چندین بار تایپ نمودن این دنباله از دستورالعملها می‌توان فقط نام آنها را در برنامه درج نمود. شکل کلی ماکروها بصورت زیر می‌باشد.

```
name    MACRO
        :
        ENDM
```

الف) name عبارتست از نام macro

ب) ENDM انتهای macro را مشخص می‌نماید.

ج) بین MACRO و ENDM دنباله‌ای از دستورالعملهای برنامه مبداء قرار می‌گیرد.

د) در زمان ترجمه برنامه به زبان ماشین ابتدا در برنامه به جای macro ها دستورالعملهای وابسته به ماکروها قرار داده می‌شوند سپس به زبان ماشین ترجمه می‌شوند.

ه) تفاوت بین روال و ماکرو در این است که ماکرو در برنامه در زمان اجرای برنامه وجود خارجی ندارد بلکه روال در زمان اجرای برنامه وجود داشته و فراخوانی می‌گردد.

ز) macro ها بایستی در ابتدای برنامه تعریف شوند.

مثال ۸-۷

```
TOT     MACRO
        MOV  AX,X
        ADD  AX,Y
        ENDM
```

ماکرو TOT مجموع محتوی دو متغیر Y و X را محاسبه و در ثبات AX قرار می‌دهد.

حال برنامه زیر را در نظر بگیرید که از این ماکرو استفاده می‌نماید.

```
   X      DW     ?
   Y      DW     ?
   S      DW     ?
MOV    X, 1000
MOV    Y, 3000
TOT
MOV    S , AX
SUB    X, 10
TOT
ADD    S, AX
⋮
```

ابتدا این برنامه قبل از اجرا یعنی دز زمان ترجمه بصورت زیر در می‌آید.

```
   X      DW     ?
   Y      DW     ?
   S      DW     ?
MOV     X, 1000
MOV     Y, 3000
; macro جایگزین
MOV     AX, X
ADD     AX, Y
MOV     S, AX
SUB     X, 10
; macro جایگزین
MOV     AX, X
ADD     AX, Y
;
ADD     S, AX
⋮
```

ماکروها می‌توانند دارای پارامتر باشند.

مثال ۹-۷

```
ADD_W   MACRO   T1, T2, SUM
        MOV     AX , T1
        ADD     AX, T2
        MOV     SUM, AX
        ENDM
```

ماکرو داده شده دارای سه پارامتر بنامهای T1, T2, SUM می‌باشد. در موقع استفاده از این ماکرو بایستی سه مقدار را بعنوان جـایگزینی ایـن پارامترهـا بـدهیم. اگر بنویسیم

```
ADD_W   PRICE, TAX, COST
```

دستورالعملهای ذیل به جای این دستورالعمل قرار می‌گیرد.

```
MOV     AX, PRICE
ADD      AX, TAX
MOV     COST, AX
```

چنانچه تعداد پارامترها در فراخوانی ماکرو کمتر از تعداد پارامترهـای مـاکرو باشد ایجاد خطا می‌نماید.

مثال ۱۰–۷

ماکرو EXCHG داده شـده در ذیـل دو مقـدار از نـوع word را جابـه‌جـا می‌نماید. لازم به توضیح است که به منظور جلوگیری از پاک شدن مقدار قبلی ثبات AX از دستورالعملهای PUSH و POP استفاده گردیده است.

```
EXCHG    MACRO    W1, W2
 PUSH    AX
 MOV     AX, W1
 XCHG    AX, W2
 MOV     W1, AX
 POP     AX
         ENDM
```

حال اگر ماکرو را بصورت زیر فراخوانی نمائیم.

```
EXCHG     X , Y
```

دستورالعملهای ذیل به جای آن در source برنامه قرار می‌گیرد.

```
PUSH    AX
MOV     AX, X
XCHG    AX, Y
MOV     X, AX
PUSH    AX
```

۱-۳-۷- دیرکتیوها Macro directives

در Macro Assembfer می‌توان از directive های متعـددی در ماکروهـا استفاده نمود که بشرح زیر می‌باشند.

```
LOCAL   directive
EXITM   directive
IFB     directive
IFNB    directive
IRP     directive
IRPC    directive
REPT    directive
```

در ذیل هر یک از موارد بالا مورد بحث قرار می‌گیرد.

شکل کلی IFB بصورت زیر می‌باشد. IFB مخفف IF BLANK می‌باشد.

```
IFB         < argument >
  ⋮
ENDIF

              یا

IFB         < argument >
  ⋮
ELSE
  ⋮
ENDIF
```

IFB ، درست است اگر، مقدار argument در موقع فراخوانی ماکرو blank یا خالی باشد.

مثال ۱۱-۷

```
ADD_B   MACRO     N1, N2, N3, N4
MOV     AL, N1
IFB     <N2>
ADD     AL, 20
ELSE
```

```
 ADD      AL, N2
ENDIF
 IFB      < N3 >
 ADD      AL , 30
 ELSE
 ADD      AL, N3
ENDIF
 IFB      < N4 >
 ADD      AL ,5
 ELSE
 ADD      AL, N4
ENDIF
ENDM
```

حال اگر ماکرو فوق را بصورت زیر فراخوانی نمائیم.

```
ADD_B      NUM1, , , NUM4
```

دستورالعمل ذیل جایگزین می‌گردد.

```
MOV   AL, NUM1
ADD   AL, 20
ADD   AL, 30
ADD   AL, NUM4
```

به جای دستورالعمل ADD _ B NUM1, NUM2 دستورالعملهای ذیـل قرار می‌گیرد.

```
MOV   AL, NUM1
ADD   AL, NUM2
ADD   AL, 30
ADD   AL, 5
```

شکل کلی IFNB بصورت زیر می‌باشد. IFNB درست است اگر، مقدار argument در موقع فراوانی ماکرو blank نباشد. IFNB به معنی IF NOT BLANK می‌باشد.

```
IFNB      < argument >
  ⋮
ELSE
  ⋮
ENDIF
```

یا

```
IFNB      < argument >
  ⋮
ENDIF
```

مثال ۱۲-۷

```
ADD_B    MACRO    X1, X2, X3, X4
  PUSH   AX
   MOV   AX, X1
   IFNB  < X2 >
   ADD   AX, X2
 ENDIF
   IFNB  < X3 >
   ADD   AX, X3
 ENDIF
   IFNB  < X4 >
   ADD   AX , X4
 ENDIF
 ENDM
```

حال اگر دستورالعمل زیر را بنویسیم.

```
ADD_B      N1, , N3, N4
```

دستورالعملهای ذیل جایگزین می‌گردد.

```
PUSH   AX
MOV    AX, N1
ADD    AX, N3
ADD    AX, N4
```

۲-۳-۷- دستورالعمل EXITM

شکل کلی دستور العمل EXITM بصورت زیر می‌باشد:

```
EXITM
```

از EXITM فقط در ماکروها استفاده می‌گردد. EXITM باعث توقف بسط ماکرو می‌شود بسته به نتایجی که بوسیله directive های شرطی (IF) بوجود می‌آید.

مثال ۱۳-۷

```
IFB       < NUM1 >
EXITM
ENDIF
```

چنانچه در موقع فراخوانی ماکرو مقدار NUM1 داده نشود بسط ماکرو متوقف می‌شود. و از کلیه دستورالعملهائی که در ماکرو بعد از EXITM داده شده صرفنظر می‌گردد.

۳-۳-۷- دستورالعمل IRP

شکل کلی دستورالعمل IRP بصورت زیر می‌باشد.

```
IRP     dummy, < argument-list >
        ⋮
ENDM
```

بـــه تعـــداد آرگومانهـــائی کـــه در < argument –list > وجـــود دارد دستورالعملهای بین IRP و ENDM تکرار خواهد شد. در هر بـار تکـرار بترتیـب به جای dummy مقادیر داخل < argument-list > قرار داده می‌شود.

مثال ١٤-٧

```
IRP     VALUE, <1,2,3,5,11,13,17,19,23>
DW      VALU*VALUE*VALUE
ENDM
```

ایجاد یک آرایه 9 عنصری از نـوع word بـا مقـادیر مکعـب نـه عـدد اول می‌نماید. یعنی با مقادیر 12167 ,…, 1331, 125, 27, 8, 1 .

1
8
27
125
1331
2197
4913
6859
12167

در حقیقت ایجاد دستورالعملهای ذیل می‌نماید.

```
DW      1 * 1 * 1
DW      2 * 2 * 2
DW      3 * 3 * 3
DW      5 * 5 * 5
DW     11 * 11 * 11
DW     13 * 13 * 13
DW     17 * 17 * 17
DW     19 * 19 * 19
DW     23 * 23 * 23
```

٤-٣-٧- دستورالعمل IRPC

شکل کلی دستورالعمل IRPC بصورت زیر می‌باشد.

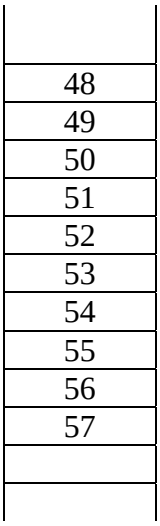
```
IRPC    dummy , string
  ⋮
ENDM
```

دستورالعملهای بین IRPC و ENDM به تعداد کارکترهای رشته string تکرار می‌گردد. در هر دفعه تکرار دستورالعملها به جای dummy بترتیب یکی از کارکترهای رشته string جایگزین می‌گردد. شبیه IRP می‌باشد با این تفاوت که آرگومانهایش بجای اعداد، متغیرهای رشته‌ای هستند.

مثال ٥-١٧

```
IRPC    CHAR, 0123456789
 DB     CHAR
ENDM
```

ایجاد یک رشته ده‌بایتی می‌نماید با مقادیر کد اسکی برای ارقام 0 تا 9.



در حقیقت ایجاد دستورالعملهای زیر می‌نماید.

```
DB   '0'
DB   '1'
DB   '2'
DB   '3'
DB   '4'
DB   '5'
DB   '6'
DB   '7'
DB   '8'
DB   '9'
```

۵-۳-۷- دستورالعمل REPT

شکل کلی دستورالعمل REPT بصورت زیر می‌باشد.

```
REPT    expression
  ⋮
ENDM
```

دستورالعملهای بین REPT و ENDM را به تعداد دفعاتی که توسط expression مشخص می‌گردد تکرار می‌کند.

مثال ۱۶-۷

```
ALLOCATE  MACRO  TLABEL , LNGTH
  TLABEL  EQU  THIS  BYTE
    VALU  = 0
    REPT  LNGTH  VALUE = VALUE + 1
          DB       VALUE

  ENDM

          ENDM
```

دقت کنید که در اینجا دو تا ENDM وجود دارد. یکی انتهای REPT و دیگری انتهای MACRO را مشخص می‌نماید. این ماکرو LNGTH بایت حافظه را تخصیص می‌دهد و مقادیر آنرا بترتیب 1 تا LNGTH قرار می‌دهد. پس از تعریف نمودن ماکرو ALLOCATE، می‌توان از این ماکرو با استفاده از دستورالعملهای زیر یک آرایه 40 بایتی بنام TABLE 1 ایجاد نمود.

```
 DATA_SEG       GSEGMENT  PARA  DATA  'DATA'
ALLOCATE   TABLE1 , 40
DATA_SEG   ENDS
```

از دیرکتیو EQU برای تخصیص ساده اسامی به اعداد، آدرسهای ترکیبی پیچیده و ... استفاده می‌گردد.

مثال ۱۷-۷

```
K    EQU   1024
```

در برنامه هر جا از نام K استفاده می‌گردد به جای آن 1024 قرار داده می‌شود.

مثال ۱۸-۷

```
SPEED    EQU    RATE
```

در برنامه هر جا از کلمه SPEED استفاده می‌شود به جای آن RATE قرار داده می‌شود.

مثال ۱۹-۷

```
TABLE    EQU  DS:[BP] [SI]
```

در برنامه هر جا از نام TABLE استفاده گردیده باشد به جای آن DS:[BP][SI] جایگزین می‌گردد.

مثال ۲۰-۷

```
COUNT    EQU  CX
```

هر جا از نام COUNT در برنامه استفاده شده باشد به جای آن ثبات CX قرار می‌دهد.

مثال ۲۱-۷

```
DBL_SPEED     EQU  2*SPEED
```

در برنامه هر جا از نام DBL_SPEED استفاده شده باشد به جای آن عبارت 2 * SPEED قرار می‌گیرد.

مثال ۲۲-۷

```
MINS_DAY    EQU  60*24
```

در برنامه هر جا از نام MINS_DAY استفاده شده باشد به جای آن 24 * 60 قرار می‌گیرد.

دیرکتیو= مانند EQU می‌باشد با این تفاوت که می‌توان آنرا مجدداً تعریف نمود یا به مقدار قبلی آن ارجاع نمود.

مثال ۲۳-۷

```
CONST=56        ; معادل EQU می‌باشد
CONST=75        ; مقدار آنرا مجدداً تعریف می‌نماید
CONST=CONST+1   ; به مقدار قبلی مراجعه می‌نماید
```

٦-۳-۷- دیرکتیو LOCAL

شکل کلی دیرکتیو LOCAL بصورت زیر می‌باشد.

```
LOCAL    dummy-list
```

این دیرکتیو باعث می شود که اسمبلر برای هر عنصر در dummy-list ایجاد یک سیمبل منحصر بفرد بنماید.

در حقیقت در ماکروها وقتی از label یا برچسب استفاده می‌نمائیم بسط آنها دچار اشکال می‌گردد. بعنوان مثال ماکرو زیر را در نظر بگیرید.

```
WAIT   MACRO    COUNT
PUSH   CX
MOV    CX , COUNT
NEXT :  LOOP   NEXT
POP    CX
ENDM
```

حال اگر در برنامه سه بار از این macro استفاده گردد.

```
  ⋮
WAIT    COUNT1
  ⋮
WAIT    COUNT2
  ⋮
WAIT    COUNT3
  ⋮
```

و ماکروها را بسط دهیم برنامه به صورت زیر نمایان می‌گردد.

```
⋮
PUSH   CX
MOV    CX, COUNT 1
NEXT : LOOP    NEXT
POP    CX
⋮
PUSH   CX
MOV    CX, COUNT 2
ENXT : LOOP    NEXT
POP     CX
⋮
PUSH    CX
MOV    CX , COUNT 3
NEXT : LOOP   NEXT
POP     CX
⋮
```

همانطوریکه مشاهده می‌نمائید دارای سهlabel یـا آدرس NEXT مـی‌باشـد که ایجاد ابهام می‌نماید. برای رفع این اشکال از دیرکتیو LOCAL استفاده می‌گردد.

```
 WAIT   MACRO       COUNT
LOCAL   NEXT
 PUSH   CX
  MOV   CX   ,  COUNT
NEXT :  LOOP     NEXT
  POP   CX
 ENDM
```

حال اگر در برنامه داده شده ماکرو WAIT را بسط دهیم برنامه بصورت زیر در می‌آید.

```
   ⋮
  PUSH     CX
  MOV      CX , COUNT
NEXT00 :   LOOP  NEXT00
  POP      CX
   ⋮
  PUSH     CX
  MOV      CX , COUNT
NEXT01:    LOOP NEXT01
  POP      CX
   ⋮
  PUSH     CX
  MOV      CX, COUNT
NEXT02:    LOOP NEXT02
  POP      CX
   ⋮
```

بایستی توجه داشت که در تعریف یک ماکرو می‌توان ماکروهای مختلفـی را فراخوانی نمود. در صورتیکه در تعریف یـک مـاکرو خـود آن مـاکرو را فراخـوانی نمائیم ماکرو را بازگشتی یا recursive macro نامیده میشود.

مثال ۲٤-۷

```
ADD_W   MACRO N1, N2, N3, N4, N5
  IFB   < N1 >
  MOV   AX, 0 ;; initialize sum
  ELSE
ADD_W   N2, N3, N4, N5
  ADD   AX , N1
ENDIF
        ENDM
```

در تعریف بالا چنانچه آرگومانی وجود نداشته باشد، مجموع برابر صفر و در غیر اینصورت مجموع تمام آرگومانها به جزء آرگومان اولی را محاسبه نموده و سپس اولین آرگومان به آن اضافه می‌نمائیم.

همانطوریکه قبلا متذکر شدیم ماکروهائی که در یک برنامه مورد استفاده قرار می‌گیرند بایستی در ابتدای برنامه تعریف نمود. از طرف دیگر می‌توان یک macro library ایجاد نمود. macro library یک فایل روی دیسک می‌باشد بنام MACRO.LIB که شامل تمام تعاریف ماکروهای مورد استفاده در برنامه‌های مختلفه می‌باشد. چنانچه از macro library بخواهیم استفاده نمائیم کافی است که دستورالعمل زیرا به عنوان اولین دستورالعمل برنامه قرار دهیم و دیگر نیازی به تعریف ماکروهای مورد نیاز برنامه در ابتدای برنامه نمی‌باشد.

```
INCLUDE     MACRO.LIB
```

چون فایل MACRO.LIB ممکن است دارای ماکروهائی باشد که مورد نیاز برنامه‌ای که می‌خواهد اجرا شود نباشد برای جلوگیری از اتلاف فضای حافظه می‌توان ماکروهای غیر ضروری برای برنامه را در دستور PURGE قرار داد تا آنها را به حافظه منتقل نکند.

مثال ۲۵-۷

```
INCLUDE   MACRO.LIB
PURGE     SHOW, CLS, LOCATE
```

ماکروهای SHOW, CLS, LOCATE به حافظه منتقل نمی‌گردند چون در برنامه این ماکروها فراخوانی نشده‌اند.

۷-۳-۷- عملگرهای ماکرو

دو عملگر ماکرو در Macro Assembler عبارتند از

&
;;

شکل کلی عملگر ;; عبارتست از

```
;; comment
```

عملگر ;; باعث می‌شود که اسمبلر comments ها را در موقع بسط ماکرو حذف نماید.

برنامه‌های بدون comments فضای کمتری را اشغال نموده و سریعتر اجرا می‌شوند. چنانچه بخواهید comment ای را در موقع بسط ماکرو حذف نکنید بایستی از ; استفاده گردد.

```
; comment
```

۸-۳-۷- عملگر &

شکل کلی عملگر & بصورت زیر می‌باشد.

```
text & text
```

این عملگر دو text یا سیمبل را بهم وصل نموده در کنار یکدیگر قرار می‌دهد.

مثال ۲۶-۷

```
DEF_TABLE   MACRO   SUFFIX, LNGTH
TABLE & SUFFIX  DB  LNGTH  DUP (?)
ENDM
```

حال اگر ماکرو فوق را به صورت زیر فراخوانی نمائیم.

```
DEF_TABLE     A , 5
```

آنگاه اسمبلر آنرا به TABLEA DB 5 DUP(2) تبدیل می‌نماید.

ماکرو زیر باعث نمایش یک کاراکتر روی صفحه مانتور می‌شود.

```
SHOW   MACRO   character
;; Display  the  specified  character.
PUSH  AX

PUSH    DX
MOV    AH,2     ;; select, display option
MOV    DL,      'character'
INT    21H      ;; call type 21 interrupt
POP    DX
POP    AX
ENDM
```

به منظور نمایش کارکتر * روی صفحه مانیتور از دستورالعملهای ذیل استفاده می‌گردد:

```
MOV     AH , 2
MOV     DL , '*'
INT     21H
```

٤-٧- وقفه‌ها (Interrupts)

ریزپردازنده در یک کامپیوتر برنامه‌ها را بسادگی اجرا نمی‌کند. بلکه بعنوان تنظیم کننده سیستم درگیر چیزهائی میشود که اتفاق می‌افتند. بعنوان مثال وقتی که کلیدی روی صفحه کلید فشار می‌دهید ریزپردازنده بایستی دریابد که کدام کلید فشار داده شده و عمل مناسب آن کلید را انجام دهد. بعنوان مثال وقتی کلید Ctrl-Break را فشار می‌دهید عملی که بایستی انجام شود کاملاً متفاوت است با وقتی که کلید T را فشار می‌دهید. وقتی که data از disk به حافظه یا بالعکس منتقل می‌گردد این مسئولیت ریزپردازنده است که دستورالعملهای مناسب برای اینکار را اجرا نماید. همانطوریکه گفته شد ریزپردازنده در کلیه کارهای کامپیوتر نقشی ایفاء می‌نماید. حال سوالی که مطرح می‌شود اینست که ریزپردازنده چگونه با وسائل جانبی درگیر می‌گردد؟ حقیقت امر این است که ریزپردازنده‌ها و وسائل جانبی بطرق مختلف با هم ارتباط برقرار می‌کنند. کاری که انجام می‌شود بدین صورت است که ریزپردازنده شروع به اجرای برنامه می‌نماید و به اجرای برنامه ادامه می‌دهد تا زمانیکه یک وسیله جانبی مانند صفحه کلید، دیسک، یا مانیتور به ریزپردازنده اعلام نماید که به کمک ریزپردازنده نیاز دارد. البته وسائل جانبی در حقیقت با ریزپردازنده صحبت نمی‌کنند بلکه آنها سیگنال نیاز به کمک خود را از طریق وقفه یا Interrupt ارسال می‌نمایند.

۱-٤-٧- نحوه کار وقفه‌ها

وقتی که یک وسائل جانبی اقدام به ارسال سیگنال وقفه می‌نماید شماره شناسائی خود را که type code نامیده می‌شود نیز ارسال می‌کند. هر وسیله جانبی

از قبیل keyboard ، disk drive ، floppy drive و ... دارای type code مختلفی می‌باشد. در حقیقت 256 نوع مختلف type code وجود دارد. هر وقت درخواست کمکی از طرف یک وسیله جانبی شود (Interrupt) ریزپردازنده اگر کاری که در حال انجام آن می‌باشد بتواند موقتاً رها نماید، اینکار را انجام داده و به کمک وسیله جانبی می‌رود (البته با حفظ موقعیت فعلی). پس از تکمیل کار وسیله جانبی مجدداً کار قبلی خود را از سر می‌گیرد. در صورتیکه ریزپردازنده نتواند کار فعلی خود را رها نماید پس از تکمیل این کار به کمک وسیله جانبی می‌رود.

همانطور که متذکر شدیم 256 تا type code وجود دارد به شماره‌های 255 تا 0. ریزپردازنده از این type code استفاده نموده آدرسی را در ابتدای حافظه محاسبه نموده و از آدرس محاسبه شده آدرس دیگری را می‌خواند. این آدرس جدید Interrupt Vector نام دارد که در حقیقت آدرس برنامه‌ای است که کار آن وقفه را عهده‌دار می‌باشد. در مورد وسیله جانبی استاندارد، برنامه‌های سرویس دهنده وقفه‌ها (Interrupt servicing programs) در تراشه ROM ذخیره می‌گردد. در بسیاری از کامپیوترها این ROM به Basic Input/Output system یا BIOS معروفست.

حال که ریزپردازنده می‌داند کدام برنامه را بایستی اجرا کند این کار را انجام داده یعنی برنامه را اجرا نموده و پس از اتمام اجرای برنامه به کار قبلی خود برگشته و اجرای آن را دنبال می‌کند.

۲-٤-۷- منابع وقفه‌ها

وقفه‌هائی که تاکنون بیان شده است وقفه‌های خارجی (External Interrupt) می‌باشد که توسط وسائل یا تجهیزات جانبی فعال می‌گردند. این نوع وقفه‌ها قسمتی از 256 نوع وقفه را پوشش می‌دهد. سایرین می‌توانند یکی از دو نوع وقفه ذیل باشند.

۱-برنامه‌ها همچنین می‌توانند با استفاده از دستورالعملهای خاص ایجاد وقفه در برنامه، وقفه‌ها را فعال نمایند.
۲-درموارد خاص ریزپردازنده حتی می‌تواند به خودش وقفه بدهد. بعنوان مثال وقتی که شما سعی در تقسیم بر صفر دارید.

۳-٤-۷- وقفه‌های رزرو شده (Reserved Interrupts)

از 256 وقفه، 32 تای اول یعنی شماره 0 تا 31 بوسیله Intel رزرو گردیده است. وقفه‌های نوع 255 تا 32 برای موارد دیگر استفاده می‌گردند. نمونه‌ای از وقفه‌های رزرو شده در ذیل داده شده است:

Type 0,	Divide Error
Type 1,	Single – Step
Type 2,	Nonmaskable Interrupts
Type 3,	Breakpoint
Type 4,	Overflow
Type 5,	Bound Range Exceeded
Type 6,	Invalid Table Limit Too Small
Type 7,	Processor Extension Not Available
Type 8,	Interrupt Table Limit Too Small
Type 9,	Processor Extension Segment Overrun
Type 13,	Segment Overrun
Type 16	Processor Extension Error

٤-٤-۷- وقفه‌های سیستم

در کامپیوترها، 1024 بایت اول حافظه یعنی محلهای حافظه باآدرس 0 تا 3FF تخصیص به جدولی دارد که این جدول بنام interrupt vector table معروفست. این جدولی است با آدرسهای 32 بیتی که به interrupt service routines در کامپیوتر اشاره می‌کنند. 256 وقفه مختلف به شماره‌های 0 تا 255 یا FF تا 0 در مبنای شانزده وجود دارد. ریزپردازنده Intel ، 32 وقفه اول یعنی

وقفه‌های شماره 1FH تا 0 را برای استفاده خودش در نظر می‌گیرد. 32 وقفه بعدی یعنی شماره‌های 3FH تا 20H برای استفاده سیستم عامل DOS در نظر گرفته شده است.

وقفه از طریق دستورالعمل‌های وقفه در برنامه یا تجهیزات خارجی (external devices) در سیستم فعال می‌گردد.

وقتی که ریزپردازنده یک وقفه دریافت می‌نماید شماره وقفه را در 4 ضرب نموده تا آدرس interrupt vector در جدول را بدست آورده سپس محتوی آدرس بدست آمده را در ثبات IP و ثبات CS قرار می‌دهد و شروع به اجرای دستورالعملها در آن آدرس می‌نماید. بعنوان مثال اگر وقفه از نوع 4AH باشد.

$$4AH * 4 = 128H$$

محلی از حافظه که آدرس آن 128H می‌باشد شامل آدرس Interrupt service routine وقفه نوع 4AH می‌باشد.

۵-٤-۷- وقفه‌های DOS

همانطور که قبلاً متذکر شدیم وقفه‌های نوع 3FH تا 20H برای سیستم عامل DOS رزرو گردیده است. اکثر این وقفه‌ها در سیستم عامل DOS کاربرد دارند بنابراین در اینجا آنها را مورد بحث قرار نمی‌دهیم. فقط دقت داشته باشید که نوع 21 دارای گزینه‌هائی برای برقراری ارتباط باkeyboard ، display ، printer، diskو وسائل ارتباطی غیر همزمان (asynchronous communicationsdevice) می‌باشد که در بخش بعدی این فصل مورد بحث قرار می‌گیرد.

شماره وقفه	نام
20	Terminate Program
21	Function Calls
22	Terminate
23	Ctrl-Break Exit Address
24	Critical Error Handler
25	Absolute Disk Read
26	Absolute Disk Write
27	Terminate, But Stay Resident
28	Reserved for Dos

٦-٤-٧- دستورالعملهای وقفه

در زبان اسمبلی در مورد وقفه‌های برنامه می‌توان از دستورالعملهای ذیل استفاده نمود:

INT
INTO
IRET

شکل کلی دستورالعمل INT بصورت زیر است:

INT Interrupt-type

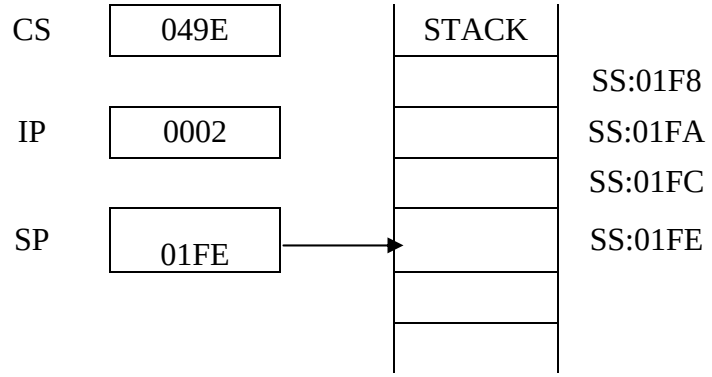
الف) interrupt – type یکی از نوع‌های وقفه 0 تا 255 می‌باشد.
ب) مقدار فلگهای IF و TF را صفر قرار می‌دهد. و روی سایز فلگها اثر ندارد.
ج) محتوی ثبات فلگ (flag register) را وارد stack می‌نماید.
د) محتوی ثبات CS را وارد stack می‌نماید.
ه) آدرس interrupt vector را محاسبه می‌نماید (شماره وقفه را در 4 ضرب می‌کند.)
و) محتوی ثبات IP را وارد stack می‌نماید.

ز) محتوی word اول interrupt vector را در ثبات IP و محتوی word دوم آنرا در ثبات CS قرار می‌دهد.

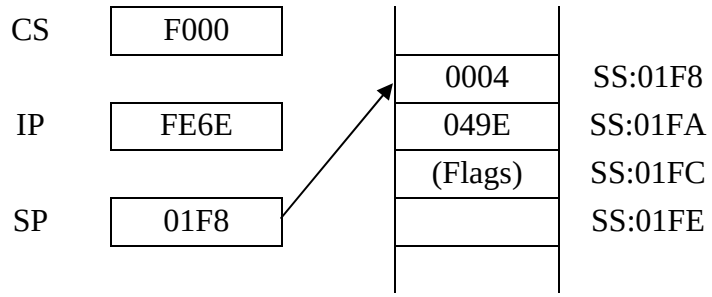
مثال ۲۷-۷

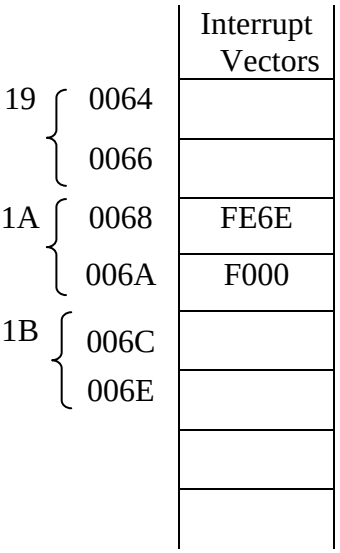
INT 1 AH

مقادیر ثباتها و پشته قبل از اجرای دستور فوق:



مقادیر ثباتها و پشته بعد از اجرای دستور فوق:





شکل کلی دستورالعمل INTO بصورت زیر می‌باشد. این دستورالعمل یک دستورالعمل وقفه شرطی است و مخفف کلمات Interrupt If Overflow می‌باشد.

INTO

الف) دارای عملوند نمی‌باشد.

ب) مقادیر فلگهای TF و IF را صفر می‌کند.

ج) این دستورالعمل ایجاد وقفه می‌نماید اگر مقدار فلگ OF برابر یک باشد.

د) این دستورالعمل وقفه نوع 4 را فعال می‌نماید.

شکل کلی دستورالعمل IRET بصورت زیر می‌باشد. IRET مخفف Interrupt Returnمی‌باشد.

IRET

الف) روی تمام فلگها اثر دارد.

ب) این دستورالعمل دارای عملوند نمی‌باشد.

ج) این دستورالعمل برای وقفه‌ها همان کاری را انجام می‌دهد که RET برای procedure callها. یعنی باعث برگشت کنترل به برنامه اصلی می‌شود.

بهمین دلیل IRET بایستی آخرین دستورالعملی باشد که ریزپردازنده در یک Interrupt Service Routine اجرا می‌نماید.

د) IRET باعث می‌شود که سه مقدار 16 بیتی از پشته خارج شده و بترتیب در ثباتهای Flag register, CS, IP قرار گیرد.

ه) مقادیر سایر ثباتها ممکن است از بین بروند مگر اینکه Interrupt service routine صریحاً آنها را ذخیره نماید.

۷-٤-۷- فراخوانی تابع وقفه نوع 21

در صفحه بعد قسمتی از فراخوانی‌های تابع وقفه نوع 21H یعنی Type 21H function calls برای سیستم عامل DOS داده شده است.

جدول ١-٧

Function calls with the Type 21 interrupt.

AH	Operation	Input Values	Results*
	Asynchronous Communications Functions		
3	Wait for Asynchronous Input character	None	AL=Character
4	Output a character to asynchronous device	DL = Character	None
	File Management Functions		
D	Reset default disk drive	None	None
E	Select default disk drive	DL=Drive Number (0=A,1= B, 2 = C)	AL=Number of disk drive (2 for single drive)
19	Get default drive code	None	AL=Default drive code (0 = A, 1 = B, 2=C)
2E	Set verify state (See also function 54)	DL=0 AL=0 to turn verify off =1 t turn verify on	None
30	Get DOS version number	None	AL=Version number (3 or 2) (Version 1.x returns 0) AH= Revision number BX,CX=0

Function calls with the Type 21 interrupt (continued).

AH	Operation	Input Values	Results*
	Interrupt Vector Functions		
25	Set interrupt vector	DS:DX=Vector address AL =Interrupt number	None
35	Read interrupt vector address	AL =Interrupt number	ES:BX= Vector address
	Directofy Functions		
39	Create a directory (MKDIR)	DS:DX=Address Of ASCIIZ String for directory	None **
3A	Remove a directory (RMDIR)	DS:DX=Address Of ASCIIZ String for directory	None **
3B	Change the directory (CHDIR)	DS:DX=Address Of ASCIIZ String for new directory	None **
47	Get current directory	DL=Drive Number (0 = default, 1=A, etc.) DS:SI=Address Of 64-byte buffer	DS:SI = Address of ASCIIZ string **

Function calls with the Type 21 interrupt (continued).

AH	Operation	Input Values	Results*
	Extended File Management Functions		
36	Get free disk space	DL=Drive number (0=default, 1=A, etc.)	AX=0FFFFH if invalid =Sectors per cluster BX=No. of free clusters DX=Total no. of clusters CX=Bytes per sector
3C	Create a file	DS:DX=Address Of ASCIIZ string CX=Attribute of file	AX=File handle **
3D	Open a file	DS:DX=Address of ASCIIZ string AL=0 to open for reading = 1 to open for writing = 2 to open for reading and writing	AX=File handle **
3E	Close a file handle	BX=File handle	None **
3F	Read from file or device	BX=File handle CX=No. of bytes to read DS:DX=Buffer address	AX=No. of bytes read** = 0 if read from end of file
40	Write to a file or device	BX=File handle	AX=No. of bytes written **

Function calls with the Type 21 interrupt (continued).

