دانشگاه آزاد اسلامی

واحد لاهیجان

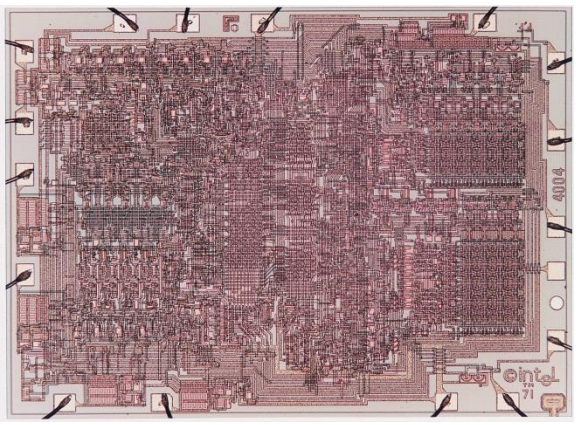
جزوه کمک آموزشی

زبان ماشین و اسمبلی

برنامه سازی سیستم

مهندسی کامپیوتر (نرم افزار – سخت افزار)

مدرس : دلدار



ویرایش سوم زمستان ۱۳۹۰

مراجع

- " زبان ماشین و اسمبلی و کاربرد آن در کامپیوترهای شخصی " ، دکتر حسن سید رضی
- " مرجع کامل برنامه نویسی به زبان اسمبلی از 8086 تا پنتیوم " ، عین ا... جعفرنژاد قمی
- " برنامه نویسی به زبان اسمبلی برای کامپیوترهای شخصی " ، پیتر ایبل ترجمه جابر هاشمی
- " برنامه نویسی سیستم برای کامپیوترهای شخصی (دو جلدی) " ، مایکل تیشر ترجمه امیر صادقی
- " مروری بر اسمبلی " ، هوروش فلاتی
- " اصول اساسی برنامه نویسی به زبان اسمبلی ویژه کامپیوترهای IBM " ، ریچارد دتمر ترجمه جابر هاشمی
- " برنامه نویسی سیستمی " ، محمد عادلی نیا
- " زبان اسمبلی طراحی و ارتباط کامپیوترهای آی بی ام 80X86 و سازگار با آن " ، محمد علی مزیدی
- " مدارهای واسط طراحی و ارتباط کامپیوترهای آی بی ام 80X86 و سازگار با آن " ، محمد علی مزیدی
- " زبان ماشین و اسمبلی " ، حمید رضا مقسمی
- " ریزپردازنده ها سری اینتل " ، باری بری ترجمه جمال میرحسینی
- " اصول و مبانی سخت افزار کامپیوترهای شخصی " ، استفن رومن ترجمه رضا خوش کیش

فهرست

فصل ۱

مقدمه

کامپیوترها و همه پردازشگرهای دیجیتال براساس منطق صفر و یک دیجیتال عمل می کنند .

اصولاً زبان های برنامه نویسی کامپیوتر به سه دسته کلی زبانهای سطح بالا ، زبانهای سطح میانه و زبانهای سطح پایین تقسیم می شوند . زبانهای سطح بالا شامل نرم افزارهایی هستند که با کاربر ارتباط بهتری برقرار می کنند مانند زبانهای برنامه نویسی پاسکال و VB در مقابل زبان های سطح پایین از لحاظ ساختار و ترجمه ، زبان هایی محسوب می شوند که با سخت افزار ارتباط نزدیکتری برقرار می کنند مانند زبان ماشین که به زبان صفر و یک معروف است .

معایب و محاسن زبان ماشین و زبان اسمبلی :

زبان اسمبلی اغلب هنگام ارتباط با سیستم عامل ، دسترسی مستقیم به خواص کلیدی ماشین و همچنین بهینه کردن قسمتهای حساس و مهم در یک برنامه کاربردی استفاده می شود . برنامه نویسی زبان اسمبلی نسبت به زبانهای سطح بالا دشوارتر است زیرا برنامه نویس بایستی به جزئیات توجه بیشتری نشان دهد و همچنین بایستی اطلاعات کافی نسبت به پردازنده داشته باشد . اما این برنامه سریع تر و با حافظه کمتری نسبت به زبانهای سطح بالا اجرا می شود .

چرا بایستی اسمبلی بیاموزیم ؟

البته یادگیری این زبان بایستی همراه با مفاهیم سیستم عامل و سخت افزار CPU (معماری کامپیوتر) همراه باشد تا درک بهتری از برنامه ها بدست آید . زبان اسمبلی وسیله خوبی جهت نحوه کار کامپیوتر ، کامپایلرها و زبانهای سطح بالا است . این گونه برنامه ها سریع تر ، کوچکتر و با توانایی هایی بیشتر از زبان های دیگر هستند و نیاز به حافظه و زمان کمتری برای اجرا دارند . برخی اعمال در زبان های سطح بالا همراه با محدودیتهایی مواجه هستند مانند دسترسی مستقیم به ثباتهای داخلی پردازنده و ... این محدودیتها در زبان اسمبلی جبران شده است . اکثر برنامه های سیستم عامل و همچنین برنامه های کامپیوترهای دستگاههای صنعتی و میکروکنترولرها به زبان اسمبلی می باشد .

مبناء ها :

اکثر کامپیوترها دارای یک زبان مشترک می باشند . زبان ماشین که کامپیوتر با آن کار می کند از مجموعه ای از صفر ها و یک ها تشکیل شده است . به عنوان مثال دستورات زیر دو عدد را با یکدیگر جمع کرده و نتیجه را نشان می دهد .

```
10110001 ——————→ IN
10110001 ——————→ IN
00110011 ——————→ ADD
00100010 ——————→ OUT
```

از آنجایی که کار با زبان ماشین مشکل می باشد زبان اسمبلی توسط سازندگان کامپیوتر عرضه شد .یکی از موارد مهم در زبان اسمبلی چگونگی ذخیره اطلاعات است بنابراین نیاز به مبناء و تبدیل مبناء می باشد . کامپیوتر و حافظه آن قادر است فقط اعداد صفر و یک را در خود ذخیره کند . (مبناء ۲) اما از آنجایی که کار با صفر و یک مشکل می باشد اغلب مبناء ۱۶ بکار برده می شود . بنابراین هرگاه سوالی در مورد چگونگی ذخیره داده ها در کامپیوتر بیان شود بایستی آنرا یا در مبناء ۲ و یا در مبناء ۱۶ بنویسیم (برای راحتی در مبناء ۱۶ می نویسیم)

اصولاً یک سیستم اعداد در مبناء n دارای n رقم می باشد .

تبدیل مبناءهایی که باید بیاموزیم :

تبدیل ۱۰ به ۲ ، تبدیل ۱۰ به ۸ ، تبدیل ۱۰ به ۱۶ ، تبدیل ۲ به ۱۰ ، تبدیل ۸ به ۱۰ ، تبدیل ۱۶ به ۱۰ ، تبدیل ۲ به ۸ و بلعکس ، تبدیل ۲ به ۱۶ و بلعکس ، تبدیل ۸ به ۱۶ و بلعکس

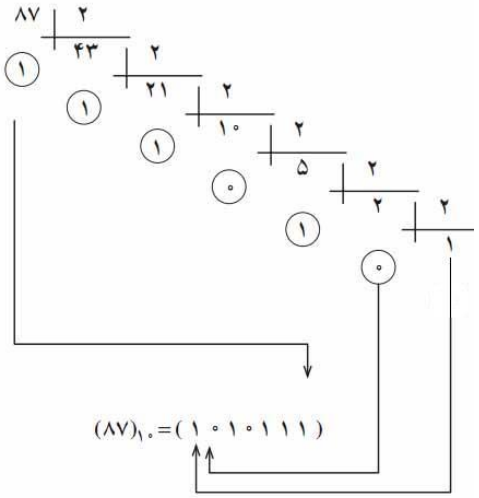
با یادگیری تبدیل های فوق می توان تبدیل های هر مبناء دیگری را نیز انجام داد .

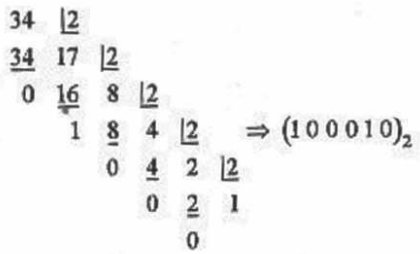
مبناء ۱۰ :

در این مبناء اعداد از ارقام ۰ الی ۹ تشکیل شده اند (۱۰ رقم) هر رقم به ضریبی از ۱۰ به توان عددی مشخص مرتبط است که آن عدد مشخص را مرتبه یا جایگاه و یا ارزش مکانی رقم گویند . به مثال زیر دقت کنید :

$$234 = 2\times10^2 + 3\times10^1 + 4\times10^0 = 200 + 30 + 4$$

مبناء ۲ (BINARY) : این سیستم تنها از دو وضعیت یا دو رقم (ارقام ۰ و ۱) تشکیل شده است . همانطور که از گذشته می دانیم راحت ترین راه جهت تبدیل هر عدد صحیح از مبناء ۱۰ به مبناء ۲ از طریق تقسیم های متوالی بر عدد ۲ و نگهداری باقیمانده ها و آخرین خارج قسمت انجام می شود . در مثال روش تبدیل عدد ۳۴ از مبناء ۱۰ به مبناء ۲ نشان داده شده است .

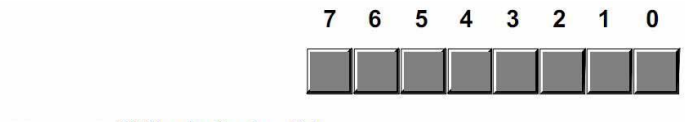




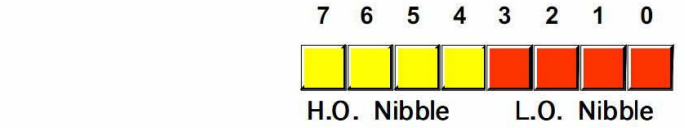
تعریف بیت : به کوچکترین واحد اطلاعات در سیستم دودویی که می تواند یک یا صفر باشد بیت گویند .

تعریف بایت : به تعداد هشت بیت یک بایت گفته می شود . در یک بایت اعداد 0 الی 255 را می توان نوشت .

تعریف نیبل : به تعداد چهار بیت یک نیبل گفته می شود .

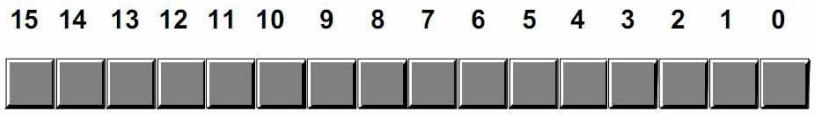


Bit Numbering in a Byte

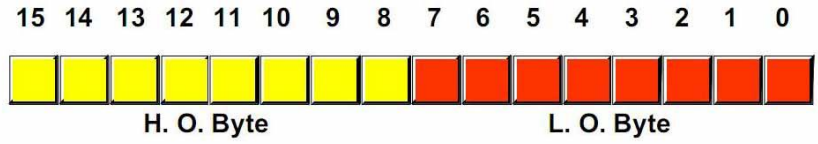


The Two Nibbles in a Byte

تعریف کلمه : به تعداد شانزده بیت یک کلمه گفته می شود .



Bit Numbers in a Word



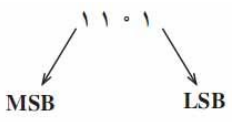
The Two Bytes in a Word

سوال : به نظر شما DOUBLEWORD و QUADWORD به ترتیب دارای چند بیت و چند بایت هستند ؟ (کلمه مضاعف و چهار کلمه)

نکته ۱ :

EB	PB	TB	GB	MB	KB
$2^{60}B$	$2^{50}B$	$2^{40}B$	$2^{30}B$	$2^{20}B$	$2^{10}B$

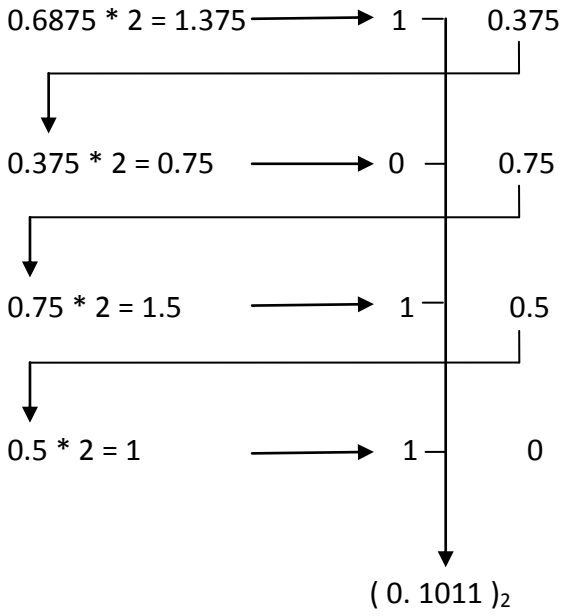
نکته ۲ : به عدد دودویی 1101 توجه کنید . به کم ارزش ترین بیت آن (LSB (Least Significant Bit و به پرارزشترین بیت آن MSB (Most Significant Bit) گویند .



نکته : هر بایت ۲۵۶ وضعیت مختلف از صفر ها و یک ها ایجاد می کند بنابراین یک بایت اعداد ۰ الی ۲۵۵ را در بر می گیرد .

تبدیل اعداد اعشاری از مبناء ۱۰ به مبناء ۲ : در این روش تبدیل بجای تقسیم از ضرب های متوالی استفاده می شود . به مثال زیر توجه کنید :

در این مثال می خواهیم عدد $(0.6875)_{10}$ را به سیستم دودویی تبدیل کنیم .



تبدیل اعداد از مبناء ۲ به مبناء ۱۰ : این نوع تبدیل با استفاده از جایگاه یا وزن ارقام صورت می گیرد . ضرایب ۲ به توان جایگاه ارقام می باشد . به مثال های زیر دقت کنید :

شماره‌های بیت‌ها	۷	۶	۵	۴	۳	۲	۱	۰
مقدار بیت‌ها	۰	۰	۰	۱	۱	۰	۰	۱
وزن‌های بیت‌ها	$۲^۷$	$۲^۶$	$۲^۵$	$۲^۴$	$۲^۳$	$۲^۲$	$۲^۱$	$۲^۰$

جایگاه یا وزن ارقام ← 3 2 1 0

$$(1101)_2 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 8 + 4 + 0 + 1 = 13$$

جایگاه یا وزن ارقام ← 3 2 1 0 -1-2-3

$$(1010.101)_2 = (?)_{10}$$

$$= 1\times2^3+0\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=8+0+2+0+0.5+0+0.125=10.625$$

نکته ۳ : جهت تبدیل اعداد دهدهی به هر مبناء دیگری می توان از روش تقسیم های متوالی استفاده کرد فقط تقسیم مورد نظر بجای عدد ۲ بر همان عددی انجام می شود که قرار است تبدیل به آن مبناء انجام شود . (تبدیل های ۱۰ به ۲ ، ۱۰ به ۸ و ۱۰به ۱۶ به همین روش انجام می شود)

نکته ۴ : جهت تبدیل یک عدد از یک مبناء مشخص به مبناء مشخص دیگری بهتر آن است که ابتداء عدد مورد نظر به مبناء ۱۰ تبدیل و سپس طبق نکته ۳ عمل کنیم .

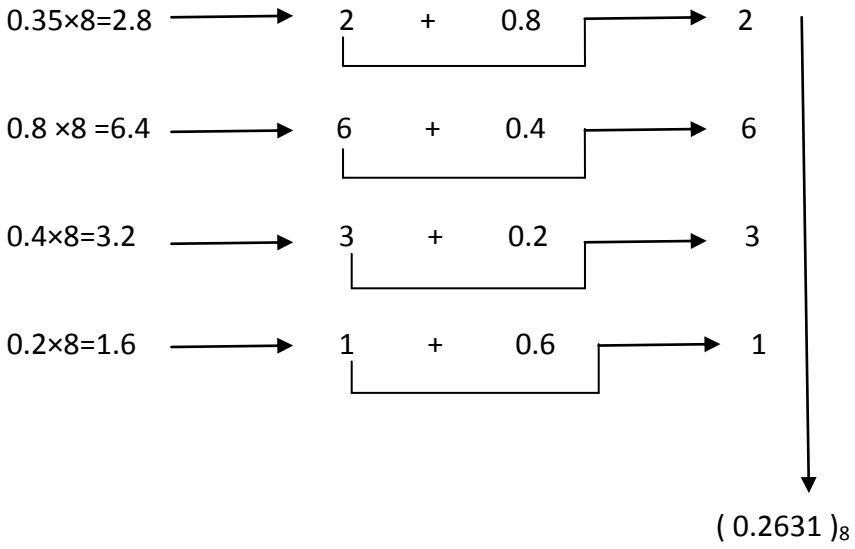
نکته ۵ : مبناء ۱۶ یا HEX در واقع روش فشرده تری را برای نمایش اعداد باینری ارائه می دهد بعبارت دیگر جهت ساده تر بیان کردن مبناء ۲ از مبناء ۱۶ استفاده می شود .

چند مثال مختلف :

$(75)_{10} = (?)_8 \longrightarrow (\ 113\ )_8$

$172 = (\ ?\ )_8 \longrightarrow (\ 254\ )_8$

$(0.35)_{10} = (?)_8$



DEC	BIN	OCT	HEX
0	0000	0	**0**
1	0001	1	**1**
2	0010	2	**2**
3	0011	3	**3**
4	0100	4	**4**
5	0101	5	**5**
6	0110	6	**6**
7	0111	7	**7**
8	1000	10	**8**
9	1001	11	**9**
10	1010	12	**A**
11	1011	13	**B**
12	1100	14	**C**
13	1101	15	**D**
14	1110	16	**E**
15	1111	17	**F**
16	10000	20	**10**
17	10001	21	**11**
18	10010	22	**12**

مثال : تبدیل خواسته شده را انجام دهید ؟

$(142)_5 = (?)_9$

$(142)_5 \longrightarrow (?)_{10} \longrightarrow (?)_9$

$(142)_5 = 1\times5^2+4\times5^1+2\times5^0=25+20+2=(47)_{10} \longrightarrow (52)_9$

نکته ۵ : از آنجایی که $2^3=8$ بنابراین هر رقم در مبناء ۸ معادل ۳ بیت در مبناء ۲ است و بلعکس .

نکته ۶ : از آنجایی که $2^4=16$ بنابراین هر رقم در مبناء ۲ است و بلعکس .

مثال مهم : به تبدیلات زیر به خوبی دقت کنید .

$$(\frac{10}{2}\ \frac{110}{6}\ \frac{001}{1}\ \frac{101}{5}\ \frac{011}{3}.\frac{111}{7}\ \frac{100}{4}\ \frac{000}{0}\ \frac{110}{6})_2=(26153.7406)_8$$

$$(\frac{10}{2}\ \frac{1100}{C}\ \frac{0110}{6}\ \frac{1011}{B}\quad \frac{1111}{F}\ \frac{0010}{2})_2=(2C6B.F2)_{16}$$

$$(673.124)_8=(\frac{110}{6}\ \frac{111}{7}\ \frac{011}{3}.\frac{001}{1}\ \frac{010}{2}\ \frac{100}{4})_2$$

$$(306.D)_{16}=(\frac{0011}{3}\ \frac{0000}{0}\ \frac{0110}{6}.\frac{1101}{D})_2$$

0	A	B	C	D	Hexadecimal
0000	1010	1011	1100	1101	Binary

جمع در مبناء های مختلف :

a	b	a+b
0	0	00
0	1	01
1	0	01
1	1	10

مثال : عملیات جمع زیر را به صورت دودویی انجام دهید .

1101

+1001

10110

13 ⟶ 00001101

\+ 9 ⟶ 00001001

00001101

+00001001

00010110 ⟶ 22

		1	1		
	1	1	0	1	1
+	0	0	0	0	1
	1	1	1	0	0

نکته : جمع اعداد در مبناء ۱۶ به صورت زیر انجام می گیرد .

$23D9^H$

$+94BE^H$

$B897^H$

9+E=23 ⟶ 23-16=7

1+D+B=25 ⟶ 25-16=9

1+3+4=8

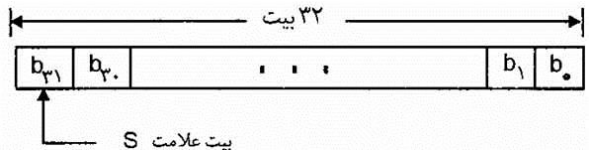
2+9=B

	1		1	
	7	E	C	6
+	3	4	0	A
	B	2	D	0

6 + A = 6 + 10 = 16 => 10h
C + 0 + 1 = 12 + 0 + 1 = 13 => Dh
E + 4 = 14 + 4 = 18 => 12h
7 + 3 + 1 = 11 => Bh

7 + 3 = A
6 + 7= D
F + 1 = 10
10 + 30 = 40
F + F = 1E
38 + 18 = 50
FF + 1 = 100

روش نمایش اعداد :



اعداد صحیح دودویی به دو صورت نمایش داده می شوند :

۱- اعداد صحیح بدون علامت UNSIGNED INTEGER شامل اعداد صحیح مثبت : در این روش کلیه بیت ها به داده مورد نظر اختصاص می یابد .

۲- اعداد صحیح علامت دار SIGNED INTEGER شامل اعداد صحیح مثبت و منفی : دراین روش یکی از بیت ها به علامت عدد مورد نظر اختصاص می یابد و سایر بیتها به داده مورد نظر اختصاص می یابد . معمولاً سه روش برای نمایش اعداد علامت دار وجود دارد :

الف) مکمل ۱

ب) مکمل ۲ ⟵ کامپیوتر های امروزی

ج) روش علامت مقدار ⟵ کامپیوتر های اولیه

سوال مهم : چگونه می توان تشخیص داد یک عدد دودویی زوج هست یا فرد ؟

بایستی به بیت سمت راست داده نگاه کنید . اگر این بیت صفر باشد داده زوج و اگر یک باشد داده فرد است .

مکمل ها :

در کامپیوتر اعداد منفی به کمک مکمل ها نمایش داده می شوند . مخصوصاً کامپیوتر ها جهت محاسبه عمل تفریق از مکمل ها استفاده می کنند . (اعداد منفی در کامپیوتر به کمک مکمل ۲ نمایش داده می شوند)

الف) مکمل ۱ : برای بدست آوردن مکمل ۱ بیت های آن عدد را معکوس می کنیم . (تبدیل ۱ ها به صفر ها و تبدیل صفرها به ۱ ها)

ب) مکمل ۲ : مکمل ۱ + ۱

مثال : مکمل ۲ عدد دودویی $(10011101)_2$ را بدست آورید .