AH	Operation	Input Values	Results*
	Extended File Management Functions (continued)		
		CX=No. of bytes to write DS:DX=Buffer address	
41	Delete a file	DS:DX=Address of ASCIIZ string	None**
43	Get file attribute	AL=0 DS:DX=Address of ASCIIZ string for file	CX=Attribute**
43	Set file attribute	AL=1 DS:DX=Address of ASCIIZ string for file CX=Attribute	None **
54	Get verify state (See also function 2E)	None	AL=0 if verify is off = 1 if verify is on
56	Rename a file	DS:DX=Address of ASCIIZ string for old name ES:DI=Address of ASCIIZ string	None **

Function calls with the Type 21 interrupt (continued).

AH	Operation	Input Values	Results*
	Extended File Management Functions (continued)		
	For new Name		
Process Management functions			
31	Keep process	AL=Return code DX=Memory size, in paragraphs	None
4B	Load and execute a program	AL=0 DS:DX=Address of ASCIIZ string for program ES:BX=Address of parameter block	None **
4B	Load overlay	AL=3 DS:DX=Address of ASCIIZ string for program ES:BX=Address of parameter block	None **
4C	End process	AL=Return code	None
4D	Get return code of child	None	AX=Return code process

Function calls with the Type 21 interrupt (continued).

AH	Operation	Input Values	Results*
	Process Management Functions (continued)		
62	Get PSP	None	BX=Segment address of PSP
	Memory Management Functions		
48	Allocate memory	BX=Number of paragraphs requested	AX=Segment address of allocated memory**
49	Free allocated memory	ES=Segment address of memory to be freed	None **
4A	Set block	BX=Number of paragraphs ES= Segment address of memory area	None **
	Get Extended Error Function		
59	Get extended error	BX=0	AX= Extended code BH= Error class BL= Suggested action CH= Locus

*Besides the registers listed in this column, only AX and the flags are affected.
**If an error occurs, these function calls return CF=1 and an error code in AX. See the folowing table for the meanings of the error codes.

Function Call Error Reports

Most of the Directory and Extended File Management functions return CF= 0 if the operation is successful and CF= 1 if an error occurred. Along with CF=1, they return and error code in AX; the folowing table tells what these codes mean.

جدول ۲-۷

Error codes for DOS function calls.

Code	Meaning
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files (no handles left)
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
15	Invalid drive was specified
16	Attempted to remove the current directory
17	Not same device
18	No more files
19	Disk is write-protected
20	Bad disk unit
21	Drive not ready
22	Invalid disk command
23	CRC error
24	Invalid length (during disk operation)
25	Seek error
26	Not an MS-DOS disk
27	Sector not found
28	Out of paper
29	Write fault
30	Read fault
31	General failure
32	Sharing violation

Error codes for DOS function calls (continued).

Code	Meaning
33	Lock violation
34	Invalid disk change
35	FCB unavailable
50	Network request not supported
51	Remote computer not listening
52	Duplicate name on network
53	Network name not found
54	Network busy
55	Network device no longer exists
56	Net BIOS command limit exceeded
57	Network adapter hardware error
58	Incorrect response from network
59	Unexpected network error
60	Incompatible remote adapt
61	Print queue full
62	Queue not full
63	Not enough space for print file
64	Network name was deleted
65	Access denied
66	Network device type incorrect
67	Network name not found
68	Network name limit exceeded
69	Net BIOS session limit exceeded
70	Temporarily paused
71	Network request not accepted
72	Print or disk redirection is paused
80	File exists
82	Cannot make
83	Interrupt 24 failure
84	Out of structures
85	Already assigned
86	Invalid password
87	Invalid parameter
88	Net write fault

With DOS Microsoft has also included a Get Extended Error function call (AH=59H) that provides more comprehensive error.

دستورالعملهای ذیل باعث پاک شدن صفحه مانیتور می‌شود.

```
MOV   AH , 00
MOV   AL, 03
INT    10 H
```

دستورالعملهای ذیل مکان نما یا Cursor را در سطر 13 ستون 40 قرار می‌دهد. همانطور که می‌دانید صفحه مانیتور بصورت یک صفحه شطرنجی در نظر گرفته می‌شود که دارای 80 ستون و 25 سطر می‌باشد. شماره سطرها و ستونها از صفرشروع می‌شوند.

```
MOV   AH, 2
MOV   BH, 0
MOV   DH, 13 ; row
MOV   DL, 40 ; column
INT    10 H
```

دستورالعملهای ذیل یک کارکتر را از صفحه کلید گرفته، آنرا در ثبات AL قرار می‌دهد و سپس روی صفحه مانیتور نمایش می‌دهد.

```
MOV   AH, 1
INT    21H
```

دستورالعملهای ذیل یک کارکتر را از صفحه کلید گرفته، آنرا در ثبات AL قرار می‌دهد. (روی صفحه مانیتور نمایش نمی‌دهد)

```
MOV   AH, 7
INT    21H
```

دستورالعملهای ذیل باعث می‌شوند که کنترل به سیستم عامل DOS برگردد.

```
MOV     AX, 4C00H
INT     21H
```

دستورالعملهای ذیل باعث می‌شود که کارکتر A روی صفحه مانیتور نمایش داده شود.

```
MOV     AH, 02H
MOV     DL, 65
INT     21H
```

دستورالعملهای ذیل باعث می‌شود مکان نما بابتدای سطر بعد منتقل گردد.

```
MOV     AH, 02H
MOV     DL, 0DH
INT     21H
MOV     DL, 0AH
INT     21H
```

بایستی توجه داشت که دستور MOV AH, 02H پس از دستور INT 21H نیازی نمی‌باشد.

برنامه ذیل جهت پاسخگوئی Y یا N به یک پیغام می‌باشد.

```
GET_KEY:    MOV     AH, 1
            INT     21H
            CMP     AL, 'Y'
            JE      YES
            CMP     AL, 'N'
            JE      NO
            JNE     GET_KEY
```

در ذیل جدول مربوط به عملیات صفحه کلید برای وقفه نوع 21 داده شده است.

جدول ۳-۷

Keyboard operations with Type 21 interrupt.

AH	Operation	Input Values	Results
1	Wait for keyboard character, then display it (with Ctrl-Break check)	None	AL=Character
6	Read keyboard character (no Ctrl-Break check)	DL=0FFH	AL=Character, if available =0, if no character is available
7	Wait for keyboard character, but do not display it (no Ctrl-Break check)	None	AL=Character
8	Same as function 7, but with Ctrl-Break check	None	AL=Character
A	Read keyboard string into buffer	DS:DX=Buffer address First buffer byte= Buffer size	Second buffer byte =Number of chars. read
B	Read keyboard status	None	AL=0FFH if no character is available =0 if character is available
C	Clear keyboard buffer and call a keyboard function	AL=Keyboard function number (1, 6, 7, 8, or A)	Per keyboard function

جدول مربوط به عملیات صفحه نمایش برای وقفـه 21 در ذیـل داده شـده است.

جدول ٤-٧

Video operations with Type 21 interrupt.

AH	Operation	Input Values	Results
2	Display a character (with Ctrl-Break check)	DL= Character	Cursor follows character
5	Print a character	DL=Character	None
6	Display a character (no Ctrl-Break check)	DL=Character	Cursor follows character
9	Display a string	DS:DX= String address. String must end with $.	Cursor follows string

۵-۷- خواندن رشته‌ها

در بسیاری از برنامه‌ها نیاز به وارد نمودن نام یا آدرس یا بطور کلی رشته‌ای از طریق صفحه کلید می‌باشد. برای اینکار شما بایستی در data segment فضائی باندازه ماکزیمم تعداد کارکترهائی که ممکن است از صفحه کلید داده شود بعلاوه 2 روزو نمود. این شامل

الف)یک فضا از نوع بایت که مقدار آن معادل تعداد ماکزیمم کارکترهای ورودی بعلاوه یک می‌باشد.

ب)یک فضا از نوع بایت که مقدار آن بوسیله تابع A یا 10H مشخص می‌گردد و عبارتست از تعداد کارکترهای واقعی که از صفحه کلید داده شده است.

ج) یک بلوک از بایتها بدون مقدار اولیه که شامل کارکترهائی است که عملاً از طریق صفحه کلید وارد گردیده است.

بنابراین شکل کلی رشته بصورت زیر می‌باشد.

```
Stringname DB key-plus-one, keys-plus-one DUP(?)
```

بعنوان مثال برای رزرو فضائی تا 50 کارکتر دستورالعمل زیر را بایستی در data segment قرار داد.

```
USER_STRING DB 51,51 DUP (?)
```

حال برای دریافت رشته از صفحه کلید دستورالعملهای زیر را بایستی اجرا نمود.

```
LEA   DX, USER_STRING; make DX point to buffer
MOV   AH, 0AH; read the string
INT   21H
```

برنامه کامل آن بصورت زیر می‌باشد.

```
READ_KEYS   PROC  FAR
        PUSH   AX
        MOV    AX, DATA_SEG
        MOV    DS, AX
        LEA    DS, AX
        LEA    DX, USER_STRING
        MOV    AH, 0AH
        INT    21H
        SUB    CH, CH; Read Character count into CX
        MOV    CL, USER_STRING+1
        ADD    DX, 2; Make DX point to text
        POP    AX
        RET
READ_KEYS                ENDP
```

در بسیاری از موارد در برنامه‌ها پیغامی نمایش داده میشود و سپس از کاربر انتظار پاسخگوئی می‌باشد. بعنوان مثال شما از کاربر می‌خواهید که نامش را وارد نماید. برای اینکار ابتدا پیغامی که می‌خواهید روی صفحه مانیتور نمایش داده شود را در Data segment تعریف می‌نمائید. سپس با استفاده از روال READ_KEYS داده شده در بالا اینکار را انجام می‌دهید.

مثال ۲۸-۷

```
DATA_SEG PARA DATA 'DATA'
GET_NAME DB 'Please enter your name:  $'
DATA_SEG ENDS
```

و سپس در code segment دستورالعملهای ذیل را بایستی بدهیم.

```
LEA     DX, GET_NAME; display the prompt
MOV     AH , 9
INT     21h
CALL    READ_KEYS; Read the response
```

بایستی توجه داشت که DS:DX اشاره می‌کند به رشته‌ای که شـامل نام می‌باشد و محتوی CX برابر با تعداد کاراکترهای رشته می‌باشد.

٦-٧- عملیات date و time

جدول ذیل مربوط به فراخوانی‌های تابع وقفه نوع 21 در زمینه زمان و تاریخ می‌باشد.

جدول ٥-٧

AH	Operation	Input Values	Results *
Note : All time and date values are in binary.			
2A	Get date	None	CX=Year DH=Month DL=Day
2B	Set date	CX=Year (1980-2099) DH=Month DL=Day	AL=0 if date is valid =FF if date is invalid
2C	Get time	None	CH=Hours CL=Minutes DH=Seconds DL=1/100 seconds
2D	Set time	CH=Hours (0-23) CL=Minutes DH=Seconds DL=1/100 Seconds	AL=0 if time is valid = FF if time is invalid

* Besides the registers listed in this column, only AX and the flags are affected.

۱-٦-۷- اندازه‌گیری زمان اجرای برنامه‌ها

با توجه باینکه می‌توان زمان را برحسب صدم ثانیه در کامپیوترها مطرح نمائیم از این ویژگی برای اندازه‌گیری زمان اجرای یک برنامه یا قسمتی از یک برنامه استفاده می‌شود. برای اینکار قبل از شروع اجرای برنامه زمان یا time را می‌خوانیم آنگاه برنامه را اجرا نموده سپس زمان یا time را می‌خوانیم. تفاضل دو زمان خوانده شده زمان اجرا را نشان می‌دهد. در ذیل برنامه‌ای داده شده که زمان اجرای یک روال بنام SORT را اندازه می‌گیرد.

Calculate execution time.

```
;       This sequence calculates the execution time of a program
;       Called SORT.
;       Results:    CH= Hours
;                   CL= Minutes
;                   DH= Seconds
;                   DL= 1/100 Seconds
```

; Put these temporary locations in the data segment:

```
HRS     DB    ?
MINS    DB    ?
SECS    DB    ?
HSECS   DB    ?
```

; Here is the timing sequence:

```
        MOV   AH, 2CH       ; Read the start time
        INT   21H
        MOV   HRS,CH        ; and save it
        MOV   MINS,CL
        MOV   SECS,DH
        MOV   HSECS,DL
       CALL   SORT          ; Execute the procedure
        MOV   AH, 2CH       ; Read the end time
        INT   21H
```

```
                SUB   DL,HSECS     : Calculate the difference
                JNC   SUB_SECS
                ADD   DL, 100
                DEC   DH
SUB_SECS:       SUB   DH, SECS
                JNC   SUB_MINS
                ADD   DH, 60
                DEC   CL
SUB_MINS:       SUB   CL, MINS
                JNC   SUB_HRS
                ADD   CL, 60
                DEC   CH
SUB_HRS:        SUB   CH, HRS
                JNC   DONE
                ADD   CH, 24
DONE:           RET
```

۲-٦-۷- ایجاد تأخیر (Generating delays)

در بعضی از برنامه‌ها نیاز به ایجاد تأخیر در تولید صوت از طریـق speaker یا در نمایش اشکال گرافیکی روی صفحه نمایش می‌باشد.

برنامه‌هائی که ایجاد تأخیر می‌نمایند بصورت زیر عمل می‌نمایند.

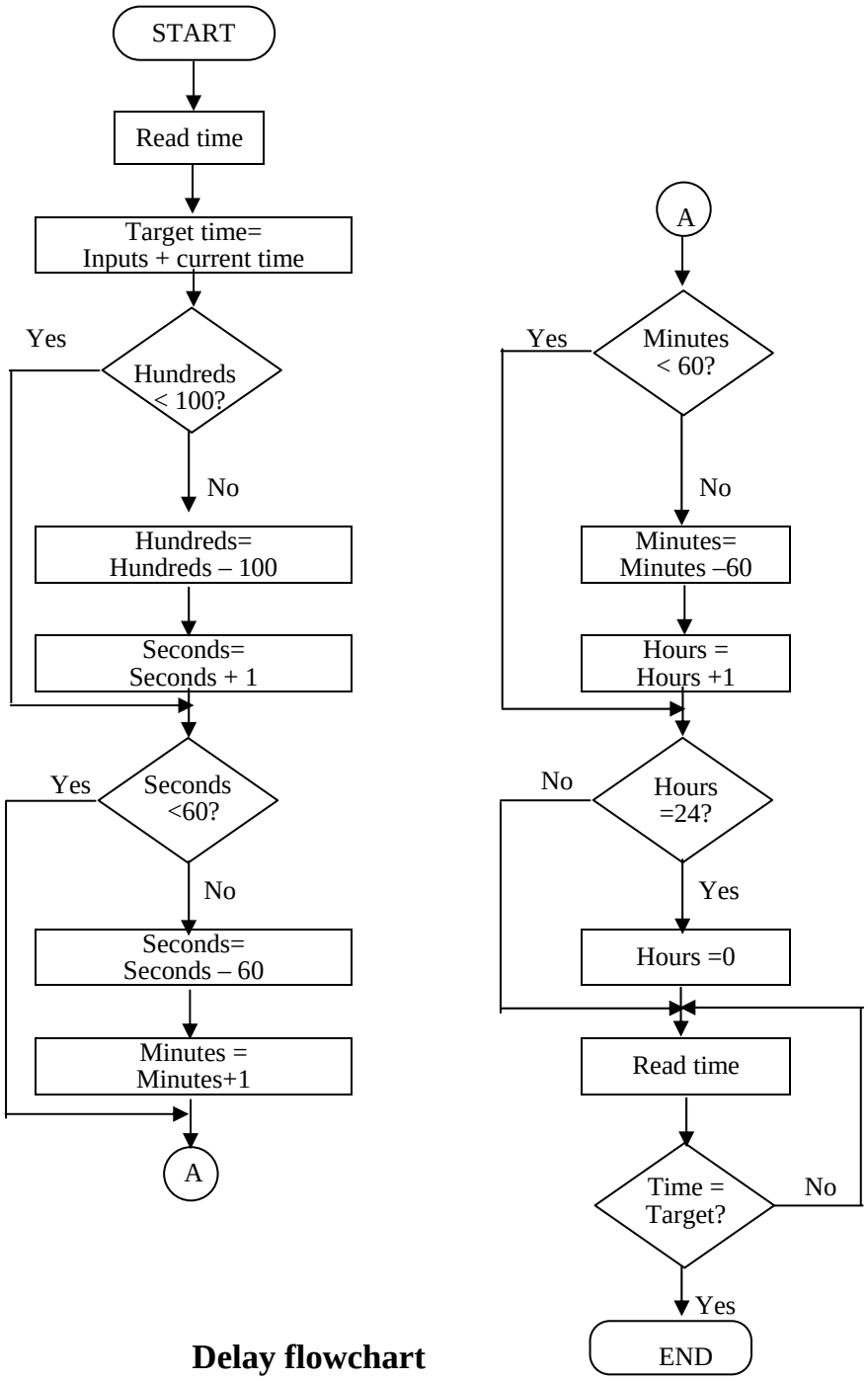
الف) وقت فعلی (Current time) را می‌خوانند.

ب) مقدار تأخیر را به وقت فعلی اضافه نموده تا زمان هدف (target time) بدست آید.

ج) زمان هدف را تنظیم می‌نمایند بطوریکه ساعت آن از 23 و دقیقه و ثانیـه آن از 59 بیشتر نشود.

د) زمان یا time را مکرراً می‌خوانند تا زمان فعلی بیشتر از زمان هدف شود.

نمودار ذیل مراحل ایجاد تأخیر را نشان می‌دهد. مدت زمان تأخیر برحـسب دقیقه، ثانیه، و صدم ثانیه داده می‌شود.



Delay flowchart

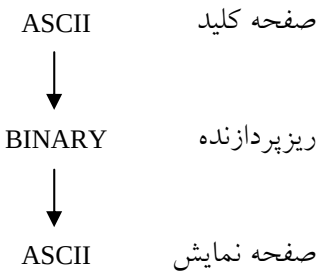
برنامه ایجاد تأخیر نیز در ذیل داده شده است.

```
Generate a delay.
;               Make the processor wait for a specified period.
;               Inputs:     AL= Minutes
;                           BH= Seconds
;                           BL=1/100 Seconds
;               All registers are preserved.

;               Assemble with: MASM DELAY;
;               Link with: LINK callprog + DELAY;


                PUBLIC      DELAY
CSEG            SEGMENT     PARA  PUBLIC     'CODE'
                ASSUME      CS: CSEG
DELAY           PROC        FAR
                PUSH        AX    ; Save affected registers
                PUSH        BX
                PUSH        CX
                PUSH        DX
                MOV         AH, 2CH   ; Read current time
                INT         21H
; Add the current time to the input values.
                MOV     AH, CH    ; Hours
                ADD     AL, CL    ; Minutes
                ADD     BH, DH    ; Seconds
                ADD     BL, DL    ; Hundredths
; Propagate any carryover.
                CMP     BL, 100       ;Hundredths must be <100
                JB      SECS
                SUB     BL, 100
                INC     BH
        SECS:   CMP     BH, 60        ; Seconds must be <60
                JB      MINS
                SUB     BH,60
                INC     AL
        MINS:   CMP     AL, 60        ; Minutes must be < 60
                JB      HRS
                SUB     AL, 60
                INC     AH
         HRS:   CMP     AH, 24        ; Hours must be < 24
                JNE     CHECK
                SUB     AH, AH
; wait for interval to elapse.
CHECK:          PUSH    AX            ; Read the time again
```

```
        MOV     AH, 2CH
        INT     21H
        POP     AX
        CMP     CX, AX      ; Compare hours and minutes
        JA      QUIT
        JB      CHECK
        CMP     DX, BX      ; Compare seconds and hunds
        JB      CHECK
QUIT:   POP     DX          ; Restore registers
        POP     CX
        POP     BX
        POP     AX
        RET                 ; Return to calling program
DELAY   ENDP
CSEG    ENDS
        END
```

۷-۷- کدهای اسکی و دودوئی

مقادیر یا اعدادی که در برنامه‌ها بعنوان ورودی از طریق صفحه کلید داده می‌شوند اگر بخواهیم مورد استفاده ریزپردازنده قرار گیرد جهت انجام عملیات ریاضی بایستی به دودوئی یا باینری تبدیل گردند. و چنانچه بخواهیم نتایج محاسبات را روی صفحه نمایشگر یا دستگاه چاپگر به نمایش درآوریم یا چاپ نمائیم بایستی تبدیل به اسکی گردند.



۱-۷-۷- تبدیل رشته‌های ASCII به دودوئی

همانطوریکه می‌دانیم کارکترهای 0 تا 9 دارای کداسکی 48 تا 57 می‌باشند. بشرح ذیل:

ASCII Value (Hex)	Decimal Digit
30	0
31	1
32	2
33	3
34	4
35	5
36	6
37	7
38	8
39	9

از طرف دیگر همانطوریکه می‌دانیم هر عدد را بصورت یک سری از توانهای 10 می‌توان نمایش داد.

مثال ۲۹-۷

$$472 = (2 *1) + (7*10) + (2*100)$$

یا

$$472=2*10^2 + 7*10^1 + 2* 10^0$$

و با توجه به آنکه در موقع ورود اعداد، در هر لحظه فقط یک رقم را وارد می‌نمائیم الگوریتم تبدیل بایستی وزن رقم را مدنظر قرار دهد. بعنوان مثال اگر کاربر عدد 95 را تایپ نماید به محض دریافت 9 الگوریتم بایستی آنرا در 10 ضرب نماید قبل از آنکه با عدد 5 جمع نماید. بطور کلی فرآیند تبدیل بایستی بطریق ذیل عمل نماید.

الف)الگوریتم تبدیل بایستی اولین رقم (با ارزشترین) را با حذف چهار بیت مرتبه بالا (بیت‌های 7 تا 4) کد اسکی به دودوئی تبدیل نماید و مقدار باینری بدست آمده را نگهداری نماید.
ب)الگوریتم بایستی ارقام بعدی را به دودوئی تبدیل نموده و نتیجه قبلی به دست آمده در مرحله الف را در 10 ضرب نموده با نتیجه بدست آمده در این مرحله جمع نماید.

مثال ۳۰-۷

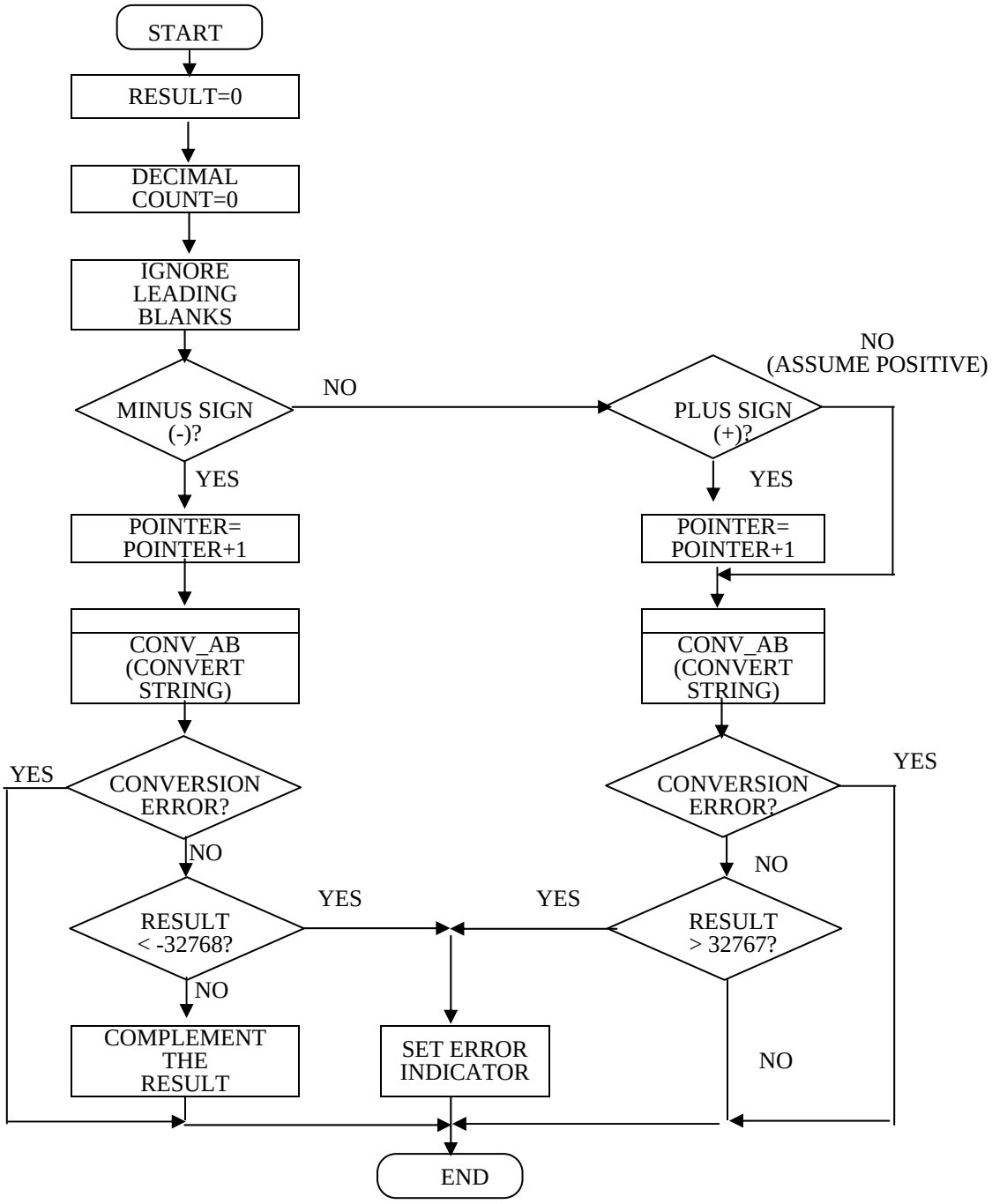
عدد 726 را در نظر بگیرید.

00110111	00110010	00110110
7	2	6

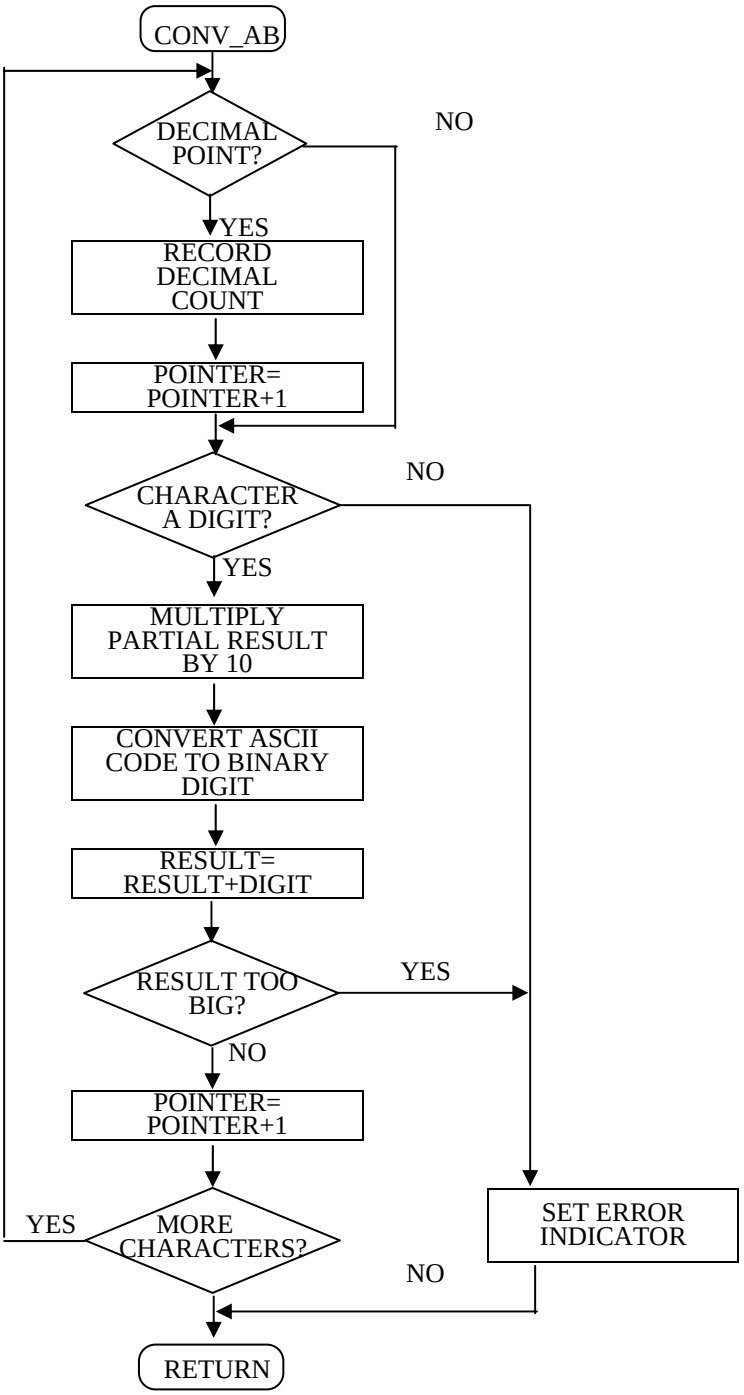
حال با توجه به الگوریتم داده شده

0111	7	
0010	2	
1000110	7*10	70
1001000	70+2	72
0110	6	
1011010000	72*10	720
1011010110	720+6	726

از طرف دیگر الگوریتم تبدیل بایستی قادر باشد که بتواند اعداد منفی را که بصورت رشته‌ای از کداسکی داده می‌شود به دودوئی تبدیل نماید. نمودار ذیل تبدیل یک مقدار بین 32767+ تا 32768– را انجام می‌دهد. در حقیقت نتیجه یک مقدار شانزده بیتی می‌باشد. روال CONV_AB عملاً کار تبدیل را بعهده دارد.


START
RESULT=0
DECIMAL COUNT=0
IGNORE LEADING BLANKS
MINUS SIGN (-)?
NO
PLUS SIGN (+)?
NO (ASSUME POSITIVE)
YES
YES
POINTER= POINTER+1
POINTER= POINTER+1
CONV_AB (CONVERT STRING)
CONV_AB (CONVERT STRING)
YES
CONVERSION ERROR?
CONVERSION ERROR?
YES
NO
NO
RESULT < -32768?
YES
YES
RESULT > 32767?
NO
COMPLEMENT THE RESULT
SET ERROR INDICATOR
NO
END

Algorithm to convert an ASCII string to binary.



Convert an ASCII String to Binary

```
;                   Converts an ASCII string to its 16-bit, two's –complement
;                   Binary equivalent.
;                   Inputs:     DS:DX= starting address of string
;                               CX= Character count
;                   Results:    CF= 0 indicates no error
;                               AX= Binary value
;                               DX=count of digits after decimal point
;                               DI=0FFH
;                               CF=1 indicates error
;                               DS:DI=Address of non-convertible character
;                   DX and CX are unaffected.

;                   Assemble with: MASM ASC_BIN;
;                   Link with: LINK callprog + ASC_BIN;
                    PUBLIC    ASCII _ BIN

CSEG                SEGMENT     PARA 'CODE'
                    ASSUME      CS : CSEG
ASCII _ BIN    PROC    FAR
                    PUSH        BX          ;Save BX and CX
                    PUSH        CX
                    MOV         BX,DX       ;Put offset in BX
                    SUB         AX,AX       ; To start, result =0
                    SUB         DA,DX       ; decimal count =0
                    MOV     DI,0FFH                 ;assume no bad characters
                    CMP     CX,7                    ; String too long?
                    JA      NO_GOOD                 ; If so, go set CF and exit
  BLANKS:           CMP     BYTE PTR [BX]           ; ' ' Scan past leading blanks
                    JNE     CHK_NEG
                    INC     BX
                    LOOP    BLANKS
  CHK_NEG:          CMP     BYTE PTR [BX]           ; '-' Negative number?
                    JNE     CHK_POS
                    INC     BX                      ; If so, increment pointer
                    DEC     CX                      ;decrement the count,
                    CALL    CONV_AB                 ; convert the string
                    JC      THRU
                    CMP     AX,32768                ; Is the number too small?
                    JA      NO_GOOD
                    NEG     AX                      ; No. complement the result
                    JS      GOOD
                    CMP     BYTE PTR [BX],'+'       ; Positive number?
                    JNE     GO_CONV
                    INC     BX                      ; If so, increment pointer
```

```
            DEC     CX              ; Decrement the count
GO_CONV:    CALL    CONV_AB         ; Convert the string
            JC      THRU
            CMP     AX,32767        ; Is the number too big?
            JA      NO_GOOD
GOOD:       CLC
            JNC     THRU
NO_GOOD:    STC                     ; If so, set carry flag
THRU:       POP     CX              ; Restore registers
            POP     BX
            RET                     ; and exit
ASCII_BIN   ENDP
```

; This procedure performs the actual conversion.

```
CONV_AB     PROC
            PUSH    BP              ;Save scratch registers
            PUSH    BX
            MOV     BP,BX           ;Put pointer in BP
            SUB     BX,BX           ; and clear BX
CHK_PT:     CMP     DX,0            ; was a decimal point found?
            JNZ     RANGE           ; If so, skip following check
            CMP     BYTE PTR DS:[BP], '.'     ; Decimal point?
            JNE     RANGE
            DEC     CX              ;If so, decrement count,
            MOV     DX,CX           ;and record it in DX
            JZ      END_CONV        ; Exit if CX=0
            INC     BP              ; Increment pointer
RANGE:      CMP     BYTE PTR DS:[BP], '0'     ; If the character is not a digit …
            JB      NON_DIG
            CMP     BYTE PTR DS:[BP],'9'
            JBE     DIGIT
NON_DIG:    MOV     DI,BP           ; put its address in DI,
            STC                     ;set the carry Flag,
            JC      END_CONV        ; and exit
DIGIT:      IMUL    AX,10           ;Multiply the digit in AX by 10
            MOV     BL, DS:[BP]     ;Fetch ASCII code
            AND     BX,0FH          ; save only high bits,
            ADD     AX,BX           ;and update partial result
            JC      END_CONV        ; Exit if result is too big
            INC     BP              ;Otherwise, increment BP
            LOOP    CHK_PT          ; and continue
```

```
                CLC             ; When done, clear carry flag
END_CONV:       POP     BX      ; Restore registers
                POP     BP
                RET             ; and return to caller
CONV_AB         ENDP
CSEG            ENDS
                END
```

به منظور بررسی درستی جواب بطریق ذیل عمل می‌نمائیم.

```
 CALL    ASCII_BIN    ; Call the conversion procedure
  JNC    VALID        ; Is the answer valid?
   OR    DI, DI       ; No. find the error condition
  JNZ    INV_CHAR
   OR    AX, AX
  JNZ    RANGE_ER

    ⋮                 ; String was too long

RANGE_ER:             ; Number out-of-range
INV_CHAR:             ; Invalid character
  VALID:              ; The answer is valid
```

به منظور نمایش نتایج روی صفحه نمایش یا چاپ آنها با استفاده از دستگاه چاپگر بایستی ابتدا آنها را از باینری به ASCII تبدیل نمود. برنامه زیر این کار را انجام می‌دهد.

BIN_ASC- Convert Binary to ASCII

```
;       Converts a signed binary number to a six-byte ASCII
;       String (sign plus five digits) in the data segment.
;       Inputs: AX= Number to be converted
;               DS: DX= starting address of string buffer
;       Results: DS:DX= Starting address of string
                CX= Character count
;       Other registers are preserved.

;       Assemble with: MASM BIN_ASC;
;       Link with: LINK callprog + BIN_ASC;
```

```
            PUBLIC      BIN_ASCII
CSEG        SEGMENT     PARA PUBLIC 'CODE'
            ASSUME      CS: CSEG
BIN_ASCII   PROC        FAR
            PUSH    DX  ; Save the caller's registers

                PUSH    BX
                PUSH    SI
                PUSH    AX
                MOV     BX, DX          ; Put offset in BX
                MOV     CX, 6           ; Fill buffer with spaces
FILL_ BUFF : MOV    BYTE  PTR  [BX], '  '
                INC     BX
                LOOP    FILL_BUFF
                MOV     SI, 10          ;Get ready o divide by 10
                OR      AX, AX          ; If value I negative,
                JNS     CLR_DVD
                NEG     AX              ; make if positive
CLR_DVD:        SUB     DX, DX          ; Clear upper half of dividend
                DIV     SI              ; Divide AX by 10
                ADD     DX, '0'         ; Convert remainder to ASCII digit
                DEC     BX              ; Back up through buffer
                MOV     [BX], DL        ; Store char. In the string
                INC     CX              ;Count converted character
                OR      AX,AX           ; All done?
                JNZ     CLR_DVD         ; If not, get next digit
                POP     AX              ; Yes. Get original value

                OR      AX,AX                   ;was it negative?
                JNS     NO_MORE
                DEC     BX                      ; Yes. Store sign
                MOV     BYTE PTR [BX], '-'
                INC     CX                      ; and increase character count
NO_MORE:        POP     SI                      ;Restore registers
                POP     BX
                POP     DX
                RET                             ; and exit
BIN_ASCII       ENDP
CSEG            ENDS
                END
```

مروری بر مطاب فصل

در این فصل پشته یا Stack تعریف گردیده پشته خاصیت LIFO دارد یعنی آخرین ورودی اولین خروجی از پشته می‌باشد. پشته در حقیقت قسمتی از حافظه را اشغال می‌نماید. عملیاتی که روی پشته انجام می‌شود عبارتست از PUSH و POP.

در این غصل همچنین نحوه تعریف ماکرو داده شده هر ماکرو با کلمه macro شروع و به endm ختم می‌شود. از ماکروها برای سهولت در نوشتن و تایپ برنامه‌ها استفاده می‌شود. ماکروها را در ابتدای برنامه بایستی تعریف نمود. امکان استفاده از روال نیز در زبان اسمبلی وجود دارد. روال‌ها در موقع اجرای برنامه فراخوانی می‌شوند. تفاوت ماکرو و روال در اینست که روال در زمان اجرای برنامه فراخوانی می‌شود در صورتیکه ماکروها در زمان ترجمه جایگزین می‌گردند. از تعدادی عملگر در ماکروها استفاده می‌شود که در این فصل بحث گردیده. معمولاً در اکثر برنامه‌ها از وقفه استفاده می‌گردد. در این فصل توابع مربوط به وقفه بصورت کامل بحث شده است. برای دادن داده‌ها به کامپیوتر بایستی از فرم ASCII آنها را به باینری تبدیل نمود و همچنین برای نمایش داده‌ها بایستی از فرم باینری آنها را به شکلASCII تبدیل نمود که برنامه‌های مخصوص آنها در این فصل تهیه و نوشته شده است. همچنین نحوه محاسبه زمان اجرای یک برنامه نیز بحث گردیده است.

⌘ تمرین

۱-تفاوت macro با procedure چیست؟

۲-marcroها در چه قسمتی از برنامه قرار می‌گیرند؟

۳-Procedure ها در چه قسمتی از برنامه قرار می‌گیرند؟

٤-کار عملگرهای ماکرو چیست؟

۵-کار REPT, EXITM, IFNB چیست؟ یک مثال برای هر کدام ارایه کنید.

٦-کار LOCAL چیست؟ یک مثال ارایه کنید.

۷-یک Macro بنویسید که مقادیر ثباتها را ذخیره نماید.

۸-Interrupt Vector چیست؟

۹-در چه قسمتی از حافظه جدول Interrupt vector قرار دارد؟

۱۰-برنامه‌ای بنویسید که یک پیغام دلخواه را نمایش دهد.

۱۱-روالی بنویسید که دو مقدار صحیح را گرفته مجموع آنها را نمایش دهد.

۱۲-روالی بنویسید که فاصله زمانی بین فشار دو دکمه را محاسبه نماید.

۱۳-روالی بنویسید که صفحه مانیتور را پاک نموده و مکان نما را در سطر 10 ستون 40 قرار داده آنگاه کارکتر * را نمایش دهد.

۱٤-یک Macro بنویسید که مقدار N از نوع بایت را گرفته مجموع زیرا را محاسبه نماید. $1+2+3+\ldots+N$

۱۵-یک Procedure بنویسید که وقت را بصورت یک عدد شش رقمی گرفته مشخص نماید که پس از گذشت 5 ساعت و 55 دقیقه و 50 ثانیه وقت چیست و آنرا نمایش دهد.

۱٦-یک Procedure بنویسید که آرایه N عنصر X از نوع بایت را بصورت نزولی مرتب نماید.

۱۷-یک Macro بنویسید که مینیمم N مقدار از نوع word را مشخص نماید.

۱۸-یک Macro بنویسید که مشخص نماید عدد صحیح و مثبت N از نوع word اول می‌باشد یا خیر؟

۱۹-یک Macro بنویسید که دو مقدار M , N از نوع word و مثبت را گرفته کوچکترین مضرب مشترک آنها را محاسبه نماید.

۲۰-اگر N یک عدد صحیح و مثبت و مجذور کامل باشد یک Macro بدهید که جذر آنرا محاسبه نماید.

فصل هشتم

عملیات پردازش رشته‌ها

هدف کلی

معرفی رشته ها و پردازش آنها.

اهداف رفتاری

پس از مطالعه این فصل با مفاهیم زیر آشنا خواهید شد.

۱- تعریف رشته (String).

۲- انتقال یا جابجائی رشته‌ها.

۳- مقایسه رشته‌ها.

٤- بررسی یا جستجوی رشته‌ها.

۵- سایر عملیات مربوط به رشته‌ها.

۸-۱- رشته (String)

رشته عبارت است از یک بلوک پشت سر هم از بایت‌ها یا word ها در حافظه اصلی کامپیوتر. طول رشته می‌تواند تا 64K بایت باشد. بایستی توجه داشت که با دستورالعملهائی که تاکنون بحث نموده‌ایم می‌توان عملیات روی رشته‌ها را انجام داد. ولی استفاده از دستورالعملهای پردازش رشته‌ای کارآمدتر می‌باشد. عملیات پردازش رشته‌ای عبارتند از:

Move	جابه‌جائی	MOVS
Compare	مقایسه	CMPS
Load	بارکردن	LODS
Scan	جستجو	SCAS
Store	ذخیره	STOS

همانطوریکه قبلا بیان گردید علاوه بر Data segment در بعضی از دستورالعملهای پردازش رشته‌ای استفاده از segment دیگری بنام Extra segment نیز ضروری می‌باشد.

```
EXTRA_SEG   SEGMENT     PARA        'EXTRA'
    ⋮
EXTRA_SEG               ENDS
```

۸-۱-۱- دستورالعمل MOVS

از این دستورالعمل برای کپی نمودن یک رشته از محلی از حافظه به محل دیگری از حافظه استفاده می‌گردد. رشته‌ای که از آن می‌خواهیم کپی تهیه نمائیم

رشته مبداء یا Sourceو رشته بدست آمده را رشته مقصد یا Destination می‌نامند. شکل کلی این دستورالعمل عبارتست از

MOVS

الف) چنانچه رشته از نوع بایت باشد از MOVSB و اگر رشته از نوع word باشد از MOVSW استفاده می‌گردد.

ب) یک عنصر از رشته مبداء را به رشته مقصد انتقال می‌دهد.

ج) دستورالعمل MOVS بر هیچ فلگی اثر ندارد.

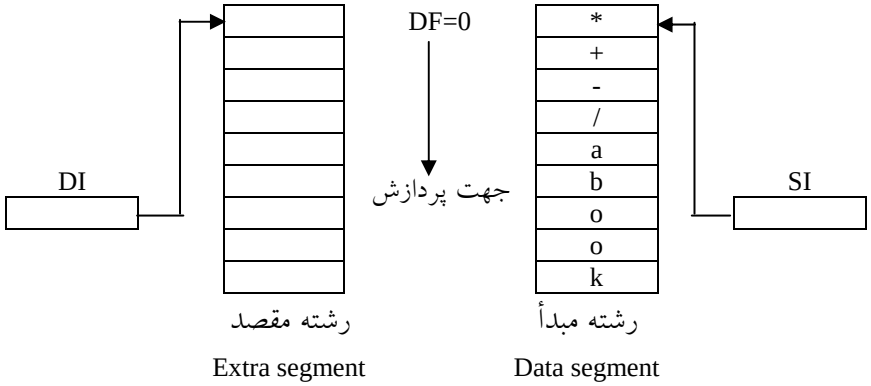
قبل از استفاده از این دستورالعمل بایستی آدرس شروع رشته مقصد را در ثبات DI قرار داده و رشته مقصد را در Extra segment تعریف نمود. همچنین آدرس شروع رشته مبداء را در ثبات SI قرار داده و رشته مبداء را در Data segment تعریف نمود. ضمناً مقدار فلگ DF مشخص کننده این است که عمل جابه‌جائی از اولین عنصر به طرف آخرین عنصر می‌باشد یا بلعکس. چنانچه مقدار DF برابر با صفر باشد عمل جابه‌جائی از اولین عنصر به طرف آخرین عنصر و چنانچه مقدار DF برابر با یک باشد عمل جابه‌جائی از آخرین عنصر بطرف اولین عنصر انجام می‌شود.

مثال ۱-۸

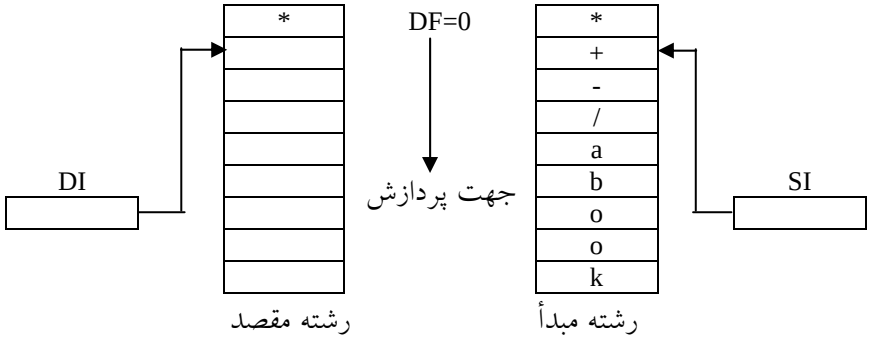
```
MOV     SI, OFFSET   SOURCE_STR
MOV     DI, OFFSET   DEST _STR
CLD ;   Forward  movement
MOVSB
```

دستورالعملهای فوق آدرس شروع رشته مبداء را در ثبات SI قرار داده و آدرس شروع رشته مقصد را در ثبات DI قرار داده و جهت حرکت از اولین عنصر به طرف آخرین عنصر مشخص نموده است. دستورالعمل MOVSB باعث می‌شود که یک عنصر از رشته مبداء که در آدرس DS:SI قرار دارد به آدرس ES:DI کپی گردد و مقدار DI و SI یک واحد افزایش یابند.

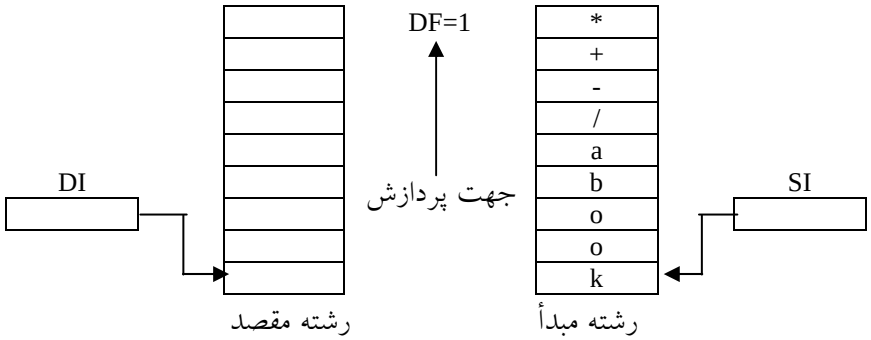
مقـــادیر ثباتها و رشته‌هـــــا قبل از اجرای دستورالعمل MOSV



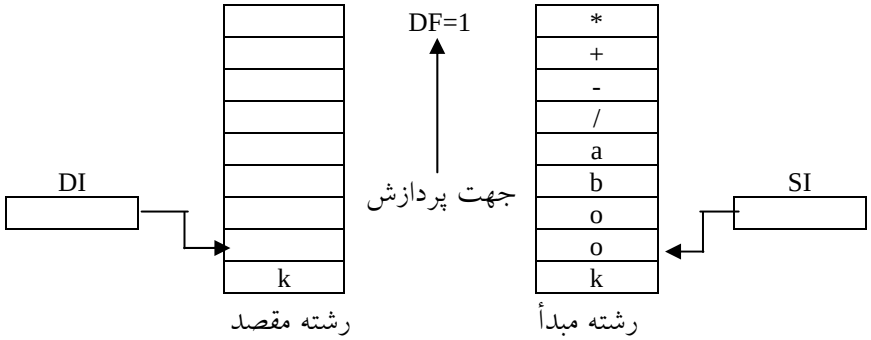
مقادیر ثباتها و رشته‌ها پس از اجرای دستورالعمل MOVS عبارتند از :



چنانچه به جای دستورالعمل CLD از دستورالعمل STD استفاده گردد مقادیر ثباتها و رشته‌ها بصورت زیر در می‌آید:

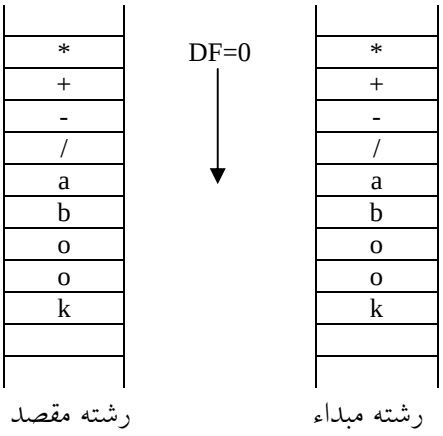


مقادیر ثباتهای SI و DI یک واحد کاهش می‌یابند.



حال برای انتقال سایر عناصر رشته بایستی تعداد عناصـر رشـته را در ثبـات CX قرار داد و از پیشوند REP استفاده نمود.

```
MOV     SI, OFFSET   SOURCE_STR
MOV     DI, OFFSET   DEST _STR
CLD
MOV     CX, 9 ;      تعدادعناصر رشته
REP     MOVSB
```



پیشوند REP تا مادامیکه مقدار CX مخالف صفر می‌باشد باعث اجرای دستورالعمل MOVSB می‌گردد. به محض صفر شدن مقدار ثبات CX اجرای دستورالعمل MOVSB متوقف می‌گردد. همانطور که قبلاً گفته شد به جای دستورالعمل‌های

```
MOV         SI, OFFSET   SOURCE_STR
MOV         DI, OFFSET   DEST _STR
```

از دستورالعمل‌های زیر می‌توان استفاده نمود.

```
LEA     SI,   SOURCE_STR
LEA     DI,   DEST _STR
```

ضمناً همانطوریکه گفته شد دستورالعمل‌های پردازش رشته‌ای در مقابل استفاده از دستورالعمل‌های معمولی کارآمد و مؤثرتر می‌باشند. برنامه فوق را می‌توان با استفاده از دستورالعمل‌های ذیل نیز نوشت.

```
MOV     SI, OFFSET      SOURCE_STR
MOV     DI, OFFSET      DEST_STR
CLD
MOV     CX, 9
JCXZ    LAB5
LAB1 :  MOVSB
LOOP LAB1
LAB5:
                ⋮
```

قطعه برنامه زیر باعث کپی شدن 100 بایت از رشته‌ای بنام SOURCE_D به DEST_D می‌شود. هر دو رشته در data segment قرار دارند.

```
PUSH    DS
POP     ES
CLD
LEA     SI, SOURCE_D
LEA     DI, DEST_D
MOV     CX, 100
REP     MOVSB
```

دستورالعملهای ذیل یک بایت از آدرس HERE به آدرس THERE منتقل می‌کند. بایستی توجه داشت که هر دو رشته در Extra segment قرار دارند.

```
LEA     SI, ES: HERE
LEA     DI, ES: THERE
MOVSB
```

۲-۱-۸- دستورالعمل STOS

این دستورالعمل باعث می‌شود که یک بایت یا یک word را از ثبات AL یا ثبات AX به یک عنصر رشته مقصد منتقل نماید. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

```
STOS
```

الف) این دستورالعمل برروی هیچ فلگی اثر ندارد.

ب)چنانچه رشته از نوع بایت باشد از دستورالعمل STOSB و چنانچه از نوع word باشد از دستورالعمل STOSW استفاده می‌گردد.

ج) مقداری که در رشته قرار می‌گیرد چنانچه از نوع بایت باشد در ثبات AL قرار داده می‌شود. و اگر از نوع word باشد در ثبات AX قرار داده می‌شود.

د) از دستورالعملهای CLD و STD برای مشخص نمودن جهت پردازش می‌توان استفاده نمود.

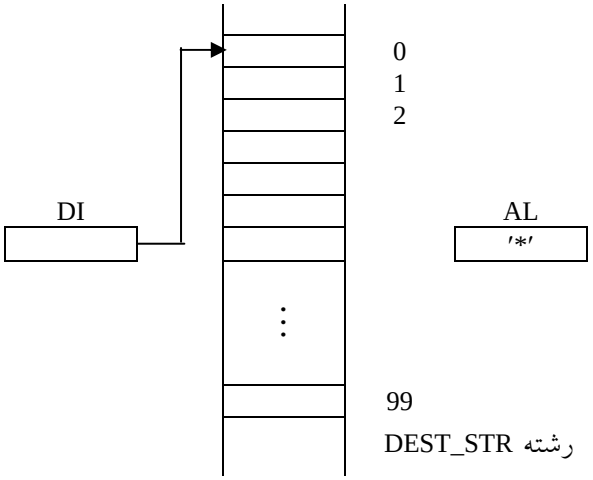
ه) از پیشوند REP نیز می‌توان استفاده نمود.

ز) رشته را بایستی در extra segment تعریف نمود و آدرس شروع آنرا در ثبات DI قرار داد.

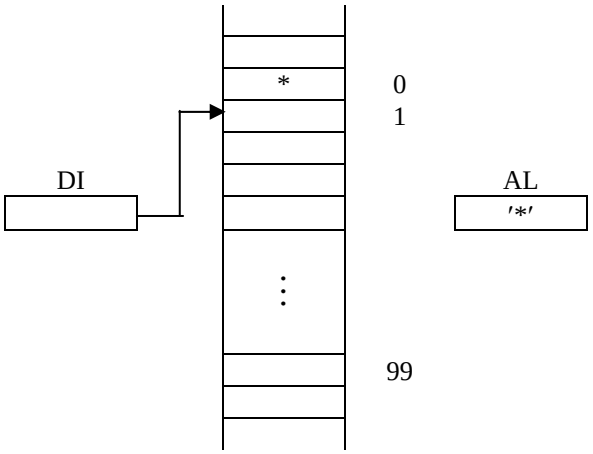
مثال ۲-۸

```
DEST_STR    DB   100 DUP (?)
     MOV    AL, '*'
      CLD
      LEA    DI, DEST_STR
     STOSB
```

مقدار * را در عنصری از رشته که آدرس آن بوسیله ES:DI مشخص می‌شود قرار می‌دهد.



پس از اجرای دستورالعمل STOSB یک واحد به ثبات DI اضافه می‌گردد.



از پیشوند REP نیز می‌توان استفاده نمود. این پیشوند مادامیکه محتوی CX مخالف صفر می‌باشد باعث اجرای دستورالعمل STOSB می‌گردد.

مثال ۳-۸

```
DSTR    DB   100  DUP(?)
MOV          CX , 50
MOV          AL, '*'
MOV     DI, OFFSET DSTR
CLD
REP          STOSB
```

قطعه برنامه فوق باعث می‌شود که مقدار 50 عنصر اول رشته DSTR برابر با کارکتر * گردند.

مثال ٤-۸

```
CLD
LEA     DI, W_STRING
MOV          AX,0
MOV          CX, 200
REP          STOSW
```

قطعه برنامه فوق مقدار دویست word اول رشته W_STRING را معادل صفر قرار می‌دهد. دقت داشته باشید که در اینجا افزایش DI باندازه دو واحد می‌باشد چون رشته از نوع word می‌باشد.

۳-۱-۸- دستورالعمل LODS

این دستورالعمل یک عنصر رشته مبداء را در AL یا AX قرار می‌دهد بسته به اینکه رشته از نوع بایت باشد یا word. شکل کلی این دستورالعمل عبارتست از

LODS

الف) چنانچه رشته از نوع بایت باشد از LODSB و چنانچه از نوع word باشد از LODSW استفاده می‌گردد.

ب) این دستورالعمل بر روی هیچ فلگی اثر ندارد.

ج) استفاده از پیشوند REP با این دستورالعمل امکان پذیر می‌باشد.

د) رشته بایستی در data segment تعریف گردد و آدرس شروع رشته در ثبات SI قرار داده می‌شود.

مثال ۵-۸

```
LEA     SI, SOURCE_STR
LODSB
```

اولین عنصر رشته SOURCE_STR در داخل ثبات AL قرار می‌گیرد.

٤-۱-۸- دستورالعمل CMPS

این دستورالعمل دو رشته مبداء و مقصد را با هم مقایسه می‌نماید.دستورالعمل CMPS مانند دستورالعمل CMP عمل می‌نماید.عنصر رشته مبداء را از عنصر متناظر رشته مقصد کم نموده و فلگ‌ها را براساس نتیجه بدست

آمده تنظیم می‌نماید. این دستورالعمل باعث تغییر مقدار هیچکدام از عملوندها نمی‌شود. شکل کلی دستورالعمل بصورت زیر می‌باشد.

CMPS

الف) چنانچه رشته‌ها از نوع بایت باشند از CMPSB و چنانچه از نوع word باشند از CMPSW استفاده می‌گردد.

ب) از پیشوندهای REPE یا REPZ می‌توان استفاده نمود.

ج)از پیشوندهای REPNE یا REPNZ می‌توان استفاده نمود.

د) رشته مبداء را بایستی در data segment تعریف نمود و آدرس شروع آنرا در ثبات SI قرار داد.

ه) رشته مقصد را بایستی در Extra Segment تعریف نموده و آدرس شروع آنرا در ثبات DI قرار داد.

مثال ٦-٨

```
MOV     SI, OFFSET  SOURCE_STR
MOV     DI, OFFSET   DEST_STR
CMPSB
```

دو عنصر اول رشته‌های DEST_STR , SOURCE_STR را با هم مقایسه می‌نماید. از پیشوندهای REPE یا REPZ با مفهوم repeat equal یا repeat zero با دستورالعمل CMPS می‌توان استفاده نمود. وقتی از پیشوند REPE یا REPZ استفاده می‌نمائیم شرط تکرار دستورالعمل اینستکه CX< > 0 and ZF=1 باشد.

مثال ۷–۸

```
        LEA     SI, STR1
        LEA     DI, STR2
        MOV     CX, 100
        CLD
        REPE    CMPSB
        JZ      FOUND
        ⋮
FOUND
        ⋮
```

قطعه برنامه فوق 100 بایت از دو رشته STR1 و STR2 را با هم مقایسه می‌نماید. از پیشوند REPNE یا REPNZ با مفهوم repeat not equal یا repeat not zero با دستورالعمل CMPS می‌توان استفاده نمود. شرط تکرار دستورالعمل CMPS آن است که CX < > 0 and ZF=0 باشد. قطعه برنامه زیر دو رشته STRG1 و STRG2 را با هم مقایسه می‌کند در صورتیکه مساوی باشند کنترل به SAME منتقل می‌گردد. صفت LENGTH مشخص کننده طول رشته می‌باشد.

```
        MOV     SI, OFFSET      STRG1
        MOV     DI, OFFSET      STRG2
        MOV     CX, LENGTH      STRG3
        CLD
NEXT:   CMPSB
        JNE     EXIT
        LOOP    NEXT
        JMP     SAME
EXIT:
        ⋮
SAME:
        ⋮
```

۵-۱-۸- دستورالعمل SCAS

از دستورالعمل SCAS برای جستجوی یک رشته جهت وجود داشتن یا نداشتن یک عنصر رشته‌ای معین بکار می‌رود. شکل کلی بصورت زیر می‌باشد.

SCAS

الف) رشته مورد جستجو بایستی رشته مقصد باشد. یعنی رشته در extra segment تعریف شده و آدرس شروع آن در ثبات DI قرار گیرد.
ب) از پیشوندهای REPE و REPNE می‌توان برای این دستورالعمل استفاده نمود.
ج) عنصر مورد جستجو چنانچه از نوع بایت باشد در ثبات AL و در صورتیکه از نوع word باشد در ثبات AX قرار داده می‌شود.
د) از فلگ DF برای تعیین جهت پردازش رشته استفاده می‌گردد.
ه)چنانچه رشته از نوع بایت باشد SCASB و چنانچه از نوع word باشد از SCASW استفاده می‌گردد.

مثال ۸-۸

قطعه برنامه زیر در رشته STRG به جستجوی * می‌پردازد.

```
STRG    DB  50  DUP (?)
MOV     AL , '*'
MOV     CX, 50
LEA     DI, STRG
CLD
REPNE   SCASB
```

دستورالعملهای ذیل یک رشتهٔ 80 کارکتری که به کارکتر فاصله ختم شده را جستجو نموده در صورتیکه کلیه عناصر رشته کارکتر فاصله باشد کنترل به NOT_FOUND منتقل گردیده در غیر اینصورت با اولین عنصر مخالف blank شروع نموده و یک زیر رشته 30 کارکتری از رشته اولیه را به رشته SYMBOL انتقال می‌دهد.

```
MOV    DI, OFFSET    LINE
MOV    CX, 80
MOV    AL, 20H   ; ASCII FOR BLANK
 CLD
 NEXT:    SCAS      LINE
          LOOPE     NEXT
          JE        NOT_FOUND
          MOV       SI,DI
          DEC       SI
          MOV       DI, OFFSET SYMBOL
          MOV       CX, 31
          FILL:     STOS      SYMBOL
          LOOP      FILL
          MOV       DI, OFFSET SYMBOL
          MOV       CX, 31
          JMP       SCANE
          MOVE:     STOS   SYMBOL
 SCANE:   LODS      LINE
          CMP       AL, 20H
          LOOPNE         MOVE
             ⋮
```

مثال ۹–۸

قطعه برنامه زیـر رشـته STRING را در نظـر مـی‌گیـرد و بـدنبال کـارکتر & می‌گردد که به کارکتر blank یا فاصله تبدیل نماید.

```
 STRLEN    EQU  15
 STRING    DB  'The time & is  now'
   CLD
   MOV       AL, '&'
   MOV       CX, STRLEN
   LEA       DI, STRING
  REPNE      SCASB
   JNZ       NOT FOUND
   DEC       DI
   MOV       BYTE PTR[DI], 20H
      ⋮
   NOT FOUND:
             ⋮
```

قطعه برنامه زیر ایجاد رشته‌ای با الگو داده شده می‌نماید.

------***---***---***---***---***---***---

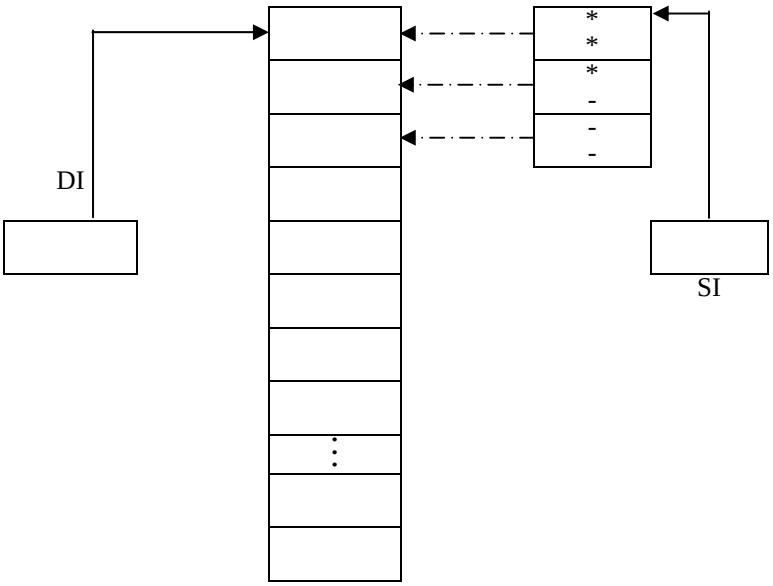
برای اینکار ابتدا رشته PATTERN را بصورت زیر تعریف می‌نمائیم.

***---

حال

```
PATTERN     DB  '***---'
DISPAREA    DB 42 DUP (?)
    ⋮
CLD
MOV             CX, 21
LEA         DI, DISPAREA
LEA         SI, PATTERN
REP             MOVSW
```

بایستی توجه داشت که وقتی به انتهای رشته PATTERN رسیدیم بصورت اتوماتیک مجدداً از سرگرفته می‌شود و این روش خوبی است برای کپی کردن الگوها.



مروری بر مطالب فصل

دستورالعمل‌های پردازش رشته‌ای برای پردازش رشته‌ها بکار می‌روند.

این دستورالعمل‌ها عبارتند از جابه‌جائی MOVS ، مقایسه CMPS ، بارکردن LODS ، جستجو SCAS و ذخیره کردن STOS. معمولاً از دو رشته مبدأ و مقصد استفاده می‌گردد. آدرس شروع رشته مقصد را در رجستر DI و آدرس شروع رشته مبدأ را در رجستر SI قرار داده میشوند. رشته مبدأ را در data segment و رشته مقصد را در extra segment تعریف می‌نمائیم. فلگ DF جهت پردازش را مشخص می‌کند. از پیشوندهای REP، REPE، REPNE، REPNZ، REPZ می‌توان با دستورالعمل‌های پردازش رشته‌ای استفاده نمود.

⌘ تمرین

۱-پیشوند REP در مورد کدام دستورالعملهای رشته‌ای کاربرد دارد؟

۲-پیشوند REPZدر مورد کدام دستورالعملهای رشته‌ای کاربرد دارد؟

۳-دستورالعملهای STOS و SCAS روی کدام فلگ‌ها اثر دارند؟

٤-برنامه‌ای بنویسید که یک رشته 100 کارکتری با استفاده از الگـوی ـ ـ ـ ـ ـ ـ ـ ـ زیر ایجاد نماید. (راهنمائی 20 مرتبه الگو را کپی نماید.)

٥-رشته 100 کاراکتری STRG را در نظر گرفته کلیـه کاراکترهـای blank آنـرا بـه * تبدیل نمائید.

٦-رشته 100 کاکتری STRG را در نظر بگیرید کلیـه کاراکترهـای آنـرا بـه * تبـدیل نمائید.

۷-رشته 100 کاراکتری STRG را در نظر بگیرید اولین کاراکتر * در رشـته را بـه & تبدیل نمائید.

۸-رشته 100 کاراکتری STRG1 را در نظر گرفته 20 کاراکتر وسط رشته را به رشـته STRG2 منتقل نمائید.

۹-رشته 50 کـاراکتری STRG1 و رشـته 100 کـاراکتری STRG2 را در نظـر بگیـرد مشخص نمائید که آیا 50 کاراکتر آخر رشـته STRG2 معـادل STRG1 مـی‌باشـد یـا خیر؟

۱۰-رشته 50 کاراکتری STRG را در نظر بگیرید کلیه عناصر * در آنرا حذف نمائید. (کاراکتر بعدی را جایگزین * نمائید)

فصل نهم
برنامه‌های نمونه

هدف کلی

نحوه نوشتن و ایجاد برنامه معرفی و چند برنامه نمونه.

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا می‌شوید.

۱- اجزای مختلفه یک برنامه.

۲- نحوه نوشتن یک برنامه و اجرای آن.

۹-۱- اجزای برنامه

همانطوریکه گفته شد در نوشتن برنامه‌ها می‌توان از چهار سگمنت زیر استفاده نمود.

```
STACK   SEGMENT
DATA    SEGMENT
EXTRA   SEGMENT
CODE    SEGMENT
```

در STACK SEGMENT پشته مورد نیاز برنامه اعلان می‌گردد. در DATA SEGMENT کلیهٔ متغیرهای مورد نیاز برنامه اعلان و تعریف می‌شود.

در EXTRA SEGMENT کلیه متغیرهای برنامه که جهت پردازش دستورالعملهای رشته‌ای مورد نیاز می‌باشد اعلان و تعریف می‌گردد.

CODE SEGMENT شامل کلیه دستورالعملهای برنامه می‌باشد.

در زبان اسمبلی در حقیقت برنامه بعنوان یک روال از نوع FAR نوشته می‌شود که وقتی بدستور RET برسیم کنترل به سیستم عامل برمی‌گردد.