10011101 ⟶ 01100011

10011101 ⟶ مکمل ۱ : 01100010 + 1

01100010

\+ 1

01100011

نکته : اولین صفرها و اولین یک کم ارزش ثابت و الباقی تبدیل صفرها به یک ها و یک ها به صفرها

1111 ⟶ 0001

1101100 ⟶ 0010100

101101 ⟶ 010011

مکمل ۹ : برای بست آوردن مکمل ۹ کافی است تمامی ارقام را از عدد ۹ کم کنید .

مکمل ۱۰ : برای بدست آوردن مکمل ۱۰ باید رقم اول را از عدد ۱۰ کم کنید و الباقی ارقام را از عدد ۹ کم کنید .

مثال : مکمل ۹ و ۱۰ اعداد زیر رابدست آورید .

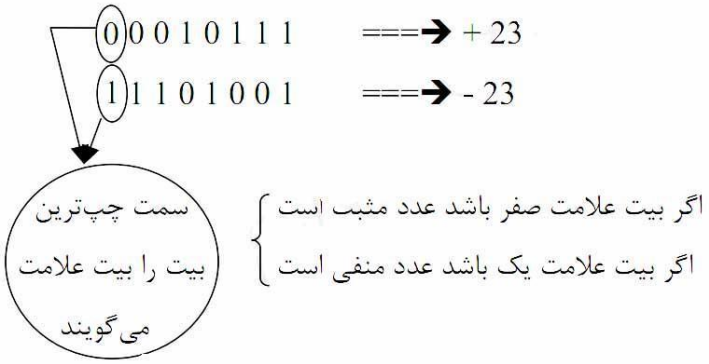
546700 مکمل ۱۰ ⟶ 453300 مکمل ۹ ⟶ 453299

012398 مکمل ۱۰ ⟶ 987602 مکمل ۹ ⟶ 987601

اعداد دودویی علامت دار (علامت مقدار) : همانطور که گفتیم در کامپیوتر جهت مشخص کردن علامت مثبت و منفی از بیت علامت استفاده می شود . معمولاً سمت چپ ترین بیت ، به علامت اختصاص داده می شود . بزرگترین عدد بدون علامت زمانی است که همه بیت ها برابر ۱ باشد و کوچکترین عدد باینری برابر صفر است . بزرگترین عدد باینری n بیتی برابر 2^n-1 است .

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 $2^{31}-1$

به سمت چپ‌ترین بیت ، بیت علامت می‌گویند که نشان دهنده نوع عدد است.



مثال ۱ : عدد 12+ و 12- را به روشهای زیر نشان دهید :

الف) روش علامت مقدار

ب) روش مکمل ۱

ج) روش مکمل ۲

جواب : الف)

$+12 = (00001100)_2 = 0C^H$ $-12 = (10001100)_2 = 8C^H$

ب)

$+12 = (00001100)_2 = 0C^H$ $-12 = (11110011)_2 = F3^H$

ج)

$+12 = (00001100)_2 = 0C^H$ $-12 = (11110100)_2 = F4^H$

مثال : عدد 20- در کامپیوتر به چه صورتی نمایش داده می شود ؟

روش حل : ۱- ابتداء عدد را بدون در نظر گرفتن علامت به دودویی تبدیل می کنیم .

-20 ⟶ 10100

۲- اگر تعداد ارقام کمتر از ۸ رقم باشد بایستی آنقدر رقم 0 در سمت چپ اضافه کنیم تا ارقام ۸ رقم گردد .

00010100

۳- اگر تعداد ارقام بیشتر از ۸ رقم باشد بایستی آنقدر رقم 0 در سمت چپ اضافه کنیم تا ارقام ۱۶ رقم گردد .

۴- سپس نتیجه حاصله را مکمل۲ می گیریم .

11101100 ⟶ -20

مثال : عدد FC^H معادل چه مقداری در مبناء ده می باشد ؟

از آنجایی که این عدد معادل عدد دودویی 11111100 می باشد و با در نظر گرفتن بیت علامت که منفی می باشد بنابراین از عدد مورد نظر مکمل ۲ می گیریم که نتیجه 00000100 حاصل می شود که عدد ۴ را نشان می دهد بنابراین عدد FC^H معادل عدد ۴- می باشد .

تفریق در کامپیوتر :

از آنجایی که در کامپیوترهای امروزی برای نمایش اعداد منفی از مکمل ها استفاده می شود (مکمل ۲) و از آنجایی که در کامپیوتر از عمل جمع برای بدست آوردن حاصل تفریق استفاده می شود بنابراین بطور خلاصه تفریق در کامپیوتر عبارتست از جمع با مکمل بصورت زیر :

حاصل تفریق = عدد دوم – عدد اول

حاصل تفریق = مکمل عدد دوم (عدد منفی) + عدد اول (عدد مثبت)

نکته مهم : عملیات محاسباتی مخصوصاً تفریق در هر مبنایی و با هر مکملی حتماً و بایستی یک مقدار مشخص و ثابت بدست آید حاصل تفریق 8-12 در هر مبناء و با هر مکملی همیشه عدد ۴ می باشد . درهنگام محاسبات در نظر گرفتن نوع مکمل اختیاری است .

مثال : حاصل تفریق های زیر را به کمک مکمل ها بدست آورید .

(1) 72532-3250=

می دانیم که حاصل تفریق فوق ۶۹۲۸۲+ می باشد . می خواهیم حاصل تفریق فوق را به کمک مکمل ۱۰ بدست آوریم :

03250 96750

72532+96750 = 169282 +69282

حال همان تفرق را به کمک مکمل ۹ حل می کنیم :

03250 96749

72532+96749 = 169281 169281

\+ 1

69282

همانطور که دیده می شود حاصل یک جواب مشترک و ثابت می باشد .

---

(2) 3250-72532=

مکمل ۱۰ :

72532 27468
3250+27468 = 30718 -(69282)

مکمل ۹ :

72532 27466
3250+27467 = 30717 -(69282)

---

(3) 1010100-1000011=

مکمل ۲ :

1000011 0111101
1010100+0111101=10010001 0010001

مکمل ۱ :

1000011 0111100
1010100+0111100=10010000 0010001

---

(4) 1000011-1010100=

مکمل ۲ :

1010100 0101100
1000011+0101100=1101111 -(0010001)

مکمل ۱ :

1010100 0101011
1000011+0101011=1101110 -(0010001)

---

(5) 27-20=

مکمل ۹ :

20 ⟶ 79

27+79=106 1 + 06 = 07

مکمل ۱۰ :

20 ⟶ 80

27 + 80 = 107 ⟶ 07

مکمل ۱ :

27 ⟶ 00011011

20 = (00010100)$_2$ ⟶ 11101011

00011011 – 00010100 = 00011011 + 11101011 = 100000110

00000110 + 1 = 00000111 ⟶ (07)$_{10}$

مکمل ۲ :

20 = (00010100)$_2$ ⟶ 11101100

00011011 – 00010100 = 00011011 + 11101100 = 100000111 00000111 (07)$_{10}$

---

(6) 20-27=

مکمل ۹ :

27 ⟶ 72

20 + 72 = 92 ⟶ -(07)

مکمل ۱۰ :

27 ⟶ 73

20 + 73 = 93 ⟶ -(07)

مکمل ۱ :

27 00011011 ⟶ 11100100

00010100 – 00011011 = 00010100 + 11100100 = 11111000 -(0000111) -(07)$_{10}$

مکمل ۲ :

27 00011011 ⟶ 11100101

00010100 – 00011011 = 00010100 + 11100101 = 11111001 -(00000111) -(07)$_{10}$

---

(7) $24F^H – 129^H$ =

F – 9 = 15 – 9 = 6

4 – 2 = 2

2 – 1 = 1

$24F^H – 129^H = 126^H$

مکمل ۱۵ :

$129^H \longrightarrow ED6^H$

$24F^H + ED6^H = 1125^H$ $125^H+1^H = 126^H$

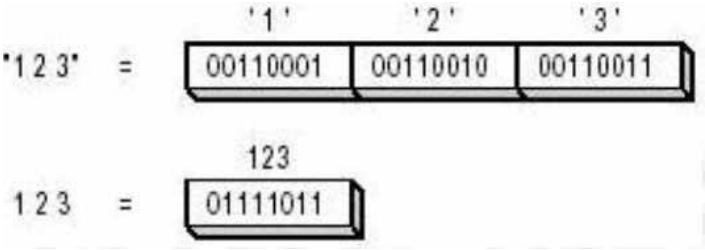
مکمل ۱۶ :

$129^H \longrightarrow ED7^H$

$24F^H + ED7^H = 1126^H \longrightarrow 126^H$

برخی کدهای دودویی :

کد BCD : Binary Coded Decimal هرگاه به جای آنکه یک عدد مستقیماً به مبناء ۲ برده شود ، تک تک ارقام جداگانه به مبناء ۲ برده و مجموعه صفر ها و یک ها به ترتیب جایگاهشان در کنار هم قرارگیرند کد BCD حاصل می شود . در نمایش عدد دهدهی از آنجایی که ده رقم داریم بنابراین چهار بیت برای نمایش آن کافیست . از ترکیبات 0 الی 9 استفاده کرده و از شش ترکیب دیگر استفاده نمی شود .



395 ⟶ دودویی ⟶ $(110001011)_2$

395 ⟶ BCD ⟶ $(0011\ 1001\ 0101)_2$

کد گری : هنگامی که از یک سیستم دیجیتال برای پردازش یک سیستم آنالوگ استفاده می شود بایستی که داده آنالوگ را به مقادیر دیجیتال تبدیل کنیم . در این تبدیل رشته ای از اعداد صفر و یک پشت سرهم تولید می شود . استفاده کد گری که در آن بین هر دو عدد متوالی فقط وضعیت یک بیت تغییر می کند امکان وقوع خطا را کاهش خواهد داد . این کد یک کد چهار بیتی می باشد که همانطور که در جدول زیر دیده می شود در گذر از یک عدد به عدد دیگر تنها وضعیت یک بیت تغییر می کند .

اعداد	کد گری
0	0000
1	0001
2	0011
3	0010
4	0110
.	.
.	.
.	.

کد اسکی : ASCII

این روش برای نمایش حروف الفباء و کاراکترهای مختلف و همچنین نقل و انتقالات آنها در دستگاههای جانبی مختلف بکار رفته که از یک استاندار خاص تبعیت شده است . این کد معمولاً از هفت بیت و گاهی از هشت بیت تشکیل شده است . در مجموع شامل ۱۲۸ کاراکتر به شرح زیر می باشد : $2^7=128$ $2^8=256$

۲۶ کد شامل حروف کوچک انگلیسی

۲۶ کد شامل حروف بزرگ انگلیسی

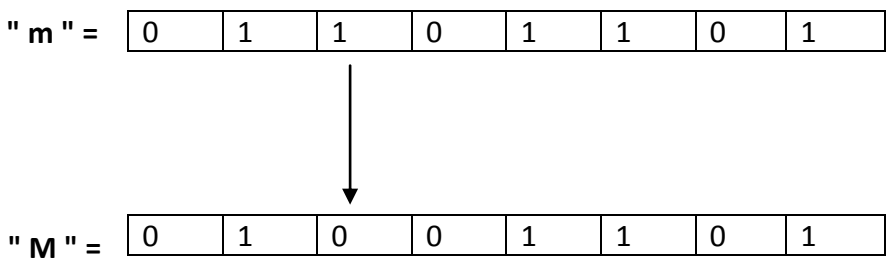
۱۰ کد شامل اعداد 0 الی 9

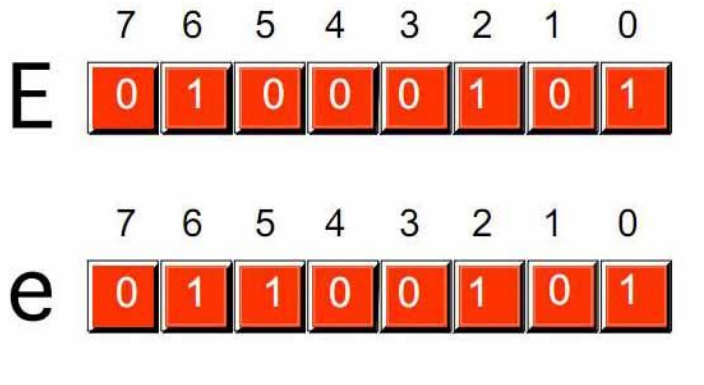
۳۲ کد شامل کاراکترهای قابل چاپ مانند * $ @ % &

۳۴ کد شامل کاراکترهای غیر قابل چاپ (کنترلی) Ctrl Enter Delete

حرف	کد باینری	حرف	کد باینری
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	#	010 1010
S	101 0011	)	010 1001
T	101 0100	-	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

نکته ۱ : تفاوت حروف بزرگ با حروف کوچک فقط در بیت ششم است (بیت d5) این بیت در حروف بزرگ صفر و در حروف کوچک یک است .





نکته ۲ : یک روش کد گذاری که اخیراً بجای اسکی استفاده می شود UNICODE می باشد . تنها تفاوت بین اسکی و UNICODE آن است که در کد اسکی برای هر کاراکتر ۱ بایت درنظر گرفته می شود در حالی که در UNICODE برای هر کاراکتر ۲ بایت درنظر گرفته می شود و در واقع کاراکترهای بیشتری توسط این کد می تواند نمایش یابد و این جهت نمایش کلیه کاراکترهای سایر زبانهای دنیا کاربرد دارد . در زیر نمایش حرف A در کدهای فوق نشان داده شده است :

A : ASCII $\longrightarrow 41^H$

A : UNICODE $\longrightarrow 0041^H$

بیت توازن :

اطلاعات دودویی در هنگام ارسال و دریافت توسط سیستمهای دیجیتال ممکن است دچار اختلال شود . هر پارازیت یا نویزی می تواند صفرها یا یک ها را معکوس کند . معمولی ترین روش جهت تشخیص خطا بکار بردن بیت توازن است . این بیت از روی تعداد زوج یا فرد بودن یک ها در یک کد دودویی حاصل می شود .

P توازن فرد	P توازن زوج	
1	0	0000
0	1	0001
0	1	0010
1	0	0011
0	1	0100
1	0	0101
1	0	0110
.	.	.
.	.	.

تمرین های فصل ۱ :

۱- تبدیل مبناء های زیر را انجام دهید :

$421 = (\ldots\ldots)_2$

$153.513 = (\ldots\ldots)_2$

$589 = (\ldots\ldots)_H$

$3BA4^H = (\ldots\ldots)_{10} = (\ldots\ldots)_2$

$(10010000)_2 = (\ldots\ldots)_{10}$

$(927)_{10} = (\ldots\ldots)_8$

$(1637)_8 = (\ldots\ldots)_{10}$

$(A36)_{16} = (\ldots\ldots)_8$

$(753)_8 = (\ldots\ldots)_{16}$

$(111011.0101)_2 = (\ldots\ldots)_8 = (\ldots\ldots)_{16} = (\ldots\ldots)_{10}$

۲- اعداد زیر در کامپیوتر به چه صورتی نمایش داده می شوند :

+1116

-97

۳- به نظر شما اگر عددی در حافظه کامپیوتر بصورت $(11001000)_2$ ذخیره شده باشد آن عدد در مبناء ۱۰ چه عددی می باشد ؟

۴- در حافظه کامپیوتر اعداد $0D43^H$ و $B2EB^H$ ذخیره شده است . معادل دهدهی آنها را پیدا کنید .

۵- حاصل عملیاتهای زیر را بدست آورید .

19+7 =

100011 + 1101011 =

F34H+5D6H=

2CH+3FH=

FE9H-5CCH=

9FF25H-4DD99H=

۶- حاصل تفریق زیر را به کمک مکمل ۲ بدست آورید .

31-14=

۷- حاصل تفریق های زیر را با کمک مکمل ۱۰ بدست آورید .

5250	1753	20	1200
-1321	-8640	-100	-250

۸- اعمال ریاضی زیر را در سیستم دودویی با استفاده از مکمل ۲ علامت دار بصورت جداگانه انجام دهید .

16	42
-27	-13

۹- آیا می توانید کد اسکی رشته زیر را در مبنای شانزده بنویسید ؟

" IRAN is a country "

۱۰- یک Paragraph چند بیت و چند بایت است ؟

۱۱- یک قلم داده 64 بیت چند کلمه است ؟

۱۲- حاصل تفریق های زیر را با کمک مکمل ۲ بدست آورید .

1000100	1100101
- 100000	-1101111

۱۳- یک عدد در مبناء شانزده در هر یک از حالات زیر چند رقم می باشد ؟

الف) یک بایت ب) یک کلمه ج) یک کلمه مضاعف

سوالات چهار گزینه ای :

۱- عدد یک بایتی 129- در کامپیوتر چگونه ذخیره می شود ؟

۱) $FFF1^H$ ۲) $FF81^H$ ۳) FE^H ۴) $FF7F^H$

۲- حاصل عبارت $(?)_{16} = (742)_8 + (3260)_8$ کدام است ؟

۱) 892 ۲) 792 ۳) 882 ۴) 782

۳- حاصل تفریق دو عدد باینری 110000 و 100 کدام گزینه است ؟

۱) 101100 ۲) 10011 ۳) -101100 ۴) -11111

فصل ۲

ساختار کامپیوتر و ریزپردازنده های 80X86

داده ← پردازش ← اطلاعات

ابر رایانه ← رایانه بزرگ ← رایانه کوچک و متوسط ← ریز رایانه

واژه میکروپروسسور در صنعت نیمه هادی توسط شرکت اینتل ابداع شد . آنها این واژه را برای توصیف یک مدار مجتمع ماشین حساب چهار بیتی که تازه طراحی کرده بودند به کار بردند . امروزه میکروپروسسور به آی سی هایی گفته می شود که اساس یک میکرو کامپیوتر را تشکیل می دهند.

بعضی سازندگان به کار بردن چند میکروپروسسور در یک کامپیوتر را مفید تشخیص داده اند . یکی از میکروپروسسورها برای کنترل صفحه کلید ، دومی برای پرداختن به عملیات ورودی - خروجی ، سومی برای کنترل وسایل ذخیره سازی انبوه (دیسک گردانها)و چهارمی به عنوان پروسسور اصلی سیستم می توانند به کار روند . این تکنیک پردازش توزیع شده Distribued Processing نام دارد .

شاید بتوان علت ساخت پردازنده ها را در سادگی ، کم بودن حجم و برنامه پذیری خلاصه کرد اما در واقع هر سیستم کامپیوتری بر اساس نحوه دریافت و پردازش اطلاعات به صورتهای زیر تقسیم می شود :

کامپوتر آنالوگ : دارای ورودی آنالوگ

کامپیوتر دیجیتال : دارای ورودی دیجیتال

کامپیوتر ترکیبی : دارای ورودی آنالوگ و دیجیتال (یک کاربرد در هواشناسی)

رایانه نسل اول	لامپ خلاء	زبان ماشین
رایانه نسل دوم	ترانزیستور	زبان اسمبلی
رایانه نسل سوم	آی سی	زبان سطح بالا
رایانه نسل چهارم	VLSI	زبان سطح بالا
رایانه نسل پنجم		هوش مصنوعی
رایانه نسل ششم		شبکه های عصبی مصنوعی

ریز پردازنده مدار الکترونیکی بسیار گسترده و پیچیده ای میباشد که دستورات برنامه های ذخیره شده را انجام می دهد . پردازنده دو عمل مهم انجام می دهد:

۱-کنترل تمام محاسبات و عملیات

۲-کنترل قسمت های مختلف

پردازنده ها وظایف اصلی زیر را برای رایانه انجام می دهد:

۱-دریافت داده هااز دستگاه های ورودی

۲-انجام عملیات و محاسبات وکنترل ونظارت بر آنها

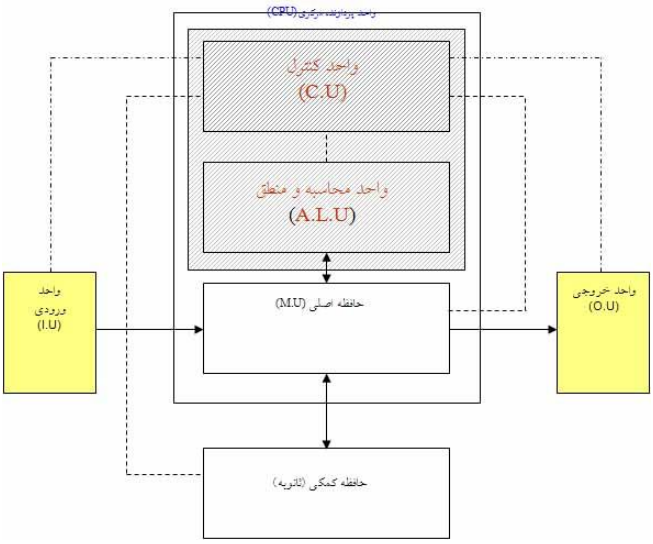
۳-ارسال نتایج عملیات با دستگاه های خروجی

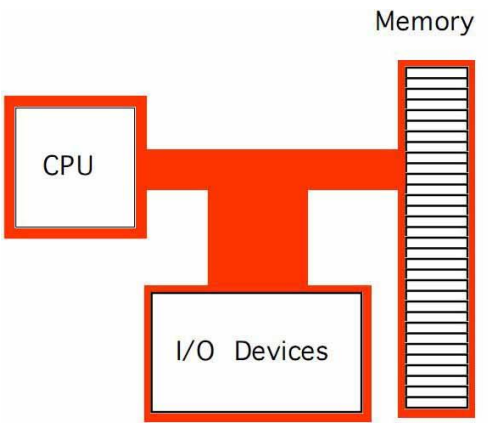
همچنین عملکرد یک پردازنده از نظر فنی با دو ویژگی تعیین می شود :

۱- تعداد بیت هایی که یک پردازنده در هر لحظه پردازش می کند . طول این کلمات معمولاً ۴و۸و۱۶و۳۲و۶۴ بیتی می باشد .

۲- تعداد پالس های الکترونیکی که در یک ثانیه تولید شده است و با واحد مگاهرتز سنجیده می شود.

اجزای یک سیستم کامپیوتری





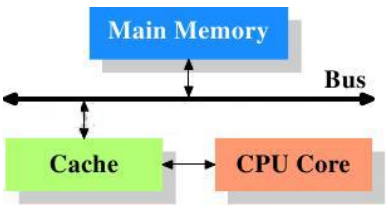
نکته ۱ : پردازنده بطور مستقیم با حافظه اصلی در ارتباط است .

نکته ۲ : ظرفیت حافظه های اصلی خیلی بالا نیست بنابراین لازم است از حافظه های کمکی نیز استفاده شود که دارای سرعت کمتری می باشد .

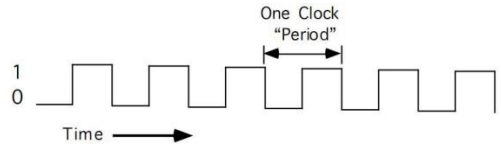
بخشهای داخلی یک ریزپردازنده CPU :

- ALU (ARITHMATIC LOGIC UNIT) : وظیفه این بخش انجام عملیاتهای محاسباتی (ریاضی) و منطقی است .