۹-۲- یک برنامه نمونه

در ذیل یک برنامهٔ نمونه داده شده است. این برنامه مقادیر یک آرایه چهار بایتی را وارون نموده و به یک آرایه دیگر منتقل می نماید.

```
STACK_SEG   SEGMENT PARA STACK 'STACK'
DW          32  DUP (?)
STACK_SEG   ENDS
DATA_SEG    SEGMENT PARA 'DATA'
SOURCE      DB  10, 20, 30, 40
DEST        DB  4 DUP (?)
DATA_SEG    ENDS
CODE_SEG    SEGMENT PARA 'CODE'
```

```
ASSUME          CS:CODE_SEG, DS:DATA_SEG, SS:STACK_SEG
OUP_PROG        PROC    FAR
;Set up         the stack to contain the proper
; Valves        so this program can return to Dos.
PUSH            DS; Put return segment address on stack
MOV             AX , 0
PUSH            AX ; put zero return address on stack
; Initialize the data segment address
MOV             AX, DATA_SEG ; Initialize DS
MOV             DS , AX
;Initialize DEST With zeroes.

    MOV         DEST, 0         ;First byte
    MOV         DEST +1, 0      ;Second byte
    MOV         DEST +2, 0      ;Third byte
    MOV         DEST +3, 0      ;Fourth byte
    ;Copy SOURCE table into DEST table in reverse order.
    MOV         AL, SOURCE
    MOV         DEST +3, AL
    MOV         AL, SOURCE+1
    MOV         DEST+2, AL
    MOV         AL, SOURCE+2
    MOV         DEST+1, AL
    MOV         AL, SOURCE+3
    MOV         DEST, AL
    RET         ; Far return to DOS
    OUR_PROG    ENDP
    CODE_SEG    ENDS
    END         OUR_PROG
```

همانطوریکه ملاحظه می‌شود. شکلی کلی SEGMENT بصورت زیرمی‌باشد.

اسم سگمنت SEGMENT

⋮

اسم سگمنت ENDS

دستورالعمل ASSUME باعث می شود که آدرس شروع سگمنت CODE در ثبات CS، آدرس شـروع سـگمنت DATA در ثبـات DS و آدرس شـروع سـگمنت STACK در ثبات SS قرار گیرد.

دستورالعمل END انتهای برنامه را مشخص می‌نماید. در موقع ترجمه برنامه به زبان ماشین به محض آنکه مترجم به دستور END رسید ترجمه برنامه متوقف می‌گردد.

۳-۹- نحوهٔ اجرای برنامه

دستورالعملهای برنامه را در یک فایل با پسوند ASM. قرار داده سپس دستور زیر را می‌دهیم.

C:\> MASM اسم برنامه.ASM

در این فاز برنامه از نظر نحوی بررسی شده چنانچه اشتباهات نحوی داشته باشد اسمبلر اشتباهات را متذکر می‌شود. چنانچه اشتباهی وجود داشته باشد آنرا رفع نموده مجدداً دستور MASM را می‌دهیم.
چنانچه برنامه دارای اشتباه نحوی نباشد اسمبلر پیغام

NO WARNINGS
NO ERRORS

را می‌دهد. آنگاه فرمان زیر را می‌دهید.

C:\ASSEMBLY > LINK اسم برنامه

در دستورالعمل فوق نیازی به پسوند ASM نمی‌باشد. چنانچه مشکلی در LINK باشد گزارش می‌شود که بایستی آنرا رفع نمود و برای اینکار مجدداً از دستور LINK بایستی استفاده نمود. حال برای اجرای برنامه کافیست که دستور ذیل را بدهیم.

C:\ASSEMBLY > اسم برنامه

در اینجا نیز نیازی به پسوند ASM . نمی‌باشد.

٤-۹- برنامه‌های اسمبلی نوشته شده

در این بخش تعدادی برنامه که بزبان اسمبلی نوشته شده ارائه می‌گردد. اولین برنامه ابتدا صفحه مانیتور را پاک نموده سپس کارکتر A را در سطح 25 و ستون 13 صفحه مانیتور قرار می‌دهد.

```
STACK_SEG     SEGMENT PARA STACK 'STACK'
DW            32    DUP   (?)
STACK_SEG     ENDS
;

DATA_SEG      SEGMENT PARA 'DATA'
MESSAGE       DB 'Nikmehr', 13, '$'
DATA_SEG      ENDS
CODE_SEG      SEGMENT PARA 'CODE'
BEGIN         PROC FAR
ASSUME        SS:STACK_SEG,CS:CODE_SEG, DS:DATA-SEG
START:
PUSH    DS
SUB     AX , AX
PUSH    AX
MOV     AX , DATA_SEG
MOV     DS , AX
;
MOV     AH , 00
MOV     AL , 03
INT     10 H
;
MOV     AH , 2
MOV     BH , 0
MOV     DH , 13 ; row
MOV     DL , 25 ; column
INT     10H
;
;
MOV     AH , 2
MOV     DL , 65
INT     21 H
;
RET
BEGIN   ENDP
CODE_SEG    ENDS
END         START
```

برنامه بعدی ارقام 0 تا 9 را بدنبال هم روی صفحه مانیتور ظاهر می‌سازد. دستورالعمل NOP به معنی NO OPERATION می‌باشد و عملاً کاری انجام نمی‌دهد.

```
STACK_SEG          SEGMENT  PARA  STACK  'STACK'
DW                 32 DUP (?)
STACK_SEG          ENDS
;
;
CODE_SEG           SEGMENT  PARA  STACK  'STACK'
BEGIN PROC FAR
ASSUME SS:STACK_SEG, CS:CODE_SEG
START:
PUSH DS
SUB AX,AX
PUSH  AX
;
MOV      AH, 00
MOV      AL, 03
INT      10 H
;
MOV      AH , 2
MOV      BH , 0
MOV      DH, 13    ;ROW
MOV      DL, 25 ; COLUMN
INT      10 H
;
;character input without echo
;
MOV      CX, 10
LOOP1:NOP
MOV      AH ,7
INT      21H
;
MOV      AH ,2
MOV      DL , AL
INT      21H
LOOP     LOOP1
;
RET
BEGIN    ENDP
CODE-SEG ENDS
END START
```

برنامه بعدی یک مقدار باینری را روی صفحه مانیتور نمایش می‌دهد.

```
STACK_SEG SEGMENT PARA ′STACK′
DW 32 DUP(?)
STACK_SEG
ENDS
;
;
DATA-SEG SEGMENT PARA ′DATA′
ASCII-VAL DB 8 DUP (?)
BINARY_VAL
DW?
ASCII_LENGTH DW 8
DATA_SEG
ENDS
CODE_SEG SEGMENT PARA ′CODE′
BEGIN PROC
FAR
ASSUME SS:STACK_SEG,
CS:CODE_SEG,DS:DATA_SEG
START:
PUSH  DS
SUB AX,AX
PUSH AX
MOV AX, DATA_SEG
MOV DS,AX
;
MOV      AH, 00
MOV      AL , 03
INT      10 H
;
MOV      AH, 2
MOV      BH,0
MOV      DH, 13    ;ROW
MOV      DL, 25     ; COLUMN
INT      10H
;
MOV BINARY _ VAL, 32456
CALL BINARYTOASCII
                    ; displaying & printing binary numbers
                    ;
LEA SI, ASCII_VAL
MOV CX, ASCII_LENGTH
LOOP1: MOV AH,2
MOV DL, [SI]
INT 21H
; PRINTING
MOV AH,5
MOV DL, [SI]
INT 21H
INC SI
LOOP LOOP1
```

```
;
;
RET
BEGIN ENDP
;
;

BINARYTOASCII PROC NEAR
                MOV CX, 10
                LEA SI,ASCII_VAL+7
                MOV AX,BINARY_VAL
   LABE12:
                CMP AX,10
                JB LABE13
                XOR DX,DX
                DIV CX
                OR DL,30H
                MOV [SI], DL
                DEC  SI
                JMP   LABLE12
  LABEL13:
                OR AL, 30H
                MOV [SI], AL
                RET
                BINARYTOASCII   ENDP
                CODE-SEG  ENDS
                END START
```

خروجی برنامه عبارتند از

```
3
2
4
5
6
32456
```

برنامهٔ بعدی نحوهٔ استفاده از Macro در برنامه‌ها را نشان می‌دهد.

```
NIK MACRO
ASSUME SS:STACK_SEG, CS:CODE_SEG,DS:DATA_SEG
START:
PUSH DS
SUB AX,AX
PUSH  AX
MOV AX,DATA_SEG
MOV DS,AX
;
ENDM
STACK_SEG SEGMENT PARA STACK 'STACK'
DW 32 DUP (?)
STACK_SEG ENDS
;
;
DATA_SEG SEGMENT PARA 'DATA'
MESSAGE DB 'DARYOUSH NIKMEHR' ,13, '$'
DATA _SEG ENDS
CODE –SEG SEGMENT PARA 'CODE'
NIK
BEGIN PROC FAR
MOV     AH, 00
MOV     AL, 03
INT     10H
;
MOV     AH, 2
MOV     BH, 0
MOV     DH, 13    ; ROW
```

```
MOV     DL, 25; COLUMN
INT     10H
;
;
MOV     AH , 2
MOV     DL , '*'
INT     21H
;
;
RET
BEGIN ENDP
CODE_SEG
ENDS
END START
```

برنامه بعـدی نحـوهٔ قـرار دادن روالهـای از نـوع NEAR در برنامـه‌هـا را در CODE SEGMENT نشان می‌دهد.

```
CODE_SEG    SEGMENT PARA 'CODE'
BEGIN       PROC FAR
ASSUME      CS:CODE_SEG , . . .
START:
                ⋮
CALL        PROCB
                ⋮
CALL        PROCC
                ⋮
RET     ;       کنترل به سیستم عامل برمی‌گردد
BEGIN   ENDP
PROCB   PROC    NEAR
                ⋮
RET     ;       کنترل به دستورالعمل بعد از CALL PROCB منتقل شود
PROCB   ENDP
PROCC   PROC    NEAR
                ⋮
RET     ;       کنترل به دستورالعمل بعد از PROCC منتقل شود
PROCC       ENDP
CODE_SEG    ENDS
END         START
```

در ذیل برنامه ضرب دو مقدار 32 بیتی داده شده است؛ در این برنامه این دو مقدار بصورت بدون علامت در نظر گرفته شده‌اند.

```
;               Multiplies two 32-bit unsigned number and generates a
;               64-bit product.
;               Inputs: CX:BX=Multiplier
;                       DX:AX=Multiplicand
;               Result : Product in DX, CX, BX, and AX (high to low order).

;               To assemble: MASM MULU32;
;               To Link: LINK callprog+MULU 32;

                PUBLIC MULU32
DSEG            SEGMENT PARA 'DATA'
HI_MCND         DW      ?
LO_MCND         DW      ?
HI_PP1          DW      ?
LO_PP1          DW      ?
HI_PP2          DW      ?
LO_PP2          DW      ?
HI_PP3          DW      ?
LO_PP3          DW      ?
HI_PP4          DW      ?
LO_PP4          DW      ?
DSEG            ENDS

CSEG            SEGMENT         PARA 'CODE'
                ASSUME  CS:CSEG,DS:DSEG
MULU32          PROC    FAR
                PUSH    DS                      ;Save caller's DS and DI
```

```
        PUSH    DI
        MOV     DI, DSEG        ;Initialize DS
        MOV     DS,DI
        MOV     HI_MCND,DX      ;Save multiplicand in memory
        MOV     LO_MCND,AX
        MUL     BX              ;Form partial product #1
        MOV     HI_PP1,DX       ;and save it in memory
        MOV     LO_PP1, AX
        MOV     AX, HI_MCND     ;Form partial product #2
        MUL     BX
        MOV     HI_PP2, DX      ;and save it in memory
        MOV     LO_PP2, AX
        MOV     AX,LO_MCND      ;Form partial product #3
        MUL     CX
        MOV     HI_PP3,DX       ;and save it in memory
        MOV     LO_PP3, AX
        MOV     AX, HI_MCND     ;Form partial product #4
        MUL     CX
        MOV     HI_PP4, DX      ;and save it in memory
        MOV     LO_PP4, AX
; Add the partial products to form the final 64-bit product.
        MOV     AX,LO_PP1       ;Low 16 bits
        MOV     BX,HI_PP1       ;Form mid-lower 16 bits
        ADD     BX,LO_PP2       ; with sum #1
        ADC     HI_PP2,0
        ADD     BX,LO_PP3       ; and sum #2
        MOV     CX,HI_PP2       ;Form mid-upper 16 bits
        ADC     CX,HI_PP3       ;with sum #3
```

```
        ADC     HI_PP4, 0
        ADD     CX,LO_PP4       ;and sum #4
        MOV     DX,HI_PP4       ;Form high 16 bits
        ADC     DX,0            ;including propagated carry
        POP     DI              ;Restore caller's registers
        POP     DS
        RET
MULU32  ENDP
CSEG    ENDS
        END
```

برنامه داده شده در ذیل حاصلضرب دو مقدار 32 بیتی علامتدار را تعیین می‌نماید.

```
;       Multiplies two 32-bit signed numbers and generates
;       A 64-bit product.
;       Inputs:         CX:BX = Multiplier
;                       DX:AX = Multiplicand
;       Result : Product in DX, CX, BX, and AX (high to low order)
;       Calls MULU32

;       To assemble: MASM MULS 32;
;       To link: LINK callprog+MULS32+MULU32;

                EXTRN           MULU32 : FAR
                PUBLIC          MULS 32
DSEG            SEGMENT         PARA 'DATA'
NEG_IND         DB      ?
DSEG            ENDS

CSEG            SEGMENT PARA 'CODE'
MULS32          PROC    FAR
                ASSUME          CS:CSEG , DS:DSEG
; Initialize the data segment address
                PUSH    DS              ;Save caller's DS and DI
                PUSH    DI
                MOV     DI, DSEG        ;Initialize DS
                MOV     DS , DI

                MOV     NEG_IND,0       ;Negative indicator=0
                CMP     DX, 0           ;Multiplicand negative?
                JNS     CHKCX           ;No. Go check multiplier
                NOT     AX              ;Yes.2s-comp.multiplicand
                NOT     DX
                ADD     AX,1
                ADC     DX,0
                NOT     NEG_IND         ;and 1s-comp. Indicator
CHKCX:          CMP     CX,0            ;Multiplier negative?
                JNS     GOMUL           ;No. Go multiply
                NOT     BX              ;Yes. 2s-comp. Multiplier
                NOT     CX
                ADD     BX,1
                ADC     CX,0
                NOT     NEG_IND         ;and 1s-comp. Indicator
```

```
GOMUL:  CALL    MULU32      ;Perform unsigned multiplication
        CMP     NEG_IND,0   ;Does product have right sign?
        JZ      DONE        ;Yes. Exit.
        NOT     AX          ;No. 2s-comp. Product
        NOT     BX
        NOT     CX
        NOT     DX
        ADD     AX,1
        ADC     BX,0
        ADC     CX,0
        ADC     DX,0
DONE:   POP     DI          ;Restore caller's registers
        POP     DS
        RET
MULS32  ENDP
CSEG    ENDS
        END
```

برنامه ذیل تقسیم دو مقدار را انجام می‌دهد.

```
;       This divide procedure determines the correct quotient
;       And remainder, regardless of overflow.
;       Inputs: BX = Divisor
;               DX:AX=Dividend
;       Results : BX:AX=Quotient
;
;                       DX = Remainder
;       To assemble: MASM DIVUO;
;       To link: LINK callprog+DIVUO;

        PUBLIC      DIVUO
CSEG    SEGMENT     PARA 'CODE'
        ASSUME      CS: CSEG
DIVUO   PROC    FAR
        CMP     BX,0            ; Divisor = 0?
        JNZ     DVROK
DVROK:  INT     0               ; Yes. Abort the divide
        PUSH    ES              ; Save working registers
        PUSH    DI
        PUSH    CX
        MOV     DI,0            ; Fetch current INT 0 vector
        MOV     ES, DI
        PUSH    ES: [DI]        ; and save it on the stack
        PUSH    ES:[DI+2]
        LEA     CX, OVR_INT     ;Make INT 0 vector
        MOV     ES: [DI], CX    ;point to OVR_INT
        MOV     CX, SEG OVR_INT
        MOV     ES: [DI+2], CX
        DIV     BX              ;Perform the division
```

```
RESTORE:  SUB    BX, BX              ;If no overflow, BX=0
          POP    ES: [DI+2]          ;Restore INT 0 vector
          POP    ES: [DI]
          POP    CX                  ;Restore registers
          POP    DI
          POP    ES
          RET

;         This interrupt service routine executes if the divide
;         Operation produces overflow.

OVR_INT:  POP    CX                  ;Modify ret. Addr. Offset
          LEA    CX, RESTORE         ; to skip SUB BX,BX
          SUSH   CX
          SUSH   AX
          MOV    AX,DX               ;Set up 1st dividend, 0-Y1
          SUB    DX,DX
          DIV    BX                  ;Q1 is in AX, R1 is in DX
          POP    CX                  ;Pop orig. AX into CX
          PUSH   AX                  ;Save Q1 on stack
          MOV    AX,CX               ;Set up 2nd dividend, R1-Y0
          DIV    BX                  ;Q0 is in AX, R0 is in DX
          POP    BX                  ; Final quotient is in BX:AX
          IRET
DIVUO     ENDP
CSEG      ENDS
          END
```

برای محاسبه جذر یک مقدار 32 بیتی از روش تکراری نیوتون استفاده می‌شود. روش نیوتون بدین صورت است که اگر A جذر تقریب مقدار N باشد آنگاه A1 تقریب بهتری برای جذر N می‌باشد.

$$A1 = (N / A + A) / 2$$

بعنوان مثال اگر N=10000 باشد و A=200 آنگاه اولین تقریب برابر با 10000 / 200 + 2 = 52 می‌باشد.

10000/52 = 192, (192+52) / 2 = 122
10000/122 = 81, (122+81) / 2 = 101
10000/101 = 99, (101+99) / 2 = 100
10000 / 100 = 100

برنامه زیر جذر یک مقدار 32 بیتی را محاسبه می‌نماید.

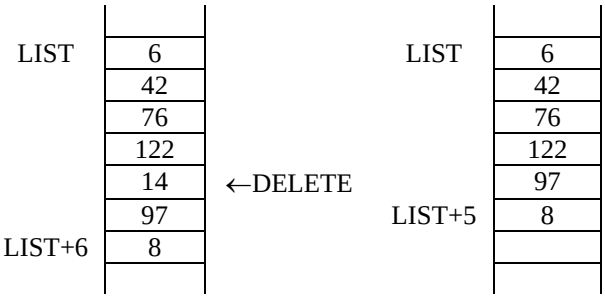
```
;       Culculates the square root of a 32-bit integer.
;       Input : DX:AX = Integer
;       Result : BX = square root
;       The original number in DX:AX is unaffected.

;       To assemble: MASM SQRT 32;
;       To link: LINK callprog+SQRT32;

        PUBLIC      SQRT32
CSEG    SEGMENT     PARA 'CODE'
        ASSUME      CS: CSEG
SQRT32  PROC    FAR
        PUSH    BP          ;Save contents of BP
        PUSH    DX          ;and source number DX:AX
        PUSH    AX
```

```
        MOV     BP,SP       ;BP points to AX on the stack
        MOV     BX. 200     ;As a first approx,
        DIV     BX          ;divide source number by 200,
        ADD     AX, 2       ;then add 2
NXT_APP: MOV    BX, AX      ;Save this approx. in BX
        MOV     AX, [BP]    ;Read source number again
        MOV     DX, [BP+2]
        DIV     BX          ;Divide by Last approx.
        ADD     AX,BX       ;Average last two approxs.
        SHR     AX,1
        CMP     AX,BX       ;Last two approxs. Identical?
        JE      DONE
        SUB     BX,AX       ;No. Check for diff. Of 1
        CMP     BX,1
        JE      DONE
        CMP     BX,-1
        JNE     NXT_APP
DONE:   MOV     BX,AX       ;Put result in BX
        POP     AX          ;Restore source number
        POP     DX
        POP     BP          ;and scratch register BP
        RET
SQRT32  ENDP
CSEG    ENDS
        END
```

برنامه زیر یک عنصر از یک لیست نامرتب را حذف می‌نماید.

```
        ;       Deletes the value in AX from an unordered list in the
        ;       Extra segment, if that value is in the list.
        ;       Inputs: DI = starting address of the list
        ;               First location = Length of list (words)
        ;       Results: None
        ;       DI and Ax are unaltered.

        ;       Assemble with : MASM DEL_UL;
        ;       Link with: LINK callprog + DEL_UL;
                PUBLIC      DEL_UL
CSEG            SEGMENT     PARA 'CODE'
                ASSUME      CS:CSEG
DEL_UL          PROC        FAR
                CLD                     ;Make DF=0, to scan forward
                PUSH        BX          ;Save scratch register BX
                PUSH        DI          ; and starting address
                MOV         CX, ES:[DI] ;Fetch element count
                ADD         DI, 2       ;Make DI point to 1st data el.
REPNE           SCASW                   ;Value in the list?
                JE          DELETE      ;If so, go delete it.
                POP         DI          ; Otherwise, restore registers
                POP         BX
                RET                     ;and exit.

        ;       The following instructions delete an element from the list,
        ;       As follows:
        ;           (1)     If the element lies at the end of the list,
        ;                   Delete it by decreasing the element count by 1.
```

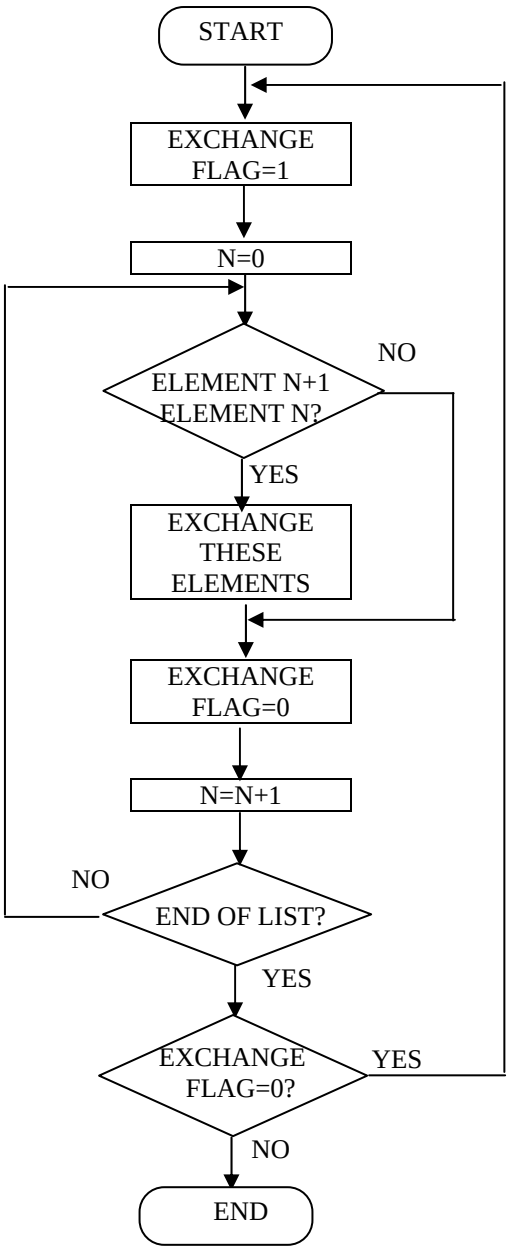
```
;               (2)     Otherwise, delete the element by moving all
;                       Subsequent elements up by one position.

DELETE:     JCXZ    DEC_CNT             ;If (CX)= 0, delete last el.
NEXT_EL:    MOV     BX,ES: [DI]         ;Move one element up. In list
            MOV     ES; [DI-2], BX
            ADD     DI, 2               ;Point to next element
            LOOP    NEXT_EL             ;Repeat until all els. Moved
DEC_CNT:    POP     DI                  ;Decrease el. Count by 1
            DEC     WORD PTR ES:[DI]
            POP     BX                  ;Restore contents of BX
            RET                         ;and exit
DEL_UL      ENDP
CSEG        ENDS
            END
```



برنامه ذیل مقدار ماکزیمم و مینیمم یک لیست نامرتب را مشخص می‌نماید.

```
;       Finds the maximum and minimum words in an unordered
;       List in the extra segment.
;       Inputs: ES: DI = Starting address of the list
;               First location = Length of list (words)
;       Results : AX = Maximum
;                 BX = Minimum
;       DI is unaltered.

;       Assemble with : MASM MINMAX;
;       Link with: LINK callprog + MINMAX;

        PUBLIC      MINMAX
CSEG    SEGMENT     PARA 'CODE'
        ASSUME      CS: CSEG
MINMAX  PROC    FAR
        PUSH    CX
        PUSH    DI              ;Save starting address
        MOV     CX, ES:[DI]     ;Fetch element count
        DEC     CX              ;Get ready for count-1 compares
        ADD     DI,2            ;point to first element
        MOV     BX, ES:[DI]     ;Declare it both minimum
        MOV     AX,BX           ;and maximum
CHKMIN: ADD     DI,2            ;Point to next element
        CMP     ES: [DI], BX    ;Compare element to minimum
        JAE     CHKMAX          ;New minimum found?
        MOV     BX,ES : [DI]    ;Yes. Put is in BX
        JMP     SHORT NEXTEL
CHKMAX: CMP     ES:[DI], AX     ;Compare element to maximum
        JBE     NEXTEL          ;New maximum found?
        MOV     AX, ES:[DI]     ;Yes. Put it in AX
NEXTEL: LOOP    CHKMIN          ;Check entire list
        POP     DI              ;Restore starting address
        POP     CX
        RET                     ; and exit
MINMAX  ENDP
CSEG    ENDS
        END
```

در ذیل الگوریتم مرتب سازی حبابی داده شده است:



برنامه زیر تعدادی عناصر را بصورت صعودی بروش حبابی مرتب می‌نماید.

```
;       Arranges the 16-bit elements of a list in the extra
;       Segment in ascending order, using bubble sort.
;       Inputs: ES:DI = starting address of the list
;               First location = Length of list (words)
;       DI is unaltered.

;       Assemble with: MASM B_SORT;
;       Link with: LINK callprog + B_SORT;

DSEG        SEGMENT     PARA 'DATA'
SAVE_CNT    DW      ?
START_ADDR  DW      ?
DSEG        ENDS

            PUBLIC      B_SPRT
CSEG        SEGMENT     PARA 'CODE'
            ASSUME      CS:CSEG,DS:DSEG
B_SORT      PROC    FAR
            PUSH    DS                      ;Save caller's registers
            SUSH    CX
            SUSH    AX
            SUSH    BX
            MOV     AX, DSEG            ;Initialize DS
            MOV     DS,AX
            MOV     START_ADDR,DI       ;Save starting address
            MOV     CX,ES:[DI]          ;Fetch element count
            DEC     CX                  ;Get ready for count-1 compares
            MOV     SAVE_CNT,CX         ;Save this value in memory
```

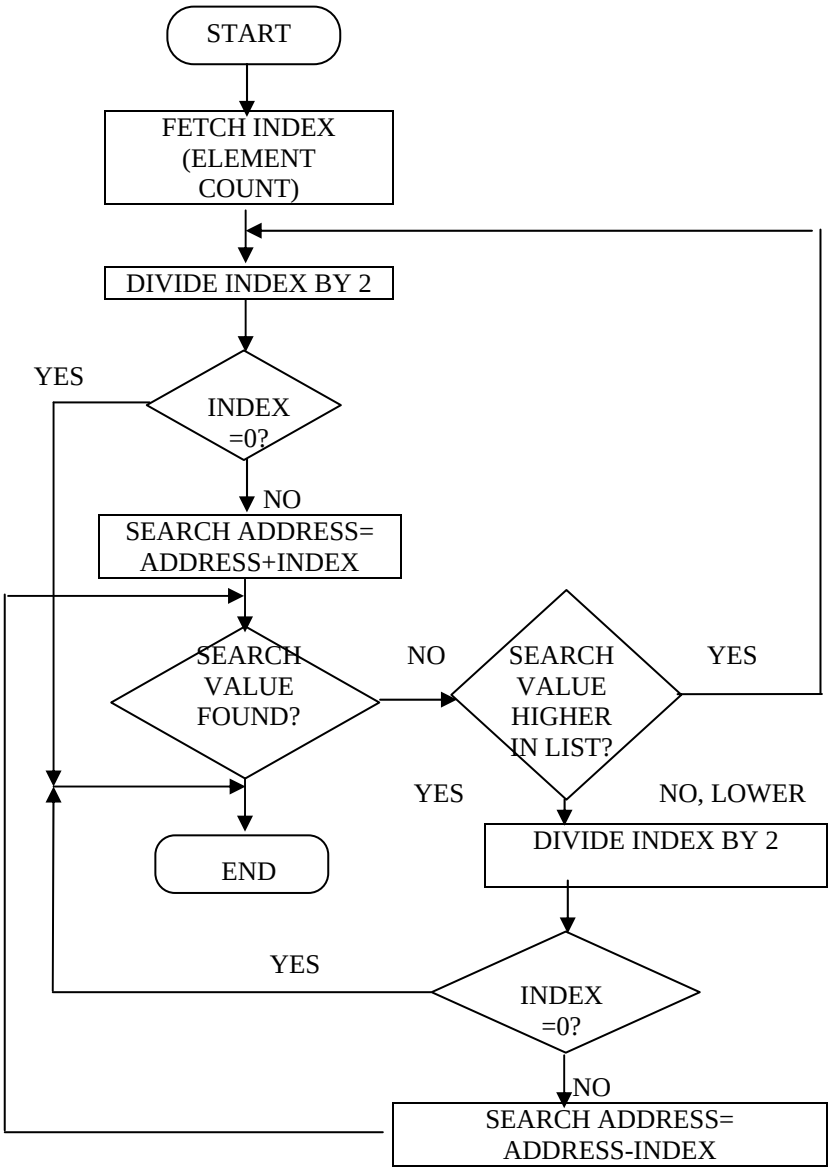
```
INIT:   MOV     BX,1            ;Exchange flag (BX) = 1
        MOV     CX,SAVE_CNT     ;and load this count into CX
        MOV     DI,START_ADDR   ;Load start address into DI
NEXT:   ADD     DI,2            ;Address a data element
        MOV     AX,ES:[DI]      ;and load it into AX
        CMP     ES:[DI+2],AX    ;Is next el. <this el.?
        JAE     CONT            ;No. Go check next pair
        XCHG    ES:[DI+2],AX    ;Yes. Exchange these elements.
        MOV     ES:[DI],AX
        SUB     BX,BX           ;and make exchange flag 0
CONT:   LOOP    NEXT            ;Process entire list
        CMP     BX,0            ;Any exchanges made?
        JE      INIT            ;If so, process list again
        MOV     DI,START_ADDR   ;If not, restore registers
        POP     BX
        POP     AX
        POP     CX
        POP     DS
        RET                     ;and exit
B_SORT  ENDP
CSEG    ENDS
        END
```

برنامه زیر روش بهتری برای مرتب نمودن عناصر یک آرایـه بـروش حبـابی ارائه می‌دهد.

```
;       Arranges the 16-bit elements of a list in the extra
;       Segment in ascending order, using bubble sort.
;       Inputs: ES:DI = starting address of the list
;               First location = Length of list (words)
;       DI is unaltered.

;       Assemble with: MASM BUBBLE;
;       Link with: LINK callprog + BUBBLE;

DSEG        SEGMENT   PARA 'DATA'
SAVE_CNT    DW    ?
START_ADDR  DW    ?
DSEG        ENDS

            PUBLIC    BUBBLE
CSEG        SEGMENT   PARA 'CODE'
            ASSUME    CS:CSEG,DS:DSEG
BUBBLE      PROC  FAR
            PUSH  DS                ;Save caller's registers
            PUSH  CX
            PUSH  AX
            PUSH  BX
            MOV   AX, DSEG          ;Initialize DS
            MOV   DS,AX
            MOV   START_ADDR,DI
            MOV   CX,ES:[DI]        ;Fetch element count
            MOV   SAVE_CNT,CX       ;Save this value in memory
```

```
INIT:   MOV     BX,1            ;Exchange flag (BX) = 1
        DEC     SAVE_CNT        ;Get ready for count-1 compares
        JZ      SORTED          ;Exit if SAVE_CNT is 0
        MOV     CX,SAVE_CNT     ;and load this count into CX
        MOV     DI, START_ADDR  ;Load start address into DI
NEXT:   ADD     DI,2            ;Address a data element
        MOV     AX,ES:[DI]      ;and load it into AX
        CMP     ES:[DI+2],AX    ;Is next el. <this el.?
        JAE     CONT            ;No. Go check next pair
        XCHG    ES:[DI+2],AX    ;Yes. Exchange these elements.
        MOV     ES:[DI],AX
        SUB     BX,BX           ;and make exchange flag 0
CONT:   LOOP    NEXT            ;Process entire list
        CMP     BX,0            ;Any exchanges made?
        JE      INIT            ;If so, process list again
SORTED: MOV     DI, START_ADDR  ;If not, restore registers
        POP     BX
        POP     AX
        POP     CX
        POP     DS
        RET                     ;and exit
BUBBLE  ENDP
CSEG    ENDS
        END
```

در ذیل الگوریتم جستجوی دودوئی مطرح گردیده است.



برنامه ذیل روش جستجوی دودوئی برای مقادیر 16 بیتی ارائه می‌دهد.

```
;       Searches an ordered list in the extra segment for the
;       Word value contained in AX.
;       Inputs : ES:DI = starting address of the list
;                First location = Length of list (words)
;       Results: if the value is in the list,
;                CF=0
;                SI=offset of matching element
;                If the value is not in the list,
;                CF = 1
;                SI = offset of last element compared
;       AX and DI are unaffected.

;       Assemble with: MASM B_SEARCH;
;       Link with: LINK callprog+B_SEARCH;

  DSEG      SEGMENT   PARA 'DATA'
START_ADDR  DW    ?
  DSEG      ENDS

            PUBLIC    B_SEARCH
  CSEG      SEGMENT   PARA 'CODE'
            ASSUME    CS:CSEG,DS:DSEG
B_SEARCH    PROC      FAR
            PUSH      DS              ;Save caller's DS register
            PUSH      AX
            MOV       AX,DSEG         ;Initialize DS
            MOV       DS,AX
            POP       AX
```

```
;           Find out if AX lies beyond the boundaries of the list.
            CMP     AX,ES:[DI+2]    ;Search value <or=first el.?
            JA      CHK_LAST        ;No. Go check last element
            LEA     SI,ES:[DI+2]    ;Yes. Fetch addr. Of first el.
            JE      EXIT            ;If value = 1 st element, exit
            STC                     ; If value < 1 st element, set CF
            JMP     EXIT            ;and then exit
CHK_LAST:   MOV     SI,ES:[DI]      ;Point to last element
            SHL     SI,1
            ADD     SI,DI
            CMP     AX,ES:[SI]      ;Search value > or = last el.?
            JB      SEARCH          ;No. Go search list
            JE      EXIT            ;Yes. Exit if value = last el.
            STC                     ;If value > last element, set CF
            JMP     EXIT            ;and then exit
;           Search for value within the list.
SEARCH:     MOV     START_ADDR,DI   ;Save starting address in memory
            MOV     SI, ES:[DI]     ;Fetch index
EVEN_IDX:   TEST    SI,1            ;Force index to an even value
            JZ      ADD_IDX
            INC     SI
ADD_IDX:    ADD     DI,SI           ;Calculate next search address
COMPARE:    CMP     AX, ES:[DI]     ;Search value found?
            JE      ALL_DONE        ; If so, exit
            JA      HIGHER          ; Otherwise, find correct half
```

```
        ;       These instructions are executed if the search value is lower
        ;       In the list.
                CMP     SI , 2                  ;Index = 2?
                JNE     IDX _ OK
NO_MATCH:       STC                             ;If so, set CF
                JE      ALL_DONE                ;and exit
  IDX_OK:       SHR     SI ,1                   ;If not, divide index by 2
                TEST    SI, 1                   ;Force index to an even value
                JZ      SUB_IDX
                INC     SI
 SUB_IDX:       SUB     DI, SI                  ;Calculate next address
                JMP     SHORT COMPARE           ; Go check this element

        ;       These instructions are executed if the search value is higher
        ;       In the list.

  HIGHER:       CMP     SI,2                    ;Index = 2?
                JE      NO_MATCH                ;If so, go set CF and exit
                SHR     SI,1                    ;If not, divide index by 2
                JMP     SHORT EVEN_IDX          ;Go check next element

        ;       Following are exit instructions.
ALL_DONE:       MOV     SI,DI                   ;Move compare address into ST
                MOV     DI,START_ADDR           ;Restore starting address
    EXIT:       POP     DS
                RET                             ;and exit
B_SEARCH        ENDP
   CSEG         ENDS
                END
```

برنامه زیر یک عنصر را به یک لیست مرتب اضافه می‌نماید.

```
;       Adds the element in AX to an ordered list in the
;       Extra segment, if that value is not already in the list.
;       Inputs: DI = starting address of the list
;               First location = List length (words).
;       Result : None
;       DI and AX are unaltered.
;       The B_SEARCH procedure (Example 5-6) is used to conduct
;       The search.

;       Assemble with: MASM ADD_2_OL;
;       Link with: LINK callprog + ADD_2_OL +B_SEARCH;

        EXTRN   B_SEARCH:FAR
        PUBLIC  ADD_TO_OL
CSEG    SEGMENT PARA 'CODE'
        ASSUME  CS:CSEG
ADD_TO_OL PROC  FAR
        PUSH    CX          ; Save caller's registers
        PUSH    SI
        PUSH    BX
        CALL    B_SEARCH    ; Is the value in the list?
        JNC     GOODBYE     ; If so, exit
        MOV     BX,SI       ; if not, copy compare addr. To BX
        MOV     CX,ES: [DI] ; Find address of last element
        SHL     CX,1
        ADD     CX,DI       ; and put it in CX
        PUSH    CX          ; Save this address on the stack
        SUB     CX,SI       ; Calculate no. of words to be moved
        SHR     CX,1
```

```
                CMP     AX,ES: [SI]         ; should compare el. Be moved, too?
                JA      EXCLUDE
                INC     CX                  ; Yes. Increase move count by 1
                JNZ     CHECK_CNT
EXCLUDE:        ADD     BX,2                ; No. Adjust insert pointer
CHECK_CNT:      CMP     CX,0                ; Move count = 0?
                JNE     MOVE_ELS
                POP     SI                  ; If so, store value at end of list
                MOV     ES: [SI+2],AX
                JMP     SHORT INC_CNT       ; Then go increase element count
MOVE_ELS:       POP     SI                  ; Load start address for move
                PUSH    BX                  ; Save insert address on stack
MOVE_ONE:       MOV     BX, ES:[SI]         ; Move one element down in list
                MOV     ES: [SI+2], BX
                SUB     SI,2                ; Point to next element
                LOOP    MOVE_ONE            ; Repeat until all are moved
                POP     BX                  ; Retrieve insert address
                MOV     ES: [BX], AX        ; Insert AX in the list
INC_CNT:        INC     WORD PTR ES:[DI]    ; Add 1 to element count
GOODBYE:        POP     BX                  ; Restore registers
                POP     SI
                POP     CX
                RET                         ; and exit
ADD_TO_OL       ENDP
CSEG            ENDS
                END
```

برنامه داده شده در ذیل یک مقدار را از لیست مرتب داده شده حذف می‌نماید.

```
;       Deletes the value in AX from the ordered list in the
;       Extra segment, if the value is in the list.
;       Inputs: ES: DI = Starting address of the list
;               First location = Length of list (words)
;       AX and DI are unaffected.
;       The B_SEARCH procedure (Example 5-6) is used to conduct
;       The search.

;       Assemble with: MASM DEL_OL;
;       Link with: LINK callprog + DEL_OL + B_SEARCH;

        EXTRN   B_SEARCH : FAR
        PUBLIC  DEL_OL
CSEG    SEGMENT PARA 'CODE'
DEL_OL  PROC    FAR
        ASSUME  CS;CSEG
        PUSH    CX          ; Save caller's registers
        PUSH    SI
        PUSH    BX
        CALL    B_SEARCH    ; Is the value in the list?
        JC      ADIOS       ; If not, exit
        MOV     CX,ES: [DI] ; If so, find addr. Of last element
        SHL     CX,1
        ADD     CX,DI       ; and put it in CX
        CMP     CX,SI       ; Is the last el. To be deleted?
        JE      CNT_M1      ; Yes. Go decrement el. Count
        SUB     CX,SI       ; No. calculate move count
        SHR     CX,1
```

```
MOVEM:    MOV     BX,ES: [SI+2]       ; Move one element up in list
          MOV     ES: [SI], BX
          ADD     SI,2                ; Point to next element
          LOOP    MOVEM               ; Repeat until all are moved
CNT_M1:   DEC     WORD PTR ES:[DI]    ; Decrease element count by 1
ADIOS:    POP     BX                  ; Restore registers
          POP     SI
          POP     CX
          RET                         ; and exit
DEL_OL    ENDP
CSEG      ENDS
          END
```

مروری بر مطالب فصل

در این فصل نحوه نوشتن برنامه‌ها به زبان اسمبلی و نحوه اجرای آنها بحث گردیده است. ضمناً چندین برنامه بصورت نمونه نوشته شده که می‌تواند الگوئی برای برنامه نویسی دانشجویان باشد.

فصل دهم

اسمبلی 80386

هدف کلی

در این فصل ریزپردازنده 80386 و زبان اسمبلی وابسته بحث می‌گردد.

اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می‌شوید.

۱- ریزپردازنده 80386 .

۲- انواع data هائی که حمایت می‌نماید و محاسبه آدرس موثر

۳- معماری و ثباتهای آن.

٤- آشنائی با مجموعه دستورالعملهای 80386.

۱-۱۰- ریز پردازنده 80386

ریز پردازنده 80386 یک ریزپردازنده 32 بیتی می‌باشد که برای سیستم عاملهائی که عمل چند وظیفه‌ای (Multitasking) را انجام می‌دهند طراحی شده است. با توجه به ثباتهای 32 بیتی آن می‌تواند آدرسها و انواع داده‌های 32 بیتی را حمایت نماید. این ریزپردازنده قادر است تا چهار گیگابایت حافظه فیزیکی و 64 ترابایت (Terabyte) حافظه مجازی را آدرس دهی نماید. با توجه به پهنای باند بالای bus، امکانات استفاده pipelining ، تراشه تبدیل آدرس، این ریزپردازنده زمان اجرای متوسط دستورالعملها را خیلی کم و به حداقل می‌رساند. این ریز پردازنده قادر است که 3 تا 4 میلیون دستورالعمل را در یک ثانیه اجرا نماید.

۲-۱۰- انواع داده‌ها

تراشه ریز پردازنده 80386 چندین نوع داده علاوه بر آنهائی که بوسیله 8086/80286 حمایت می‌شوند را پشتیبانی می‌نماید. این ریزپردازنده مقادیر صحیح با علامت و بدون علامت 32 بیتی را حمایت می‌نماید. همچنین اطلاعات بیتی از 1 تا 32 بیت را پشتیبانی می‌نماید. این ریزپردازنده انواع اشاره‌گرهای استاندارد استفاده شده بوسیله 8086/80286 و اشاره‌گرهای 32 و 48 بیتی را حمایت می‌نماید. بطور کلی 80386 از عملوندهای 32, 16, 8 بیتی می‌تواند استفاده نماید.

در ریزپردازنده 80386 بایستی به نکات ذیل توجه نمود.

الف) یک بایت از هشت بیت پیوسته تشکیل شده است.

ب)یک word از شانزده بیت پشت سر هم تشکیل شده است.

ج)یک double word از 32 بیت کنار هم تشکیل شده است.

د)یک quad word از 64 بیت پشت سر هم تشکیل شده است.

که بوسیله دیرکتیوهای زیر در زبان اسمبلی مورد استفاده قرار می‌گیرند.

DB	Define byte
DD	Define double word
DQ	Define quad word
DW	Define word

۳-۱۰- محاسبه آدرس مؤثر (Effective Address)

زمانیکه می‌خواهیم به سگمنت بزرگتر از 64k دسترسی پیدا نمائیم آدرس مؤثر می‌تواند 32 بیتی باشد. آدرس مؤثر از مجموع ثبات مبنا، ثبات شاخص و یک جابجائی بدست می‌آید.

٤-۱۰- معماری

ریزپردازنده 80386 امکان استفاده از 32 ثبات که آنها را می توان به هفت دسته زیر تقسیم نمود را فراهم می‌نماید.

۱- ثباتهای مصرف عمومی (General-purpose registers).

۲- ثباتهای سگمنت (Segment registers).

۳- فلگها و اشاره‌گر دستورالعمل (Instruction pointer and flags).

٤- ثباتهای کنترل (Control registers).

٥- ثباتهای آدرس سیستم (System address registers).

٦- ثباتهای تست (Test registers).

ثباتهای مصرف عمومی امکان ذخیره8 ، 16 ، 32 بیت داده را دارند.

تعداد این ثباتها هشت تا می‌باشد و اسامی آنها عبارتند از:

EAX	Accumulator
EBX	Base
ECX	Count
EDX	Data
ESP	Stack pointer
EBP	Base pointer

ESI	Source Index
EDI	Destination Index

	31 16	15		0
EAX		[AH]	AX	[AL]
EBX		[BH]	BX	[BL]
ECX		[CH]	CX	[CL]
EDX		[DH]	DX	[DL]

در حقیقت بیت‌های 0 تا 15 ثبات EAX همان ثبات AX می باشد که خود نیز به دو ثبات هشت بیتی AL و AH تقسیم می‌گردند. همینطور در مورد سایر ثباتهایEBX ، ECX ،EDX .

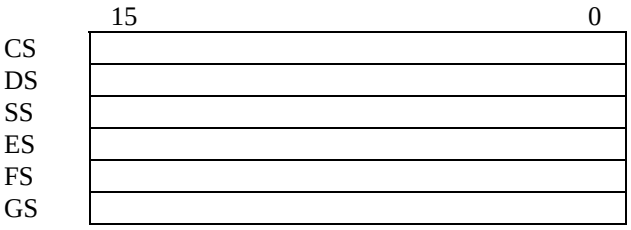
80386 دارای چهار ثبات 32 بیتی بنامهای ESP ، EBP ، ESI ، EDI می‌باشد که ثباتهای SP ، BP ، SI ، DI بترتیب شانزده بیت اول این ثباتها می باشند

SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index

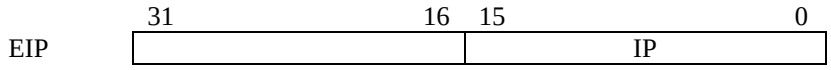
31 16	15 0	
	SP	ESP
	BP	EBP
	SI	ESI
	DI	EDI

80386 دارای شش ثبات سگمنت 16 بیتی می‌باشد. اسامی این ثباتها عبارتند از:

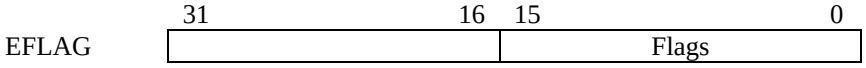
CS	Code	Segment
DS	Data	Segment
SS	Stack	Segment
ES	Extra	Segment
FS		
GS		



80368 دارای یک ثبات اشاره گر دستورالعمل می‌باشد که 32 بیتی است بنـام EIP . محتوی ثبات EIP آدرس مـؤثر دسـتورالعمل بعـدی کـه بایـستی اجـرا شـود می‌باشد. 16 بیت اول ثبات EIP بنام IP می‌باشد که بصورت جداگانه می‌توان بـه آن دسترسی پیدا نمود.



ثبات فلگ یا فلگ ثبات در ریزپردازنده 80386 به 32 بیت توسعه یافته است. ثبات فلگ بنام EFLAGS مـی‌باشـد. البتـه بایـستی توجـه داشـت کـه 16 بیـت اول EFLAGS همان فلگ ثبات مطرح شده در فصلهای قبل می‌باشد.



ساختار ثبات EFLAGS در ذیل داده شده است.

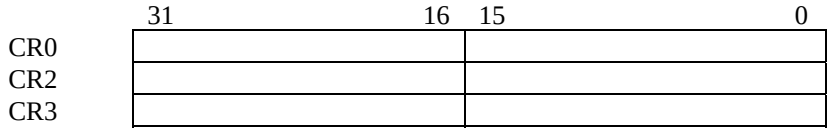
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NT	IO	PL	OF	DF	IF	TF	SF	ZF	//	AF	//	PF	//	CF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														VM	RF

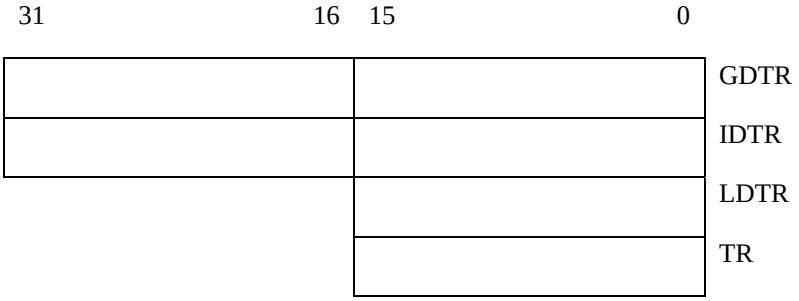
که

RF	Resume Flag
VM	Virtual Mode Flag

ریزپردازنده 80386 دارای سه ثبات کنترل 32 بیتی بنامهـای CR3, CR2, CR0 می باشد. ثبات CR0 شامل شش فلگ از قبل تعریف شده برای کنترل ریزپردازنده و وضعیت آن می‌باشد.



همچنین ریزپردازنده 80386 دارای چهار ثبات با کاربرد مخـصوص بنامهـای GDTR ، IDTR ، LDTR ، TR مـی‌باشـد. کـه GDTR و IDTR ثباتهـای 32 بیتـی و LDTR ، TR ثباتهای شانزده بیتی می‌باشند.



GETR	Global descriptor table register
IDTR	Interrupt descriptor table register
LDT	Local descriptor table register
TR	Task state segment table register

۵-۱۰- دستورالعملهای 80386

بایستی توجه داشت که کلیه مطالبی که در مورد ریزپردازنده 80286 در فصلهای قبل مطرح شد توسط ریزپردازنده 80386 کاملاً می‌توانند بدون هیچگونه تغییری مورد استفاده قرار گیرند یعنی کلیه برنامه‌هائی که تاکنون نوشته شده می‌توان بوسیله ریزپردازنده 80386 اجرا نمود. ضمناً فهرستی از دستورالعملهای 80386 در انتهای این فصل داده شده است. بایستی توجه داشت که در ریزپردازنده 80386 عملوندها می‌توانند 32 بیتی باشند که اینکار در مورد 80286 امکان پذیر نبود.

مثال

```
ADD    AX, BX
ADD    EBX , ECX
```

که دومین دستورالعمل محتوی ECX که 32 بیتی می‌باشد با محتوی EBX که 32 بیتی می‌باشد جمع نموده نتیجه در EBX قرار می‌دهد.

مثال دیگر

```
ADD    AL , 4
ADD    EAX , 98765432H
```

که مقدار ثابت داده شده را با محتوی EAX جمع می‌نماید.

یا در مورد AND می‌توانیم دستورالعملهای ذیل را بکار نبریم.

```
AND    AX , BX
AND    EAX , ECX
```

در مورد دستورالعمل CBW، همانطوریکه قبلاً گفته می‌شود می‌توان برای تبدیل بایت به word استفاده نمود. برای این کار مقدار مورد نظر را در AL قرار می‌دهیم و پس از اجرای دستورالعمل CBW نتیجه در AX قرار می‌گیرد. اما برای

تبدیل word به double word کافی است که مقدار مورد نظـر را در ثبـات AX قـرار داده پس از اجرای دستورالعمل CBW نتیجه در EAX قرار می‌گیرد.

در مورد ADC می‌توانیم دستورالعملهای ذیل را استفاده نمائیم.

```
ADC     AL , 4
ADC     AX , 298
ADC     EBX , 22334455H
ADC     TABLE [SI] , 2
ADC     NUMBE, 12345678
ADC     DL , BL
ADC     SI , X
ADC     X, SI
```

در مورد CMP می‌توانیم از دستورالعلهای ذیل استفاده نمائیم:

```
CMP     BL , 5
CMP     EAX , 0FFFF0000H
CMP     AL , 7
CMP     AX , BX
CMP     EDX , EAX
CMP     EBX , Y
```

در مورد MOV از دستورالعملهای ذیل بعنوان مثال می‌توانیم استفاده نمائیم:

```
MOV     AX , Y
MOV     ECX , X
MOV     X , AL
MOV     BX , ES
MOV     TABLE [BX] , SS
MOV     DS , AX
MOV     EAX , ECX
MOV     CX, [BP][SI]
MOV     EAX, 12345678H
```

در مورد دستورالعمل ضرب بعنوان مثـال مـی‌تـوانیم از دسـتورالعمل ذیـل اسـتفاده نمائیم:

```
MOV     EAX, 0FCAB1234H
MUL     EBX
```

که نتیجه در EDX : EAX ذخیره می‌گردد.

در مورد NEG می‌توان از دستورالعملهای ذیل استفاده نمود:

```
NEG     AL
NEG     EBX
```

در مورد NOT می‌توانیم بعنوان مثال دستورالعملهای ذیل را بکار بریم:

```
NOT     AL
NOT     AX
NOT     EBX
```

در مورد PUSH و POP می‌توانیم بعنوان مثال از دستورالعملهای ذیل استفاده نمائیم:

```
POP     CX
POP     EAX
POP     SS
PUSH    BX
PUSH    EAX
```

مجموعه کامل دستورالعملهای 80386

جدول ۱-۱۰

Instruction	Meaning	Assembler Format	
AAA	ASCII Adjust after Add	AAA	
AAD	ASCII Adjust before Divide	AAD	
AAM	ASCII Adjust after Multiply	AAM	
AAS	ASCII Adjust after Subtract	AAS	
ADC	Add with Carry	ADC	Dest,src
ADD	Add	ADD	Dest,src
AND	Logical AND	AND	Dest,src
ARRL	Adjust RPL Field of Selector	ARPL	Sel,reg
BOUND	Check Array Bounds	BOUND	Reg,bound
BSF	Bit Scan Forward	BSF	Dest,src
BSR	Bit Scan Reverse	BSR	Dest,src
BT	Bit Test	BT	Base,offset
BTC	Bit Test and complement	BTC	Base,offset
BTR	Bit Test and Reset	BTR	Base,offset
BTS	Bit Test and Set	BTS	Base,offset
CALL	Call Procedure	CALL	Dest
CBW	Convert Byte to Word	CBW	
CDQ	Convert Double Word to Quad Word	CDQ	
CLC	Clear Carry Flag	CLC	
CLD	Clear Direction Flag	CLD	
CLI	Clear Interrupt Flag	CLI	
CLTS	Clear Task-Switched Flag	CLTS	
CMC	Complement Carry Flag	CMC	
CMP	Compare	CMP	Dest,src
CMPS	Compare Strings	CMPS	Dest,src
CWD	Convert Word to Double Word*	CWD	
CWDE	Convert Word to Double Word*	CWDE	
DAA	Decimal Adjust after Add	DAA	
DAS	Decimal Adjust after Subtract	DAS	
DEC	Decrement by 1	DEC	Dest
DIV	Unsigned Divide	DIV	Acc,src
ENTER	Make Stack Frame for Procedure	ENTER	Storage,level
ESC	Escape	ESC	
HLT	Halt	HLT	
IDIV	Signed Divide	IDIV	Acc,src
IMUL	Signed Multiply	IMUL	Acc,src
IN	Input from Port	IN	Acc,port
INC	Increment by 1	INC	Dest
INT	Software Interrupt (Trap)	INT	Inttype
INTO	Interrupt If Overflow	INTO	
IRET	Interrupt Retum	IRET	

Instruction	Meaning	Assembler Format	
JA	Jump If Above	JA	Dest
JAE	Jump If Above or Equal	JAE	Dest
JB	Jump If Below	JB	Dest
JBE	Jump If Below or Equal	JBE	Dest
JC	Jump If Carry	JC	Dest
JCXZ	Jump If CX Is Zero	JCXZ	Dest
JE	Jump If Equal	JE	Dest
JECXZ	Jump If ECX Is Zero	JECXZ	Dest
JG	Jump If Greater	JG	Dest
JGE	Jump If Greater or Equal	JGE	Dest
JL	Jump If Less	JL	Dest
JLE	Jump If Less or Equal	JLE	Dest
JMP	Jump Unconditionally	JMP	Dest
JNA	Jump If Not Above	JNA	Dest
JNAE	Jump If Not Above or Equal	JNAE	Dest
JNB	Jump If Not Below	JNB	Dest
JNBE	Jump If Not Below or Equal	JNBE	Dest
JNC	Jump If No Carry	JNC	Dest
JNE	Jump If Not Equal	JNE	Dest
JNG	Jump If Not Greater	JNG	Dest
JNGE	Jump If Not Greater or Equal	JNGE	Dest
JNL	Jump If Not Less	JNL	Dest
JNLE	Jump If Not Less or Equal	JNEL	Dest
JNO	Jump if No Overflow	JNO	Dest
JNP	Jump if Parity Odd	JNP	Dest
JNS	Jump if Sign Positive	JNS	Dest
JNZ	Jump if Not Zero	JNZ	Dest
JO	Jump if Overflow	JO	Dest
JP	Jump if parity Even	JP	Dest
JPE	Jump if parity Even	JPE	Dest
JPO	Jump if parity Odd	JPO	Dest
JS	Jump if Sign Negative	JS	Dest
JZ	Jump if Zero	JZ	Dest
LAHF	Load Flags into AH Register	LAHF	
LAR	Load Access Rights Byte	LAR	Reg,src
LDS	Load DS Register	LDS	Reg,src
LEA	Load Effective Address	LEA	Reg,src
LEAVE	Leave Procedure	LEAVE	
LES	Load ES Register	LES	Reg,src
LFS	Load FS Register	LFS	Reg,src
LGS	Load GS Register	LGS	Reg,src
LGDT	Load GDT Register	LGDT	Src
LIDT	Load IDT Register	LIDT	Src

Instruction	Meaning	Assembler Format	
LLDT	Load LDT Register	LLDT	Src
LMSW	Load Machine Status Word	LMSW	Src
LOCK	Lock Bus	LOCK	
LODS	Load String	LODS	Src
LOOP	Loop with CX Counter	LOOP	Dest
LOOPE	Loop If Equal	LOOPE	Dest
LOOPNE	Loop If Not Equal	LOOPNE	Dest
LOOPNZ	Loop If Not Zero	LOOPNZ	Dest
LOOPZ	Loop If Zero	LOOPZ	Dest
LSL	Load Segment Limit	LSL	Reg,src
LSS	Load SS Register	LSS	Reg,src
LTR	Load Task Register	LTR	Src
MOV	Move Data	MOV	Dest,src
MOV	Move to/from Special Regs	MOV	Dest,src
MOVS	Move String	MOVS	Dest,src
MOVSX	Move with Sign-Extend	MOVSX	Reg,src
MOVZX	Move with Zero-Extend	MOVZX	Reg,src
MUL	Unsigned Multiply	MUL	Acc,src
NEG	2's Complement Negation	NEG	Dest
NOP	No Operation	NOP	
NOT	1's Complement Negation	NOT	Dest
OR	Logical Inclusive OR	OR	Dest,src
OUT	Output to Port	OUT	Port,acc
OUTS	Output String	OUTS	DX,src
POP	Pop Operand off Stack	POP	Dest
POPA	Pop All General Registers	POPA	
POPF	Pop Flags off Stack	POPF	
PUSH	Pop Operand Onto Stack	PUSH	Src
PUSHA	Push All General Registers	PUSHA	
PUSHF	Push Flags onto Stack	PUSHF	
RCL	Rotate Left through Carry	RCL	Dest,count
RCR	Rotate Right through Carry	RCR	Dest,count
REP	Repeat	REP	
REPE	Repeat while Equal	REPE	
REPNE	Repeat while Not Equal	REPNE	
REPNZ	Repeat while Not Zero	REPNZ	
REPZ	Repeat while Zero	REPZ	
RET	Repeat from Procedure	RET	
ROL	Rotate Left	ROL	Dest,count
ROR	Rotate Right	ROR	Dest,count
SAHF	Store AH Register in Flags	SAHF	
SAL	Shift Arithmetic Left	SAL	Dest,count
SAR	Shift Arithmetic Right	SAR	Dest,count

Instruction	Meaning	Assembler Format	
SBB	Subtract with Borrow	SBB	Dest,src
SCAS	Compare String	SCAS	Dest
SETcc	Set Byte on Condition	SETcc	Dest
SGDT	Store GDT Register	SGDT	Dest
SHL	Shift Logical Left	SHL	Dest, count
SHLD	Double Precision Shift Left	SHLD	Dest, src,count
SHR	Shift Logical Right	SHR	Dest,count
SHRD	Double Precision Shift Right	SHRD	Dest,src,count
SIDT	Store IDT Register	SIDT	Dest
SLDT	Store LDT Register	SLDT	Dest
SMSW	Store Machine Status Word	SMSW	Dest
STC	Set Carry Flag	STC	
STD	Set Direction Flag	STD	
STI	Set Interrupt Flag	STI	
STOS	Store String	STOS	Dest
STR	Store Task Register	STR	Dest
SUB	Subtract	SUB	Dest,src
TEST	Logical Compare	TEST	Dest,src
VERR	Verify Segment for Reading	VERR	Sel
VERW	Verify Segment for Writing	VERW	Sel
WAIT	Wait until BUSY#Negated	WAIT	
XCHG	Exchange Operand, Register	XCHG	Dest,src
XLAT	Table Lookup	XLAT	Source-table
XOR	Logical Exclusive OR	XOR	Dest,src

*CWD sign extends register AX into registers DX and AX, whereas CWDE sign extends AX into EAX.

جدول ۲-۱۰

DEC	Subtract 1
INC	Add 1
NOT	Logical NOT (complement or invert)
ROL	Rotate left
ROR	Rotate right
SBB	Subtract with borrow
SHL	Shift logical left
SHR	Shift logical right
SUB	Subtract
TEST	Bit test
Program control	
CALL	Call subroutine
INT	Interrupt (trap)
JA	Jump if above
JAE	Jump if above or equal
JB	Jump if below
JBE	Jump if below or equal
JC	Jump if carry
JE	Jump if equal
JMP	Jump unconditionally
JNC	Jump if not carry
JNE	Jump if not equal
JNS	Jump if not sign
JNZ	Jump if non zero
JS	Jump if sign
JZ	Jump if zero
RET	Return from subroutine

جدول ٣-١٠

Frequently Used 80386 Instructions

ADC	Add with carry
ADD	Add
AND	Logical AND
CALL	Call subroutine
CMP	Compare
DEC	Subtract 1
IN	Input
INC	Add 1
INT	Interrupt (trap)
JA	Jump if above
JAE	Jump if above or equal
JB	Jump if below
JBE	Jump if below or equal
JC	Jump if carry
JE	Jump if equal
JMP	Jump unconditionally
JNC	Jump if not carry
JNE	Jump if not equal
JNS	Jump if sign positive
JNZ	Jump if not zero
JS	Jump if sign negative
LEA	Load effective address
MOV	Move
NOT	Logical NOT
	(complement or invert)
OUT	Output
POP	Load from stack
PUSH	Store on stack
RET	Return from subroutine
ROL	Rotate left
ROR	Rotate right
SBB	Subtract with borrow

مروری بر مطالب فصل

در این فصل در مورد ریز پردازنده ۸۰۳۸٦ بحث گردید. همانطوری که گفته شد این ریز پردازنده یک ریزپردازنده ۳۲ بیتی می باشد و تمام دستورالعمل هایی که روی ریزپردازنده ۸۰۲۸٦ اجرا می شود روی این ریزپردازنده نیزقابل اجرا می باشد. ثبات های ۳۲ بیتی آن عبارتند از EAX ، EBX ، ECX ، EDX . در ضمن می توان از ثبات های AX ، BX ، CX ، DX ، AL ، AH ، BL ، BH ، CL ، CH ، DL ، DH و ... استفاده نمود.درنهایت جدول کلی فرمت دستورالعملها برای ریزپردازنده ۸۰۳۸٦ داده شده است که با مرور آن می توان برنامه های مختلفی را به راحتی نوشت و بر روی این ریزپردازنده اجرا نمود

ضمیمه ۱

عملـگرها (OPERATORS)

Operator	Function
Arithmetic	
+	Format: value1 + value2 Adds value1 and value2.
-	Format: value1 – value2 Subtracts value2 from value1.
*	Format: value1 * value2 Multiplies value2 by value1.
/	Format: value1 / value2 Divides value1 by value2, and returns the quotient.
MOD	Format: value1 MOD value2 Divides value1 by value2, and returns the Remainder.
SHL	Format: value SHL expression Shifts value left by expression bit positions.
SHR	Format: value SHR expression Shifts value right by expression bit positions.

Operator	Function
Logical	
AND	Format: value1 AND value2 Takes logical AND of value1 and value2.
OR	Format: value1 OR value2 Takes logical inclusive-OR of value1 and value2.
XOR	Format: value1 XOR valure2 Takes logical exclusive-OR of value1 and Value2.
NOT	Format: NOT value Reverses the state of each bit in value; that is, it takes the one's complement.
Relational	
EQ	Format: operand1 EQ operand2 True if the two operands are identical.
NE	Format: operad1 NE operand2 True if the two operands are not identical.
LT	Format: operand1 LT operand2 True if operand1 is less than operand2.
GT	Format: operand1 GT operand2 True if operand1 is greater than operand2.
LE	Format: operand1 LE operand2 True if operand1 is less than or equal to operand2.
GE	Format: operand1 GE operand2 True if operand1 is greater than or equal to operand2.
Value - Returning	
$	Format: $ Returns the current value of the location counter.
SEG	Format: SEG variable Or SEG label Returns the segment value of variable or label.

Operator	Function
OFFSET	Format: OFFSET variable Or OFFSET label Returns the offset value of variable or label.
LENGTH	Format: LENGTH variable Returns the length in units (bytes or words) for any variable defined using DUP.
TYPE	Format: TYPE variable Or TYPE label For variables, TYPE returns 1 (BYTE), 2(WORD), or 4 (DOUBLEWORD). For labels, it returns –1 (NEAR) or-2(FAR).
Value-Returning	
SIZE	Format: SIZE variable Returns the product of LENGTH times TYPE.
Attribute	
PTR	Format: type PTR expression Overrides the type (BYTE or WORD) or distance (NEAR or FAR) of a memory address operand. Type is the new attribute and expression is the identifier whose attribute is to to be overridden.
DS: **ES:** **SS:** **CS:**	Format: seg –reg: addr-expr or seg-reg: label or seg-reg: variable Overrides the segment attribute of a label, variable, or address expression.
SHORT	Format: JMP SHORT labe1 Tells the assembler that the JMP target label is no farther than 127 bytes past the next instruction.
THIS	Format: THIS attribute or THIS type Creates a memory address operand of either distance attribute (NEAR or FAR) or either type attribute (BYTE or WORD) at an offset equal to the current value of the location counter and a segment attribute of the enclosing segment.

Operator	Function
HIGH	Format: HIGH value or HIGH expression Returns the high-order byte of a 16-bit numeric value or address expression.
Attribute	
LOW	Format: LOW value or LOW expression Returns the low-order byte of a 16-bit numeric value or address expression.