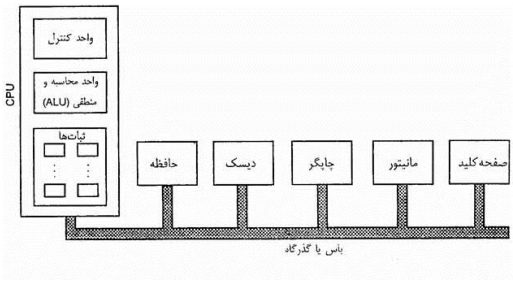
- CU (CONTROL UNIT) : این قسمت فرمانهای لازم جهت کنترل و نظارت بر سیستم را به قسمتهای مختلف ارسال و یا از قسمتهای مختلف دریافت می کند .
- REGISTER : ثباتها حافظه های کوچک و سریعی هستند که در درون پردازنده جهت ذخیره موقت اطلاعات بکار می روند و بسته به نوع ریزپردازنده می توانند ۸ بیتی ، ۱۶ بیتی ، ۳۲ بیتی و ۶۴ بیتی و ... باشند .
- CACHE : این قسمت جهت ذخیره اطلاعاتی که بطور مداوم استفاده می شود بکار می رود .



- ACCUMULATOR : ثباتی است که از سایر ثباتها متمایز بوده و عمدتاً در متن اجرای برخی دستورات قرار دارد که جهت نگهداری نتجه مورد استفاده قرار می گیرد . (برگه کار موقت)
- IR (INSTRUCTION REGISTER) : این قسمت معنای هر دستور و مراحلی که یک پردازنده بایستی برای اجرای آن در پیش بگیرد را مشخص می کند . دستورات پس از دریافت از حافظه وارد این قسمت شده تا پس از رمزگشایی اجرا شوند . (فرهنگ لغت کامپیوتر)
- PC (PROGRAM COUNTER) : از آنجایی که دستورات یک برنامه بایستی به ترتیب اجرا شوند پردازنده بایستی به طریقی بداند که دستور بعدی را که بایستی اجرا کند کدام است . وظیفه این ثبات نگهداری آدرس دستور بعدی است که قرار است اجرا شود . با اجرای هر دستور پردازنده به طور خودکار یک واحد به این ثبات اضافه می کند تا به دستور بعدی اشاره کند . (همان ثبات IP در برخی کامپیوترها)
- CP : هر دستورالعمل در یک سری مراحل اجرا می شود . CPU برای همگام کردن سیکل اجرای دستور از یک سری پالس های ساعت استفاده می کند . (سیکل ساعت مجموعه ای از پالس های ساعت برای اجرای هر دستور است) مدارهای الکترونیکی کامپیوتر از این پالس های ساعت برای انجام صحیح عملیات خود استفاده می کنند . مثلاً وقتی یک کامپیوتر 2GHz می خرید در واقع فرکانس پالس های ساعت آن 2GHz است . یعنی در هر ثانیه ۲ میلیارد پالس تولید می کند . اصطلاح MIPS (Milion Instruction Per Second) مربوط به سرعت cpu در اجرای دستورالعمل ها است .

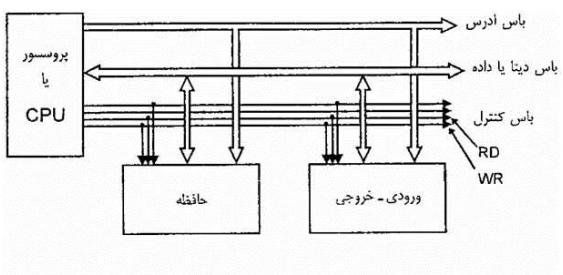


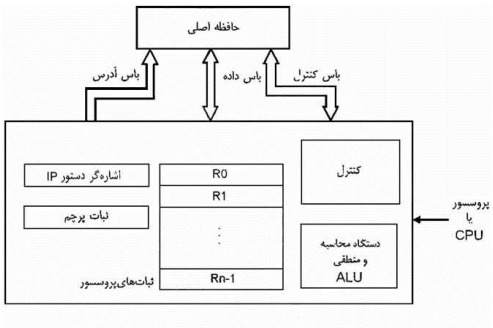
- BUS : سیمهای ارتباطی بین عده ای وسایل در کامپیوتر باس یا گذرگاه نامیده می شود .



گذرگاه سیستم : داده ، آدرس ، کنترل

گذرگاه ورودی/خروجی





گذرگاه داده DATA BUS : تمامی اطلاعاتی که بایستی در یک سیستم کامپیوتری جابجا شود از این گذرگاه عبور می کند . هرچقدر تعداد پایه های این گذرگاه بیشتر باشد انتقال داده با سرعت بیشتری انجام می شود . این گذرگاه دو طرفه بوده و عرض آن برابر تعداد بیتهای ثبات های داخلی پردازنده می باشد . معمولاً بصورت D16 D15 D14 D13 .... D0

80x86 Processor Data Bus Sizes

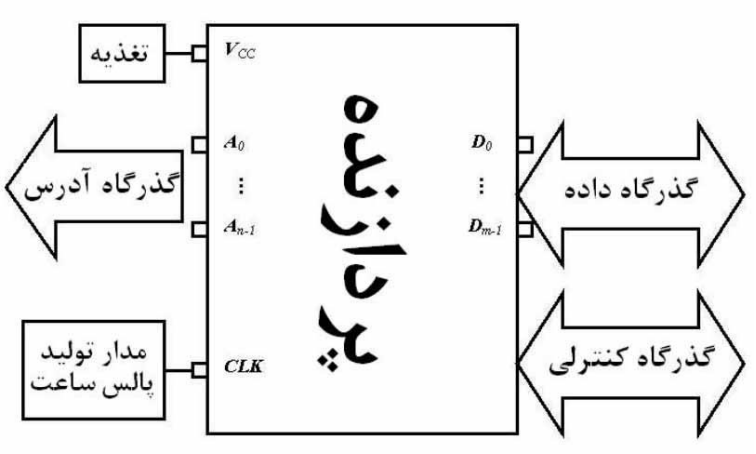
Processor	Data Bus Size
8088	8
80188	8
8086	16
80186	16
80286	16
80386sx	16
80386dx	32
80486	32
80586 class/ Pentium (Pro)	64

گذرگاه آدرس ADDRESS BUS : این گذرگاه مشخص می کند که گذرگاه داده در هر لحظه بایستی در اختیار چه وسیله ای باشد . به منظور شناسایی یک وسیله ، پردازنده یک آدرسی به آن اختصاص می دهد . CPU این آدرس را روی گذرگاه آدرس قرار داده و یک مدار دیکد وسیله مورد نظر را پیدا می کند . هرچقدر تعداد پایه های این گذرگاه بیشتر باشد پردازنده تعداد وسایل بیشتری را می تواند آدرس دهی کند . (n پایه آدرس قابلیت آدرس دهی کردن 2^n وسیله را دارد) همچنین به کمک این گذرگاه ماکزیمم حافظه قابل دسترس نیز مشخص می شود . معمولاً بصورت A16 A15 A14 A13 .... A0

Table 18: 80x86 Family Address Bus Sizes

Processor	Address Bus Size	Max Addressable Memory	In English!
8088	20	1,048,576	One Megabyte
8086	20	1,048,576	One Megabyte
80188	20	1,048,576	One Megabyte
80186	20	1,048,576	One Megabyte
80286	24	16,777,216	Sixteen Megabytes
80386sx	24	16,777,216	Sixteen Megabytes
80386dx	32	4,294,976,296	Four Gigabytes
80486	32	4,294,976,296	Four Gigabytes
80586 / Pentium (Pro)	32	4,294,976,296	Four Gigabytes

گذرگاه کنترل CONTROL BUS : ارتباط دو گذرگاه قبلی را کنترل و نظم می بخشد که شامل سیگنالهای کنترلی RD و WR و ... می باشد .



سوال : منظور از cache L1 یا cache L2 چیست ؟

سوال تحقیقی : همانطور که می دانید نسل دوم پردازنده های جدید اینتل براساس معماری Core2 با اسم رمز Sandy bridge ساخته شده است . مشخصه های پردازنده Ivy bridge را بنویسید . همچنینی مشخصه کلی ریزپردازنده extreme 965 intel Core i7 که دارای معماری 2QuadCore می باشد را بنویسید .

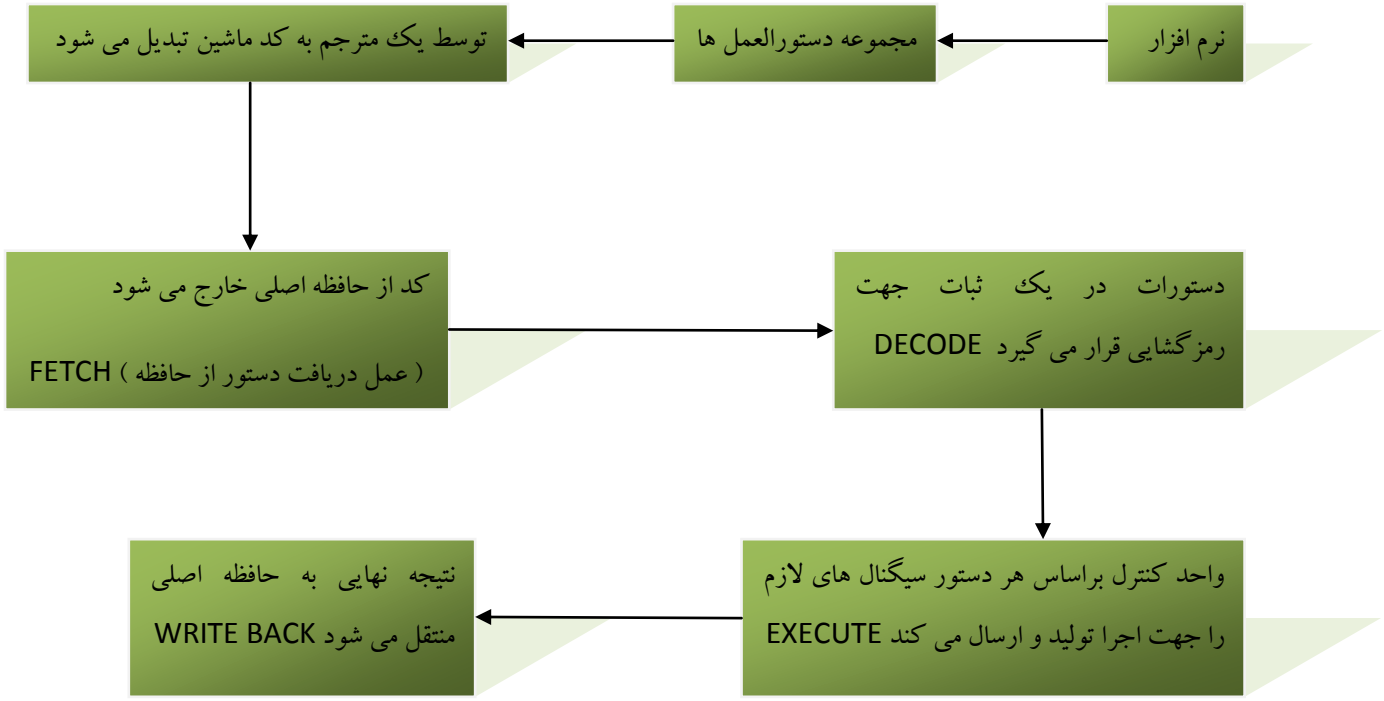
پردازنده های 80X86 اینتل

4004	این پردازنده یک پردازنده ۴ بیتی بود که جهت ماشین حسابهای دستی طراحی شده بود و از ۲۳۰۰ ترانزیستور PMOS ساخته شده بود.
8008	این پردازنده یک پردازنده ۸ بیتی ۱۸ پایه بود که از ۳۰۰۰ ترانزیستور PMOS ساخته شده بود و دارای ۶۶ دستورالعمل بوده و قابلیت آدرس دهی 16KB را دارا می باشد .
8080	این پردازنده ۴۰ پایه بود که از تکنولوژی NMOS ساخته شده بود و دارای ۱۱۱ دستورالعمل بوده و قابلیت آدرس دهی 64KB را دارا می باشد .
8085	اینتل ۳ پردازنده فوق را در یک تراشه تحت عنوان 8085 قرار داد .
8086	این پردازنده یک پردازنده ۱۶ بیتی بوده و قابلیت آدرس دهی 1MB حافظه را دارا بود .
8088	تا قبل از استفاده شرکت IBM از پردازنده های اینتل ، شرکت اینتل بعنوان تولید کننده تراشه های حافظه شناخته می شد . اینتل این پردازنده را برای کاهش هزینه های بوردهای 8085 طراحی نمود . پردازنده ای که در درون ۱۶ بیتی ولی دارای ۸ پایه گذرگاه داده بود و از ۲۹۰۰۰ ترانزیستور ساخته شده بود .
80286	تمامی پردازنده های قبلی بصورت بسته بندی DIP ساخته شده بودند . این پردازنده یک پردازنده ۱۶ بیتی ، دارای ۲۴ پایه گذرگاه آدرس و بصورت بسته بندی LCC و از ۱۳۰۰۰۰ ترانزیستور ساخته شده بود که توانایی اجرای کلیه دستورات ۸۰۸۶ و ۸۰۸۸ را دارا بود .
80386	قبل از ساخت این پردازنده اینتل تولید تراشه های حافظه را متوقف کرد . این پردازنده ۳۲ بیتی دارای ۳۲ پایه گذرگاه آدرس و ۱۳۲ پایه که بصورن بسته بندی PGA و از ۲۷۵۰۰۰ ترانزیستور CMOS ساخته شده بود .
80486	قبل از ساخت این پردازنده اینتل مجوز تولید تراشه های ۸۰۸۶ و ۸۰۸۸ را برای سایر شرکت ها صادر و خود تنها به ساخت تراشه ۸۰۳۸۶ پرداخت . این تراشه اولین ریزپردازنده ای بود که شامل ۱/۲ میلیون ترانزیستور بود . یک پردازنده ۳۲ بیتی با ۱۶۸ پایه در بسته بندی PGA و قابلیت آدرس دهی 4GB حافظه .
PENTIUM	اولین پردازنده پنتیوم با بیش از ۳ میلیون ترانزیستور BiCMOS که با پردازنده های قبلی سازگار بود ساخته شد . این پردازنده دارای گذرگاه داده ۶۴ بیتی و ثباتهای داخلی ۳۲ بیتی بود . دارای دو عدد حافظه نهان داخلی (یکی برای داده و یکی برای آدرس) و مدارهای پیش بینی پرش و ۲۷۳ پایه ای و تقریباً دو برابر سریعتر از ۸۰۴۸۶ بود .

انواع RAM : به طورکلی دو نوع و گاهاً سه نوع RAM وجود دارد

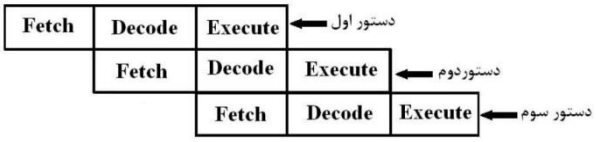
۱- SRAM : در ساخت این نوع حافظه از فلیپ فلاپها استفاده می شود . جهت ساخت هر سلول حافظه از ۶ ترانزیستور استفاده می شود (امروزه ۴ ترانزیستور) در ساخت حافظه های CACHE به دلیل سرعت بالاتر این نوع حافظه ها استفاده می شود .
۲- DRAM : در ساخت این نوع حافظه از خازن استفاده می شود . جهت ساخت هر سلول حافظه از ۱ ترانزیستور استفاده می شود .
۳- NV-RAM : این نوع حافظه ترکیبی از مزایای RAM و ROM را دارا می باشد . دارای یک باطری لیتیوم داخلی بعنوان منبع انرژی پشتیبان جهت نگهداری داده می باشد به همین دلیل به آن RAM غیرفرار گویند .

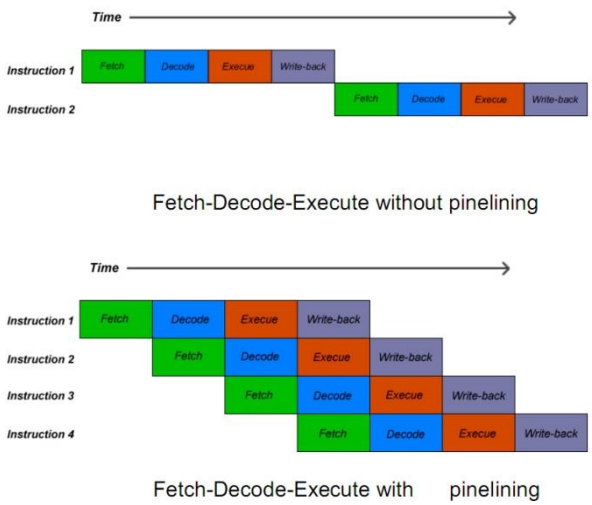
کار پردازنده :



تکنیک خط لوله ای PIPELINE :

پردازنده ۸۰۸۵ و قبل از آن برای اجرای هر دستور سه مرحله دریافت ، رمزگشایی و اجرا را بطور مجزا و مستقل از دستورات قبلی و بعدی انجام می دادند . این موضوع باعث می شود که نتوانیم از امکانات یک پردازنده بطور بهینه استفاده کنیم . زیرا زمانی که پردازنده یک دستور را از حافظه دریافت می کند وآن را جهت رمزگشایی به بخش دیکد می سپارد در حین رمزگشایی واحد دریافت اطلاعات بیکار است . این درحالی است که دریافت دستور دوم هیچگونه ارتباطی به رمزگشایی دستور اول ندارد . بطور مشابه وقتی در ۸۰۸۵ دستور اول را به بخش اجرا می سپارد بخشهای دریافت و رمزگشایی بیکار هستند . اما در ۸۰۸۶ و پردازنده های نسل های بعد همزمان با رمزگشایی دستور اول از آنجایی که واحد دریافت اطلاعات بیکار است ، عمل دریافت دستور دوم نیز بطور همزمان انجام می شود و این عمل سرعت پردازش را بالا می برد . به نحو مشابه همزمان با اجرای دستور اول ، رمزگشایی دستور دوم و دریافت دستور سوم نیز انجام می شود . درواقع بجای آوردن آب بصورت سطل به سطل از چاه می توان یک خط لوله از چاه به منزل کشید تا آب بصورت جاری و پشت سر هم منتقل شود .





انواع پردازنده از نظر نوع اجرای دستور :

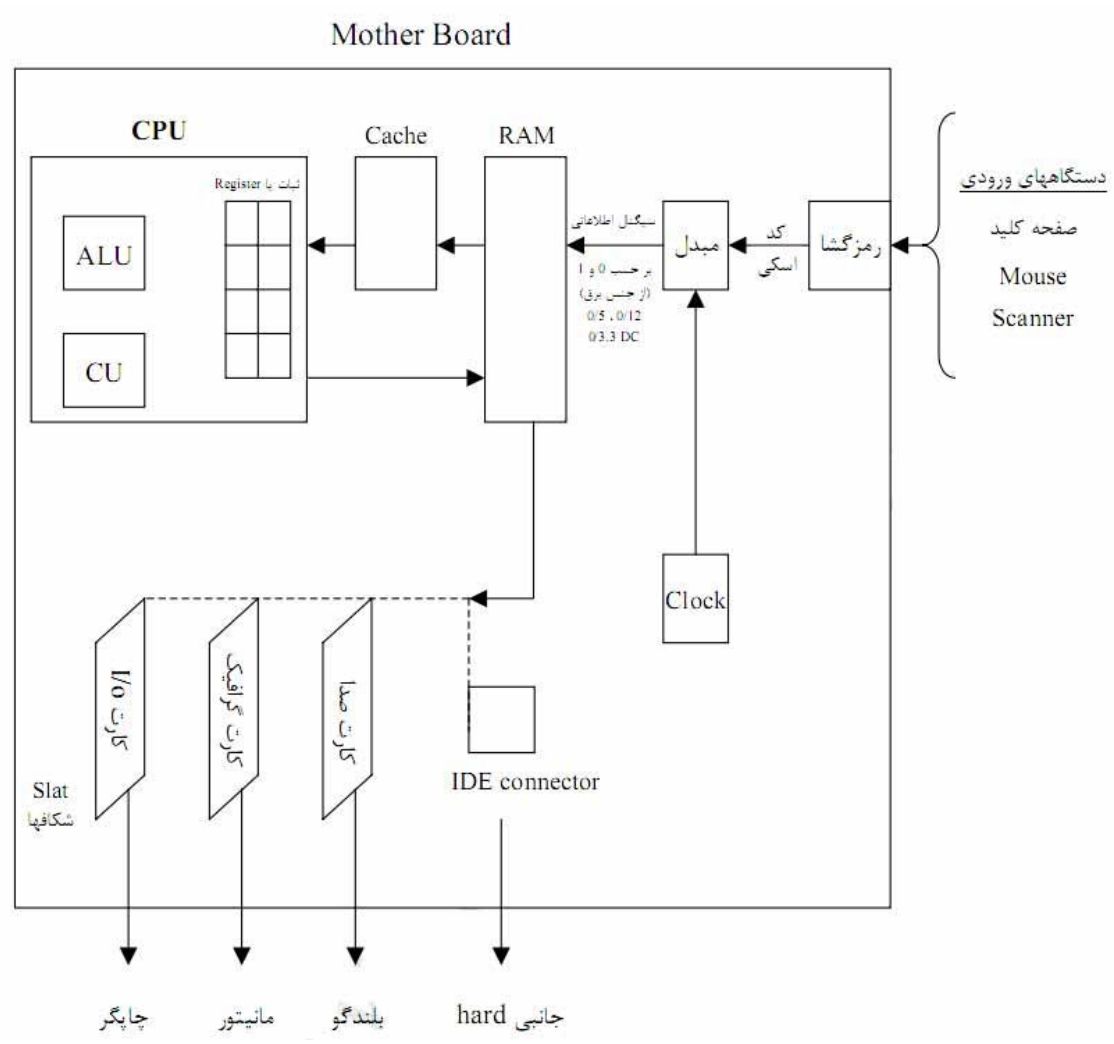
CISC-COMPLEX INSTRUCTION SET COMPUTER : پردازنده هایی که مجموعه دستورالعمل های کاملی با پشتیبانی سخت افزاری برای انواع وسیعی از عملیاتها را دارا می باشند .در عملیاتهای علمی و مهندسی و ریاضی معمولاً اکثر کارها را در کوتاهترین زمان انجام می دهند . پردازنده ۸۰۸۶ و ۸۰۸۸ و ۸۰۲۸۶ و ۸۰۳۸۶ از این نوع می باشد .

RISC-REDUCED INSTRUCTION SET COMPUTER : پردازنده هایی که مجموعه دستورالعمل فشرده و کوچکی دارند . در کاربردهای تجاری و برنامه هایی که توسط کامپایلر ایجاد می شود معمولاً اکثر کارها را در کوتاهترین زمان (یک پالس) انجام می دهند . اغلب پردازنده های کنترلی تک منظوره مانند میکروکنترولرها را شامل می شود .

HYBRID : ترکیبی از روش RISC و CISC را دارا هستند و سعی در برقراری تعادل بین هر دو روش دارند . ۸۰۴۸۶ و پنتیوم

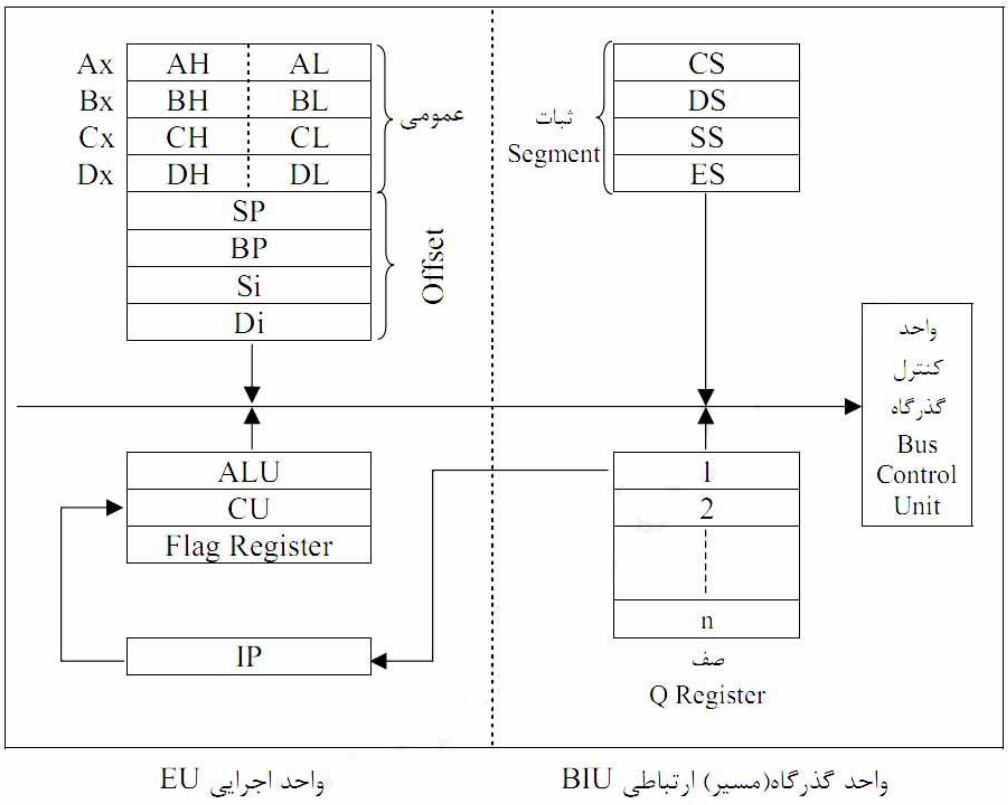
SPECIAL PURPOSE : پردازنده هایی که برای وظایف خاص بهینه شده اند . DSP و CO-PROCESSOR

سوال : آیا می دانید چرا سرعت انتقال داده در پردازنده ۸۰۸۶ بیشتر از پردازنده ۸۰۸۸ است ؟



فصل سوم

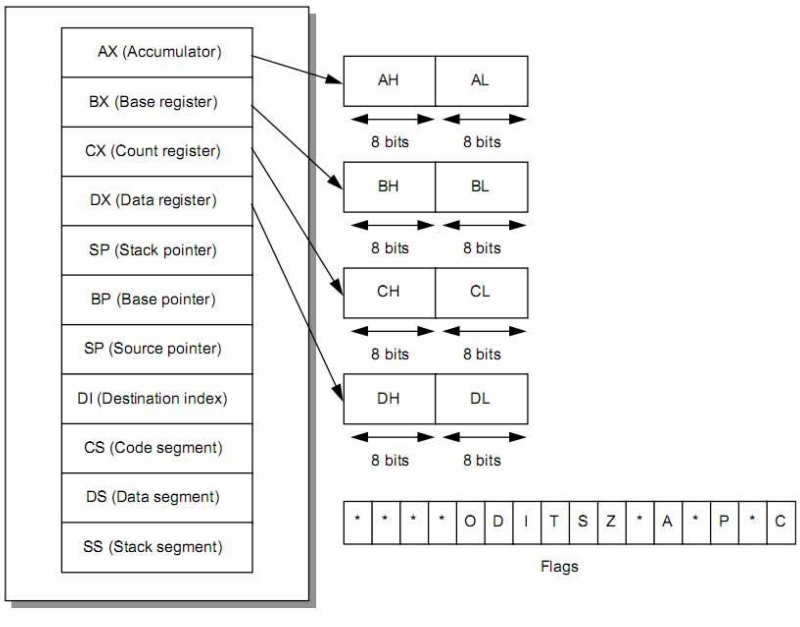
آشنایی با معماری پردازنده های 80X86



ثباتها :

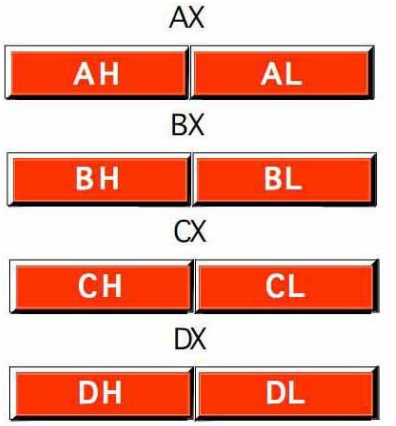
ثباتهای پردازنده ۸۰۸۶ به شش دسته تقسیم می شوند که همه این ثباتها شانزده بیتی هستند :

۱- ثباتهای عمومی (همه منظوره) AX , BX , CX , DX
۲- ثباتهای قطعه DS , CS , SS , ES
۳- ثباتهای اشاره گرها SP , BP
۴- ثبات دستور IP
۵- ثباتهای شاخص (اندیس) SI , DI
۶- پرچم FR

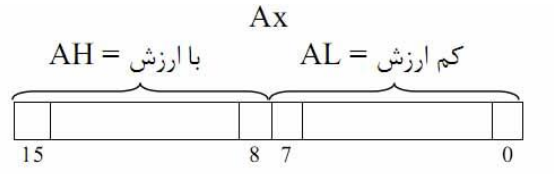


8086/88 registers

ثباتهای همه منظوره : این ثباتها برای مقاصد مختلفی بکار می روند . ویژگی منحصر به فرد این ثباتها قابلیت استفاده آنها بصورت ۸ بیتی و ۱۶ بیتی است . مثلاً ثبات ۱۶ بیتی AX شامل یک بخش AH (هشت بیت بالا) و AL (هشت بیت پایین) است .



ثبات AX (ACCUMULATOR) :این ثبات در دستورالعملهای محاسباتی و ورودی/ خروجی به عنوان ثبات نتیجه عملیات به کار می رود .



AX=39F7H ➔ AL=7FH , AH=39H

ثبات BX (BASE) : این ثبات جهت نگهداری آدرس پایه حافظه به کار می رود . کاربرد دیگر آن در انجام محاسبات است .

ثبات CX (COUNTER) : از این ثبات معمولاً برای شمارش دفعات تکرار یک حلقه و نیز محاسبات استفاده می شود .

ثبات DX (DATA) : از این ثبات در عملیاتهای ورودی/خروجی به عنوان آدرس پورت استفاده می شود .

نکته : قسمت BIU (واحد ارتباط گذرگاه) وظیفه دارد دستورالعمل ها و داده های مورد نظر واحد اجرا (EU) را فراهم کند . واحد اجرا مسئول اجرای دستورات است . این دو قسمت بصورت موازی کار می کنند یعنی زمانی که EU در حال اجرای دستور جاری باشد BIU دستور دیگری را از حافظه واکشی می کند .

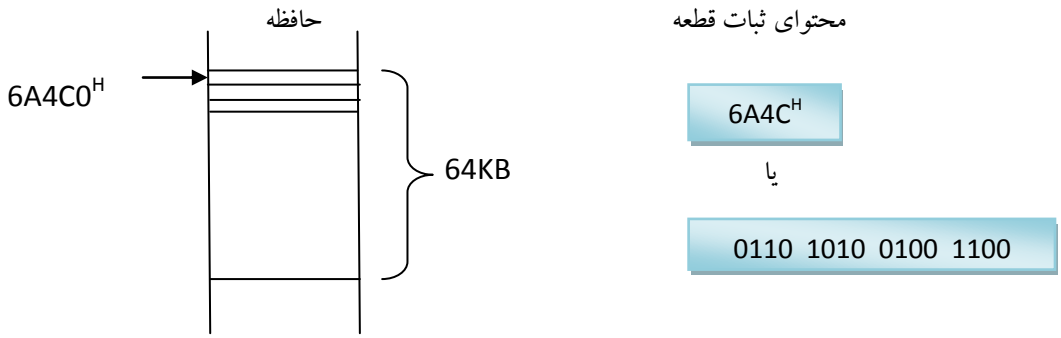
قطعه : هر برنامه اسمبلی حداقل از سه قطعه تشکیل شده است . قطعه کد که دستورات برنامه در آن قرار دارد ، قطعه داده که اطلاعات موجود در برنامه در آن قرار دارد و قطعه پشته که برای اطلاعات موقت بکار می رود . فرض کنید شما می خواهید عملیات ضرب زیر را انجام دهید : ابتدا ۵ را در ۷ ضرب کرده و از آنجایی که جواب آن ۳۵ می شود ۵ را نوشته و ۳ را در ذهن خود می سپارید سپس ۵ را در ۱ ضرب کرده و با ۳ ذهن جمع کرده و حاصل را که ۸ هست می نویسید . این الگو محاسبه در کامپیوتر به شکل قطعه بصورت زیر نوشته می شود :

۱۷

× ۵

قطعه کد	دستورات	× و + و =
قطعه داده	اطلاعات	۱۷ و ۵ و ۸۵
قطعه پشته	اطلاعات موقت	۳

تعریف محدوده قطعه : ناحیه ای از حافظه 64KB می باشد و از آدرس های قابل قسمت بر $(16)_{10}$ آغاز می شود (آدرس هایی که سمت راست آن صفر باشد) سایز قطعه 64KB می باشد زیرا ۸۰۸۵ به دلیل داشتن ۱۶ خط آدرس حداکثر تا 64KB حافظه فیزیکی را پشتیبانی می کرد و این محدودیت به طراحی های ۸۰۸۶ و ۸۰۸۸ نیز جهت سازگاری منتقل گشت . از آنجایی که سمت راست آدرس های شروع محدوده قطعه عدد صفر می باشد یعنی چهار بیت پایین آن صفر می باشد بنابراین نیازی به ذخیره آن در ثبات های قطعه که خود ۱۶ بیتی هستند نیست .



همانطور که در شکل فوق دیده می شود آدرس ابتداء قطعه در ثبات قطعه ذخیره می شود .

نکته : از آنجایی که ۸۰۸۶ دارای بیست خط آدرس می باشد بنابراین حداکثر تا 1MB را پشتیبانی می کند این مقدار در واقع بیشترین حافظه قابل دسترس فیزیکی می باشد .

وظیفه ثبات CS : محل نگهداری آدرس ابتدای قطعه کد

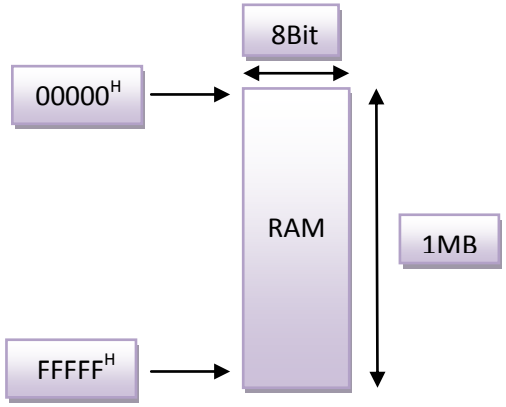
وظیفه ثبات DS : محل نگهداری آدرس ابتدای قطعه داده

وظیفه ثبات SS : محل نگهداری آدرس ابتدای قطعه پشته

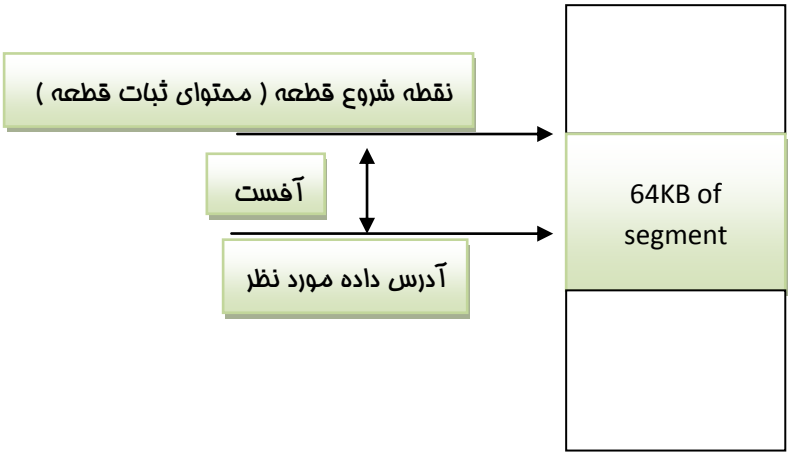
وظیفه ثبات ES : محل نگهداری آدرس ابتدای قطعه اضافی

انواع آدرس : در کاتالوگ ۸۰۸۶ سه نوع آدرس ذکر شده است : آدرس فیزیکی ، آدرس تفاوت مکان ، آدرس منطقی

آدرس فیزیکی : یک آدرس بیست بیتی است که دقیقاً روی پایه های گذرگاه آدرس ۸۰۸۶ قرار دارد . محدوده آن از 00000^H الی $FFFFF^H$ می باشد (1MB)



آدرس تفاوت مکان (آفست) : این آدرس در محدوده 64KB قطعه قرار داشته و یک آدرس ۱۶ بیتی می باشد . معمولاً فاصله نقطه شروع قطعه تا مکانی که داده مورد نظر درآنجا قرار دارد را آفست گویند . محدوده آن از 0000^H الی $FFFF^H$ می باشد .



آدرس منطقی : جهت مشخص کردن یک خانه حافظه همزمان می بایست از یک جفت ثبات سگمنت و آفست استفاده نمود . از آنجایی که ثباتهای قطعه در ۸۰۸۶ شانزده بیتی می باشند و محدوده حافظه 1MB (به دلیل داشتن ۲۰ خط آدرس) بنابراین جهت اشاره به یک مکان بیست بیتی به کمک ثباتهای شانزده بیتی از آدرس منطقی استفاده می شود . آدرس منطقی در هر قطعه ای جداگانه تعریف می شود اما روال کلی یک آدرس منطقی بصورت زیر است :

آدرس تفاوت مکان : محتوای ثبات قطعه

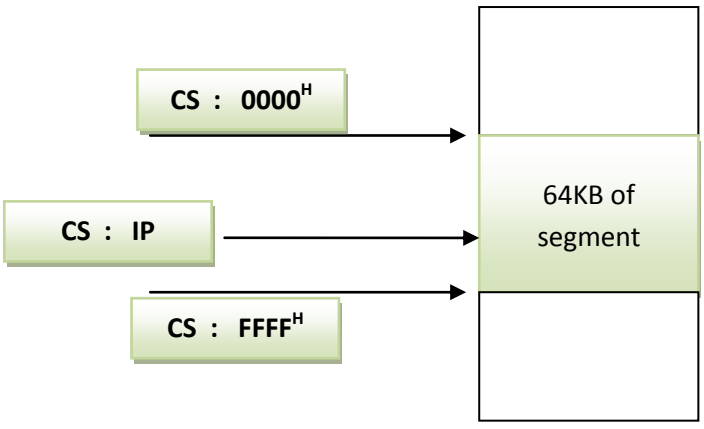
به عنوان نمونه ادرس منطقی در قطعه کد بصورت زیر است :

محتوای قطعه کد : **CS**

تفاوت مکان : **IP**

CS : IP

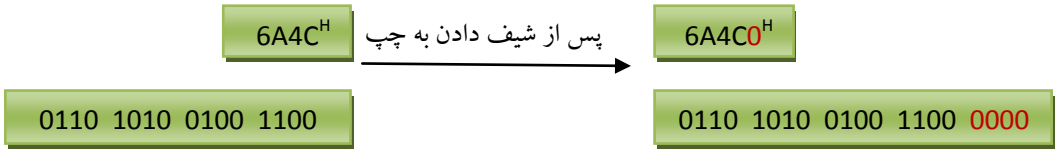
نکته : از انجایی که تفاوت مکان در محدوده 64KB قطعه قرار دارد بنابراین محدوده آدرس منطقی از CS : 0000 الی CS : FFFF قرار دارد .



بدست آوردن آدرس فیزیکی از روی آدرس منطقی :

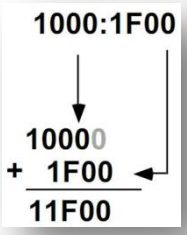
حال نکته مهم آن است که ۸۰۸۶ چگونه می تواند به کمک یک آدرس ۱۶ بیتی ، مکان یک داده ۲۰ بیتی را بیابد . فرض کنید که داده مورد نظر در قطعه کد قرار دارد (سایر قطعه ها نیز به شیوه مشابه بدست می آید)

۱- محتوای CS به سمت چپ شیفت پیدا کند با این کار آن چهار بیت پایین که صفر بود و در ثبات قطعه ذخیره نشده بود به محل خود باز می گردد . مثلاً فرض کنید محتوای قطعه کد برابر $6A4C^H$ باشد نتیجه به صورت زیر است :



۲- سپس نتیجه بدست آمده را با محتوای تفاوت مکان IP جمع می کنیم . حاصل یک آدرس ۲۰ بیتی بدست می آید .

نکته : برای بدست آوردن محدوده آدرس فیزیکی کافیست نتیجه بدست آمده در مرحله ۱ را با 0000^H (محدوده پایین) و با $FFFF^H$ (محدوده بالا) جمع کنیم . و یا به مثال زیر نگاه کنید :



مثال : با توجه به آدرس منطقی 2500 : 95F3 موارد زیر را بدست آورید :

الف) آدرس فیزیکی را بدست آورید .

ب) محدوده آدرس فیزیکی را بدست آورید .

حل : الف)

25000
+ 95F3
2E5F3

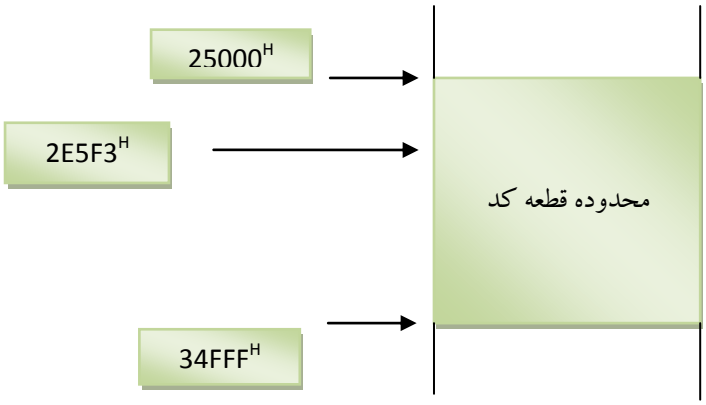
ب)

محدوده پایین آدرس فیزیکی :

25000
+0000
25000

محدوده بالای آدرس فیزیکی :

25000
+FFFF
34FFF



MEMORY

مثال : اگر محتوای قطعه کد $204E^H$ و آدرس تفاوت مکان $2F0A^H$ باشد موارد خواسته شده را بدست آورید :

الف) آدرس منطقی را بدست آورید .

ب) آدرس فیزیکی را بدست آورید .

ج) محدوده آدرس منطقی را بدست آورید .

د) محدوده آدرس فیزیکی را بدست آورید .

حل : الف)

$204E^H : 2F0A^H$

ب)

$204E0^H + 2F0A^H = 233EA^H$

ج)

محدوده بالا $204E^H : FFFF^H$ محدوده پایین $204E^H : 0000^H$

د)

محدوده بالا $204E0^H + FFFF^H = 304DF^H$ محدوده پایین $204E0^H + 0000^H = 204E0^H$

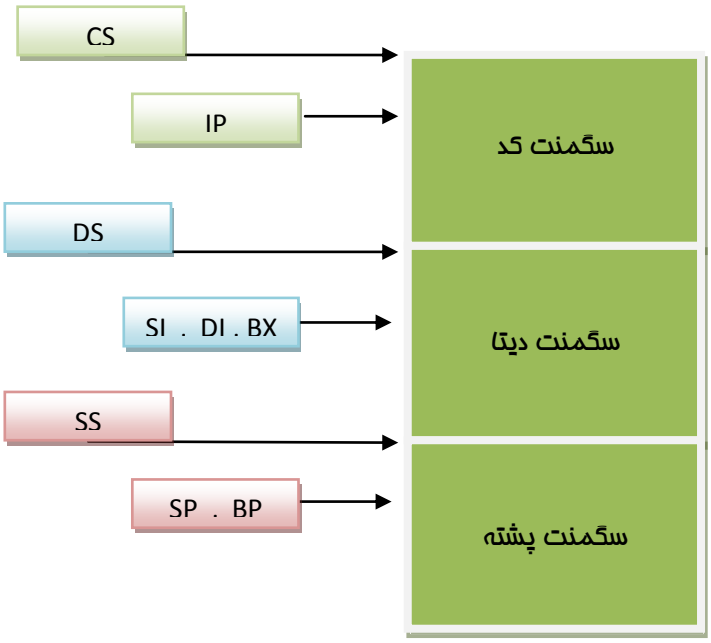
تمرین : مثال فوق را با $CS=24F6^H$ و $IP=634A^H$ تکرار کنید .

اشاره گرها POINTER : ثباتهای اشاره گر ۱۶ بیتی نگهدارنده بخش آفست در آدرس دهی هستند و همراه یکی از ثباتهای قطعه به محلی از حافظه اشاره می کنند . (قبلاً دیدیم که ثبات IP با ثبات CS اشاره به یک آدرس منطقی یک دستور در حافظه می کند .)

IP : همراه با ثبات CS به دستورالعمل بعدی که بایستی توسط پردازنده اجرا شود اشاره می کند .

SP : آفست مکانی از محدوده قطعه پشته است که عمل قرار گرفتن داده در پشته صورت می گیرد . SS : SP

BP : برای دسترسی به متغیرهایی که در پشته قرار دارند استفاده می شود .