ضمیمه شماره ۲

Instruction Set Summary

For each instruction, it shows the general assembler format and which flags are affected. In the Flags column, -means unchanged, * means may have changed, and? Means undefined.

mnemonic	Assembler	format	Flags								
			OF	DF	IF	TF	SF	ZF	AF	PF	CF
AAA	AAA		?	-	-	-	?	?	*	?	*
AAD	AAD		?	-	-	-	*	*	?	*	?
AAM	AAM		?	-	-	-	*	*	?	*	?
AAS	AAS		?	-	-	-	?	?	*	?	*
ADC	ADC	Destination,source	*	-	-	-	*	*	*	*	*
ADD	ADD	Destination,source	*	-	-	-	*	*	*	*	*
AND	AND	Destination,source	0	-	-	-	*	*	?	*	0
BOUND	BOUND	Reg16,source	-	-	-	-	-	-	-	-	-
CALL	CALL	Target	-	-	-	-	-	-	-	-	-
CBW	CBW		-	-	-	-	-	-	-	-	-
CLC	CLC		-	-	-	-	-	-	-	-	0
CLD	CLD		-	0	-	-	-	-	-	-	-
CLI	CLI		-	-	0	-	-	-	-	-	-
CMC	CMC		-	-	-	-	-	-	-	-	*
CMP	CMP	Destination,source	*	-	-	-	*	*	*	*	*
CMPS	CMPS	Dest-string,source-string	*	-	-	-	*	*	*	*	*
CMPSB	CMPSB		*	-	-	-	*	*	*	*	*
CMPSW	CMPSW		*	-	-	-	*	*	*	*	*
CWD	CWD		-	-	-	-	-	-	-	-	-
DAA	DAA		?	-	-	-	*	*	*	*	*
DAS	DAS		?	-	-	-	*	*	*	*	*
DEC	DEC	Destination	*	-	-	-	*	*	*	*	-
DIV	DIV	Source	?	-	-	-	?	?	?	?	?
ENTER	ENTER	Immed16,level	-	-	-	-	-	-	-	-	-
ESC	ESC	Ext,opcode,source	-	-	-	-	-	-	-	-	-
HLT	HLT		-	-	-	-	-	-	-	-	-
IDIV	IDIV	source	?	-	-	-	?	?	?	?	?
IMUL	IMUL	Source	*	-	-	-	?	?	?	?	*
IMUL	IMUL	Dest[,source],immed	*	-	-	-	?	?	?	?	*
IN	IN	Accumulator,port	-	-	-	-	-	-	-	-	-
INC	INC	Destination	*	-	-	-	*	*	*	*	-
INS	INS	Dest-string,DX	-	-	-	-	-	-	-	-	-
INT	INT	Interrupt-type	-	-	0	0	-	-	-	-	-
INTO	INTO		-	-	0	0	-	-	-	-	-

mnemonic	Assembler	format	Flags								
			OF	DF	IF	TF	SF	ZF	AF	PF	CF
IRET	IRET		*	*	*	*	*	*	*	*	*
JA/JNBE	JA	Short-label	-	-	-	-	-	-	-	-	-
JAE/JNB	JAE	Short-label	-	-	-	-	-	-	-	-	-
JB/JNAE/JC	JB	Short-label	-	-	-	-	-	-	-	-	-
JBE/JNA	JBE	Short-label	-	-	-	-	-	-	-	-	-
JCXZ	JCXZ	Short-label	-	-	-	-	-	-	-	-	-
JE/JZ	JE	Short-label	-	-	-	-	-	-	-	-	-
JG/JNLE	JG	Short-label	-	-	-	-	-	-	-	-	-
JGE/JNL	JGE	Short-label	-	-	-	-	-	-	-	-	-
JL/JNGE	JL	Short-label	-	-	-	-	-	-	-	-	-
JLE/JNG	JLE	Short-label	-	-	-	-	-	-	-	-	-
JMP	JMP	Target	-	-	-	-	-	-	-	-	-
JNC	JNC	Short-label	-	-	-	-	-	-	-	-	-
JNE/JNZ	JNE	Short-label	-	-	-	-	-	-	-	-	-
JNO	JNO	Short-label	-	-	-	-	-	-	-	-	-
JNP/JPO	JNP	Short-label	-	-	-	-	-	-	-	-	-
JNS	JNS	Short-label	-	-	-	-	-	-	-	-	-
JO	JO	Short-label	-	-	-	-	-	-	-	-	-
JP/JPE	JP	Short-label	-	-	-	-	-	-	-	-	-
JS	JS	Short-label	-	-	-	-	-	-	-	-	-
LAHF	LAHF		-	-	-	-	-	-	-	-	-
LDS	LDS	Reg16,mem32	-	-	-	-	-	-	-	-	-
LEA	LEA	Reg16,mem16	-	-	-	-	-	-	-	-	-
LEAVE	LEAVE		-	-	-	-	-	-	-	-	-
LES	LES	Reg16,mem32	-	-	-	-	-	-	-	-	-
LOCK	LOCK		-	-	-	-	-	-	-	-	-
LODS	LODS	Source-string	-	-	-	-	-	-	-	-	-
LODSB	LODSB		-	-	-	-	-	-	-	-	-
LODSW	LODSW		-	-	-	-	-	-	-	-	-
LOOP	LOOP	Short-label	-	-	-	-	-	-	-	-	-
LOOPE/LOOPZ	LOOPE	Short-label	-	-	-	-	-	-	-	-	-
LOOPNE/LOOPNZ	LOOPNE	Short-label	-	-	-	-	-	-	-	-	-
MOV	MOV	Destination, source	-	-	-	-	-	-	-	-	-
MOVS	MOVS	Dest-string,source-string	-	-	-	-	-	-	-	-	-
MOVSB	MOVSB		-	-	-	-	-	-	-	-	-
MOVSW	MOVSW		-	-	-	-	-	-	-	-	-
MUL	MUL	Source	*	-	-	-	?	?	?	?	*
NEG	NEG	Destination	*	-	-	-	*	*	*	*	*
NOP	NOP		-	-	-	-	-	-	-	-	-
NOT	NOT	Destination	-	-	-	-	-	-	-	-	-
OR	OR	Destination, source	0	-	-	-	*	*	?	*	0
OUT	OUT	Port, accumulator	-	-	-	-	-	-	-	-	-
OUTS	OUTS	DX, source-string	-	-	-	-	-	-	-	-	-
POP	POP	destination	-	-	-	-	-	-	-	-	-

mnemonic	Assembler	format	Flags								
			OF	DF	IF	TF	SF	ZF	AF	PF	CF
POPA	POPA		-	-	-	-	-	-	-	-	-
POPF	POPF		*	*	*	*	*	*	*	*	*
PUSH	PUSH	Source	-	-	-	-	-	-	-	-	-
PUSH	PUSH	Immediate	-	-	-	-	-	-	-	-	-
PUSHA	PUSHA		-	-	-	-	-	-	-	-	-
PUSHF	PUSHF		-	-	-	-	-	-	-	-	-
RCL/RCR	RCL	Destination,1	*	-	-	-	-	-	-	-	*
RCL/RCR	RCL	Destination, CL	?	-	-	-	-	-	-	-	*
RCL/RCR	RCL	Destination, count	?	-	-	-	-	-	-	-	*
REP	REP		-	-	-	-	-	-	-	-	
REPE/REPZ	REPE		-	-	-	-	-	-	-	-	-
REPNE/REPNZ	REPNE		-	-	-	-	-	-	-	-	-
RET	[pop-value]		-	-	-	-	-	-	-	-	-
ROL/ROR	ROL	Destination, 1	*	-	-	-	-	-	-	-	*
ROL/ROR	ROL	Destination, CL	?	-	-	-	-	-	-	-	*
ROL/ROR	ROL	Destination, count	?	-	-	-	-	-	-	-	*
SAHF	SAHF		-	-	-	-	*	*	*	*	*
SAL/SHL	SAL	Destination, 1	*	-	-	-	*	*	?	*	*
SAL/SHL	SAL	Destination, CL	?	-	-	-	*	*	?	*	*
SAL/SHL	SAL	Destination, count	?	-	-	-	*	*	?	*	*
SAR	SAR	Destination, 1	0	-	-	-	*	*	?	*	*
SAR	SAR	Destination, CL	?	-	-	-	*	*	?	*	*
SAR	SAR	Destination, count	?	-	-	-	*	*	?	*	*
SBB	SBB	Destination, source	*	-	-	-	*	*	*	*	*
SCAS	SCAS	Dest-string	*	-	-	-	*	*	*	*	*
SCASB	SCASB		*	-	-	-	*	*	*	*	*
SCASW	SCASW		*	-	-	-	*	*	*	*	*
SHR	SHR	Destination,1	*	-	-	-	0	*	?	*	*
SHR	SHR	Destination, CL	?	-	-	-	0	*	?	*	*
SHR	SHR	Destination, count	?	-	-	-	0	*	?	*	*
STC	STC		-	-	-	-	-	-	-	-	1
STD	STD		-	1	-	-	-	-	-	-	-
STI	STI		-	-	1	-	-	-	-	-	-
STOS	STOS	Dest-string	-	-	-	-	-	-	-	-	-
STOSB	STOSB		-	-	-	-	-	-	-	-	-
STOSW	STOSW		-	-	-	-	-	-	-	-	-
SUB	SUB	Destination, source	*	-	-	-	*	*	*	*	*
TEST	TEST	Destination, source	0	-	-	-	*	*	?	*	0
WAIT	WAIT		-	-	-	-	-	-	-	-	-
XCHG	XCHG	Destination, source	-	-	-	-	-	-	-	-	-
XLAT	XLAT	Source-table	-	-	-	-	-	-	-	-	-
XOR	XOR	Destination, source	0	-	-	-	*	*	?	*	0

ضمیمه شماره ۳

Instruction times

Instruction		Clocks	Bytes
AAA		3	1
AAD		14	2
AAM		16	1
AAS		3	1
ADC	Register, Register	2	2
ADC	Register, memory	7*	2-4
ADC	Memory, register	7*	2-4
ADC	Register, immediate	3	3-4
ADC	Memory, immediate	7*	3-6
ADC	Accumulator; immediate	3	2-3
ADD	Register, register	2	2
ADD	Register, memory	7*	2-4
ADD	Memory, register	7*	2-4
ADD	Register, immediate	3	3-4
ADD	Memory, immediate	7*	3-6
ADD	Accumulator; immediate	3	2-3
AND	Register, register	2	2
AND	Register, memory	7*	2-4
AND	Memory, register	7*	2-4
AND	Register, immediate	3	3-4
AND	Memory, immediate	7*	3-6
AND	Accumulator; immediate	3	2-3
BOUND reg16,source		13*	2
CALL	Near-proc	7+m	3
CALL	Far-proc	13+m	5
CALL	Memptr16	11+m*	2-4
CALL	Regptr16	7+m	2
CALL	Memptr32	16+m	2-4

Instruction		Clocks	Bytes
CBW		2	1
CLC		2	1
CLD		2	1
CL1		3	1
CMC		2	1
CMP	Register, register	2	2
CMP	Register, memory	6*	2-4
CMP	Memory, register	7*	2-4
CMP	Register, immediate	3	3-4
CMP	Memory, immediate	6*	3-6
CMP	Accumulator; immediate	3	2-3
CMPS	Dest-string, source-string	8	1
CMPS	(repeat)dest-string,source-string	5+9(rep)	1
CWD		2	1
DAA/DAS		3	1
DEC	Register	2	1-2
DEC	Memory	7*	2-4
DIV	Reg8	14	2
DIV	Reg16	22	2
DIV	Mem8	17*	2-4
DIV	Mem16	25*	2-4
ENTER	Immed16,0	11	4
ENTER	Immed16,1	15	4
ENTER	Immed16,level	12+4(L)	4
ESC	Immediate, memory	9-20*	2-4
ESC	Immediate, register	2	2
HLT		2	1
IDIV	Reg8	17	2
IDIV	Reg16	25	2

Instruction		Clocks	Bytes
IDIV	Mem8	20*	2-4
IDIV	Mem16	28*	2-4
IMUL	Reg8	13	2
IMUL	Reg16	21	2
IMUL	Mem8	16*	2-4
IMUL	Mem16	24*	2-4
IMUL	Dest-reg,reg16,immediate	21*	3-4
IMUL	Dest-reg, memory, immediate	24*	3-4
IN	Accumulator, immed8	5	2
IN	Accumulator, DX	5	1
INC	Register	2	1-2
INC	Memory	7*	2-4
INS	Dest-string, DX	5	1
INS	(rep) dest-string,DX	5+4(rep)	1
INT	Immed8	23+m	1-2
INTO		24+m or 3	1
IRET		17+M	1
All conditional jump instructions except JCXZ:			
Jccc	Short-label	7+m or 3	2
JCXZ	Short-label	8+m or 4	2
JMP	Short-label	7+m	2
JMP	Near-label	7+m	3
JMP	Far-label	11+m	5
JMP	Memptr16	11+m*	2-4
JMP	Regptr16	7+m	2
JMP	Memptr32	15+m*	2-4
LAHF		2	1
LDS	Reg16, mem32	7*	2-4
LEA	Reg16,mem16	3*	2-4

Instruction		Clocks	Bytes
LEAVE		5	1
LES	Reg16, mem32	7*	2-4
LOCK		0	1
LODS	Source-string	5	1
LODS	(repeat)source-string	5+4(rep)	1
LOOP Short-label		8+m or 4	2
LOOPE/LOOPZ Short-label		8+m or 4	2
LOOPNE/LOOPNZ Short-label		8+m or 4	2
MOV	Memory, accumulator	3	3
MOV	Accumulator, memory	5	3
MOV	Register, register	2	2
MOV	Register, memory	5*	2-4
MOV	Memory, register	3*	2-4
MOV	Register, immediate	2	2-3
MOV	Memory, immediate	3*	3-6
MOV	Seg-reg,reg16	2	2
MOV	Seg-reg-mem16	5*	2-4
MOV	Reg16,seg-reg	2	2
MOV	Memory, seg-reg	3*	2-4
MOVS	Dest-string,source-string	5	1
MOVS	(repeat)dest-string,source-string	5+4(rep)	1
MUL	Reg8	13	2
MUL	Reg16	21	2
MUL	Mem8	16*	2-4
MUL	Mem16	24*	2-4
NEG	Register	2	2
NEG	Memory	7*	2-4
NOP		2	1
NOT	Register	2	2
NOT	Memory	7*	2-4
OR	Register, register	2	2
OR	Register, memory	7*	2-4

Instruction		Clocks	Bytes
OR	Memory, register	7*	2-4
OR	Register, immediate	3	3-6
OR	Memory, immediate	7*	3-6
OR	Accumulator; immediate	3	2-3
OUT	Immed8, accumulator	3	2
OUT	DX, accumulator	3	1
OUTS	DX,source-string	5	1
OUTS	(rep)DX, source-string	5+4(rep)	1
POP	Register	5	1
POP	Memory	5*	2-4
POPA		19	1
POPE		5	1
PUSH	Register	3	1
PUSH	Memory	5*	2-4
PUSH	Immediate	3	2-3
PUSHA		17	1
PUSHF		3	1
RCL/RCR/ROL/ROR register,1		2	2
RCL/RCR/ROL/ROR register, CL		5+1/bit	2
RCL/RCR/ROL/ROR memory,1		7*	2-4
RCL/RCR/ROL/ROR memory, CL		8*+1/bit	2-4
RCL/RCR/ROL/ROR reg, count		5+1/bit	3
RCL/RCR/ROL/ROR memory, count		8*+1/bit	3-5
REP		0	1
REPE/REPZ		0	1
REPNE/REPNZ		0	1
RET	(near, no pop)	11+M	1
RET	(near, pop)	11+M	3
RET	(far, no pop)	15+M	1
RET	(far, pop)	15+M	3
SAHF		2	1

Instruction		Clocks	Bytes
SAL/SHL/SAR/SHR register, 1		2	2
SAL/SHL/SAR/SHR register, CL		5+1/bit	2
SAL/SHL/SAR/SHR memory,1		7*	2-4
SAL/SHL/SAR/SHR memory, CL		8*+1/bit	2-4
SAL/SHL/SAR/SHR reg, count		5+1/bit	3
SAL/SHL/SAR/SHR memory, count		8*+1/bit	3-5
SBB	Register, register	2	2
SBB	Register, memory	7*	2-4
SBB	Memory, register	7*	2-4
SBB	Register, immediate	3	3-4
SBB	Memory, immediate	7*	3-6
SBB	Accumulator; immediate	3	2-3
SCAS	Dest-string	7	1
SCAS	(repeat) dest-string	5+8(rep)	1
STC/STD/STI		2	1
STOS	Dest-string	3	1
STOS	(repeat) dest-string	4+3(rep)	1
SUB	Register, register	2	2
SUB	Register, memory	7*	2-4
SUB	Memory, register	7*	2-4
SUB	Register, immediate	3	3-4
SUB	Memory, immediate	7*	3-6
SUB	Accumulator; immediate	3	2-3
TEST	Register, register	2	2
TEST	Register, memory	6*	2-4
TEST	Register, immediate	3	3-4
TEST	Memory, immediate	6*	3-6
TEST	Accumulator; immediate	3	2-3
WAIT		3	1
XCHG	Accumulator, reg16	3	1
XCHG	Memory, register	5*	2-4
XCHG	Register, register	3	2
XLAT	Source-table	5	1

Instruction		Clocks	Bytes
XOR	Register, register	2	2
XOR	Register, memory	7*	2-4
XOR	Memory, register	7*	2-4
XOR	Register, immediate	3	3-4
XOR	Memory, immediate	7*	3-6
XOR	Accumulator; immediate	3	2-3

ضمیمه شماره ٤

کد ماشین دستورالعمل‌ها

کد	ثبات ١٦ بیتی	ثبات ٨ بیتی	ثبات سگمنت
000	AX	AL	ES
001	CX	CL	CS
010	DX	DL	SS
011	BX	BL	DS
100	SP	AH	
101	BP	CH	
110	SI	DH	
111	DI	BH	

کدگذاری آدرس موثر

R/m	Mod=00	mod=10 یا Mod=01
000	[BX+SI]	[مقدار جابجایی+BX+SI]
001	[BX+DI]	[مقدار جابجایی+BX+DI]
010	[BP+SI]	[مقدار جابجایی+BP+SI]
011	[BP+DI]	[مقدار جابجایی+BP+DI]
100	[SI]	[مقدار جابجایی +SI]
101	[DI]	[مقدار جابجایی +DI]
110	(حالت مستقیم)	[مقدار جابجایی +BP]
111	[BX]	[مقدار جابجایی +BX]

فیلد "mod" در این حالت‌ها تعیین کننده آن است که چند بایت بعنوان مقدار جابجایی وجود دارد. مقدار 00 به این معناست که بایت جابجایی در کد ماشین وجود ندارد. یعنی حالت آدرس دهی غیر مستقیم با ثبات یا آدرس دهی شاخص دار با ثبات پایه بدون مقدار جابجایی اختیاری است. مقدار 10 در فیلد "mod" به این معنی است که دو بایت جابجایی در کد ماشین وجود دارد، این کلمه به مقداری که از ثبات شاخص و یا ثبات مبنا بدست می‌آید اضافه می‌شود. مقدار 01 در فیلد "mod" به این معنی است که یک بایت جابجایی در کد ماشین وجوددارد، این بایت بعنوان یک عدد علامت دار در نظر گرفته می‌شود و قبل از آنکه با مقداری که از ثبات شاخص و یا ثبات مبنا بدست می‌آید جمع گردد به یک کلمه توسعه پیدا می‌کند.

باید توجه داشت که عملوند [BP] در جدول بالا نیامده است. ترکیب منطقی mod=00 و r/m=110 یک جفت ویژه ای است که حالت مستقیم حافظه را مشخص می‌کند. این بدان معناست که هیچ حالت آدرس دهی غیر مستقیم ثبات با ثبات bp وجود ندارد. اسمبلر [BP] را به [BP+0] تبدیل کرده و از حالت mod=01 استفاده کرده و بایت جابجایی را برابر صفر قرار می‌دهد.

کلمه آدرس در جدول بعدی در پرانتز قرار گرفته است زیرا تمام عملوندهای حافظه به بایت‌های اضافی هدف نیاز ندارند. اگر mod=00 و r/m=110 (آدرس دهی مستقیم) باشد، دو بایت اضافی وجود خواهد داشت.

ضمیمه شماره ۵

جدول کد اسکی

ASCII Character Sets

LSD \ MSD		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	“	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	,	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	i	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

NUL	-Null	DLE	-Data Link Escape
SOH	-Start of Heading	DC	-Device Control
STX	-Start of Text	NAK	-Negative Acknowledge
ETX	-End of Text	SYN	-Synchronous ldle
EOT	-End of Transmission	ETB	-End of Transmission Block
ENQ	-Enquiry	CAN	-Cancel
ACK	-Acknowledge	EM	-End of Medium
BEL	-Bell	SUB	-Substitute
BS	-Backspace	ESC	-Escape
HT	-Horizontal Tabulation	FS	-File Separator
LF	-Line Feed	GS	-Group Separator
VT	-Vertical Tabulation	RS	-Record separator
FF	-Form Feed	US	-Unit Separator
CR	-Carriage Return	SP	-Space (Blank)
SO	-Shift Out	DEL	-Delete
SI	-Shift In		

ضمیمه شماره ٦

کد دستورالعملها

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
00	add mem8, reg8	mod reg r/m, (address)
01	add mem 16, reg16	mod reg r/m, (address)
02	add reg8, mem8	mod reg r/m, (address)
	add reg8, reg8	11 dest_reg source_reg
03	add reg16, mem16	mod reg r/m, (address)
	add reg16, reg16	11 dest_reg source_ reg
04	add al, imm8	immediate byte
05	add ax,imm16	immediate word
06	push es	(none)
07	pop es	(none)
08	or mem8, reg8	mod reg r/m, (address)
09	or mem16, reg16	mod reg r/m, (address)
0a	or reg8, mem8	mod reg r/m, (address)
	or reg8, reg8	11 dest_reg source_ reg
0b	or reg16, mem16	mod reg r/m, (address)
	or reg16, reg16	11 dest_reg source_ reg
0c	or al, imm8	immediate byte
0d	or ax, imm16	immediate word
0e	push cs	(none)
0f		
10	adc mem8, reg8	mod reg r/m, (address)
11	adc mem16, reg16	mod reg r/m, (address)
12	adc reg8, mem8	mod reg r/m, (address)
	adc reg8, reg8	11 dest_reg source_ reg

13	adc reg16, mem16	mod reg r/m, (address)
	adc reg16, reg16	11 dest_reg source_ reg
14	adc al, imm8	immediate byte
15	adc ax, imm16	immediate word
16	push ss	(none)
17	pop ss	(none)
18	sbb mem8, reg8	mod reg r/m, (address)
19	sbb mem16, reg16	mod reg r/m, (address)
1a	sbb reg8, mem8	mod reg r/m, (address)
	sbb reg8, reg8	11 dest_reg source_ reg
1b	sbb reg16, mem16	mod reg r/m, (address)
	sbb reg16, reg16	11 dest_reg source_ reg
1c	sbb al, imm8	immediate byte
1d	sbb ax, imm16	immediate word
1e	push ds	(none)
1f	pop ds	(none)
20	and mem8, reg8	mod reg r/m, (address)
21	and mem16, reg16	mod reg r/m, (address)
22	and reg8, mem8	mod reg r/m, (address)
	and reg8, reg8	11 dest_reg source_ reg
23	and reg16, mem16	mod reg r/m, (address)
	and reg16, reg16	11 dest_reg source_ reg
24	and al, imm8	immediate byte
25	and ax, imm16	immediate word
26	es segment override prefix byte	
27	daa	(none)
28	sub mem8, reg8	mod reg r/m, (address)
29	sub mem16, reg16	mod reg r/m, (address)
2a	sub reg8, mem8	mod reg r/m, (address)
	sub reg8, reg8	11 dest_reg source_ reg

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
2b	sub reg16, mem16	mod reg r/m, (address)
	sub reg16, reg16	11 dest_reg source_ reg
2c	sub al, imm8	immediate byte
2d	sub ax, imm16	immediate word
2e	cs segment override prefix byte	
2f	das	(none)
30	xor mem8, reg8	mod reg r/m, (address)
31	xor mem16, reg16	mod reg r/m, (address)
32	xor reg8, mem8	mod reg r/m, (address)
	xor reg8, reg8	11 dest_reg source_ reg
33	xor reg16, mem16	mod reg r/m, (address)
	xor reg16, reg16	11 dest_reg source_ reg
34	xor al, imm8	immediate byte
35	xor ax,imm16	immediate word
36	ss segment override prefix byte	
37	aaa	(none)
38	cmp mem8, reg8	mod reg r/m, (address)
39	cmp mem16, reg16	mod reg r/m, (address)
3a	cmp reg8, mem8	mod reg r/m, (address)
	cmp reg8, reg8	11 dest_reg source_ reg
3b	cmp reg16, mem16	mod reg r/m, (address)
	cmp reg16, reg16	11 dest_reg source_ reg
3c	cmp al, imm8	immediate byte
3d	cmp ax, imm16	immediate word
3e	ds segment override prefix byte	
3f	aas	(none)
40	inc ax	(none)
41	inc cx	(none)
42	inc dx	(none)

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
43	inc bx	(none)
44	inc sp	(none)
45	inc bp	(none)
46	inc si	(none)
47	inc di	(none)
48	dec ax	(none)
49	dec cx	(none)
4a	dec dx	(none)
4b	dec bx	(none)
4c	dec sp	(none)
4d	dec bp	(none)
4e	dec si	(none)
4f	dec di	(none)
50	push ax	(none)
51	push cx	(none)
52	push dx	(none)
53	push bx	(none)
54	push sp	(none)
55	push bp	(none)
56	push si	(none)
57	push di	(none)
58	pop ax	(none)
59	pop cx	(none)
5a	pop dx	(none)
5b	pop bx	(none)
5c	pop sp	(none)
5d	pop bp	(none)
5e	pop si	(none)
5f	pop di	(none)

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
60-6f		
70	jo	displacement byte
71	jno	displacement byte
72	jb/jnae/jc	displacement byte
73	jnb/jae/jnc	displacement byte
74	je/jz	displacement byte
75	jne/jnz	displacement byte
76	jbe/jna	displacement byte
77	jnbe/ja	displacement byte
78	js	displacement byte
79	jns	displacement byte
7a	jp/jpe	displacement byte
7b	jnp/jpo	displacement byte
7c	jl/jnge	displacement byte
7d	jnl/jng	displacement byte
7e	jle/jng	displacement byte
7f	jnle/jg	displacement byte
80	add mem8, imm8	mod 000 r/m, (address), immediate byte
	add reg8, imm8	11 000 reg, immediate byte
	or mem8, imm8	mod 001 r/m, (address), immediate byte
	or reg8, imm8	11 001 reg, immediate byte
	adc mem8, imm8	mod 010 r/m, (address), immediate byte
	adc reg8, imm8	11 010 reg, immediate byte
	sbb mem8, imm8	mod 011 r/m, (address), immediate byte
	sbb reg8, imm8	11 011 reg, immediate byte
	and mem8, imm8	mod 100 r/m, (address), immediate byte
	and reg8, imm8	11 100 reg, immediate byte
	sub mem8, imm8	mod 101 r/m, (address), immediate byte
	sub reg8, imm8	11 101 reg, immediate byte

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	xor mem8, imm8	mod 110 r/m, (address), immediate byte
	xor reg8, imm8	11 110 reg, immediate byte
	cmp mem8, imm8	mod 111 r/m, (address), immediate byte
	cmp reg8, imm8	11 111 reg, immediate byte
81	and mem16,imm16	mod 000 r/m, (address), immediate word
	and reg16,imm16	11 000 reg, immediate word
	or mem16,imm16	mod 001 r/m, (address), immediate word
	or reg16,imm16	11 001 reg, immediate word
	adc mem16,imm16	mod 010 r/m, (address), immediate word
	adc reg16,imm16	11 010 reg, immediate word
	sbb mem16,imm16	mod 011 r/m, (address), immediate word
	sbb reg16,imm16	11 011 reg, immediate word
	and mem16,imm16	mod 100 r/m, (address), immediate word
	and reg16,imm16	11 100 reg, immediate word
	sub mem16,imm16	mod 101 r/m, (address), immediate word
	sub reg16,imm16	11 101 reg, immediate word
	xor mem16,imm16	mod 110 r/m, (address), immediate word
	xor reg16,imm16	11 110 reg, immediate word
	cmp mem16,imm16	mod 111 r/m, (address), immediate word
	cmp reg16,imm16	11 111 reg, immediate word
82		
83		mod 000 r/m, (address), immediate byte
	and reg16,imm8	11 000 reg, immediate byte
	or mem16, imm8	mod 001 r/m, (address), immediate byte
	or reg16,imm8	11 001 reg, immediate byte
	adc mem16, imm8	mod 010 r/m, (address), immediate byte
	adc reg16,imm8	11 010 reg, immediate byte
	sbb mem16, imm8	mod 011 r/m, (address), immediate byte
	sbb reg16,imm8	11 011 reg, immediate byte

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	and mem16, imm8	mod 100 r/m, (address), immediate byte
	and reg16,imm8	11 100 reg, immediate byte
	sub mem16, imm8	mod 101 r/m, (address), immediate byte
	sub reg16,imm8	11 101 reg, immediate byte
	xor mem16, imm8	mod 110 r/m, (address), immediate byte
	xor reg16,imm8	11 110 reg, immediate byte
	cmp mem16, imm8	mod 111 r/m, (address), immediate byte
	cmp reg16,imm8	11 111 reg, immediate byte
84	test reg8, mem8	mod reg r/m, (address)
	test reg8, reg8	11 dest_reg source_ reg
85	test reg16, mem16	mod reg r/m, (address)
	test reg16, reg16	11 dest_reg source_ reg
86	xchg reg8, mem8	mod reg r/m, (address)
	xchg reg8, reg8	11 dest_reg source_ reg
87	xchg reg16, mem16	mod reg r/m, (address)
	xchg reg16, reg16	11 dest_reg source_ reg
88	mov mem8, reg8	mod reg r/m, (address)
	mov mem8, reg8	11 dest_reg source_ reg
89	mov mem16, reg16	mod reg r/m, (address)
	mov reg16, reg16	11 dest_reg source_ reg
8a	mov reg8, mem8	mod reg r/m, (address)
8b	mov reg16, mem16	11 dest_reg source_ reg (address)
8c	mov mem16, sreg	mod reg r/m, (address)
	mov reg16, sreg	11 reg r/m
8d	lea reg16, mem	mod reg r/m, (address)
8e	mov sreg, mem16	mod reg r/m, (address)
	mov sreg, reg16	11 reg r/m
8f	pop mem16	mod 000 r/m, (address)

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
90	nop	(none)
91	xchg ax,cx	(none)
92	xchg ax,dx	(none)
93	xchg ax,bx	(none)
94	xchg ax,sp	(none)
95	xchg ax,bp	(none)
96	xchg ax,si	(none)
97	xchg ax,di	(none)
98	cbw	(none)
99	cwd	(none)
9a	call	offest and segment number words
	(far direct)	
9b	wait	(none)
9c	pushf	(none)
9d	popf	(none)
9e	sahf	(none)
9f	lahf	(none)
A0	mov al, mem8	offset word (direct addressing)
A1	mov ax, mem16	offset word
A2	mov mem8, al	offset word
A3	mov mem16, ax	offset word
A4	movsh	(none)
A5	movsw	(none)
A6	cmpsb	(none)
A7	cmpsw	(none)
A8	test al, imm8	immediate byte
A9	test ax, imm16	immediate word
AA	stosb	(none)
AB	stosw	(none)

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
AC	lodsb	(none)
AD	lodsw	(none)
AE	scasb	(none)
AF	scasw	(none)
B0	mov al, imm8	immediate byte
B1	mov cl, imm8	immediate byte
B2	mov dl, imm8	immediate byte
B3	mov bl, imm8	immediate byte
B4	mov ah, imm8	immediate byte
B5	mov ch, imm8	immediate byte
B6	mov dh, imm8	immediate byte
B7	mov bh, imm8	immediate byte
B8	mov ax, imm16	immediate word
B9	mov cx, imm16	immediate word
BA	mov dx, imm16	immediate word
BB	mov bx, imm16	immediate word
BC	mov sp, imm16	immediate word
BD	mov bp, imm16	immediate word
BE	mov si, imm16	immediate word
BF	mov di, imm16	immediate word
C0, C1		
C2	ret imm16	immediate word
	(near return)	
C3	ret	(none)
	(near return)	
C4	les reg16, mem	mod reg r/m, (address)
C5	lds reg16, mem	mod reg r/m, (address)
C6	mov mem8, imm8	mod 000 r/m, (address), immediate byte
	mov reg8, imm8	11 000 r/m, immediate byte

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
C7	mov mem16, imm16	mod 000 r/m, (address), immediate word
	mov reg1, imm16	11 000 r/m, immediate word
C8,C9		
CA	ret imm16	immediate word
	(far return)	
CB	ret	(none)
	(far return)	
CC	int 3	(none)
CD	int imm8	immediate byte
CE	into	(none)
CF	iret	(none)
D0	rol mem8,1	mod 000 r/m, (address)
	rol reg8,1	11 000 reg
	ror mem8,1	mod 001 r/m, (address)
	ror reg8,1	11 001 reg
	rcl mem8,1	mod 010 r/m, (address)
	rcl reg8,1	11 010 reg
	rcr mem8,1	mod 011 r/m, (address)
	rcr reg8,1	11 011 reg
	shl/sal mem8,1	mod 100 r/m, (address)
	shl/sal reg8,1	11 100 reg
	shr mem8,1	mod 101 r/m, (address)
	shr reg8,1	11 101 reg
	sar mem8,1	mod 111 r/m, (address)
	sar reg8,1	11 111 reg
D1	rol mem16,1	mod 000 r/m, (address)
	rol reg16,1	11 000 reg
	ror mem16,1	mod 001 r/m, (address)
	ror reg16,1	11 001 reg

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	rcl mem16,1	mod 010 r/m, (address)
	rcl reg16,1	11 010 reg
	rcr mem16,1	mod 011 r/m, (address)
	rcr reg16,1	11 011 reg
	shl/sal mem16,1	mod 100 r/m, (address)
	shl/sal reg16,1	11 100 reg
	shr mem16,1	mod 101 r/m, (address)
	shr reg16,1	11 101 reg
	sar mem16,1	mod 111 r/m, (address)
	sar reg16,1	11 111 reg
		(110 not used with 8088)
D2	rol mem8,cl	mod 000 r/m, (address)
	rol reg8,cl	11 000 reg
	ror mem6,cl	mod 001 r/m, (address)
	ror reg8,cl	11 001 reg
	rcl mem8,cl	mod 010 r/m, (address)
	rcl reg8,cl	11 010 reg
	rcr mem8,cl	mod 011 r/m, (address)
	rcr reg8,cl	11 011 reg
	shl/sal mem8, cl	mod 100 r/m, (address)
	shl/sal reg8, cl	11 100 reg
	shr mem8, cl	mod 101 r/m, (address)
	shr reg8, cl	11 101 reg
	sar mem8, cl	mod 111 r/m, (address)
	sar reg8, cl	11 111 reg
D3	rol mem16, cl	mod 000 r/m, (address)
	rol reg16, cl	11 000 reg

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	ror mem16, cl	mod 001 r/m, (address)
	ror reg16, cl	11 001 reg
	rcl mem16, cl	mod 010 r/m, (address)
	rcl reg16, cl	11 010 reg
	rcr mem16, cl	mod 011 r/m, (address)
	rcr reg16, cl	11 011 reg
	shl/sal mem16, cl	mod 100 r/m, (address)
	shl/sal reg16,cl	11 100 reg
	shr mem16, cl	mod 101 r/m, (address)
	shr reg16, cl	11 101 reg
	sar mem16, cl	mod 111 r/m, (address)
	sar reg16, cl	11 111 reg
D4	aam	0a
D5	aad	0a
D6		
D7	xlat	(none)
D8	esc 0	mod 000 r/m, (address)
D9	esc 1	mod 001 r/m, (address)
DA	esc 2	mod 010 r/m, (address)
DB	esc 3	mod 011 r/m, (address)
DC	esc 4	mod 100 r/m, (address)
DD	esc 5	mod 101 r/m, (address)
DE	esc 6	mod 110 r/m, (address)
DF	esc 7	mod 111 r/m, (address)
E0	loopnz / loopne	displacement byte
E1	loopz/loope	displacement byte
E2	loop	displacement byte
E3	jcxz	displacement byte
E4	in al, port	port byte

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
E5	in al, port	port byte
E6	out al, port	port byte
E7	out ax, port	port byte
E8	call	displacement word
	(near relative)	
E9	jmp	displacement word
	(intra-segment relative)	
EA	jmp	offset and segment number words
	(inter-segment direct)	
AB	jmp	displacement byte
	(inter-segment relative short)	
EC	in al, dx	(none)
ED	in ax, dx	(none)
EE	out al, dx	(none)
EF	out ax, dx	(none)
F0	lock prefix byte	
F1		
F2	repne /repnz prefix byte	
F3	rep/repe/repz prefix byte	
F4	hlt	(none)
F5	cmc	(none)
F6	test mem8, imm8	mod 000 r/m, (address), immediate byte
	test reg8, imm8	11 000 reg, immediate byte
	not mem8	mod 010 r/m, (address)
	not reg8	11 010 reg
	neg mem8	mod 011 r/m, (address)
	neg reg8	11 011 reg
	mul mem8	mod 100 r/m, (address)
	mul reg8	11 100 reg
	imul mem8	mod 101 r/m, (address)
	imul reg8	11 101 reg

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	div mem8	mod 110 r/m, (address)
	div reg8	11 110 reg
	idiv mem8	mod 111 r/m, (address)
	idiv reg8	11 111 reg
F7	test mem16, imm16	mod 000 r/m, (address), immediate word
	test reg16, imm16	11 000 reg, immediate word
	not mem16	mod 010 r/m, (address)
	not reg16	11 010 reg
	neg mem16	mod 011 r/m, (address)
	neg reg16	11 011 reg
	mul mem16	mod 100 r/m, (address)
	mul reg16	11 100 reg
	imul mem16	mod 101 r/m, (address)
	imul reg16	11 101 reg
	div mem16	mod 110 r/m, (address)
	div reg16	11 110 reg
	idiv mem16	mod 111 r/m, (address)
	idiv reg 16	11 111 reg
F8	clc	(none)
F9	stc	(none)
FA	cli	(none)
FB	sti	(none)
FC	cld	(none)
FD	std	(none)
FE	inc mem8	mod 000 r/m, (address)
	inc reg8	11 000 reg
	dec mem8	mod 001 r/m, (address)
	dec reg8	11 001 reg
FF	inc mem16	mod 000 r/m, (address)

کد عمل	دستورالعمل‌ها و عملوندها	ساختار بایتها
	dec mem16	mod 001 r/m, (address)
	call	mod 010 r/m, (address)
	(far indirect)	
	call	mod 011 r/m, (address)
	(near indirect)	
	jmp	mod 100 r/m, (address)
	(intra –segment indirect)	
	jmp	mod 101 r/m, (address)
	(inter-segment indirect)	
	push mem16	mod 110 r/m, (address)

سئوالات چهار گزینه‌ای

۱- اگر AX=0A21 و BX=0010 باشد وضعیت فلگها در اثر اجرای دستورالعمل CMP AX,BX چیست؟

الف. CF=0 OF=0 SF=0 ZF=0

ب. CF=1 OF=0 SF=1 ZF=1

ج. CF=1 OF=1 SF=0 ZF=0

د. CF=1 OF=1 SF=1 ZF=1

۲- کدام یک از دستورالعملهای ذیل روی CF بی‌تأثیر هستند؟

الف. SAR

ب. ROR

ج. CMC

د. MOVS

۳- کدام یک از دستورالعملهای ذیل روی ثبات نشانه‌ها تاثیر می‌گذارد؟

الف. POP

ب. IN

ج. OR

د. PUSHF

٤- کدامیک از دستورات ذیل غیر مجاز است؟

الف. MUL BX

ب. INC NUM

ج. NOP

د. SHR AX,3

٥- کدامیک از دستورات ذیل مجاز نیست؟

الف. CMP 10,BX

ب. MOV TEMP,COUNT

ج. MOV DS,ES

د. CBW

٦- دستورالعملهای DEC و INC روی کدامیک از نشانه‌های ذیل بی‌تأثیر هستند؟

الف. CF

ب. OF

ج. SF

د. ZF

۷- بر اثر جمع کدام یک از زوج اعداد ذیل، سرریزی رخ خواهد داد (اعداد در مبنای ۱٦ هستند)؟

الف. 0A07,0FD3

ب. 0206,FFB0

ج. FFE7,FFF6

د. 483F,745A

۸- در دستورالعمل MOV AX,SEG DATA مقدار SEG DATA چه زمانی معین می‌گردد (DATA نام سگمنت داده است)؟

الف. زمان اسمبل کردن

ب. زمان LINK کردن

ج. زمان اجرای واقعی برنامه

د. هیچکدام

۹- دستور تعریف نوع ساختار ذیل چند بایت را تخصیص می‌دهد؟

```
S1  STRUCT
F1  DW ?
F2  DB  10 DUP(?)
S1  ENDS
```

الف. ۱۲ ب. ۱٤ ج. صفر د. ۱۵

۱۰- آدرس پنج رقمی 1F558 معادل کدامیک از آدرسهای ذیل است؟

الف. 1055:1018

ب. 1E00:5581

ج. 1E55:1008

د. 18A3:5B28

۱۱- اگر CX=01A2,AX=0075 باشد بعد از اجـرای دسـتورالعمل SUB AX,CX محتوای SF و AX کدام یک از موارد ذیل است؟

الف. SF=1 AX=FED3

ب. SF=0 AX=FED3

ج. SF=1 AX=0127

د. SF=0 AX=0127

۱۲- پس از اجرای قطعه کد ذیل محتوای AX چه خواهد بود؟

```
STC
MOV   AX,10
MOV   BX,4
ADC   AX,BX
```

الف. ۱٦

ب. ۱۵

ج. ۱۰

د. ۵

۱۳- کدامیک از دستورات ذیل باقیمانده تقسیم محتـوای DX بـر ۸ را محاسـبه می‌کند؟

الف. AND DX,0007

ب. OR DX,0007

ج. AND DX,0008

د. OR DX,0008

۱٤- روال PROC1 از نوع FAR دو پـارامتر N1 و N2 را از طریـق پـشته و بـه صورت ذیل دریافت می‌کند.

```
PUSH   N1
PUSH  N2
CALL    PROC1
```

در لحظه ورود به روال PROC1 آدرس N2 کدامیک از موارد ذیل است؟

الف. SP+2

ب. SP- 2

ج. SP+4

د. SP- 4

۱۵- قطعه کد ذیل چه عبارتی را محاسبه خواهد کرد؟

```
MOV   AX,X
AND   AX,AX
NEG   AX
ADD   AX,Y
SUB   AX,Z
INC   AX
```

الف. 2X+Y- Z

ب. 2(X+Y+Z)+1

ج. (- 2X)+Y- Z+1

د. 2(X+Y- Z)

۱٦- اگر AX=FFFF باشد بعد از اجرای قطعه کد ذیل محتوای AX چیست؟

```
NEG  AX
ROR  AX,1
MOV  CL,3
SAR  AX,CL
```

الف. F000

ب. 1000

ج. 0000

د. FFFO

۱۷- بعد از اجرای قطعه کد ذیل محتوای AX چیست؟

```
MOV   CX,10
MOV   AX,5
FOR:  DEC   AX
CMP   AX,0
LOOPNE   FOR
```

الف. صفر

ب. ۱

ج. ۱-

د. ۲-

۱۸- کدام دستورالعمل مجاز است؟

الف. IMUL BL

ب. XCHG TEMP, AX

ج. DIV 10,VALUE

د. CLD

۱۹- کدامیک از دستورالعملهای ذیل روی ثبات نشانه‌ها تاثیر می‌گذارند؟

الف. MOV

ب. XCHG

ج. STOS

د. MUL

۲۰- در دستورالعمل MOV AX, SEG DATA حالت عملوندهای AX, SEG DATA به ترتیب کدامیک از موارد ذیل است (DATA نام سگمنت است)؟

الف. بی اواسطه، ثبات

ب. حافظه مستقیم، ثبات

ج. حافظه غیر مستقیم از نوع دارای مبنا، ثبات

د. بلاواسطه، حافظه مستقیم

۲۱- کدام یک از موارد ذیل پیاده سازی طرح ذیل است؟

```
IF (AX<=10 OR BX=100) THEN
AX=AX+1,
```

الف.
```
CMP  AX,10
JLE  T1
.CMP. BX,100
JE   T1
T1: INC  AX
```

ب.
```
CMP  AX,10
JLE  T1
CMP  BX,100
JE   T1
JMP  T2
T1: INC  AX
T2:
```

ج.
```
CMP  AX,10
JNE  T1
CMP  BX,100
JNLE T1
INC  AX
T1:
```

د. هیچکدام

۲۲- کدامیک از موارد ذیل صحیح نیست؟

الف. نتیجه شیفت ریاضی و منطقی به طرف چپ یکسان هستند.

ب. فلگ سرریزی برای شیفت چند بیتی تعریف نشده است.

ج. عمل شیفت به طرف چپ روی فلگ PF بی‌تاثیر است.

د. در عمل شیفت چند بیتی به چپ ، بیتهایی که از طرف چپ بیرون می‌روند به دور ریخته می‌شوند،مگر آخرین بیت که در فلگ CF ذخیره می‌گردد.

۲۳- قطعه کد ذیل چه کاری انجام می‌دهد. (فرض می‌کنیم ثباتهای لازم دیگر تنظیم شده‌اند.)

```
MOV   AL,'* '
MOV   CX,10
MOV   DI,OFFSET   STR
CID
REP   STOSB
```

الف. کاراکتر '*' را در رشته STR جستجو می‌کند.

ب. تا وقتی به کاراکتر '*' نرسیده است پویش را تا ۱۰ کاراکتر اول STR ادامه می‌دهد.

ج. کاراکتر '*' را در ۱۰ کاراکتر اول STR کپی می‌کند.

د. هیچکدام

۲٤- اگر اجرای دستورهای ADD، سه پالس و اجرای MOV، ٤ پالس و LOOP، ۸ پالس زمانی مصرف کنند، در این صورت اجرای قطعه کد ذیل چند پالس زمانی نیاز دارد؟

```
  MOV   CX,5
B:ADD   AX,2
  LOOP   B
```

الف. ۱۵ ب. ۵۹ ج. ۵۵ د. ٤۹

۲۵- کدامیک از موارد ذیل مجاز است؟

الف. MOV TEMP,COUNT

ب. MUL 2

ج. PUSHF AX

د. CBW

۲٦- کدامیک از دستورات ذیل روی فلگها تاثیر می‌گذارد؟

الف. MOV

ب. INC

ج. STOS

د. LOOP

۲۷- پس از اجرای قطعه کد ذیل CX حاوی چه مقداری خواهد بـود (temp و num1 توسط دستور DW در سگمنت DATA تعریف شده‌اند)؟

```
        MOV     CX,0
        MOV     BX,10
        MOV     AX,123
LOOP1:  CWD
        DIV     BX
        MOV     NUM1,DX
        MOV     TEMP,AX
        MOV     AX,CX
        MUL     BX
        ADD     AX,NUM1
        MOV     CX,AX
        MOV     AX,TEMP
        CMP     AX,0
        JNZ     LOOP1
```

الف. ٤٤٤ ب. ٤٥٦ ج. ۱۲۳ د. ۳۲۱

۲۸- کدامیک از دستورات زیر مجاز است؟

الف. CMP BH,'*'

ب. OUT 05A4H,AL

ج. MOV DS,CS

د. MOV IP,AX

۲۹- کدامیک از موارد ذیل پیاده سازی دستور شرطی ذیل است؟

```
IF (NOT (A>=10  OR  BX<=1) ) THEN
      CX=CX+1
```

الف.

```
        CMP     A,10
        JGE     EXIT
        CMP     B,1
        JLE     EXIT
        JMP     M
EXIT:   INC     CX
M:
```

ب.

```
        CMP     A,10
        JGE     EXIT
        CMP     B,1
        JLE     EXIT
EXIT:   INC     CX
```

ج.

```
        CMP     A,10
        JNL     EXIT
        CMP     B,1
        JNG     EXIT
        INC     CX
EXIT:
```

د. هیچکدام

۳۰- پس از اجرای دستورالعملهای ذیل محتوای BX کدامیک از موارد ذیل است؟

```
MOV   CL,4
MOV   AX,FFFF
CLC
RCR   AX,CL
```

الف. 0FFF

ب. FFF0

ج. F000

د. 000F

۳۱- ماکروی ذیل را در نظر بگیرید.

```
MAC1    MACRO
        CMP     AX,0
        JE      TEST
        ADD     AX,AX
TEST:   ADD     AX,AX
        ENDM
```

پس ازاجرای قطعه کد ذیل محتوای AX چه خواهد بود؟

```
MOV  AX,3
MAC1
MAC1
```

الف. ۱۲

ب. ۹

ج. ۱۵

د. اسمبلر پیغام خطا می‌دهد.

۳۲- کدامیک از دستورات ذیل غیر مجاز است؟

الف. CMPSW

ب. IRET

ج. LEA BX,AX

د. LOOPNE FOR1

۳۳- با توجه به تعریف ماکروی ذیل کدامیک از فراخوانی‌های ذیل باعث بـروز خطا خواهد شد.

```
MAC2  MACRO  N1,N2,N3,N4
MOV          AX,N1
IFNB<N2>
ADD          AX,N2
ENDIF
IFNB<N3>
ADD          AX,N3
```

```
ENDIF
IFNB<N4>
ADD         AX,N4
ENDIF
ENDM
```

الف. MAC1 BX, CX,DX,1

ب. MAC1 BX,CX,100

ج. MAC1 BX,DX

د. هیچکدام

۳٤- دستور REPE در کدام یک از شرایط ذیل تکرار را ادامه می‌دهد؟

الف. CX=0 AND ZF=0

ب. CX< >0 OR ZF=0

ج. CX< >0 AND ZF=1

د. CX=0 OR ZF=1

۳۵- اگر در ۲۰ کاراکتر اول رشته STRING حداقل یک کاراکتر S وجود داشته باشد، پس از اجرای قطعه کد ذیل DI حاوی چیست؟ (فرض کنید ثباتهای لازم دیگر به درستی تنظیم شده است)

```
MOV     AL, 'S'
LEA     DI, STRING
  D
REPNE   SCASB
DEC     DI
```

الف. DI حاوی افست اولین وقوع کاراکتر S در رشته STRING است.

ب. DI حاوی تعداد کاراکتر S در رشته STRING است.

ج. DI حاوی افست کاراکتر قبل از اولین وقوع کاراکتر S در رشته STRING است.

د. هیچکدام

۳۶- محتوای AL چه عددی باشد تا بعد از اجرای دستورات ذیل مقدار AL صفر شود؟

```
XOR  AL,0FH
```

الف. 00

ب. FF

ج. 0F

د. F0

۳۷- اگر AX=FF15 و VALUE=0003 باشد بعد از اجرای دستور IMUL VALUE فلگهای CF و OF کدامیک از مقادیر ذیل را دارند؟

الف. OF=1 ، CF=0

ب. OF=0 ، CF=1

ج. OF=1 ، CF=1

د. OF=0 ، CF=0

۳۸- با توجه به دستور ذیل چه عددی در مبنای ۱۶ در BYTE1 ذخیره می‌شود.

```
BYTE1  DB - 128
```

الف. 70

ب. 80

ج. FF

د. 08

۳۹- بعد از اجرای دستورالعملهای ذیل DX حاوی چه عددی خواهد بود (در مبنای ۱۶)؟

```
MOV  BX,1
MOV  AX,- 1
MUL  BX
```

الف. DX تغییری نمی‌کند و مقدار قبلی خود را دارد.

ب. 1111

ج. 0000

د. FFFF

۴۰- آدرس پنج رقمی (در مبنای ۱۶) شروع سگمنت شماره $(0010)_{16}$ کـدامیک از موارد ذیل است؟

الف. 01010

ب. 00100

ج. 0011F

د. 00111

۴۱- مکمل پایه $(342)_5$ کدامیک از موارد ذیل است؟

الف. 113

ب. 213

ج. 102

د. 103

۴۲- کدامیک ازاسامی ذیل مجاز نیست؟

الف. TEMP$

ب. TEMP. NUM

ج. TEMP%

د. TEMP4

۴۳- دستورالعمل ذیل چند بایت تخصیص می‌دهد؟

B DB 2(5 DUP('*'),4 DUP('0'))

الف. 120

ب. 11

ج. 40

د. 18

٤٤- با توجه به قطعه کد ذیل در D چه عددی ذخیره می‌شود؟

```
DATA    SEGMENT
  A     DB    5DUP(9)
  B     DB    6DUP(8)
  C     DB    OFFSET    B
```

الف. 0 ب. 6 ج. 5 د. 11

٤٥- اگر AX حاوی عدد ده باشد پس از اجرای دستورات ذیل محتوای AX کدامیک از موارد ذیل است؟

```
NEG     AX
ADD     AX,AX
NEG     AX
DEC     AX
```

الف. 18 ب. 19 ج. 20 د. 21

٤٦- پس از اجرای دستورات ذیل BX حاوی چه مقداری است؟

```
        MOV             CX,5
        MOV             BX,16
LOOP:   DEC             BX
        CMP             BX,0
        LOOPNE    FOR
```

الف. 12 ب. 15 ج. 0 د. 11

٤٧- کدام یک از دستورات ذیل مجاز است؟

الف. cbw

ب. cwd ax

ج. cmp 10,ax

د. cld bx

۴۸- قطعه کد ذیل پیاده سازی کدامیک از موارد ذیل است؟

```
cmp   temp,10
Jng    e
Cmp   ali,8
Jne    e
Dec    count
E:
```

الف. if (temp > 10and ali=8)
count=count –1

ب. if (temp < 10 or ali=8)
count = count –1

ج. if (temp > 10 or ali < > 8)
count = count –1

د. if (temp < 10 and ali < > 8)
count = count –1

۴۹- فرض کنید سگمنت داده حاوی دو دستور ذیل باشد:

```
source DB  "Summe"
dest   DB   "Summi"
```

اگر آدرس شروع Source,0000 و آدرس شروع dest,0005 باشد مقادیر ثباتهای SI و DI بعد از اجرای قطعه کد ذیل چه خواهد بود؟

```
Lea      Si,source
Lea      Di,dest
Cld
Mov      Cx,6
Repne    Cmpsb
```

الف. DI=0005, SI=0000

ب. DI=0009, SI=0004

ج. DI=000A, SI=0005

د. DI=0006, SI=0001

۵۰- اگر AX=A9D7 بعد از اجرای rol AX,1 مقدار AX کدامیک از موارد ذیل است؟

الف. 54EB

ب. 53AF

ج. A9D7

د. B9D6

۵۱- دستورالعمل ذیل چند باید حافظه را تخصیص می‌دهد؟

```
temp    DB    20Dup(2Dup('*'),3Dup('1'))
```

الف. 25 ب. 120 ج. 100 د. 20

۵۲- کدامیک از دستورات ذیل مجاز است؟

الف. mov ds,12

ب. mov cs,ds

ج. mov IP,12

د. mov ax, OFFSET TE

۵۳- کدامیک از موارد ذیل مجاز است؟

الف. DW 12*10

ب. DW "ABC"

ج. mov [bx],2

د. push 2

٥٤- مکمل 2 عدد 00001010 کدامیک از موارد ذیل است؟

الف. 11110110 ب. 11110101

ج. 00000110 د. 01101111

۵۵- کدامیک از دستورالعملهای ذیل غیرمجاز است؟

الف. mul bx

ب. add 1,ax

ج. inc num

د. inc ax

۵٦- کدامیک از دستورالعملهای ذیل CF را تغییر نمی‌دهد؟

الف. cmc

ب. stc

ج. clc

د. Jc

۵۷- اگر dh=F5 باشد بعد از اجرای دستور neg dh محتوای dh چیست؟

الف. oB

ب. F4

ج. 5F

د. F5

۵۸- اگر SP=0100 باشد بعد از اجرای قطعه کد ذیل محتوای SP چیست؟

```
push   ax
push   bx
push   cx
```

الف. 0106

ب. 0104

ج. 00FA

د. 00FC

۵۹- دستور rep در کدامیک از حالات ذیل تکرار را ادامه می‌دهد؟

الف. CX≠0

ب. ZF≠0

ج. ZF=0

د. CX=0

٦۰- اگر AX=A9D7 و CL=04 باشد بعد از اجرای دستور Sar ax, cl مقـدار AX چیست؟

الف. FA9D

ب. 0F9D

ج. 9D7F

د. 9D70

٦۱- اگر BL=FF و AX=FE01 باشد بعـد از اجـرای دسـتور div bl مقـدار AX کدامیک از موارد ذیل است؟

الف. EF01

ب. 00FF

ج. FE01

د. E0E0

٦۲- اگر AL=6E و ch=0A باشد بعد از اجرای دستور imul ch مقادیر AX و CF و OF کدامیک از موارد ذیل است؟

الف. AX=044c, CF=1, OF=1

ب. AX=044c, CF=0, OF=0

ج. AX=046E, CF=0, OF=0

د. AX=046E, CF=1, OF=1

٦٣- حلقه ذیل چند بار اجرا می‌شود؟

```
        mov     cx,- 1
for:
⋮
        loop    for
```

الف. 65535

ب. 65536

ج. 0

د. 1

٦٤- کدامیک از اعداد ذیل نمی‌تواند به عنوان عملوند DW استفاده شوند؟

الف. – 32768

ب. 132768

ج. 65535

د. 32000

٦٥- کدامیک از دستورات ذیل غیرمجاز است؟

الف. lea ax,array

ب. lea ax,bx

ج. mov ax,[bx]

د. mov ax,SEG data

٦٦- اگر بخواهیم عدد FF را در مبنای 16 در خانه‌ای از حافظه ذخیره کنـیم از کدامیک از عملوندهای ذیل نمی‌توان به عنوان عملوند DB استفاده کرد؟

الف. 255

ب. – 1

ج. 11111111b

د. 88Q

٦٧- کدامیک از موارد ذیل صحیح نیست؟

الف. کد ماکرو در محل فراخوانی ماکرو کپی می‌شود.

ب. استفاده از ماکرو به جای روال معمولاً باعث طولانی شدن کد هدف می‌شود.

ج. استفاده از ماکرو به جای روال معمولاً باعث اجرای سریعتر برنامه می‌شود.

د. عمل جایگزینی دستور فراخوانی ماکرو با کد ماکرو در زمان اجرا انجام می‌شود.

٦٨- دستورالعمل ذیل چند بایت حافظه را تخصیص می‌دهد؟

ABLE, DB 10DUP(3DUP('A'), 9 DUP ('B'))

الف. ۲۲

ب. ۱۲۰

ج. ۲۷۰

د. ۳۹

٦٩- دستورالعمل ذیل چند بایت حافظه را تخصیص می‌دهد؟

DW 100H DUP(?)

الف. 100

ب. 1024

ج. 512

د. 200

۷۰- کدام یک از نامهای ذیل مجاز نیست؟

الف. TEMP. 2

ب. ?TEMP

ج. TEMP?

د. TEMP4

۷۱- با توجه به کد ذیل، آفست شروع دستور NUM DB 12 چیست؟

```
DATA    SEGMENT
TOTAL   DW      23
SUM     DB      8
NUM     DB      12
DATA    ENDS
```

الف. 0004

ب. 0003

ج. 0002

د. 0001

۷۲- کوچکترین و بزرگترین عملوندهای عددی دستور DB کدام یک از مـوارد ذیل است؟

الف. – 128,128

ب. – 127,128

ج. – 128,255

د. 255,- 255

۷۳- مکمل ۲، عدد ۱۱۰، کدام یک از موارد ذیل است؟

الف. 1001

ب. 1010

ج. 1110

د. 0111

۷٤- در مورد استفاده از دستورالعمل ING AX و دستورالعمل ADD AX,1 کدام مورد غلط است؟

الف. سرعت اجرای INC AX از ADD AX,1 بیشتر است.

ب. حافظه مصرفی INC AX از ADD AX کمتر است.

ج. نتیجه (مقدار AX) اجرای این دو دستورالعمل یکسان است.

د. هیچکدام

۷۵- در مورد قطعه کد ذیل کدام یک از موارد ذیل صحیح است؟

```
PUSH    AX
PUSH    - BX
POP     AX
POP     BX
```

الف. این قطعه کد دارای خطا بوده و ترجمه نمی‌شود

ب. این کد هیچ تغییری در مقادیر AX و BX ایجاد نمی‌کند

ج. مقدار AX و BX را تعویض می‌کند.

د. مقدار AX و BX را دو برابر می‌کند.

۷۶- قطعه کد ذیل پیاده سازی کدام یک از موارد ذیل است؟

```
        CMP     SUM,200
        JGE     A
        CMP     A VER,5
        JNE     B
A:      MOV      COUNT,100
B:
```

الف. IF (SUM>=200 OR AVER=5)
COUNT=100

ب. IF (SUM>=200 AND AVER=5)
COUNT=100

ج. IF(SUM>=200 OF AVER< >5)
COUNT=100

د. IF (SUM<=200 AND AVER< > 5)
COUNT=100

۷۷- کدام یک از دستورات ذیل مجاز است؟

الف. PUSHF AX

ب. CMP 100,AX

ج. MOV [BX],0

د. هیچکدام

۷۸- کدام یک از دستورالعملهای ذیل روی فلگها موثرند؟

الف. JCXZ

ب. PUSHF

ج. CMC

د. XCHG

۷۹- پیشوند تکرار REPZ تحت کدام یک از شرایط ذیل تکرار را ادامه می‌دهد؟

الف. zf=1 یا CX=0

ب. zf=0 یا CX=0

ج. zf=1 و CX< > 0

د. zf=0 و CX< > 0

۸۰- دستورالعمل XOR CL,11111111 معادل کدام دستورالعمل ذیل است؟

الف. NOT CL

ب. OR CL,CL

ج. AND CL,00000000

د. TEST CL 11111111

۸۱- کدام یک از موارد ذیل صحیح نیست:

الف. روالهای NEAR در همان سگمنت کدی که فرا خوانی می‌شود، تعریف می‌گردند.

ب. روالهای FAR در سگمنتی جدا از سگمنت کد فرا خواننده روال قرار دارد.

ج. دستور CALL برای فراخوانی روالهای FAR ثباتهای CS و IP و برای روال NEAR فقط IP را در پشته ذخیره می‌کند.

د. وقتی مقداری در پشته قرار می‌گیرد SP افزایش می‌یابد.

٨٢- قطعه برنامه ذیل را در نظر بگیرد.

```
MOV     AX,1
MOV     CX,10
P:

INC     AX
LOOP     P
```

پس از اجرای قطعه برنامه مقدار AX چیست؟

الف. ٩

ب. ١١

ج. ١٠

د. ٦٥٥٣٦

٨٣- اگر AX=ffff باشد بعد از اجرای دستور NEG AX محتوای AX کدام یک از مقادیر ذیل (در مبنای ده) است.

الف. ١

ب. ١-

ج. ٠

د. ٦٥٥٣٦

٨٤- اگر AX=0032 و DX=0000 و CX=000B باشد بعد از اجرای دستور IDIV CX مقادیر DX و AX کدام یک از موارد ذیل است؟

الف. DX=0006 AX=0004

ب. DX=000B AX=0006

ج. DX=0006 AX=000B

د. DX=0004 AX=0006

٨٥- اگر AX=ff50 و CX=0023 و CF=1 باشد آنگاه بعد از اجرای دستورالعمل ADC AX,CX مقادیر AX و CF کدامیک از مقادیر ذیل است؟

الف. AX=FF74 CF=0

ب. AX=FF73 CF=1

ج. AX=FF73 CF=0

د. AX=FF74 CF=1

٨٦- تعریف ذیل چند بایت حافظه را تخصیص می‌دهد؟

```
P     STRUC
C     DW      ?
C2    DW      ?
C3    DB      10DUP(?)
P     ENDS
S1    P       50DUP(< >)
```

الف. ٧٠٠

ب. ١٤

ج. ٦٤

د. ٥٠٠

٨٧- اگر اختلاف عددی کد اسکی حروف کوچک و بزرگ ٣٢ باشد آنگاه برای تبدیل حروف کوچک به حروف بزرگ و بالعکس از کدام دستورالعمل ذیل می‌توان استفاده نمود (فرض کنید حرف مورد نظر در CL باشد).

الف. XOR CL,00100000

ب. OR CL,00100000

ج. NOT CL

د. AND 00100000

٨٨- اگر DX=D0 56 باشد بعد از اجـرای دسـتور ROL DX,1 مقـدار DX چـه خواهد بود؟

الف. A0AC　　ب. 6828

ج. A0AD　　د. ADD0

٨٩- قطعه کد زیر به چه معنی است.

```
        MOV   CX,20
        MOV   BX,6
FOR:    ⋮
DEC     BX
CMP     BX,0
LOOPN1:       FOR
```

الف. یک حلقه با ٢٠ بار اجرا

ب. یک حلقه با ٦ بار اجرا

ج. یک حلقه با ٢٦ بار اجرا

د. یک حلقه با ١٣ بار اجرا

٩٠- دستورات زیر معادل کدام گزینه است.

```
CMP      AX,10
JE       L1
CMP      BX,20
JNE      LJ
ADD      AX
LE ADD   BX
```

الف.
```
IF(AX< >10  AND  BX=20)
                AX=AX+1
ELSE  BX=BX+1
```

ب.
```
IF(AX< >10  AND  BX=20)
                AX=AX+1
ELSE  BX=BX+1
```

ج.
```
IF (AX=10  AND  BX< >20)
                AX=AX+1
ELSE  BX=BX+1
```

د. هیچکدام

۹۱- کدام یک از دستورات زیر مجاز است.

الف. shr ax,4

ب. sar bx,3

ج. imul 3

د. pushf

۹۲- انجام کدام جمع زیر منجر به سرریز (overflow) خواهد شد.

الف. 0A07+01d3

ب. 0206+FFB0

ج. 483F+645A

د. هیچکدام

۹۳- دستورالعمل repne تحت کدام یک از شرایط زیر تکرار را ادامه می‌دهد؟

الف. cx< >0 or zf< >0

ب. cx< >0 or zf=0

ج. cx< >0 and zf=0

د. cx=0 and zf< >0

۹٤- اگر ax=0F و bx=0E باشد بعد از اجرای دستورالعمل زیر فلگها به چه صورت خواهد شد؟

```
cmp   bx,ax
```

الف. of=0 , cf=1, sf=1

ب. of=0 , cf=0, sf=0

ج. of=1 , cf=1, sf=1

د. of=0 , cf=1, sf=0

۹۵- پس از اجرای دستورات زیر مقدار ax (در مبنای ١٦) چیست؟

```
mov ax,   100h
stc
adc       ax,ax
```

الف. 200

ب. 201

ج. 202

د. 199

۹٦- مقدار نهایی ax پس از اجرای کد زیر چیست؟

```
mov     ax,x
mov     bx,y
add     bx,z
add     bx,bx
ncg     ax
add     ax,bx
```

الف. 2(z+y)- x

ب. 2(x+y- x)

ج. 2y+z- x

د. هیچکدام

۹۷- برای اینکه فقط بیت سوم ax را یک کنیم از کدام یک از دستورات می‌توانیم استفاده کنیم؟

الف. or ax,4

ب. and ax,4

ج. xor ax,4

د. not ax,4

۹۸- بعد از اجرای کد زیر مقدار cx چیست؟

```
mov  cx,10
l1:
  ⋮
loop l1
```

الف. 10

ب. 0

ج. 1

د. ۱-

۹۹- کدامیک از مجموعه دستورات زیر صحیح نیست؟

الف. in ax, 0ffh

ب. in ax,07ch

ج. mov cx,0717h
in ax,cx

د. mov dx,07ch
in ax,dx

۱۰۰- پس از اجرای قطعه کد زیر مقدار AX چیست؟

```
MOV    AX,1
MOV    CL,4
SHL    AX,CL
XOR    AX,AX
```

الف. 000F

ب. FFF0

ج. 0000

د. FFFF

۱۰۱- اگر SP=00F0 باشد بعد از اجرای قطعه کد زیر مقدار SP چیست؟

```
PUSH    AX
PUSH    BX
PUSH    CX
```

الف. 00EA

ب. 00F9

ج. 00F6

د. هیچکدام

۱۰۲- کدام مورد غلط است؟

الف. شیفت ریاضی و منطقی به چپ یکسان است.

ب. فلگ سرریزی برای شیف چند بیتی تعریف نشده است.

ج. شیف ریاضی و منطقی به راست یکسان هستند.