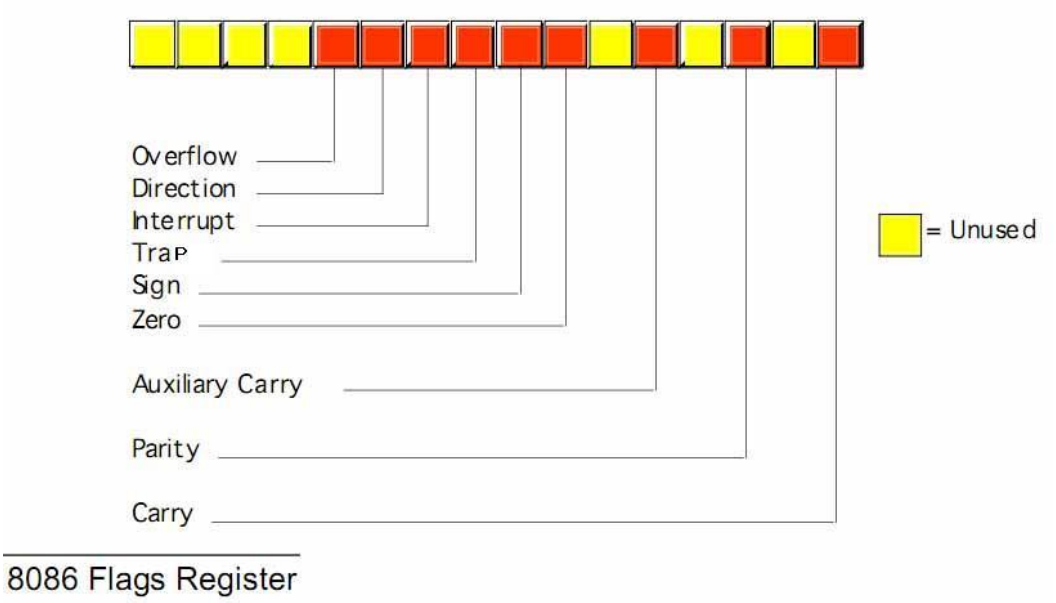


ثبات های شاخص INDEX : این دو ثبات ۱۶ بیتی اغلب به عنوان اشاره گر به همراه DS به کار می روند تا به داده های موجود در محدوده قطعه داده دسترسی شود . اما به منظورهای دیگری نیز استفاده می شوند مثلاً اگرچه نمی توانند به دو بخش هشت بیتی تجزیه شوند اما می توانند مانند ثباتهای همه منظوره به کار روند . (آفست قطعه اضافی ثبات DI می باشد)

SI : برای آدرس دهی و در عملیاتهای رشته ای به عنوان مبداء استفاده می شوند .

DI : برای آدرس دهی و در عملیات های رشته ای بعنوان مقصد استفاده می شوند .

ثبات پرچم FLAG REGISTER : بیتهای این ثبات وضعیت CPU را بعد از انجام یک عمل نشان می دهند .



رقم نقلی C (Carry) : در عملیاتهای محاسباتی و همینطور چرخش و شیفت این بیت جهت ذخیره رقم نقلی به کار می رود . (با ارزش ترین رقم نقلی در محاسبات ذخیره می شود) در واقع اگر نتیجه یک عمل محاسباتی بدون علامت آنقدر بزرگ باشد که در مقصد جا نشود این پرچم برابر یک خواهد شد .

توازن P (Parity) : این پرچم توازن بایت اول (۸ بیت کم ارزش) را چک می کند . اگر پس از انجام عملی تعداد زوجی از یک ها وجود داشته باشد این پرچم یک می شود (توازن فرد) در واقع این پرچم مخصوص تست انتقال دیتا می باشد .

رقم نقلی کمکی A (Auxiliary Carry) : در انجام عملیات ریاضی به صورت BCD اگر رقم نقلی از بیت d3 به d4 وجود داشته باشد این پرچم یک و در غیر این صورت صفر است .

پرچم صفر Z (Zero) : هرگاه نتیجه یک عملیات حسابی یا منطقی صفر گردد این پرچم یک می شود و در غیر این صورت صفر است .

علامت S (Sign) : این بیت مستقیماً به بیت پرارزش نتیجه عملیات (MSB) حاصل عملیات متصل است اگر نتیجه منفی باشد این پرچم برابر یک و اگر نتیجه عملیات مثبت باشد برابر صفر است .

تله T (Trap/Trace) : این پرچم برای اجرای برنامه بصورت گام به گام یا دستور به دستور کاربرد دارد . در واقع این پرچم مشخص می کند که آیا پردازنده پس از اجرای هر دستور متوقف می شود یا خیر . اگر این بیت ۱ باشد برنامه به صورت گام به گام اجرا می شود و اگر صفر باشد بصورت طبیعی اجرا می شود .

وقفه I (Interrupt) : جهت تشخص وقفه کاربرد دارد . اگر یک باشد فعال و اگر صفر باشد وقفه غیر فعال است .

جهت D (Direction) : جهت کنترل عملیاتهای رشته ای از قبیل انتقال یا مقایسه کاربرد دارد . اگر یک باشد عمل انتقال یا مقایسه از راست به چپ یا از پایین به بالا است و اگر صفر باشد عمل انتقال یا مقایسه از چپ به راست یا از بالا به پایین است .

سرریز O (Overflow) : اگر در حاصل یک عملیات (علامت دار) نتیجه ای بیش از ظرفیت بدست آید سرریز رخ داده است و این پرچم یک می شود .

مثلاً در عملیات جمع خطای سرریز زمانی رخ می دهد که علامت حاصلجمع دو عدد مثبت ، منفی شود و یا علامت حاصلجمع دو عدد منفی ، مثبت شود . برای برطرف کردن این خطا تعداد بیتها را افزایش می دهند مثلاً بجای هشت بیت از ده بیت استفاده می شود .در جمع دو عدد مختلف العلامت سرریز رخ نمی دهد .

نکته مهم : تفاوت بین نقلی و سر ریز $OF=CF\oplus Cin$

CF : نقلی خارج شده از بیت علامت Cin : نقلی وارد شده به بیت علامت

اگر OF=1 عدد در محدوده علامتدار نادرست است و اگر CF=1 عدد در محدوده بدون علامت نادرست است .

	علامتدار	بدون علامت	
از آنجایی که هم CF و هم OF برابر صفر است نتیجه در محدوده علامتدار وبی علامت صحیح است .	+4 -5 -1	4 +251 255	0000 0100 +1111 1011 1111 1111 CF=0,Cin=0 ➔ OF=0
OF=0 فقط محدوده علامتدار صحیح است .	-4 + 5 +1	252 + 5 1	1111 1100 +0000 0101 0000 0001 CF=1,Cin=1 ➔ OF=0
CF=0 فقط محدوده بدون علامت صحیح است .	+121 + 11 -124	121 + 11 132	0111 1001 +0000 1011 1000 0100 CF=0,Cin=1 ➔ OF=1
از آنجایی که هم CF و هم OF برابر یک است هر دو نتیجه در محدوده علامتدار وبی علامت نادرست است .	-10 -119 +127	246 +132 127	1111 0110 +1000 1001 0111 1111 CF=1,Cin=0 ➔ OF=1

مثال : ضمن بدست آوردن حاصل جمع های زیر مقادیر ثبات پرچم را نشان دهید .

$38^H + 2F^H =$

$0011\ 1000+0010\ 1111 = 0110\ 0111 \longrightarrow 67^H$

CF=0 PF=0 AF=1 ZF=0 SF=0 OF=0

$2345^H + 3219^H =$

$0010\ 0011\ 0100\ 0101+0011\ 0010\ 001\ 1001=0101\ 0101\ 0101\ 1110 \longrightarrow 555E^H$

CF=0 PF=0 AF=0 ZF=0 SF=0 OF=0

صف دستورات INSTRUCTION QUEUE : در ریزپردازنده ها یک صف دستورالعمل وجود دارد که طول آن برای پردازند های اولیه محدود بود و به تدریج در پردازنده های بعدی افزایش یافت که نهایتاً منجر به ایجاد CACHE گردید . (در ۸۰۸۶ طول صف دستور ۴ (بایت)دستور می باشد) از این صف برای ذخیره تعدادی دستورالعمل که از حافظه به ریز پردازنده منتقل می شوند استفاده می گردد .(به منظور افزایش سرعت CPU) در پردازنده های ۸۰۸۵ و ماقبل آن برای اجرای یک برنامه پردازنده می بایست دستور را از حافظه دریافت آنرا اجرا و سپس مجدداً به حافظه مراجعه کرده و دستور بعدی را به پردازنده منقل نماید و این عمل بصورت پشت سرهم انجام می شد . اما در پردازنده ۸۰۸۶ با ایجاد صف دستور همزمان با اجرای یک دستور (بطور موازی) تا ۴ دستور بعدی نیز از حافظه دریافت و در این صف ذخیره می شد .(FIFO) با این کار پردازنده برای اجرا مجبور نبود هربار به حافظه مراجعه کند ودر نتیجه زمان کلی اجرا کاهش و سرعت اجرا بالاتر خواهد رفت . بنابراین دو نوع واکشی مختلف وجود دارد . واکشی دستور از حافظه توسط BIU و واکشی دستور از صف توسط EU که می تواند همزمان انجام شود

پشته STACK

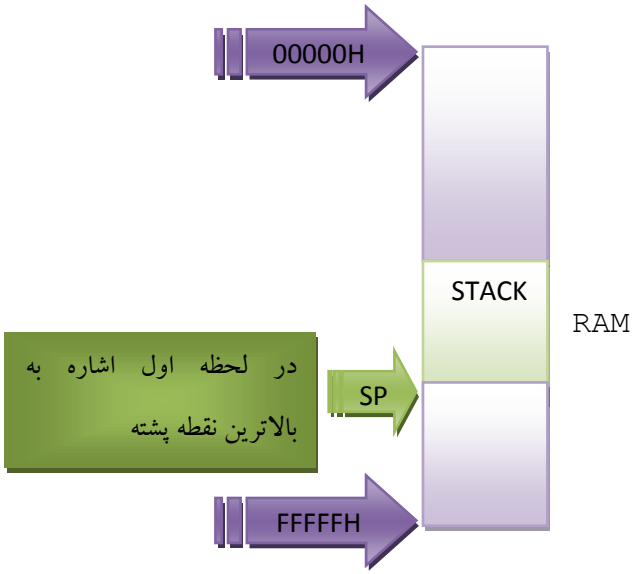
هر ثبات در ۸۰۸۶ (بجز ثباتهای قطعه و SP) قابل ذخیره سازی در پشته و دریافت از آن می باشند .

PUSH : ذخیره در پشته (درج)

POP : بار کردن (دریافت) از پشته (بازیافت)

پشته بصورت LIFO است یعنی آخرین داده ای که در آن قرار گرفته اول خوانده می شود مانند چند کتاب که روی هم قرار گرفته شده باشد . اطلاعات در حافظه پشته بصورت یک کلمه یک کلمه (دو بایت دو بایت) نوشته می شود و یا از آن خوانده می شود .

در ۸۰۸۶ ثبات SP به مکان حافظه جاری بکار رفته در بالای پشته اشاره می کند و به محض درج داده کاهش خواهد یافت (۲ واحد کاهش می یابد) و برعکس هنگام بازیافت این اشاره گر افزایش می یابد (۲ واحد افزایش)

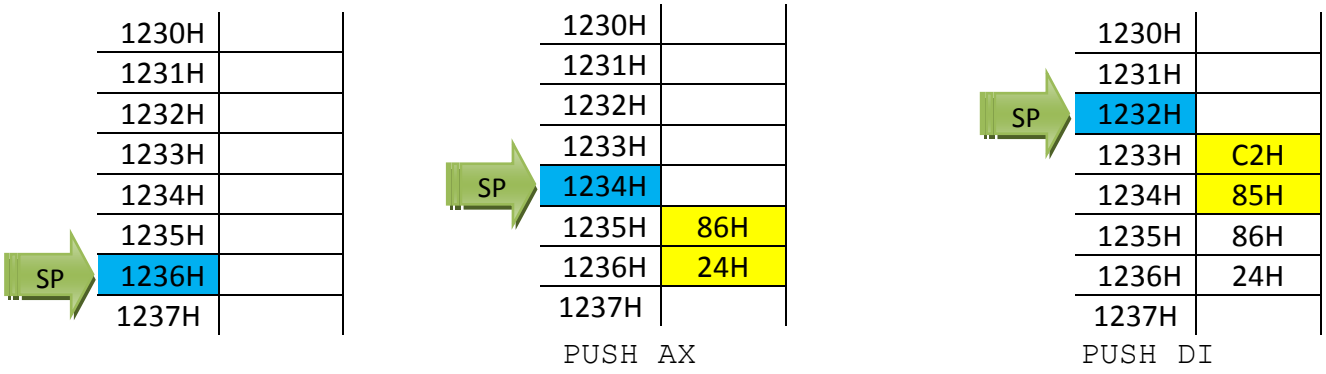


برای آنکه سگمنت کد و سگمنت پشته همپوشانی نداشته باشند (در غیر اینصورت برنامه به هم میریزد) آنها را در دو انتهای مخالف هم در RAM قرار می دهند (البته سگمنت پشته و دیتا می توانند همپوشانی داشته باشند)

مثال : با فرض AX=2486H, DX=5F93H, DI=85C2H, SP=1236H با اجرای دستورات زیر محتوای پشته و اشاره گر آن و همچنین محتوای ثبات BX و پرچم را نشان دهید .

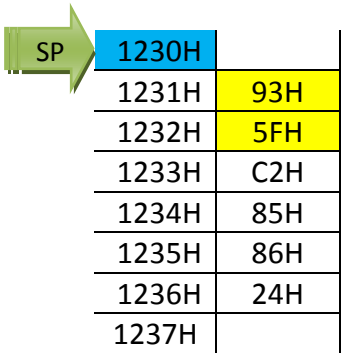
```
PUSH AX
PUSH DI
PUSH DX
POP BX
POPF
```

جواب :

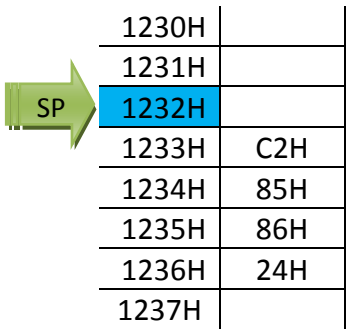


PUSH DI

PUSH AX

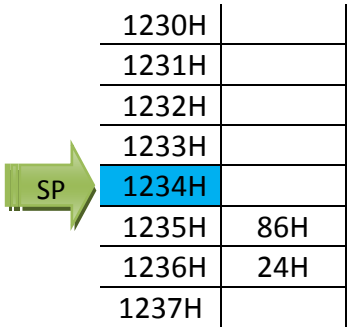


PUSH DX



POP BX

BX=5F93H



POPF

FR=85C2H

- دستور PUSH با اعداد ثابت فقط در 286 به بعد قابل استفاده است .
- دستور PUSHA (PUSH ALL) تمامی ۸ ثبات (AX, BX, CX, DX, SI, DI, BP, SP) را در پشته ذخیره می کند و POPA آنها را بازیافت می کند . این دو دستور نیز فقط در 286 به بعد قابل استفاده هستند .

تمرین های فصل ۳ :

۱- کدام یک از ثباتهای زیر نمی توانند به دو بایت بالا و پایین تقسیم شوند .

الف) CS ب) AX پ) DS ت) SS ث) BX ج) DX چ) SI

۲- اگر CS=1298H و IP=7CC8H باشد :

الف) آدرس منطقی ب) آدرس فیزیکی ج) محدوده بالا و پایین آدرس منطقی

د) محدوده بالا و پایین آدرس فیزیکی

۳- اگر SS=2000H و SP=4578H باشد ، پیدا کنید :

الف) آدرس فیزیکی ب) آدرس منطقی پ) محدوده بالا و پایین قطعه پشته

۴- وضعیت CF,PF,AF,ZF,SF را برای اعمال زیر مشخص کنید .

الف) MOV BL,9F^{H} ب) MOV DX,10FF^{H}

ADD BL,61^{H} ADD DX,1^{H}

۵- ضمن بدست آوردن حاصل جمع های زیر وضعیت ثبات پرچم را در هر یک از حالات زیر نشان دهید .

5439^{H}+456A^{H}=

FFCA^{H}+AA0B^{H}=

9C^{H}+64^{H}=

سوالات چهار گزینه ای :

۱- چنانچه $CS=5FC2^H$ و $IP=495B^H$ باشد آدرس فیزیکی در IP+50 چیست ؟

۱) $645D^H$ ۲) $645C^H$ ۳) 6457^H ۴) هیچکدام

۲- ترکیب آدرس 5678 : 1234 معادل کدامیک از آدرس های زیر است ؟

۱) 1240 :557C ۲) 1230:567B ۳) 1236:5666 ۴) 1233:5688

۳- در ریزپردازنده ۸۰۸۶ آخرین آدرس در محدوده 640K چه آدرسی است ؟

۱) 654720 ۲) 655360 ۳) 655359 ۴) به نحوه قرار گرفتن فایلهای سیستم در حافظه بستگی دارد .

۴- آدرس پنج رقمی 1F558H معادل کدام آدرس زیر است ؟

الف) 1055:1018 ب) 1E00:5581 ج) 1E55:1008 د) 18A3:5B28

فصل ۴

اصول برنامه نویسی اسمبلی

معمولاً هر برنامه زبان اسمبلی از تعدادی دستورالعمل ساخته شده است که بیانگر عملیاتی است که بایستی انجام شود . مجموعه دستور العملهای یک میکروپروسسور لیست تمام فرمانها یی است که CPU می تواند تشخیص دهد و اجرا کند.

شبه دستورات : اسمبلر دارای فرمانهایی است که کاربر را در کنترل ترجمه و تهیه لیست های برنامه یاری می کند . این فرمان ها به شبه دستورات معروف هستند که کد زبان ماشین تولید نمی کنند .

شبه دستور PAGE :

PAGE [length] , [width]

زمانی که length از برنامه اسمبل شده و لیست گردید ، کنترل لیست گرفتن به بالای صفحه بعد منتقل و یکی به شماره صفحه اضافه می شود . اگر جلوی PAGE عددی ننویسیم لیست اسمبل شده بطور خودکار پس از مواجه شدن با PAGE به صفحه جدید منتقل می شود .

شبه دستور TITLE : جهت قرار دادن یک نام برای برنامه استفاده می شود . (حداکثر ۶۰ حرف)

TITLE ' اسم برنامه .ASM '

شبه دستور END : نقطه انتهای یک برنامه یا یک سگمنت را برای اسمبلر مشخص می کند .

شبه دستور ORG : جهت مشخص کردن شروع آدرس تفاوت مکان بکار می رود .

انتخاب اسم : اسم با رقم شروع نمی شود و هرگاه از کاراکتر dat (.) در اسم استفاده شود این کاراکتر بایستی اولین کاراکتر اسم باشد . همچنین اسم نبایستی از کلمات ذخیره شده در اسمبلی باشد .

انواع ثابت ها :

۱- باینری : شامل صفر و یک ها می باشد که در انتهای آن حرف B نوشته می شود .
۲- دسیمال : شامل ارقام صفر الی ۹ می باشد (اضافه کردن حرف D در انتهای آن اختیاری می باشد)
۳- هگزادسیمال : شامل ارقام ۰ الی ۹ و حروف A الی F که در انتهای آن حرف H نوشته می شود .
۴- اکتاو : شامل ارقام ۰ الی ۷ می باشد که در انتهای آن حرف O نوشته می شود . (به جای O از Q نیز استفاده می شود)
۵- کاراکتر : شامل هر کاراکتراز کدهای اسکی می باشد که بین علامت نقل قول ' یا " قرار می گیرد .
۶- ممیز شناور Floating Point : جهت نمایش اعشاری که بصورت نمایی نوشته می شود بکار می رود .

0.26E-2

نکته : هرگاه مقداری در مبناء شانزده با حروف A تا F شروع شود بایستی به اول آن یک 0 اضافه شود تا کامپیوتر آنرا با یک برچسب یا متغیر اشتباه نگیرد .

داده ها : معمولاً عرض یک داده برابر سایز ثباتهای داخلی پردازنده می باشد (در اینجا شانزده بیتی) شبه دستورهای داده ها برای همه خانواده 80x86 یکسان و استاندارد می باشد .

تعریف متغیرها : شامل آدرس ، نوع داده و اندازه آن می باشد . (آدرس اطلاعات در حافظه)

شبه دستور DB : به متغیر مورد نظر یک بایت اختصاص می دهد .

شبه دستور DW : به متغیر مورد نظر دو بایت اختصاص می دهد .

شبه دستور DD : به متغیر مورد نظر چهار بایت اختصاص می دهد .

شبه دستور DQ : به متغیر مورد نظر هشت بایت اختصاص می دهد .

شبه دستور DT : به متغیر مورد نظر ده بایت اختصاص می دهد .

DB تنها شبه دستوری است که می تواند برای تعریف یک رشته اسکی بزرگتر از ۲ کاراکتر مورد استفاده قرار گیرد و استفاده از شبه دستور DT,DQ,DD برای رشته اسکی بیش از ۲ بایت ، خطای اسمبلی تولید خواهد کرد . همچنین شبه دستور DT معمولاً جهت ذخیره بسته های BCD بکار می رود .