د. هیچکدام

۱۰۳- بعد از اجرای قطعه کد زیر مقدار AX چقدر است؟

```
SEGMENT             DATA
DB  A               "ALI",13,10
DB  B               "REZA"
ENDS
CODE                SEGMENT
MOV AX, Offest  B
```

الف. 6

ب. 4

ج. 3

د. 10

۱۰٤- اگر si=0000 و di=0005 باشد مقادیر si,di بعد از اجرای قطعه کد زیر چیست؟

```
Source      Db      "book"
Dest        Db      "book"
Lea         Si,source
Lea         Di,dest
Mov   cx,5
Repne cmpsb
```

الف. SI=0000 , di=0005

ب. SI=0001 , di=0006

ج. SI=0004 , di=0009

د. هیچکدام

۱۰۵- اگر کد دستور add چهار بایت و mov دو بایت باشد اسمبلر برای دستورات زیر چند بایت در نظر می‌گیرد؟

```
M1    Macro   N1,n2
      Mov     Ax,n1
      Add     Ax,n2
      Endm
M1    Cx,bx
M1    Dx,cx
```

الف. 4

ب. 6

ج. 10

د. 20

۱۰٦- کدامیک از دستورالمعلهای ذیل روی فلگ‌ها اثر دارند؟

الف. CALL

ب. MOV

ج. INC

د. LOOP

۱۰۷- کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. SHL DL,5

ب. MOV CS,300

ج. MOV IP,3600

د. کلیه موارد بالا

۱۰۸- دستور تکرار زیر چند بار اجرا می‌گردد؟

```
XOR     AX,AX
MOV     CX,AX
LOOP1:
        -
        -
        -
        -
        LOOP   LOOP1
```

الف. اصلاً اجرا نمی‌شود

ب. یکبار اجرا می‌شود

ج. 65536 دفعه اجرا می‌شود

د. 32767 دفعه اجرا می‌شود

۱۰۹- با توجه به دستورالعملهای ذیل چند بایت از حافظه اشغال می‌شود؟

```
X    DB    'P LEASE YWAIT'
Y    DW    4DUP(?)
Z    DD    35000,42000
```

الف. 20

ب. 27

ج. 7

د. 30

۱۱۰- قطعه برنامه زیر چه کاری را انجام می‌دهد؟

```
MOV    AL, 'a'
AND    AL,0DFH
```

الف. حرف 'A' را به 'a' تبدیل می‌کند
ب. حرف 'a' را به 'A' تبدیل می‌کند
ج. محتوای AL را تغییر نمی‌دهد
د. هیچکدام

۱۱۱- کدام گزینه صحیح می‌باشد؟

الف. CLD مقدار DF را صفر می‌کند
ب. دستورالعمل LAHF بایت دارای ارزش کمتر رجیستر فلگ را در AH قرار می‌دهد
ج. دستور X و bx و LEA آدرس X را در رجیستر BX قرار می‌دهد
د. کلیه موارد بالا

۱۱۲- در مورد دستورالعمل CBW کدام گزینه صحیح می‌باشد؟

الف. روی هیچ فلگی اثر ندارد
ب. برای تبدیل محتوی یک بایت که در رجیستر AL قرار دارد بیک Word استفاده می‌شود
ج. الف و ب
د. در محتوی رجیستر AL را دو برابر می‌نماید

۱۱۳- در مورد دستورالعمل CMPS کدام گزینه صحیح می‌باشد؟

الف. برنامه مقایسه دو مقدار استفاده می‌گردد

ب. برای مقایسه دو رشته استفاده می‌گردد

ج. استفاده از چنین دستوری مجاز نمی‌باشد

د. هیچکدام

۱۱٤- برای مکمل نمودن بیت‌های شماره 7,5,2 رجیستر AL از چه دستورالعلمی استفاده می‌شود؟

الف. XOR AL, 0A4H

ب. AND AL,0A4H

ج. NOT AL

د. NEG AL

۱۱۵- برای جابه‌جا نمودن دو مقدار حافظـه X و Y از نـوع WORD از کـدام دستورالعمل می‌توان استفاده نمود؟

الف. MOV X,Y

ب. XCHG X,Y

ج. LEA X,Y

د. هیچکدام

۱۱٦- در مورد آدرس شروع segmentها کدام گزینه صحیح می‌باشد؟

الف. از هر آدرس دلخواهی در حافظه می‌توانند شروع شوند

ب. بایستی قابل تقسیم بر 16 باشد

ج. بایستی قابل تقسیم بر 8 باشد

د. بایستی فرد باشد

۱۱۷- پس از اجرای عمل زیر کدام گزینه صحیح می‌باشد؟

A2B4+

88F3

	ZF	SF	CF	OF
الف.	0	0	1	1
ب.	0	1	1	0
ج.	0	0	1	1
د.	1	1	1	1

۱۱۸- دستورالعمل LEA BX ,Z معادل کدام دستورالعمل می‌باشد؟

الف. MOV BX,Z

ب. MOV BX OFFSET Z

ج. MOV BX OFSET,Z

د. هیچکدام

۱۱۹- کدام گزینه صحیح است؟

الف. دستورالعمل CMP مانند دستورالعمل SUB عمل نموده ولی نتیجه در جائی ذخیره نمی‌شود

ب. دستورالعمل CMP دقیقاً مانند دستورالعمل SUB عمل می‌نماید

ج. دستورالعمل CMP روی فلگ‌های AF و CF اثر ندارند

د. استفاده از دستور CMP AX,AX مجاز نیست

۱۲۰- اگر AX=300, BL=- 5, AL=4 کدامیک از دستورالعملهای زیر مجاز نیست؟

الف. MUL BL

ب. MUL AL,BL

ج. الف و ب

د. IDIV BL

۱۲۱- کدامیک از دستورالعملهای زیر باعث تغییر مقدار فلگCF می‌شود؟

الف. CLC

ب. CMC

ج. STC

د. کلیه موارد

۱۲۲- استفاده از کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. PUSH CS

ب. MOV IP,100

ج. POP AL

د. کلیه موارد

۱۲۳- کدامیک از دستورالعملهای ذیل روی فلگ‌ها بی‌اثرند؟

الف. PUSH,POP

ب. PUSHF

ج. الف و ب

د. POPF

۱۲٤- بعد از دستورالعمل CMP AL , -20 از کدامیک از دستورالعملهای زیر نمی‌توان استفاده نمود؟

الف. JL LAB1

ب. JNGE LAB1

ج. JB LAB1

د. JLE LAB1

۱۲۵- در دستورالعمل JBE LAB1 کنترل به LAB1 منتقل می‌شود اگر

الف. CF=0 باشد

ب. ZF=0 باشد

ج. CF=1 و ZF=0 باشد

د. CF=0 و ZF=0 باشد

۱۲٦- کدام گزینه غلط می‌باشد؟

الف. در ماکروها دستورالعملهائی که با ;; شروع می‌شوند در برنامه لیست نمی‌شود.

ب. در فراخوانی ماکروها می‌توان بعضی از پارامترها را ذکر نکرد

ج. دستورالعمل STOS مقدار بعضی از فلگ‌ها را تغییر می‌دهد

د. دستورالعمل LODS روی هیچ فلگی اثر ندارد

۱۲۷- اگر CL=3 و DL=8DH و CF=1 باشد. پس از اجرای دستورالعمل CL و SAR DL مقدار ثبات DL برابر است با؟

الف. 11110001B

ب. 00010001B

ج. 10001000B

د. هیچکدام

۱۲۸- دستور تکرار زیر چند بار اجرا می‌گردد؟

```
MOV       CX,10
LOOP1:
          -
          -
          -
          -
          MOV   CX,5
          -
          -
          -
          LOOP LOOP1
```

الف. 10 بار

ب. بی‌نهایت بار

ج. 5 بار

د. 50 بار

۱۲۹- دستورالعملهای ذیل چه کاری را انجام می‌دهند؟

```
MOV     AH, 01H
INT     21H
```

الف. منتظر می‌ماند که کلیدی از صفحه کلید فشار داده شود

ب. کارکتری از صفحه کلید داده می‌شود در AL قرار می‌دهد

ج. الف و ب

د. کارکتری از صفحه کلید داده می‌شود روی صفحه مانیتور نمایش می‌دهد

۱۳۰- کدام گزینه غلط است؟

الف. انتقال اطلاعات از حافظه به ثبات امکان پذیر است.

ب. انتقال اطلاعات از ثبات به حافظه امکان پذیر است.

ج. انتقال اطلاعات از ثبات به رجیستر امکان پذیر می‌باشد

د. انتقال اطلاعات از حافظه به حافظه مستقیماً امکان پذیر است

۱۳۱- دستورالعمل تکرار زیر را در نظر بگیرید:

```
Loop1:   MOV    CX,10
          ⋮
         Loop   Loop1
```

الف. بی نهایت بار اجرا می‌شود

ب. 5 بار اجرا می‌شود

ج. اصلاً اجرا نمی‌شود

د. ۰ بار اجرا می‌شود

۱۳۲- کدامیک از دستورالعملهای ذیل مجاز است؟

الف. MOV AX,BL

ب. MOV CS,100

ج. MOV DS,CS

د. MOV AX,[BX]

۱۳۳- دستورالعملهای ذیل را در نظر بگیرید:

```
CMP    AL,- 5
JA     LAB1
```

الف. استفاده از این دو دستورالعمل بترتیب داده شده صحیح است.

ب. JA بایستی به JG تغییر کند.

ج. مقادیر منفی را نمی‌توان در دستور CMP استفاده نمود.

د. JA بایستی به JL تغییر نماید.

۱۳٤- دستورالعمل تکرار زیر چند مرتبه اجرا می‌شود؟

```
MOV     CX, 0
Loop1:   ⋮
         Loop    Loop1
```

الف. صفر مرتبه

ب. یکبار

ج. ٦٥٥٣٦ دفعه

د. دو مرتبه

۱۳۵- کدام گزینه غلط است؟

الف. دستورالعمل CBW روی هیچ فلگی اثر ندارد.

ب. دستورالعمل CWD روی هیچ فلگی اثر ندارد.

ج. در دستورالعمل CBW عملوند بایستی در ثبات AL قرار گیرد.

د. دستورالعمل CBW باعث تبدیل یک WORD به DOUBLE WORD می‌شود.

۱۳٦- کدام گزینه غلط می‌باشد؟

الف. برای مکمل نمودن تعدادی بیت در یک ثبات از دستورالعمل OR استفاده می‌گردد.

ب. برای صفر نمودن تعدادی بیت در یک ثبات از دستورالعمل AND استفاده می‌شود.

ج. دستورالعمل TEST نتیجه را در جائی ذخیره نمی‌کند.

د. دستورالعمل NOT روی هیچ فلگی اثر ندارد.

۱۳۷- دستورالعمل LEA BX,TAB+5 معادل

الف. MOV BX,OFFSET TAB+5

ب. MOV BX,TAB+5

ج. MOV TAB+5,BX

د. هیچکدام

۱۳۸- اگر محتوی AX برابر با 1010100001111001B۰ باشد و محتوی BX برابـر با 1000101011010108۰ باشد پس از اجرای دستور ADD AX,BX کدام گزینـه صحیح است؟

الف. OF=1, SF=1, AF=1, PF=1

ب. OF=1, SF=1, AF=0, PF=0

ج. OF=1, SF=0, AF=1, PF=1

د. OF=1, SF=0, AF=1, PF=1

۱۳۹- کدام گزینه صحیح است؟

الف. دستور CALL روی هیچ فلگی اثر ندارد.

ب. دستور LODS روی هیچ فلگی اثر ندارد.

ج. SUB AX,AX محتوی CF را صفر می‌کند

د. هر سه مورد

۱٤۰- برای مکمل نمودن بیت‌های شماره فرد ثبات AL از چه دستوری استفاده می‌گردد؟

الف. XOR AL,AAH

ب. XOR AL,0AAH

ج. OR AL, 0AAH

د. AND AL, 0AAH

۱٤۱- کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. PUSHF AX

ب. POP AL

ج. PUSH AL

د. کلیه موارد

١٤٢- دستورالعمل JCXZ LAB را در نظر بگیرید.

الف. کنترل به LAB منتقل می‌شود اگر CX=0 باشد.

ب. کنترل به LAB منتقل می‌شود اگر CF=0 باشد.

ج. کنترل به LAB منتقل می‌شود اگر DF=0 باشد.

د. کنترل به LAB منتقل می‌شود اگر CX معادل صفر نباشد.

١٤٣- کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. NEG AL

ب. NOT AL

ج. PUSH AL

د. هیچکدام

١٤٤- در دستور JA LAB

الف. کنترل به LAB منتقل می‌گردد اگر CF=0 باشد.

ب. کنترل به LAB منتقل می‌گردد اگر CF=1 باشد.

ج. کنترل به LAB منتقل می‌گردد اگر CF=0, ZF=0 باشد.

د. کنترل به LAB منتقل می‌گردد اگر ZF=1 باشد.

١٤٥- کدام گزینه غلط می‌باشد؟

الف. دستورهای پرش روی هیچ فلگی اثر ندارد

ب. دستور TEST روی هیچ فلگی اثر ندارد.

ج. دستور MOV روی هیچ فلگی اثر ندارد.

د. دستور LEA روی هیچ فلگی اثر ندارد.

۱٤٦- در مورد جمع دو مقدار از نوع WORD کدام گزینه غلط می‌باشد؟

الف. اگر مقدار MSB نتیجه برابر با یک شود SF برابر یک می‌شود.

ب. اگر تعداد بیت‌های یک در 8 بیت اول زوج باشد ZF برابر صفر می‌شود.

ج. اگر نتیجه جمع صفر شود ZF برابر با صفر می‌شود.

د. ب و ج

۱٤۷- کدام گزینه غلط می‌باشد؟

الف. دستور XCHG روی هیچ فلگی اثر ندارد

ب. استفاده از دستور X,Y XCHG باعث مبادلهٔ متحوی X,Y می‌گـردد. (X,Y از نوع WORD می‌باشد.)

ج. دستور AL NOT مکمل 1 مقدار AL را می‌دهد.

د. دستور AL NEG مکمل 2 مقدار AL را می‌دهد.

۱٤۸- در مورد دستورالعمل INC کدام گزینه غلط می‌باشد؟

الف. روی CF اثر دارد.

ب. روی CF اثر ندارد

ج. روی SF اثر دارد

د. روی ZF اثر دارد

۱٤۹- کدام دستورالعمل مجاز نیست؟

الف. SUB 100,AL

ب. IMUL 20

ج. DIV 100

د. کلیه موارد

۱۵۰- در دستورالعمل MUL BL

الف. اگر OF و CF برابر با یک شوند نتیجه حاصلضرب در یک بایت جا نمی‌شود.

ب. دستورالعمل MUL فقط روی OF و CF اثر دارند.

ج. در دستورالعمل MUL عملوند نبایستی مقدار ثابتی باشد.

د. کلیه موارد بالا

۱۵۱- کدامیک از دستورالعلمهای ذیل مجاز نیست؟

الف. SHL OPR, CL

ب. ROL OPR,2

ج. SAL OPR,CL

د. STD

۱۵۲- کدام گزینه صحیح است؟

الف. SCAS برای پویش یک رشته جهت وجود یا عدم وجود یک عنصر رشته‌ای معین بکار می‌رود.

ب. دستورالعمل MOVS بر روی هیچ فلگی اثر نمی‌گذارد.

ج. دستورالعمل CMPS برای مقایسه محتوی دو رشته بکار می‌رود.

د. کلیه موارد بالا

۱۵۳- کدام گزینه صحیح نیست؟

الف. پارامترها در دستور MACRO، نمادهای معمولی هستند که بوسیله علامت کاما از یکدیگر جدا می‌شوند.

ب. تعریف یک MACRO شبیه تعریف یک روال در یک زبان سطح بالا می‌باشد.

ج. یک MACRO می‌تواند در هر جای برنامه اسمبلی تعریف شود.

د. توضیحاتی که با ;; در یک MACRO شروع می‌شود هرگز لیست نمی‌شود.

۱٥٤- دستورالعملهای ذیل را در نظر بگیرید. با این دستورالعملها چند بایت حافظه تخصیص می‌یابد؟

```
X    DB    10 DUP(?)
Y    DB    'WAIT', 13, 20
Z    DW    4DUP(?)
```

الف. 23

ب. 20

ج. 24

د. 22

۱٥٥- چرا از زبان اسمبلی استفاده می‌کنیم؟ (دو پردازنده‌های معمولی مثل 68060, Pentium, 8086 و ...)

الف. بدلیل سرعت بیشتر نسبت به زبان سطح بالا

ب. بعلت قابلیت زیاد در کنترل مستقیم روی پردازنده

ج. بدلیل حجم کوچکتر برنامه نسبت به زبان سطح بالا

د. هر سه مورد

۱٥٦- ثباتهای همه منظوره کدامند؟

الف. SI, DI, SP, IP, BP

ب. CS, SS, DS, ES

ج. DX, CX, BX, AX

د. CS, IP, Flags

۱۵۷- از ثابت Flags چه استفاده‌ای می‌شود؟ (پرچم‌ها=Flags)

الف. برای انتقال اطلاعات از حافظه به داخل CPU استفاده می‌شود.

ب. برای انتقال اطلاعات از CPU به حافظه استفاده می‌شود.

ج. برای تصمیم‌گیریها از آن استفاده نمی‌شود.

د. هیچکدام

۱۵۸- پشته یا Stack چیست؟

الف. قسمتی از حافظه جانبی که به منظور ذخیره سازی اطلاعات از آن استفاده می‌شود.

ب. قسمتی از حافظه اصلی که به منظور ذخیره سازی بازیابی اطلاعات خصوصاً در هنگام Int, Call استفاده می‌شود.

ج. قسمتی از دیسک سخت که در مقابل دستیابی cpu های دیگر محافظت شده است.

د. قسمتی از حافظه کمکی که در مقابل "خواند شدن" محافظت شده است.

۱۵۹- از پشته یا Stack اطلاعات به چه صورت ذخیره می‌شوند؟

الف. LIFO (last in first out و آخرین ورود اولین خروج)

ب. FIFO (First in first out و اولین ورود اولین خروج)

ج. هم LIFO و هم FIFO

د. یا LIFO و یا FIFO

۱۶۰- چگونه از دو ثبات 16 بیتی، می‌توان یک آدرس 20 بیتی ساخت؟

الف. امکان ندارد.

ب. یکی از ثباتها را در عدد 16 ضرب کنیم و آن یکی دیگر را با آن جمع کنیم.

ج. یکی از ثباتها را چهار بیت به سمت چپ Shift دهیم و ثبات دیگر را با آن جمع کنیم.

د. موارد ب و ج

۱٦۱- ثبات ES برای یک "ثبات قطعه داده اضافی" مفید واقع شده است. اگر چنین است پس چرا بیش از یک ثبات اضافی (مثلاً GS, FS و ...) نداشته باشیم؟

الف. بیش از یکی مورد نیاز مرکز واقع نخواهد شد.

ب. تکنولوژی زمان ساخت 8086 محدودیت داشته است ... اینکه از سیستم 80386 آنرا می‌بینیم (یعنی ثباتهای اضافی علاوه بر FS)

ج. اصولاً ثبات ES بعنوان یک ثبات داده اضافی استفاده نمی‌شود.

د. ممکن است این کار عملی نباشد، اما تاکنون مشاهده شده است (یعنی استفاده از ثباتهای قطعه علاوه بر FS)

۱٦۲- تفاوت فایل COM, EXE چیست؟

الف. فایل EXE محدودیت اندازه ندارد ولی فایل COM حداکثر به اندازه یک قطعه می‌باشد.

ب. فایل COM از آدرس 100H شروع می‌شود ولی فایل EXEچنین نیست.

ج. اصولاً این دو در واقع یکی هستند و هر دو فایلهای قابل اجرا نیستند و تفاوتی ندارند.

د. موارد الف و ب

۱٦۳- تفاوت Call و INT چیست؟

الف. بوسیله Call می‌توان سیستم را وادار ساخت تا IP, CS را در پشته قرار دهد ولی بوسیله INT می‌توان این کار را نمود علاوه بر اینکه Floy را نیز ذخیره ساخت.

ب. Call برای صدا زدن یک رویه بکار می‌رود و INT وجود ندارد.

ج. تفاوتی ندارند و ارقام یکسان برای یک دستور می‌باشند.

د. در Call فقط IP ذخیره می‌شود (در رشته) ولی در INT هم CS و هم IP ذخیره می‌شود.

١٦٤- مجموعه سه دستورالعمل زیر چه عملی را انجام می دهد؟

```
XOR    AX,BX
XOR    BX,AX
XOR    AX,BX
```

الف. محتوای ثبت AX را با BX عوض می‌کند.

ب. محتوای ثبات AX را به (AX ⊕ BX) تبدیل می‌کند.

ج. محتوای ثبات BX را به (BX ⊕ AX) تبدیل می‌کند.

د. هیچکدام.

١٦٥- در 8086 CPU دو دستور وجود دارد که ظاهراً عیناً یک کار می‌کنند. چرا این دو دستورالعمل در کنار یکدیگر وجود دارند؟ (در صورتیکه وجود یکی از آنها کافی به نظر می‌رسد):

INC DX, ADD AX,1

الف. بدلیل اینکه شاید برنامه‌نویس یکی از این دو فرم را نتواند بخاطر بسپارد .

ب. دستور ADD دو بابتی است در صورتیکه دستور INC DX یک بایتی است.

ج. دستور ADD یک بایتی است، در صورتیکه دستور INC DX دو بایتی است.

د. این دو دستور دقیقاً معادل می‌باشند و یک معادل باینری کاملاً یکسان دارند

166- زیر چند بار اجرا می‌شود: loopدستور

```
        MOV     CX,0
Loble:  Loop    Loble
```

الف. بی‌نهایت

ب. اصلاً اجرا نمی‌شود

ج. FFFF بار بعلاوه 1

د. FFFF بار

١٦٧- هدف از FAT چه می‌باشد؟

الف. اختصاص فضای دیسک برای فایلها

ب. اختصاص فضای دیسک برای I/O

ج. مقیم ساختن برنامه‌ها

د. ایجاد توابع DOS

۱٦۸- هدف از ایجاد ماکرو چیست؟

الف. سادگی و کم کردن تعداد دستورالعمل

ب. ایجاد قابلیت خوانائی زیادتر

ج. موارد الف و ب

د. سرعت بخشیدن به اجرا برنامه

واژه نامه

Abacus	چرتکه
Abort	متوقف کردن–ناقص تمام شدن
Abstract	مجرد – انتزاع
Access	دسترسی
Access time	زمان دسترسی
Accounting	حسابداری
Accumulator	آکومولاتور– انباره– مخزن
Action	عمل
Action cycle	دوره یا سیکل عمل
Action rate	میزان عمل
Active	فعال
Actual	واقعی
Actual address	آدرس واقعی
Actual decimal point	نقطه اعشار واقعی
Add	جمع کردن
Addendum	ضمیمه
Adder	جمع کننده

Addition	جمع
Additional	اضافی
Address	آدرس – نشانی
Addressee	گیرنده – مخاطب
Addressing system	سیستم آدرس دهی
Adjacent	مجاور– نزدیک
Adjective	صفت
Alarm	سیگنال – آژیر
Alarm display	نمایش سیگنال
Algebra	جبر
Algorithm	الگوریتم
Alphabet	حروف الفباء
Ambiguity	ابهام
Analog	قیاسی
Analysis	آنالیز–تحلیل
And gate	مدار 'و'
And operator	عملگر 'و'
Application	کاربرد
Application program	برنامه کاربردی

Applicant	متقاضی
Approach	نزدیکی-تمایل-دسترسی
Arbitrary	اختیاری-دلخواه
Arithmetic	(علم) حساب
Arithmetical	حسابی
Arithmetic register	ثبات محاسباتی
Arithmetic section	قسمت محاسباتی
Arithmetic unit	واحد محاسباتی
Arm	بازو
Array	آرایه
Artificial intelligence	هوش مصنوعی
Artificial language	زبان مصنوعی
Ascending	صعودی
Ascending sort	مرتب نمودن صعودی
ASCII	کد آمریکائی برای مبادله اطلاعات
Assemble	مونتاژ-مونتاژ کردن
Audio	شنوائی
Audit	رسیدگی-ممیزی
Automatic	خودکار-اتوماتیک

Auxiliary	کمکی
Auxiliary operations	عملیات کمکی
Available	موجود
Background	زمینه
Back up	پشتیبانی
Back up system	سیستم پشتیبانی
Base data	داده مبنا
Base number	عدد مبنا
Base register	ثبات مبنا یا شاخص
Begin	شروع کردن – شروع
Bi Conditional	دو شرطی
Bidirectional	دو جهتی
Binary	دودویی
Binary code	کددودوئی
Binary digits	ارقام دودوئی
Binary half adder	نیم جمع کننده دودوئی
Binary logic	منطق دودوئی
Binary numbers	اعداد دودوئی
Binary notation	نمایش دودوئی

Binary operation	عملیات دودوئی
Binary variable	متغیر دودوئی
Bit	رقم 0 یا 1
Bit pattern	الگوی بیتی
Blank	خالی- فاصله
Block	بلوک
Block entry	ورودی بلوک
Boolean	بول
Boolean algebra	جبربول
Branch	شاخه
Branching	شاخه شاخه کردن- منشعب کردن
Bubble sort	مرتب کردن حبابی
Buffer	بافر-قسمتی از حافظه اصلی
Button	دکمه
Cable	کابل
Cache memory	حافظه نهان
Calculate	محاسبه کردن
Calculator	ماشین حساب
Capacity	ظرفیت

Cell	سلول
Center	مرکز
Central	مرکزی
Central processor	پردازشگر مرکزی
Chain	زنجیر
Character recognition	شناخت کارکتر
Check bit	بیت کنترل
Check digit	رقم کنترلی
Chip	تراشه
Circuit	مدار
Clear	پاک کردن
Column	ستون
Combination	ترکیب
Command	فرمان
Comment	تفسیر–ملاحظات – نظریه
Communications	ارتباطات
Compare	مقایسه کردن
Comparing unit	واحد مقایسه کننده
Comparison	مقایسه

Compatible	سازگار کردن
Compatiblity	سازگاری
Compile	ترجمه کردن
Compiler	مترجم
Compiling phase	فاز ترجمه
Complement	مکمل
Component	مؤلفه
Compound condition	شرط ترکیبی
Compress	فشرده کردن
Compute	محاسبه کردن
Computer	محاسبه کننده
Computer network	شبکه کامپیوتری
Condensed	متراکم – خلاصه
Connect time	زمان اتصال
Content	محتوی
Context	زمینه
Continuous	پیوسته
Control cycle	حلقه کنترل
Conversion	تبدیل

Counter	شمارنده
Critical	بحرانی
Critical path	مسیر بحرانی
CRT	صفحه نمایش
Cylinder	سیلندر
Data	داده
Data base	پایگاه داده
Data description	توصیف داده
Date error	خطای داده
Data location	محل داده
Data management	مدیریت داده
Data manipulation	دستکاری داده
Data preparation	تدارک داده
Data processing	پردازش داده
Data rules	قواعد داده
Data transfer	انتقال داده
Deciding	تصمیم گیری
Decimal	اعشاری
Decimal digit	رقم اعشاری

Decimal number	عدد اعشاری
Decimal notation	نمایش اعشاری
Decimal numbering system	سیستم اعداد اعشاری
Decimal point	نقطه اعشار
Decision	تصمیم گیری
Decision box	جعبه تصمیم‌گیری
Decision logic	منطق تصمیم‌گیری
Decision – making system	سیستم تصمیم‌گیری
Decision mechanism	مکانیزم تصمیم‌گیری
Decision rules	قواعد تصمیم‌گیری
Define	تعریف کردن
Definition	تعریف
Delay	تأخیر
Delay time	زمان تأخیر
Delete	حذف
Density	چگالی – دانسیته
Dependent variables	متغیرهای وابسته
Descending	نزولی
Descending sort	مرتب سازی نزولی

Design	طرح
Device	وسیله – دستگاه
Diagnosis	تشخیص
Diagram	نمودار
Difference	اختلاف
Digital clock	ساعت دیجیتال
Disk	دیسک
Disk operating system DOS	سیستم عامل دیسک
Display	نمایش
Display unit	واحد نمایش
Distance	فاصله
Distribute	توزیع کردن
Division	تقسیم
Document	مدرک– سند
Documentation	مستندات
Double length	طول مضاعف
Double precision	دقت مضاعف
Dynamic	پویا
Dynamic memory	حافظه پویا

Effective	موثر
Effective speed	سرعت مؤثر
Effective address	آدرس مؤثر
Efficiency	کارائی
End of file	انتهای فایل
Epitome	رئوس مطالب-خلاصه
Equality	کیفیت
Equipment	تجهیزات
Erase	از بین بردن-پاک کردن
Eraser	پاک کننده-پاک کن
Error	خطا
Error code	رمز خطا
Error detection	تشخیص خطا
Error message	پیغام خطا
Error rate	نرخ خطا
Exchange	مبادله
Execute	اجرا کردن
Execution	اجرا
Execution cycle	سیکل اجرا

Exit	خروج
Exit point	نقطه خروج
Expression	عبارت
Extension	بسط-توسعه-تمدید
External	خارجی
External interrupt	وقفه خارجی
External labels	برچسب‌های خارجی
Facility	تسهیلات
Factor	فاکتور -عامل - ضریب
Failure	شکست - خرابی
Fixed	ثابت
Fixed length	طول ثابت
Flag	فلگ-پرچم
Floating point	نقطه اعشار شناور
Floating point numbers	اعداد با نقطه اعشار شناور
Flowchart	فلوچارت - نمودار
Folder	پوشه
Format	قالب-شکل
Gate	دروازه مدار

General	عمومی
Generator	ایجاد کننده
Global	سراسری
Half	نصف–نیم
Halt	متوقف کردن
Hardware	سخت افزار
Heading	تیتر–عنوان
Help	کمک
High speed	سرعت بالا
High order	مرتبه بالا
Identifier	شناسه
Idle time	زمان بیکاری
Ignore	اغماض کردن
Image	تصویر
Implicit	ضمنی
Inactive	غیرفعال
Inclusive	دربرگیرنده
Increment	افزایش
Independent	مستقل

Index	اندیس–شاخص
Indexed address	آدرس شاخص دار
Indicate	نشان دادن – تعیین کردن
Indirect	غیر مستقیم
Indirect address	آدرس غیر مستقیم
Indirect reference address	آدرس ارجاع غیر مستقیم
Information	اطلاعات
Information retrieval	بازیابی اطلاعات
Input	ورودی
Input data	داده ورودی
Instruction	دستورالعمل
Instruction address	آدرس دستورالعمل
Instruction address register	ثبات آدرس دستورالعمل
Integer	صحیح
Integrated system	سیستم مجتمع
Interface	واسطه–میانجی
Internal	داخلی
Internal code	کد یا رمز داخلی
Internal sort	مرتب سازی داخلی

Interrupt	وقفه
Interrupt system	سیستم وقفه
Iteration	تکرار
Iterative	تکراری
Key	کلید
Keyboard	صفحه کلید
Label	برچسب
Language	زبان
Language compiler	مترجم زبان
Level	سطح
Library	کتابخانه–مرکز اسناد
Light	نور
Limit	محدود کردن
Limited	محدود
Link	متصل کردن–اتصال
List	فهرست
Load	بارکردن
Loader	برنامه سیستم عامل برای بار کردن برنامه در حافظه
Location	محل – جا
Logical	منطقی
Logical operation	عملیات منطقی
Logical operator	عملگر منطقی
Logical shift	شیفت منطقی
Loop	حلقه–حلقه تکرار

Low	پائین
Low order	مرتبه پائین
Low order digit	رقم مرتبه پائین
LSD	بیت باکمترین ارزش
Magnetic	مغناطیسی
Magnetic disk	دیسک مغناطیسی
Main	اصلی
Maintenance	تعمیر و نگهداری
Major	اصلی
Major key	کلید اصلی
Malfunction	خرابی–از کار افتادگی
Match	تطبیق–جور
Matching error	خطای تطبیق
Mathematical	ریاضی
Matrix	ماتریس
Mechanism	مکانیزم–روش
Medium	وسیله برای ضبط داده
Memory	حافظه
Memory address register	ثبات آدرس حافظه
Memory cycle	سیکل یا دوره حافظه
Memory dump	تخلیه حافظه
Memory exchange	مبادله حافظه
Memory hierarchy	سلسله مراتب حافظه
Memory location	محل حافظه

Memory protection	حفاظت حافظه
Memory register	ثبات حافظه
Merge	ادغام
Message	پیغام
Message exchange	مبادله پیغام
Method	روش
Microsecond	میلیونیوم ثانیه–میکروثانیه
Minimum	کوچکترین
Mistake	اشتباه
Mode	روش–طرز–طریقه
Module	ماژول–برنامه
Monitor	صفحه نمایش
Multiply	ضرب کردن
Multiplication	ضرب
Multiprocessor	کامپیوتر دارای چند ریزپردازنده
Negative	منفی
Notation	نمایش
Number	عدد
Numeric	عددی
Operand	عملوند
Operand address	آدرس عملوند
Operating system	سیستم اجرائی
Operation	عمل
Operational unit	واحد اجرائی

Operator	اپراتور–عملگر
Operator error	خطای اپراتور
Option	گزینه
Order	ترتیب
Out of range	خارج از دامنه
Output	خروجی
Output equipment	تجهیزات خروجی
Output unit	واحد خروجی
Overflow	سرریزی–سرریزشدن
Overflow check	بررسی سرریزی
Page	صفحه
Page address	آدرس صفحه
Page heading	عنوان صفحه
Paging	صفحه به صفحه کردن
Parallel	موازی
Parallel operation	عملیات موازی
Parity bit	بیت توازن
Path	مسیر
Pattern	الگو
Pattern recognition	تشخیص الگو
Perform	انجام دادن–عمل کردن
Peripheral	جانبی
Peripheral equipment	وسائل جانبی –تجهیزات جانبی
Permanent	دائم

Phase	فاز–مرحله
Point	نقطه
Pointer	اشاره‌گر
Position	موقعیت – جا
Precision	دقت
Predefined	از قبل تعریف شده
Primary	اصلی–اولیه
Printer	چاپگر
Priority	اولویت
Priority interrupt	وقفه اولویت دار
Problem	مسأله
Procedural	رویه‌ای
Procedure	روال–رویه
Process	فرآیند
Processor	پردازشگر
Production	تولید–محصول
Program	برنامه
Program compilation	ترجمهٔ برنامه
Program error	خطای برنامه
Programmer	برنامه نویس
Programming	برنامه نویسی
Program segment	قطعه یا قسمتی از برنامه
Quantity	مقدار
Queue	صف

Ram	حافظه اصلی کامپیوتر
Random	تصادفی
Random access	دسترسی تصادفی
Random number	اعداد تصادفی
Rapid memory	حافظه سریع
Rate	نرخ
Ratio	نسبت
Raw data	داده‌های خام یا اولیه
Read	خواندن
Read time	زمان خواندن
Real number	اعداد حقیقی
Receive	دریافت کردن
Receiver	دریافت کننده
Record	رکورد
Record length	طول رکورد
Register	ثبات-رجیستر
Register address	آدرس ثبات
Relative	نسبی
Relative code	کد نسبی
Repeat	تکرار کردن
Repeat counter	شاخص تکرار
Repeater	تکرار کننده
Report	گزارش کردن-گزارش
Retrieve	بازیابی

Return	برگشت
Rules	قواعد
Run	اجرا کردن برنامه–اجرای برنامه
Sample	نمونه
Scan	با دقت نگاه کردن–بررسی کردن
Screen	صفحه نمایش
Section	بخش
Segment	قطعه–قسمت
Segmentation	قسمت کردن–قطعه قطعه کردن
Separator	جدا کننده
Sequence	دنباله–ترتیب
Sequential	متوالی
Sequential processing	پردازش متوالی
Serial access	دسترسی سری
Set	قراردادن–تعیین کردن–تنظیم کردن –میزان کردن – مجموعه
Shared storage	حافظه اشتراکی
Shift	جابه‌جائی
Sign	علامت
Signed	علامتدار
Significance	بااهمیت
Software	نرم افزار
Solution	راه حل
Sort	مرتب نمودن
Sort bubble	مرتب کردن به روش حبابی

Space	فضا
Stack	پشته
Standard	استاندارد
Status	وضعیت
Step	قدم–مرحله–گام
Storage	حافظه
Storage block	بلوک حافظه
String	رشته
Symbol	نماد
Symbolic address	آدرس نمادی
Symbol table	جدول نمادی
System	سیستم – دستگاه
Table	جدول
Table lookup	جستجوی جدول
Tape	نوار مغناطیسی
Temporary	موقت
Terminal	ترمینال–نقطه نهائی
Test data	داده جهت تست
Time	زمان
Time scale	مقیاس زمان
Transmission	انتقال
Unconditional	بدون شرط
Unit	واحد–قسمت
User	کاربر

Variable	متغیر
Variable length	بطول متغیر
Verify	بررسی کردن
Virtual	مجازی
Waiting state	وضعیت انتظار
Warning	اخطار
Word	کلمه-شانزده بیت
Write	نوشتن
Zero	صفر
Zero divide	تقسیم بر صفر