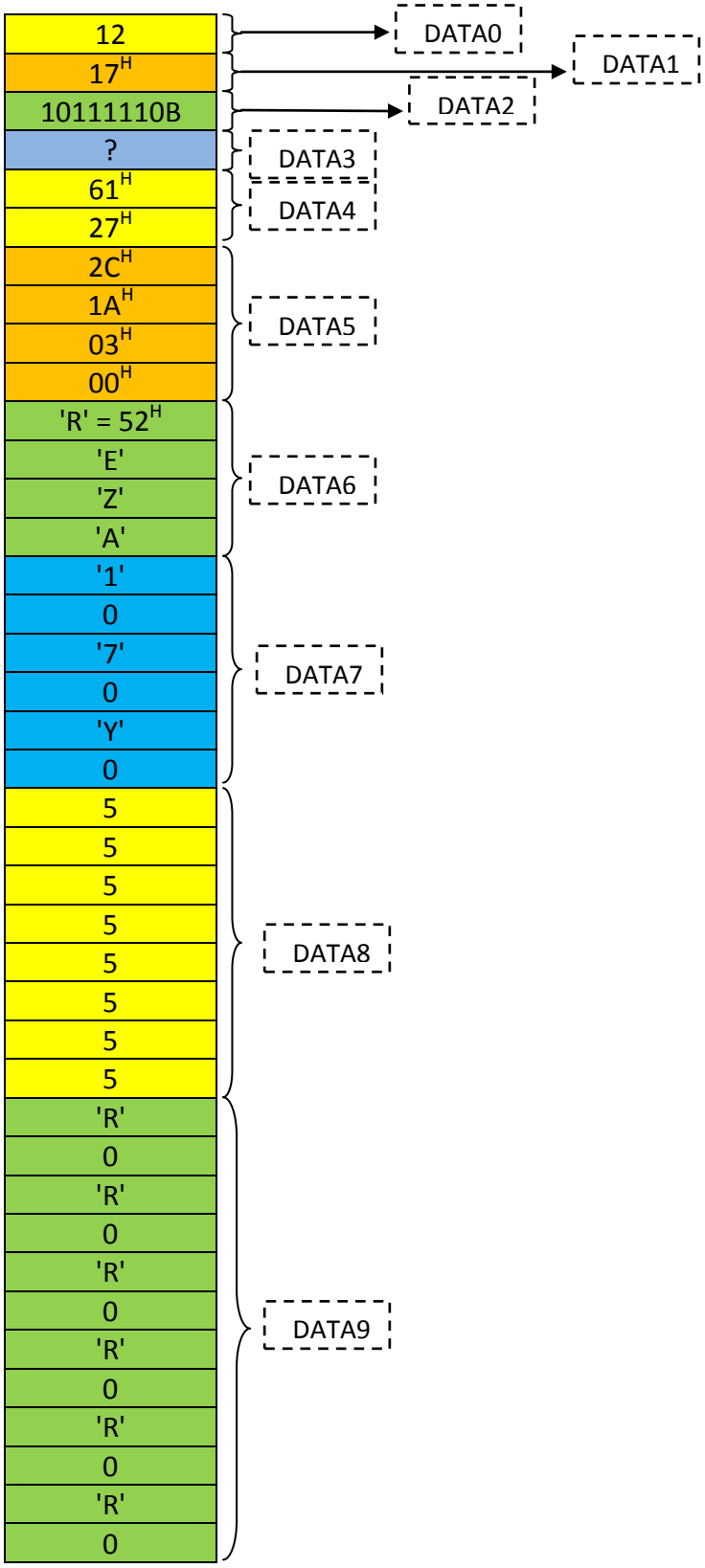
شبه دستور EQU : جهت قرار دادن یک مقدار ثابت به یک متغیر بدون اشغال مکان حافظه بکار می رود . در واقع این شبه دستور هیچ محلی را برای ذخیره کنار نمی گذارد . مزیت استفاده از این شبه دستور در این است که هرگاه مقدار ثابتی در چند محل مورد استفاده قرار گیرد با بکار بردن این شبه دستور می توان به یکباره آنها را تغییر داد و نیازی به تغییر تک تک آنها به تنهایی نیست .

شبه دستور DUP (کپی کردن) : با این شبه دستور می توان اطلاعات مساوی را در متغیرهای مختلف کپی نمود .

M1 DB 4^H, 4^H, 4^H, 4^H, 4^H M1 DB 5DUP(4^H)

مثال : ضمن رسم نقشه حافظه بنویسید که هر متغیر تعریف شده چند بایت حافظه اشغال می کند . به کل قطعه نوشته شده چند بایت حافظه اختصاص می یابد ؟

```
DATA0   DB   12
DATA1   DB   17H
DATA2   DB   10111110B
DATA3   DB   ?
DATA4   DW   2761H
DATA5   DD   31A2CH
DATA6   DB   ' REZA '
DATA7   DW   '1' , '7' , 'Y'
DATA8   DB   8DUP(5)
DATA9   DW   2DUP3DUP('R')
```



همانطور که دیده می شود برای کل متغیرها ۴۰ بایت درنظر گرفته شده است

به کار بردن اپراتورهای ریاضی در هنگام تعریف داده ها :

```
LIST DB 2*25 DUP(?)    ;50 BYTE
L1   DB 53/50          ;L1=5
X    DB 53 MOD 10      ;X=3
```

/ تقسیم صحیح و MOD باقیمانده است .

قالب کلی هر دستورالعمل زبان اسمبلی :

[lable] op-code operand ; comment

برچسب : معمولاً از برچسب به عنوان آدرس دستورالعمل در برنامه استفاده می شود . (آدرس دستور در حافظه)

برخی از انواع برچسب :

۱- برچسب دستور

```
BACK :
       MOV AL,3C^H
       .
       .
       LOOP BACK
```

۲- برچسب متغیر

```
ALI DB 85
DATA DW ?
```

۳- برچسب روال

```
START PROC FAR
.
.
START ENDP
```

۴- برچسب قطعه

```
CDSEG SEGMENT
.
.
CDSEG ENDS
```

کد دستور : این قسمت حاوی عملی است که پردازنده بایستی آنرا انجام دهد .

عملوند : این قسمت حاوی اطلاعات مورد نیاز کد عمل می باشد . برخی از دستورات دارای دو عملوند و برخی شامل یک عملوند و برخی نیز بدون عملوند هستند . در شرایطی که دستور شامل دو عملوند باشد عملوند اول را مقصد و دومی را مبداء گویند .


```
MOV     AX,BX
ADD     BL,CH
```

توضیحات : این قسمت شامل توضیحات مورد نیاز دستورالعمل یا برنامه می باشد که بوسیله ; جدا می شود .

```
MOV AX,DATA1          ;move the first word into AX
```

روشهای آدرس دهی :

روش دست یافتن پردازنده به عملوند را آدرس دهی گویند .

۱- آدرس دهی فوری : در این روش عملوند مبداء یک مقدار عددی ثابت است .

```
MOV     AL,20
```

نکته : جهت ثباتهای قطعه و پرچم از این روش آدرس دهی استفاده نمی شود .

۲- آدرس دهی ثباتی : این روش مربوط به عملیاتهای ثباتهای داخلی ریز پردازنده می باشد . (سرعت آن از سایر روشهای آدرس دهی بالاتر است)

```
MOV     BX,CX
MOV     CL,DH
```

۳- آدرس دهی مستقیم : در این روش عملوند مبداء یا داده مورد نظر در حافظه وجود دارد و آدرس آن داده موجود است .

```
MOV     AX,[2000H]
MOV     CX,DATA3
```

نکته : ثباتهای شاخص SI , DI آدرس دهی مستقیم نمی شوند .

۴- آدرس دهی غیر مستقیم : در این روش آدرس داده مورد نظر در خود دستور نیست بلکه در یکی از ثباتهای SI , DI , BP , BX است .

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

۵- آدرس دهی غیر مستقیم نسبی پایه : در این روش آدرس موثر اطلاعات در حافظه برابر محتوای یکی از ثباتهای پایه BP , BX بعلاوه مقداری جابجایی قرار دارد .

```
MOV     AX,[BX+5]
MOV     AX,[BX]+5
MOV     AX,5 [BX]
```

۶- آدرس دهی غیر مستقیم نسبی شاخص (اندیس) : در این روش آدرس موثر اطلاعات در حافظه برابر محتوای یکی از ثباتهای شاخص SI , DI بعلاوه مقداری جابجایی قرار دارد .

```
MOV   AX,[SI+3]
```

۷- آدرس دهی غیر مستقیم نسبی پایه و شاخص : در این روش آدرس موثر اطلاعات در حافظه برابر مجموع محتوای یکی از ثباتهای شاخص SI , DI بعلاوه محتوای یکی از ثباتهای پایه BP , BX بعلاوه مقداری جابجایی قرار دارد .

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

۸- آدرس دهی ضمنی : به دستوراتی که به طور مشخص به یک آدرس اطلاعات یا عملوند اشاره نمی کنند گفته می شود .

```
STC
NOP
CLD
```

تمرین فصل ۴ :

۱- شبه دستور DT چیست و معمولاً برای چه منظوری بکار می رود ؟

۲- ضمن رسم نقشه حافظه جهت متغیرهای زیر با توجه به دستورالعمل های ذیل چند بایت حافظه اشغال می شود ؟

```
X   DB   'PLEASEWAIT'
Y   DW   4DUP(?)
Z   DD   35000,42000
```

۳- به دستورالعمل ذیل چند بایت حافظه اختصاص می یابد .

```
TEMP   DB   20DUP(2DUP('*'),3DUP('1'))
```

۴- به دستورالعمل ذیل چند بایت حافظه اختصاص می یابد .

```
T5 DB 10DUP(5DUP(0),6,7,8,9,10)
```

۵- روشهای آدرس دهی را برای هر یک از موارد زیر تعیین کنید :

الف) MOV [BP+6] , AL

ب) MOV DX , [BP+DI+4]

پ) MOV [DI] , BX

ت) MOV CX , $[3000^H]$

ث) MOV BX , 5678^H

ج) MOV AL , CH

چ) MOV AL , [BX]

ح) MOV CX , DS

خ) MOV BL , [SI]+10

د) MOV [BP][SI]+12 , AX

۶- در مورد آدرس شروع SEGMENT کدام گزینه صحیح است ؟

الف) از هر آدرسی در حافظه شروع می شوند .

ب) از آدرسهای قابل قسمت بر ۱۶ شروع می شوند .

ج) از آدرسهای قابل قسمت بر ۸ شروع می شوند .

د) بایستی آدرس فرد باشد .

فصل ۵

دستورات زبان اسمبلی

اشکال مختلف دستورالعملها :

۱- انتقال اطلاعات ۲- محاسبات ریاضی ۳- مقایسه و پرش ۴- محاسبات منطقی ۵- ورودی و خروجی ۶- کنترل پرچم و توقف کامپیوتر ۷- زیر برنامه

۱- انتقال اطلاعات :

دستور MOV :

MOV مبداء , مقصد

با اجرای این دستور محتوای عملوند مبداء به عملوند مقصد منتقل خواهد شد .

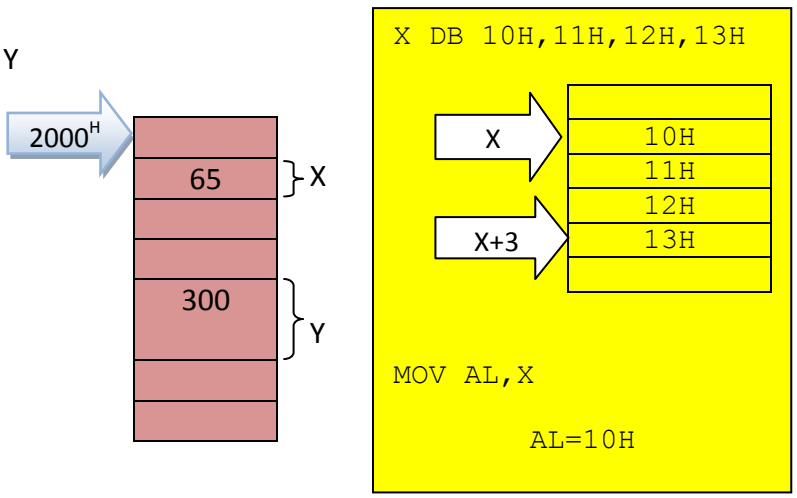
- هر دو عملوند بایستی از نوع بایت یا از نوع کلمه باشند .
- هر دو عملوند نمی توانند متغیر باشند .
- هیچکدام از عملوندها نمی توانند ثبات های IP و FR باشند .
- محتویات دو ثبات قطعه را نمی توان مستقیماً به همدیگر منتقل نمود .

```
MOV  CL , -50
MOV  X , 34H
MOV  AX , DATA1
MOV  ALI , BX
```

در مثال زیر دو دستور نوشته شده است . با توجه به نقشه حافظه نتیجه هر دستور چیست ؟

```
MOV  BX , OFFSET_Y

MOV  BX , Y
```



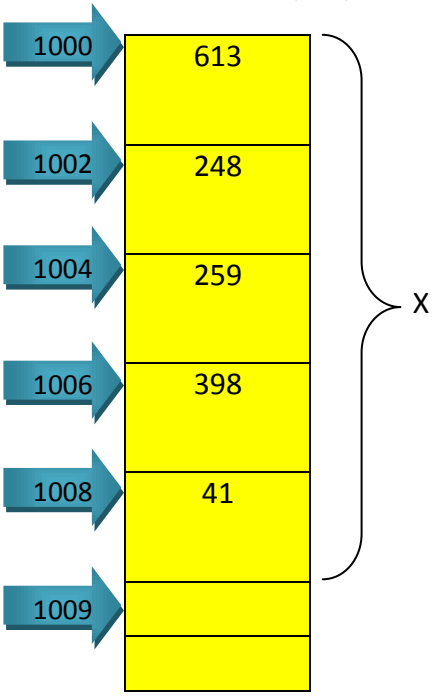
دستور اول آدرس متغیر Y را به ثبات BX منقل می کند . از آنجایی که متغیر Y دو بایت از حافظه را اشغال کرده است بنابراین آدرس آن برابر 2004^H خواهد بود . $BX=2004^H$

دستور دوم محتوای متغیر Y را به ثبات BX منقل می کند . بنابراین BX=300

مثال : با توجه به قطعه برنامه زیر محتوای AX و BX را پس از اجرای دستورات زیر بنویسید .

```
ORG   1000
X   DW   613 , 248 , 259 , 398 , 41
MOV   BX , OFFSET_X
MOV  AX , [BX]+4
MOV   DX , X+3
```

جواب : در خط اول آفست یا نقطه شروع از آدرس 1000 شروع شده است . در خط دوم یک متغیر به نام X تعریف شده است با ۵ عدد دو بایتی که اگر بخواهیم برای این متغیر نقشه حافظه را رسم کنیم بصورت زیر می شود : (البته در بسیاری از موارد رسم نقشه حافظه لازم نیست)



در خط سوم آدرس متغیر X به ثبات BX منتقل می شود بنابراین : BX = 1000

در خط چهارم یک روش آدرس دهی غیر مستقیم نسبی پایه مشاهده می کنید . در این دستور محتوای آدرس 1000+4 به ثبات AX منتقل می شود . در آدرس 1004 عدد 259 ذخیره شده است بنابراین : AX = 259

DX=398

مثال : نتیجه برنامه زیر چیست ؟

```
TABLE DB 32,47,53,-12
MOV SI,2
MOV AH,TABLE[SI]          ;AH=53
```

برخی اپراتورهای بکار رفته در دستورالعمل MOV:

1.POINTER(PTR)

EXAMPLE:

```
TOTAL DW 0F25BH

MOV AX,TOTAL              ;AX=F25BH
MOV AH,TOTAL              ;ERROR
MOV AL,BYTE PTR TOTAL     ;AL=5BH
MOV AH,BYTE PTR TOTAL+1   ,AH=F2H
```

تمرین : خروجی برنامه های زیر را بنویسید .

- ```
  SUM DB 4DH,32H
  MOV AX,WORD PTR SUM          ;AX=324DH
  ```

- ```
  K DD 3ADE68B1H
  MOV AX,WORD PTR K
  MOV BX,WORD PTR K+2          ;AX=68B1H   , BX=3ADEH
  ```

2.SEG,OFFSET

EXAMPLE :

```
LIST DB 100             ;IN ADDRESS 3F56:127B

MOV AH,LIST
MOV BX,SEG LIST         ;BX=3F56
MOV CX,OFFSET LIST      ;CX=127B
```

3.TYPE,LENGTH,SIZE

EXAMPLE:

```
LIST DW 100 DUP(?)

MOV AX,SIZE LIST          ;AX=200
MOV AX,TYPE LIST          ;AX=2
MOV AX,LENGTH LIST        ;AX=100
```

نکته : در مثال فوق اپراتور SIZE کل تعداد بایتها را برمی گرداند . اپراتور TYPE تعداد بایتهای هر بلوک متغیر را بر می گرداند در اینجا ۲ تا ۱۰۰ تایی و اپراتور LENGHT تعداد بلوک های متغیر را بر می گرداند .

دستور LEA : با اجرای این دستور آدرس متغیر مبداء در یکی از ثباتهای SI , DI , BP , BX قرار می گیرد .

Lea مقصد , مبداء

مقصد ◄—————— آفست مبداء

- مبداء می تواند یک متغیر از نوع بایت یا کلمه باشد .
- از نظر نتیجه و اجرا این دو دستور زیر با هم معادل هستند :

```
LEA   BX , X
MOV  BX , OFFSET X
```

مثال : محتوای ثبات AL پس از اجرای قطعه برنامه زیر چیست ؟

```
ORG     100
DATA3   DB   23 , 47 , 35 , 83
LEA     SI , DATA3 + 1
MOV     AL , [SI]
```

جواب : خط اول آفست برنامه یا نقطه شروع را برای متغیر DATA3 تعیین می کند .

خط دوم یک متغیر از نوع بایت می باشد با چهار رقم که هر رقم با توجه به نوع متغیر بایت می باشند . بدین ترتیب عدد 23 در آدرس 100 و عدد 83 در آدرس 103 ذخیره خواهد شد .

خط سوم آدرس DATA3 + 1 به ثبات SI منتقل خواهد شد . از آنجایی که آفست DATA3 آدرس 100 می باشد بنابراین آدرس DATA3 + 1 آدرس 101 می باشد . SI=101

خط چهارم محتوای SI یعنی محتوای آدرس 101 به ثبات AL منتقل می شود . با توجه به متغیر DATA3 نتیجه می شود که در آدرس 101 عدد 47 ذخیره شده است . بنابراین : AL=47

دستور مبادله XCHG : با اجرای این دستور محتوای عملوندهای مبداء و مقصد با هم مبادله می شوند .

مبداء ⟷ مقصد XCHG مبداء , مقصد

- مبداء و مقصد نمی توانند ثابت باشند .
- عملوندها بایستی از نوع بایت یا از نوع کلمه باشند .
- هر دو عملوند با هم نمی توانند متغیر باشند .

مثال : محتوای ثبات AX و متغیر X را پس از اجرای قطعه برنامه زیر بنویسید :

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

جواب : دو خط اول دستورهای انتقال است پس نتیجه :

AX=1000
X=3000

در خط سوم دستور مبادله بین ثبات AX و متغیر X داده شده که محتوای این دو با هم عوض می شوند بنابراین نتیجه نهایی :

AX=3000
X=1000

مثال : محتوای ثبات AL و متغیرهای X و Y را پس از اجرای دستورات زیر بنویسید .

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

جواب : خط اول X=65
خط دوم Y=48
خط سوم AL=65
خط چهارم AL=48 , Y=65
خط پنجم X=48

نتیجه : آخرین تغییرات Y=65 , X=48 , AL=48

دستور انتقال آدرس LDS: (Load Data Segment Register) این دستور جهت دسترسی به سگمنت دیتای جدید (DS جدید) بکار می رود .

```
LDS   مقصد , مبداء
```

محتوای مبداء ← بایت کم ارزش مقصد

محتوای (مبداء + ۱) ← بایت پر ارزش مقصد

محتوای (مبداء + ۲) ← بایت کم ارزش سگمنت مربوط به مقصد

محتوای (مبداء + ۳) ← بایت پر ارزش سگمنت مربوط به مقصد

مثال : با توجه به محتویات حافظه نشان داده شده نتیجه دستور LDS DI,[2000] را بنویسید .

12H	2000
17H	2001
4FH	2002
9BH	2003
00H	2004
ACH	2005

جواب : از دستور فوق نتیجه می شود که نتیجه دستور در ثبات DI و DS قرار دارد .

قسمت کم ارزش DI: 12H قسمت پر ارزش DI: 17H قسمت کم ارزش DS: 4FH قسمت پرارزش DS: 9BH

DI = 1712H

DS = 9B4FH

دستور انتقال آدرس LES: این دستور مانند LDS است فقط به جای سگمنت DS از سگمنت ES استفاده می شود .

دستور ورودی IN: با اجرای این دستور محتوای یک بایت یا یک کلمه از یک دستگاه ورودی به ثبات AL یا AX منتقل می شود .

یک بایت از پورت ورودی به قسمت کم ارزش آکومولاتور منتقل می شود . IN AL , پورت

یک کلمه از پورت ورودی به ثبات آکومولاتور منتقل می شود . IN AX , پورت

پورت : آدرس دستگاه یا واحد ورودی می باشد که عددی بین 00H الی FFH است .

بعنوان مثال در دستور IN AX,12 از پورت 12 و 13 یک کلمه شانزده بیتی در ثبات AX قرار می گیرد و یا دستور IN AL,DX نشان می دهد که اطلاعات از پورتی که آدرس آن در ثبات DX قرار دارد در ثبات AL ذخیره می شود و یا دستور IN AX,DX نشان می دهد که اطلاعات از پورتی که آدرس آن در ثبات DX و ثبات DX+1 قرار دارد در ثبات AX ذخیره شود .

دستور خروجی OUT : با اجرای این دستور محتوای ثبات های AL یا AX به دستگاههای خروجی منتقل می شود .

```
OUT    پورت , AL

OUT    پورت , AX
```

مثلاً در دستور زیر محتوای ثبات AX به پورت های خروجی شماره 32 و 33 منتقل می شود .

```
OUT    32 , AX
```

تمرین : بنویسید که دستورات زیر به چه مفهومی است ؟

```
OUT   DX,AX

OUT   DX,AL
```

۲- دستورات محاسبات ریاضی

دستور جمع ADD : با اجرای این دستور محتوای عملوند مبداء و مقصد با یکدیگر جمع و نتیجه در عملوند مقصد قرار می گیرد .

ADD مبداء , مقصد مقصد ← مبداء + مقصد

- دستور جمع بر همه بیتهای حسابی ثبات پرچم اثر دارد .

مثال : در دو قطعه برنامه زیر نتیجه متغیر X را مشخص کنید :

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

X=23

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

X=0

تمرین : نتیجه ثبات CL و CF در دستور زیر را بنویسید :

```
MOV   CL,0F5H
ADD   CL,0BH
```

دستور جمع با رقم نقلی ADC : با اجرای این دستور محتوای عملوند مبداء با مقصد با رقم نقلی (CF) با یکدیگر جمع و در عملوند مقصد قرار می گیرد .

مبداء + مقصد + رقم نقلی ⟵ مقصد ADC مبداء , مقصد

مثال : نتیجه ثبات AX پس از اجرای قطعه برنامه زیر چقدر است ؟

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

جواب : در خط چهارم محتوای ثبات AX=1000 و متغیر X=3000 با CF با همدیگر جمع شده و نتیجه در ثبات AX ذخیره می گردد .

If CF=0 ⟶ AX=4000

If CF=1 ⟶ AX=4001

نکته : تکنیکی جهت جمع ۲ عدد ۳۲ بیتی با ثبات های ۱۶ بیتی

کلمه مضاعف اول

۱۶ بیت کم ارزش = AX	۱۶ بیت کم ارزش = BX

کلمه مضاعف دوم

۱۶ بیت کم ارزش = CX	۱۶ بیت کم ارزش = DX

```
ADD   AX,BX
ADC   DX,CX
```

نتیجه جمع :

۱۶ بیت کم ارزش = AX	۱۶ بیت پر ارزش = DX

در مثال فوق ثباتهای AX, BX, CX, DX به عنوان نمونه قرار داده شده اند .

دستور تفریق SUB : با اجرای این دستور محتوای عملوند مقصد از مبداء کم شده و نتیجه در عملوند مقصد قرار می گیرد .

مبداء – مقصد ⟵ مقصد SUB مبداء , مقصد

مثال : محتوای ثبات AL و BL پس از اجرای قطعه برنامه زیر چقدر است ؟

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

جواب :

AL=4

BL=6

مثال : پس از اجرای دستورات زیر مقادیر نشان داده شده را بنویسید .

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

X=?
Y=?
Z=?
AX=?
SF=?

جواب :

AX=-313 , X=300 , Y=613 , Z=-313 , SF=1

تمرین : در قطعه برنامه زیر محتوای ثبات AX و متغیر X را پس از اجرای دستورات زیر نشان دهید .

```
X   DW   600
MOV   AX , 248
SUB   AX , 48
SUB   X , AX
```

تمرین : در برنامه زیر محتوای ثبات AL و پرچم SF را پس از اجرای دستورات زیر نشان دهید .

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

دستور تفریق با رقم نقلی SBB : با اجرای این دستور محتوای عملوند مقصد از مبداء و از رقم نقلی CF کم شده و نتیجه در عملوند مقصد قرار می گیرد .

SBB مبداء , مقصد مقصد ← مبداء – مقصد - CF

مثال : محتوای ثبات AX را یکبار با CF=0 و بار دیگر با CF=1 بنویسید .

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

جواب :

If CF=0 AX=200

If CF=1 AX=199

دستور ضرب : دستور ضرب به دو صورت عملوندهای بدون علامت و عملوندهای علامتدار می باشد .

MUL ⟶ بدون علامت

IMUL ⟶ علامتدار

- در دستور ضرب عملوند می تواند از نوع بایت یا کلمه باشد .
- عملوند می تواند متغیر یا ثبات باشد .
- عملوند نمی تواند یک عدد ثابت باشد .

ضرب بدون علامت :

❖ الف) چنانچه عملوند بصور بایت باشد :

MUL عملوند بایت

AX ← AL * عملوند بایت

همانگونه که دیده می شود دستورالعمل ضرب با یک عملوند نوشته می شود . برای انجام ضرب دو عدد هشت بیتی لازم است که یکی از ارقام از قبل در ثبات AL و دیگری را در دستور بعنوان عملوند بایت قرار دهیم . نتیجه این ضرب در ثبات AX ذخیره می شود .

❖ ب) چنانچه عملوند بصورت کلمه باشد :

MUL عملوند کلمه

DX : AX ← AX * عملوند کلمه

همانگونه که دیده می شود ضرب دو کلمه می تواند جوابی بیش از شانزده بیت داشته باشد در نتیجه قسمت کم ارزش تر در ثبات AX و قسمت پرارزش تر در ثبات DX ذخیره می گردد .

ضرب علامتدار : ضرب علامتدار نیز مانند ضرب بدون علامت می تواند عملوند بایت و کلمه داشته باشد . از نظر نوع اجرا و ذخیره سازی نتیجه این دو ضرب مانند هم می باشند با این تفاوت که در ضرب علامتدار ، علامت نیز در نظر گرفته می شود .

مثال : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

جواب : همانطور که در خط سوم دیده می شود دستور ضرب بدون علامت بکار برده شده است . از روی عملوند ضرب نتیجه می گیریم که ضرب از نوع عملوند بایت است (ثبات BL) که نتیجه این ضرب حتماً بایستی در ثبات AX باشد .

AX = AL * BL

$AX = 0B4^H * 11^H$

11^H ⟶ 17 $0B4^H$ ⟶ 180

$BF4^H$ ← 3060 = 17 * 180

AX = $0BF4^H$ یا 0000 1011 1111 0100

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

مثال : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

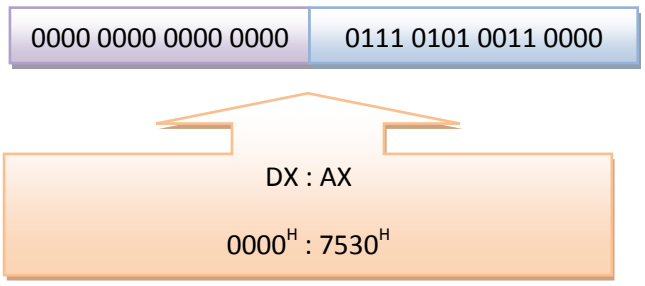
جواب : در خط سوم دستور ضرب بدون علامت از نوع کلمه نوشته شده است .

در این ضرب نتیجه در دو مکان ذخیره می شود . قسمت کم ارزش تر در ثبات AX و قسمت پرارزش تر در ثبات DX قرار دارد .

DX : AX = AX * BX

DX : AX ← 30000 = 2000 * 15

DX : AX



مثال : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

```
ORG  2002H
X  DW  5000
MOV  AX , 3000
LEA  BX , X
MUL  [BX]
```

جواب :

خط اول تعیین آفست است برای متغیر X

خط دوم تعریف یک متغیر ۱۶ بیتی

خط سوم انتقال عدد 3000 به ثبات AX که در مبناء شانزده $BB8^H$

خط چهارم آفست متغیر X را که 2002^H می باشد به ثبات BX منتقل می کند .

خط پنجم دستور ضرب بایت می باشد زیرا محتوای آدرس 2002^H یک داده هشت بیتی می باشد که شامل قسمت کم ارزش عدد 5000 می باشد . در تبدیل زیر عدد 88^H می باشد . پس عملیات ضرب بصورت زیر است :

AL * [BX] = AX

$B8^H * 88^H \longrightarrow 184 * 136 = 25024 \longrightarrow 61C0^H$

AX

0110 0001 1100 0000

مثال : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

جواب :

در خط سوم یک دستور ضرب بایت علامتدار قرار دارد عملیات بصورت زیر می باشد :

AL * BL = AX

$AL = 11^H = 0001\ 0001 = +17$

$BL = B4^H = 1011\ 0100 \longrightarrow 11001100 = -76$

$+17 * (-76) = -1292 \longrightarrow 1000\ 0101\ 0000\ 1100$

$1111\ 1010\ 1111\ 0100$

$AX = FAF4^H$

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

دستور تقسیم : دستور تقسیم نیز مانند دستور ضرب به دو صورت عملوندهای بدون علامت و عملوندهای علامتدار می باشد .

DIV ⟵ بدون علامت

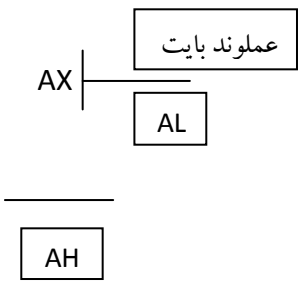
IDIV ⟵ علامتدار

- در دستور تقسیم عملوند می تواند از نوع بایت یا کلمه باشد .
- عملوند می تواند متغیر یا ثبات باشد .
- عملوند نمی تواند یک عدد ثابت باشد .

تقسیم بدون علامت :

❖ الف) چنانچه عملوند بصور بایت باشد :

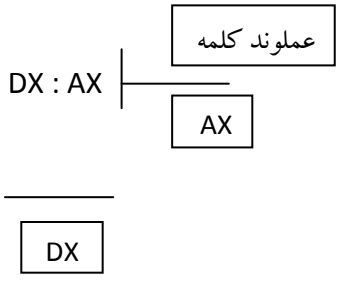
DIV عملوند بایت



نتیجه خارج قسمت و باقیمانده در ثبات AX ذخیره می شود .

❖ ب) چنانچه عملوند بصورت کلمه باشد :

DIV عملوند کلمه



تقسیم علامتدار : تقسیم علامتدار نیز مانند تقسیم بدون علامت می تواند عملوند بایت و کلمه داشته باشد . از نظر نوع اجرا و ذخیره سازی نتیجه این دو تقسیم مانند هم می باشند با این تفاوت که در تقسیم علامتدار ، علامت نیز در نظر گرفته می شود .

مثال : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

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

جواب : در خط سوم دستور تقسیم بدون علامت با عملوند بایت نوشته شده است .

130 / 5 خارج قسمت : $26 \longrightarrow 1A^H \longrightarrow AL$

باقیمانده : $00 \longrightarrow 00^H \longrightarrow AH$

$AX = 001A^H$

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

```
ORG   1000H
X   DB   10110100B
MOV   AX , 400H
DIV   X
```

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

```
ORG  8008H
Y   DW   5700
MOV   AX , 00A2H
MOV   DX , 0B1CH
DIV   Y
```

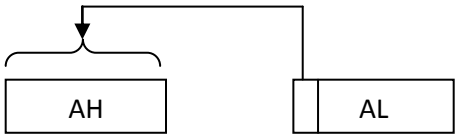
تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

```
MOV   BL , 0B4H
MOV   AX , 400H
IDIV   BL
```

دستور CBW و CWD: convert word to double word , convert byte to word

جهت تبدیل یک بایت به یک کلمه و یا تبدیل یک کلمه به یک کلمه مضاعف بکار می رود

CBW : بیت علامت ثبات AL را در ثبات AH کپی می کند



مثال : محتوای ثبات AX پس از اجرای دستورات زیر چه مقداری است ؟

```
MOV  AL,86H
CBW
```

جواب : AX = FF86H

CWD : بیت علامت ثبات AX را در ثبات DX کپی می کند .

مثال : محتوای ثبات AX و DX پس از اجرای دستورات زیر چه مقداری است ؟

```
MOV  AX,E94FH
CWD
```

جواب : AX=E94FH

DX=FFFFH

نکته ۱ : جهت انجام ضرب یک بایت در یک کلمه از دستور CBW استفاده می کنیم .

نکته ۲ : جهت انجام تقسیم یک بایت بر یک بایت از دستور CBW استفاده می کنیم .

نکته ۳ : جهت انجام تقسیم یک کلمه بر یک کلمه از دستور CWD استفاده می کنیم و یا MOV DX,0

دستور کاهش : با اجرای این دستور یک واحد از محتوای عملوند کم شده و نتیجه در عملوند قرار می گیرد .

DEC عملوند ۱- عملوند ⟵ عملوند

دستور افزایش : با اجرای این دستور یک واحد به محتوای عملوند اضافه شده و نتیجه در عملوند قرار می گیرد .

INC عملوند ۱+ عملوند ⟵ عملوند

- عملوند می تواند از نوع بایت یا از نوع کلمه باشد .
- عملوند نمی تواند یک عدد ثابت باشد .

مثال : نتیجه خط آخر در قطعه برنامه زیر چیست ؟

```
ORG   2005H
Z   DB   130
LEA   BX , Z
DEC   [BX]
```

جواب :

در خط سوم آدرس متغیر Z در ثبات BX ذخیره می شود $BX = 2005^H$

در خط چهارم از محتوای آدرس 2005^H یک واحد کم شده و نتیجه در آدرس 2005^H ذخیره می گردد . بنابراین پس از اجرای دستورات فوق در آدرس 2005^H عدد 129 ذخیره می گردد .

تمرین : نتیجه قطعه برنامه زیر چیست و مکان ذخیره نتیجه کجاست ؟

```
ORG   3000H
X   DB   10 , 20 , 26 , 44 , 6
MOV   SI , 2
MOV   BX , OFFSET_X
INC   [BX][SI]
```

دستور محاسبه مکمل ۲ : با اجرای این دستور مکمل ۲ عملوند محاسبه شده و نتیجه در عملوند قرار می گیرد .

مکمل ۲ عملوند ⟵ عملوند NEG عملوند

۳- مقایسه و پرش

عملیات مربوط به پرچم : این دستورالعملها بدون عملوند هستند .

CLC	**CF=0**
CMC	Complement CF
STC	CF=1
CLD	DF=0
STD	DF=1
CLI	IF=0
STI	IF=1

دستور LAHF :

- این دستور بدون عملوند می باشد .
- با اجرای این دستور بایت کم ارزش ثبات پرچم به ثبات AH منتقل می شود . (فقط بیتهای 7 , 6 , 4 , 2 , 0)

دستور SAHF :

- این دستور بدون عملوند می باشد .
- با اجرای این دستور محتوای ثبات AH به بایت کم ارزش ثبات پرچم منتقل می شود .

پرش غیر شرطی (JMP) : برای اجرای این دستور کامپیوتر آفست آدرس پرش را که جلوی دستور نوشته شده است در IP قرار می دهد . بدین ترتیب اجرای دستورات از این IP دنبال می شود . البته می توان شماره آدرس را نیز قرار داد . JMP 2010H

JMP آدرس

به مثال زیر دقت کنید . در خط سوم دستور پرش غیر شرطی آورده شده است . با اجرای این دستور ادامه برنامه از خط ششم ادامه می یابد و خطوط چهارم و پنجم اجرا نمی شود .

```
        MOV   AL , 5
        ADD   AL , BL
        JMP   L1
        MUL    BL
        INC   BL
L1 :    SUB   CX , 2
```

دستور پرش معمولی شامل :

الف – پرش SHORT از 128- الی 127+ `JMP **SHORT** begin`

ب- پرش NEAR (در داخل سگمنت) از 32768- الی 32767+ `JMP **NEAR** begin`

ج – پرش FAR (از یک سگمنت به سگمنت دیگر) ... **FAR** JMP

دستورات پرش شرطی : تمامی دستورات پرش شرطی از نوع SHORT هستند .

الف – پرش شرطی مبتنی بر بیت های پرچم

ب - پرش شرطی مبتنی بر اعداد علامت دار

ج - پرش شرطی مبتنی بر اعداد بدون علامت

جدول پرش شرطی مبتنی بر بیت های پرچم

دستور	شرط پرش	توضیحات
JZ (JE)	ZF=1	**پرش روی صفر**
JNZ (JNE)	ZF=0	**پرش روی غیر صفر**
JS	SF=1	**پرش روی علامت منفی**
JNS	SF=0	**پرش روی علامت مثبت**
JO	OF=1	**پرش روی سرریز**
JNO	OF=0	**پرش روی عدم سرریز**
JP(JPE)	PF=1	**پرش روی ایجاد توازن**
JNP (JJPO)	PF=0	**پرش روی عدم ایجاد توازن**
JC	CF=1	**پرش روی ایجاد رقم نقلی**
JNC	CF=0	**پرش روی عدم ایجاد رقم نقلی**

❖ دو دستور اول در پرش شرطی مبتنی بر اعداد علامت دار و بدون علامت نیز کاربرد دارند .

در مثال زیر دقت کنید :

```
        MOV    AX,-100
        ADD    AX,BX
        SUB    AX,CX
        JNZ    NEXT
        MUL    BX
        MOV    AX,BX
NEXT :  MOV    CX,10
```

در مثال فوق در خط چهارم یک دستور پرش شرطی آورده شده که می خواهیم ببینیم آیا پرش انجام می شود یا خیر ؟ شرط پرش این دستور آن است که ZF=0 شود یعنی پرش را روی نتیجه غیر صفر انجام می دهد . برای کنترل پرچم ZF بایستی به دستور قبل از پرش نگاه کنیم یعنی دستور SUB AX,CX از آنجایی که در این دستور نتیجه در ثبات AX ذخیره می شود بنابراین دستور پرش این ثبات را ملاک پرش خود قرار می دهد . اگر محتوای این ثبات پس از اجرای دستور SUB صفر نشود شود در نتیجه ZF=0 شده و پرش به برچسب NEXT انجام

می شود و ادامه برنامه از آن خط دنبال می شود . اما هرگاه پس از اجرای دستور SUB ثبات AX صفر شود آنگاه ZF=1 شده و درنتیجه عمل پرش انجام نمی شود و ادامه برنامه از خط بعد از پرش یعنی MUL BL دنبال می شود .

مثال : محتوای متغیر PH و ثبات CX را پس از اجرای قطعه برنامه زیر بنویسید .

```
        PH    DW    ?
        MOV    PH,0
        MOV    CX,10
Begin : ADD    PH,CX
        DEC    CX
        JNZ    begin
```

جواب :

همانطور که مشاهده می شود در خط آخر یک دستور پرش شرطی آورده شده ولی برچسب آن به خطوط ماقبل اشاره دارد یعنی اگر پرش انجام شود به خطوط ماقبل پرش انجام می پذیرد . شرط پرش نتیجه غیر صفر می باشد یعنی ZF=0 و این نتیجه از دستور ماقبل پرش بدست می آید یعنی ثبات CX .

بار اول : PH=10 و CX=9 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار دوم : PH=19 و CX=8 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار سوم : PH=27 و CX=7 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار چهارم : PH=34 و CX=6 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار پنجم : PH=40 و CX=5 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار ششم : PH=45 و CX=4 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار هفتم : PH=49 و CX=3 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار هشتم : PH=52 و CX=2 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار نهم : PH=54 و CX=1 می بینیم که محتوای CX صفر نشده و در نتیجه ZF=0 و پرش به برچسب begin انجام می شود .

بار دهم : PH=55 و CX=0 می بینیم که محتوای CX صفر شد و در نتیجه ZF=1 و دیگر پرش به برچسب begin انجام نمی شود .

جدول پرش شرطی مبتنی بر اعداد علامت دار

دستور	توضیحات
JG (JNLE)	**اگر عملوند اول بزرگتر باشد پرش انجام می شود (در صورت مساوی و کوچکتر پرش انجام نمی شود)**

JGE (JNL)	**اگر عملوند اول بزرگتر یا مساوی باشد پرش انجام می شود (در صورت کوچکتر بودن پرش انجام نمی شود)**
JL (JNGE)	**اگر عملوند اول کوچکتر باشد پرش انجام می شود (در صورت مساوی و بزرگتر پرش انجام نمی شود)**
JLE (JNG)	**اگر عملوند اول کوچکتر یا مساوی باشد پرش انجام می شود (در صورت کوچکتر بودن پرش انجام نمی شود)**

در مثال زیر محتوای ثبات AL با عدد 20H مقایسه می شود در صورتی که AL < 20H باشد پرش به برچسب مورد نظر انجام می شود :

```
CMP  AL , 20H
JL     NEXT
```

جدول پرش شرطی مبتنی بر اعداد علامت دار

دستور	توضیحات
JA (JNBE)	**اگر عملوند اول بزرگتر باشد پرش انجام می شود (در صورت مساوی و کوچکتر پرش انجام نمی شود)**
JAE (JNB)	**اگر عملوند اول بزرگتر یا مساوی باشد پرش انجام می شود (در صورت کوچکتر بودن پرش انجام نمی شود)**
JB (JNAE)	**اگر عملوند اول کوچکتر باشد پرش انجام می شود (در صورت مساوی و بزرگتر پرش انجام نمی شود)**
JBE (JNA)	**اگر عملوند اول کوچکتر یا مساوی باشد پرش انجام می شود (در صورت کوچکتر بودن پرش انجام نمی شود)**

دستورالعمل مقایسه (CMP) :

CMP عملوند ۱ , عملوند ۲

این دستور مانند دستور SUB عمل می کند با این تفاوت که نتیجه در جایی ذخیره نمی گردد بلکه مقادیر بیتهای ثبات پرچم را تغییر می دهد . در این دستور عملوندها تغییر نکرده و عملوند مقصد نبایستی یک عدد ثابت باشد .

	CF	SF	ZF
عملوند۲ > عملوند۱	0	0	0
عملوند۲ = عملوند۱	0	0	1
عملوند۲ < عملوند۱	1	1	0

مثال : برنامه زیر چه عملی را انجام می دهد ؟

```
CMP    AL,20
JZ     NEXT
```

جواب : در برنامه ابتداء محتوای ثبات AL با عدد 20 مقایسه می شود . در صورت برابر بودن از آنجا که ZF=1 شده و شرط پرش فراهم می شود ادامه برنامه از برچسب NEXT دنبال می شود . اما هرگاه محتوای AL با عدد 20 برابر نباشد آنگاه ZF=0 شده و درنتیجه پرش انجام نمی شود .

دستور تکرار LOOP : گاهی یک حلقه می بایست به تعداد مشخصی تکرار شود و یا اگر به شرط مشخصی رسید متوقف شود .

LOOP آدرس (برچسب)

با اجرای این دستور یک واحد از ثبات CX کم شده و در صورت صفر نبودن این ثبات تکرار حلقه از برچسب (آدرس) ادامه می یابد . معمولاً تعداد دفعات تکرار عملاً بایستی از قبل در ثبات CX ذخیره شود .

نکته : دستور LOOP معادل دستورات زیر است :

DEC CX

JNZ آدرس

دستور	اسم دیگر	شرط تکرار
LOOPZ	LOOPE	CX<>0 & ZF=1
LOOPNZ	LOOPNE	CX<>0 & ZF=0
JCXZ	-------	CX=0

نکته : دستور LOOPZ و LOOPE هر دو یکی هستند ولی برای خوانایی برنامه زمانی که بخواهیم مساوی بودن یک نتیجه را بررسی کنیم از LOOPE و زمانی که بخواهیم صفر بودن نتیجه ای را بدانیم از LOOPZ استفاده می کنیم .

تمرین : چگونه می توان با دستور LOOP در برنامه تاخیرهای مختلف ایجاد کرد ؟ کاربرد تاخیر را بنویسید .

چند مثال ساده :

- برنامه ای بنویسید که بتواند دو عدد 20 و 31 را بایکدیگر جمع نموده و نتیجه را در ثبات DL ذخیره نماید .

جواب :

روش اول

```
MOV   AL,20
MOV   BL,31
ADD   AL,BL
MOV   DL,AL
```

روش دوم

```
MOV   AL,20
MOV   DL,31
ADD   DL,AL
```

روش سوم

```
MOV    DL,20
ADD    DL,31
```

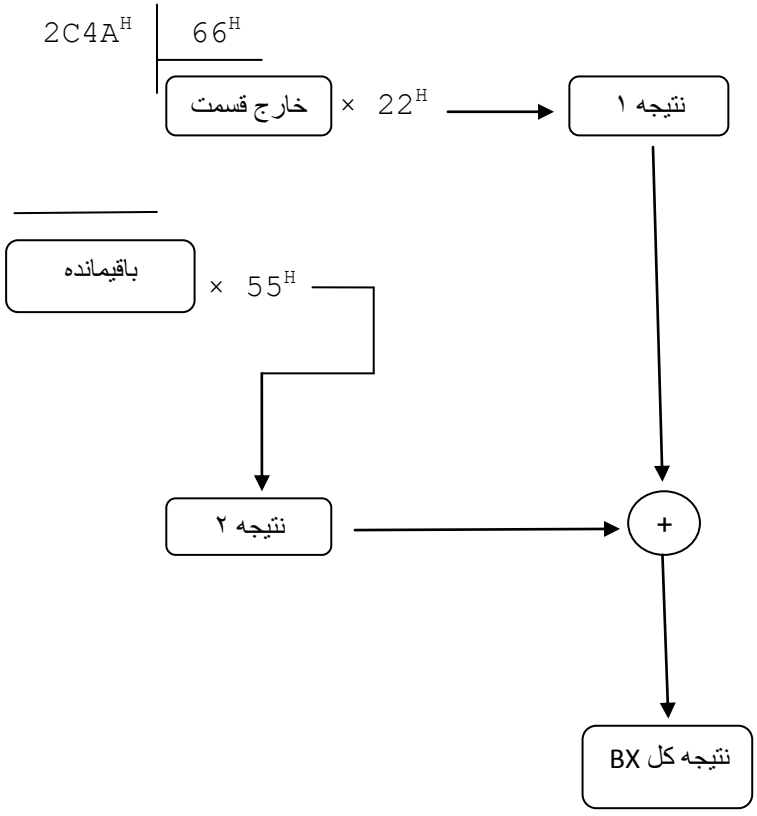
- برنامه ای بنویسید که محتویات آدرسهای 300^H الی 304^H حافظه را با یکدیگر جمع نموده و با فرض اینکه نتیجه این عمل جمع از هشت بیت بیشتر نمی شود مجموع را در ثبات AL ذخیره نماید .

```
MOV  AL,[0300H]
ADD  AL,[0301H]
ADD  AL,[0302H]
ADD  AL,[0303H]
ADD  AL,[0304H]
```

- برنامه ای بنویسید که بتواند دو عدد ۸ و ۱۵ را در هم ضرب کرده و قسمت پر ارزش نتیجه ضرب را از ۱۰ کم کند و نتیجه کل در ثبات DH ذخیره نماید .

```
MOV  AL,8
MOV  BL,15
MUL  BL
SUB  AH,10
MOV  DH,AH
```

تمرین : برنامه ای بنویسید که بتواند عملیات زیر را انجام دهد .



تمرین : برنامه ای بنویسید که ابتدا در ثبات AL عدد ۲۰ را قرار دهد . سپس ثبات BL را با عدد ۷ جمع و نتیجه را در BL قرار دهد . سپس محتوای AL را در BL ضرب و نتیجه را در DX ذخیره کند و سرانجام یک واحد از DX کم کند .

۴- دستورات منطقی

دستور NOT : این دستور مکمل ۱ عملوند را محاسبه و در عملوند قرار می دهد .

NOT عملوند

مثال : نتیجه ثبات DL در قطعه برنامه زیر چیست ؟

```
MOV   DL,8A^H
NOT   DL
```

جواب : در خط دوم از $8A^H$ مکمل ۱ گرفته می شود و نتیجه در ثبات DL ذخیره می شود . برای بدست آوردن نتیجه ابتداء بایستی $8A^H$ را به مبناء ۲ تبدیل و سپس مکمل۱ بگیریم :

$8A^H \longrightarrow 1000\ 1010^B \xrightarrow{\text{مکمل ۱}} 0111\ 0101 \longrightarrow 75^H$

$DL=75^H$

دستور AND :

عملوند۲ & عملوند۱ ⟵ عملوند۱ عملوند ۲ , عملوند ۱ AND

X	Y	X . Y
0	0	0
0	1	0
1	0	0
1	1	1

مثال : نتیجه ثبات BL در قطعه برنامه زیر چیست ؟

```
MOV   BL,35^H
AND   BL,0F^H
```

جواب : در این برنامه بین 35^H و $0F^H$ عمل AND منطقی انجام و نتیجه در ثبات BL ذخیره می شود .

```
0011 0101
0000 1111
0000 0101 ⟶ 05^H ⟶ BL=05^H
```

دستور OR :

عملوند۲ OR عملوند۱ ⟵ عملوند۱ عملوند ۲ , عملوند ۱ OR

X	Y	X + Y
0	0	0
0	1	1
1	0	1
1	1	1

مثال : نتیجه ثبات AL در قطعه برنامه زیر چیست ؟

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

جواب :

```
1010 0101
0010 1010
1010 1111 ⟶ AFH ⟶ AL=AFH
```

نکته : از دستور OR منطقی جهت تست صفر بودن یک ثبات استفاده می شود . مثلاً در دستور OR BL, 0 اگر BL صفر باشد مطمئناً ZF=1 می شود بنابراین با کنترل پرچم ZF می توان صفر بودن یک ثبات را تست نمود .

دستور XOR :

عملوند۲ XOR عملوند۱ ⟵ عملوند۱ عملوند ۲ , عملوند ۱ XOR

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

مثال : نتیجه ثبات AL در قطعه برنامه زیر چیست ؟

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

جواب :

```
0010 1101
1100 0010
1110 1111 ⟶ EFH ⟶ AL=EFH
```

نکته : جهت پاک کردن یک ثبات از XOR استفاده می شود (هر ثباتی با خودش XOR شود صفر می شود)

نکته : کد اسکی حروف بزرگ انگلیسی از 65 الی 90 یا از 41^H الی $5A^H$ می باشد . حروف کوچک نیز دارای کد اسکی بین 97 الی 122 و یا از 61^H الی $7A^H$ می باشند . تنها تفاوت بین حروف بزرگ و کوچک در بیت d5 می باشد که این بیت در حروف کوچک برابر 1 و در حروف بزرگ برابر 0 است . بنابراین جهت تبدیل حروف کوچک به بزرگ و بلعکس از دستور زیر استفاده می شود :

XOR حرف کوچک (یا بزرگ) , $0010\ 0000^B$

نکته : برای تبدیل حروف بزرگ به کوچک از دستور زیر نیز استفاده می شود :

OR حرف بزرگ , $0010\ 0000^B$

نکته : برای تبدیل حروف کوچک به بزرگ از دستور زیر نیز استفاده می شود :

AND حرف کوچک , 1101 1111^{B}

نکته : اگر بخواهیم پنج بیت اول ثبات AX معکوس شود (مکمل) از دستور زیر استفاده می کنیم :

XOR AX , 001FH

نکته : اگر بخواهیم شش بیت کوچکتر ثبات AX را جدا کنیم از دستور زیر استفاده می کنیم : (کنار گذاشتن تعدادی بیت)

AND AX , 003FH

به عبارتی دیگر دستور فوق فقط شش بیت کم ارزش باقی مانده و ده بیت دیگر صفر می شود .

نکته : با دستور OR می توان یک یا چند بیت ثباتی را برابر عدد ۱ قرار داد (صرفنظر از مقدار قبلی) دستور زیر بیت چهارم ثبات AL را برابر یک قرار می دهد :

OR AL , 10H

دستور TEST : این دستور مانند دستور AND منطقی عمل می کند بدون ذخیره نتیجه فقط بیتهای ثبات پرچم را تغییر می دهد .

TEST عملوند ۱ , عملوند ۲ عملوند۱ AND عملوند۲

قطعه برنامه زیر مشخص می کند که آیا بیتهای 0 و 2 و 5 و 7 ثبات AL برابر با یک است یا خیر؟ در صورت یک بودن پرش انجام می شود .

```
TEST  AL,10100101B
JZ    NEXT
```

اگر بخواهیم بیتهای 0 و 2 ثبات DL را یک کنیم از دستور زیر استفاده می کنیم :

```
OR  DL,0000 0101B
```

اگر بخواهیم بیتهای 3 و4 ثبات DL را صفر کنیم :

```
AND  DL,1110 0111B
```

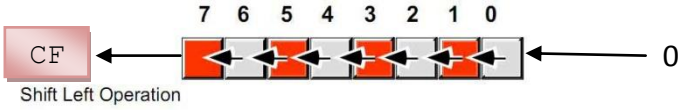
اگر بخواهیم بیتهای 1 و 6 ثبات DL را مکمل کنیم :

```
XOR  DL,0100 0010B
```

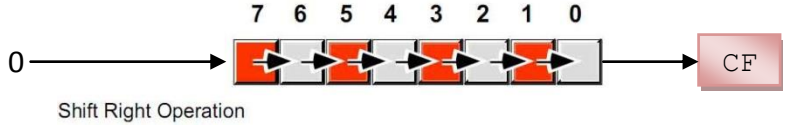
دستورات شیفت : این دستورالعملها باعث تغییر مکان بیتها به سمت چپ یا راست می شوند .

SHL	شیفت منطقی به چپ برای اعداد بدون علامت
SHR	شیفت منطقی به راست برای اعداد بدون علامت
SAL	شیفت ریاضی به چپ برای اعداد علامتدار
SAR	شیفت ریاضی به راست برای اعداد علامتدار

شیفت به چپ :



شیفت به راست :




```
SHL   DL,3        ثبات DL را ۳ بار به سمت چپ شیفت بده
SHR   AX,2        ثبات AX را دو بار به سمت راست شیفت بده
```

همانطور که دیده می شود دو دستور فوق با دو عملوند بکار می رود که عملوند دوم تعداد دفعات شیفت را تعیین می کند . در این نوع دستورات اگر تعداد دفعات شیفت ۱ بار باشد در عملوند نوشته می شود در غیر اینصورت از ثبات CL استفاده می شود .

مثال : محتوای ثبات AL پس از اجرای قطعه برنامه زیر چیست ؟

```
MOV   AL,14H
MOV   CL,3
SHR   AL,CL
```

جواب : خط سوم نشان می دهد که محتوای ثبات AL که عدد 14^H می باشد بایستی سه بار به سمت راست شیفت داده شود

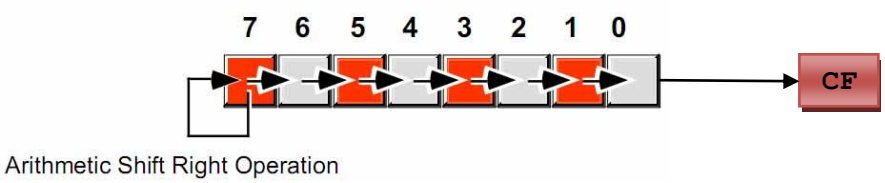
```
14H ⟶ 0001 0100
       0000 1010     بار اول
       0000 0101     بار دوم
       0000 0010     بار سوم ⟶ AL=02H
```

نکته : برای ضرب یک عدد در ۲ می توان آنرا به چپ شیفت داد .

نکته : هر بار شیفت به راست عدد را بر ۲ تقسیم می کند .

دستور SAL و SAR :

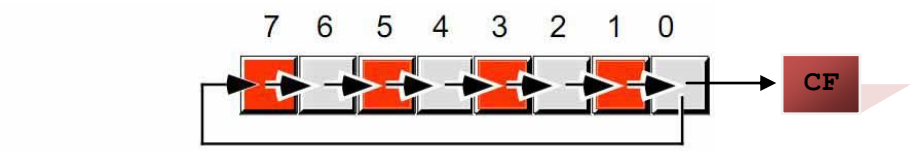
دستور SAL دقیقاً مانند SHL است و کد ماشین این دو دستور دقیقاً یکی است اما عملکرد دستور SAR بصورت زیر می باشد :



دستورات چرخش :

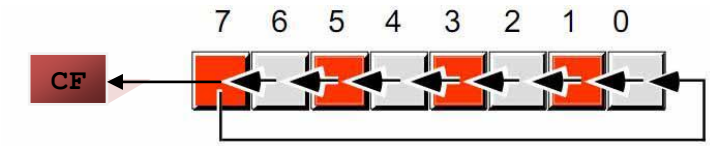
ROR	چرخش به راست بدون رقم نقلی
ROL	چرخش به چپ بدون رقم نقلی
RCR	چرخش به راست با رقم نقلی
RCL	چرخش به چپ با رقم نقلی

: ROR



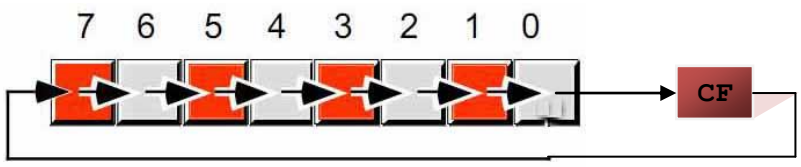
Rotate Right Operation

: ROL

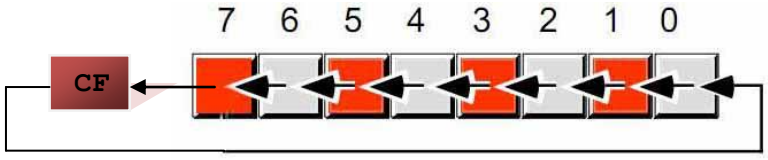


Rotate Left Operation

: RCR



: RCL



مثال : محتوای ثبات BL و CF پس از اجرای قطعه برنامه زیر چیست ؟

```
MOV   BL,35H
MOV   CL,4
ROR   BL,CL
```

جواب : خط سوم نشان دهنده ۴ بار چرخش بدون رقم نقلی محتوای ثبات BL می باشد .

```
BL    0011 0101
      1001 1010   CF=1    بـار اول
      0100 1101   CF=0    بـار دوم
      1010 0110   CF=1    بـار سوم
      0101 0011   CF=0    بـار چهـارم    BL=53H
```

مثال : محتوای ثبات AL و CF پس از اجرای قطعه برنامه زیر چیست ؟

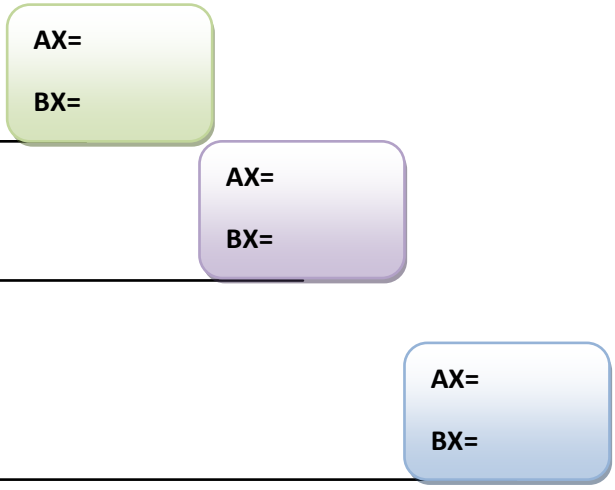
```
MOV   AL,86H
STC
RCR   AL,1
```

جواب :

```
AL    1000 0110   CF=1
      1100 0011   CF=0     AL=C3H
```

تمرین : در قطعه برنامه زیر تا مکانهای مشخص شده محتوای ثباتهای AX و BX را بدست آورید .

```
MOV   AX,22A0H
MOV   BX,180BH
MOV   CL,3
SHR   AL,CL
SHR   BH,1
AND   AL,BH
RCR   AL,1
ROL   BL,1
ADD   BH,AL
SUB   BH,BL
```



تمرین های فصل ۵:

۱- دستوری معادل دستور LEA BX,TABLE بنویسید .

۲- اگر $DH=F5^{H}$ باشد بعد از اجرای دستور NEG DH محتوای DH چقدر خواهد شد .

۳- اگر $AX=0FF50^{H}$ و $CX=0023^{H}$ و CF=1 باشد آنگاه بعد از اجرای دستور SBB AX,CX مقدار AX چقدر می شود ؟

۴- پس از اجرای دستورالعمل های زیر DX حاوی چه مقداری خواهد شد ؟

```
MOV   DX,FFFF^H
ROR   DX,1
MOV   CL,3
ROL   DX,CL
NEG   DX
```

۵- دستور العملی معادل INC AX بنویسید .

۶- قطعه برنامه زیر را در نظر بگیرید :

```
MOV   AX,1
MOV   CX,10
P :

INC   AX
LOOP  P
```

پس از اجرای قطعه برنامه مقدار AX چیست ؟

۷- محتوای ثباتهای BH و AH و AL را پس از اجرای قطعه برنامه زیر بنویسید .

```
MOV   AL,12^H
MOV   BH,8^H
MUL   BH
NOT  AX
```

❖ برنامه زیر چه عملی انجام می دهد ؟

```
MOV DH,AH
MOV DL,AL
ADD AX,AX
ADD AX,AX
ADD AX,DX
```

۱) AX در ۸ ضرب می شود .

۲) AX با ۱۰ جمع می شود .

۳) AX در DX ضرب می شود .

۴) AX در ۵ ضرب می شود .

❖ محتوای پشته ، اشاره گر پشته و شمارنده برنامه را بعد از اجرای هر خط از برنامه بدست آورید . SP=100

PC	
	ORG 20
	PUSH 25
	PUSH 45
	CALL 50
	ORG 50
	POP AX
	POP DX
	RET

فصل ۶

اجزای یک برنامه زبان اسمبلی و نمونه برنامه ها

قبلاً گفتیم که هر برنامه اسمبلی از چند سگمنت یا قطعه تشکیل شده است :

```
STACK   SEGMENT
DATA    SEGMENT
CODE    SEGMENT
EXTRA   SEGMENT
```

❖ قطعه اضافی شامل کلیه متغیرهای برنامه است که غالباً جهت پردازش دستورات رشته ای کاربرد دارد .

شبه دستور تعریف سگمنت : دستورات زیر به اسمبلر شروع و پایان سگمنت را نشان می دهند .

```
NAME    SEGMENT    [align][combine][class]

                   محتویات قطعه

NAME    ENDS
```

Align : (همترازی) عملوند مذکور اختیاری می باشد و در مواقعی بکار می رود که چندین برنامه با هم پیوند داده می شود .
به جای آن یکی از کلمات page , para , word , byte نوشته می شود .

Byte : سگمنت از هر آدرسی می تواند شروع شود .

Word : سگمنت از هر آدرس زوجی می تواند شروع شود .

Para : سگمنت از هر آدرس قابل قسمت بر ۱۶ شروع می شود (پیش فرض)

Page : سگمنت از هر آدرس قابل قسمت بر ۲۵۶ شروع می شود .

Combine : چگونگی ترکیب سگمنت درحال تعریف را با سایر سگمنت های همنام در سایر برنامه ها مشخص می کند . (در واقع مشخص می کند که در موقع پیوند برنامه آیا سگمنت مورد نظر با سایر سگمنتها پیوند داده شده است یا خیر)
این قسمت عموماً یکی از کلمات public و یا stack می باشد .

Public : دو سگمنت همنام از دو برنامه را در موقع پیوند در حافظه بطور متوالی قرار می دهد و تشکیل یک قطعه بزرگتر می نماید . لذا در برنامه های معمولی که فقط یک سگمنت پشته و دیتا و کد دارند نیازی به این کلمه نیست .

Stack : فقط در سگمنت پشته استفاده می شود و بکار بردن آن در برنامه اجباری است .

```
NAME    SEGMENT    STACK
```

❖ در موقع پیوند برنامه ، سگمنت پشته برنامه کاربر ، با سگمنت پشته سیستم عامل ترکیب می شود و یک سگمنت پشته واحد برای اجرای برنامه درست می کند .

Class : این کلمه در موقع پیوند برنامه ها ، برای ترکیب سگمنت های از یک نوع بکار می رود و عموماً یکی از کلمات 'STACK' , 'DATA' , 'CODE' می باشد .

```
NAME   SEGMENT   STACK   'STACK'
NAME   SEGMENT   'DATA'
NAME   SEGMENT   'CODE'
```

دستور العمل ASSUME : این دستور آدرس شروع قطعه کد را در ثبات CS ، آدرس شروع قطعه دیتا را در ثبات DS ، آدرس شروع قطعه پشته را در ثبات SS ذخیره می کند . این دستور به این دلیل استفاده می شود که یک برنامه اسمبلی ممکن است چندین قطعه کد و یا چندین قطعه دیتا و ... داشته باشد ولی هر بار فقط یکی از آنها توسط CPU آدرس دهی می شود . این دستور در واقع ارتباط ثبات های قطعه و نام قطعه را برقرار می کند .

❖ سیستم عامل به ثبات های قطعه مقدار می دهد زیرا سیستم عامل می داند که چه مقدار حافظه در کامپیوتر نصب و چه مقداری از آن بوسیله سیستم مورد استفاده قرار گرفته است و چه مقداری باقی مانده است . کسی نمی تواند به DOS بگوید که حتماً یک مقدار خاصی از حافظه را از آدرس مشخصی اختصاص دهد . تخصیص ناحیه خاصی از حافظه و مقدار دقیق ثباتهای قطعه به عده DOS می باشد . ولی ثباتهای DS , ES توسط کاربر آدرس دهی می شوند زیرا یک برنامه ممکن است شامل چندین سگمنت دیتا و اضافی باشد و برنامه نویس بخواهد در حین اجرای برنامه بین آنها جابجا شود .

❖ طریقه آدرس دهی ثبات DS و ES توسط کاربر بصورت زیر است :

```
MOV   AX, اسم سگمنت دیتا
MOV   DS,AX
MOV   ES,AX
```

تعریف روال : روال گروهی دستور است که جهت انجام عملی خاص در نظر گرفته شده است . یک قطعه کد می تواند یک روال کامل و یا چند روال کوچکتر باشد . این کار جهت راحتی نوشتن برنامه ها است .

```
NEME   PROC   (FAR   or   NEAR)
    محتویات روال ( دستورات برنامه )
ENDP
```

- اگر صفتی در جلوی کلمه PROC قید نشود اسمبلر صفت NEAR در نظر می گیرد .

NAME PROC or NAME PROCE NEAR

-درصورتی که برچسب از نوع FAR باشد استفاده از کلمه FAR اجباری می باشد .

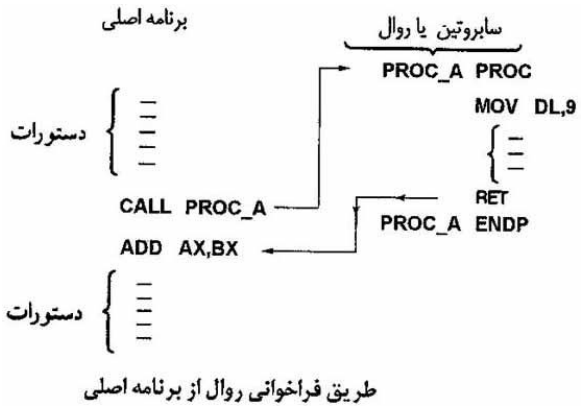
-در صورتی که برچسب از نوع NEAR باشد بعد از اسم یا برچسب می توان علامت : استفاده کرد و دیگر نیازی به کلمه NEAR نیست .

NAME : or NAME PROC NEAR

دستورالعمل CALL : جهت فراخوانی یک زیربرنامه از این دستور استفاده می شود .

CALL اسم زیربرنامه

معمولاً در زیربرنامه قبل از پایان روال از دستور RET استفاده می شود که کنترل برنانه را به خط بعد از CALL منتقل می کند .



طریق فراخوانی روال از برنامه اصلی

دستور CALL دو عمل راانجام می دهد : آدرس PC را عوض می کند و آدرس RET را در STACK ذخیره می کند .

در مثال زیر چگونگی استفاده از ، قطعه ، روال و دستور CALL نشان داده شده است :

```
PAGE 100,110
TITLE 'CALL_SUB.ASM'
;------------------------
;              1- Define stack segment
;             -------------------------
```

```
STACKSG  SEGMENT STACK 'STACK'
         DW  32 DUP (?)
STACKSG  ENDS
;------------------------
;              2-Define data segment
;              ---------------------------
DATASG   SEGMENT  'DATA'
         X1   DB 19H
         X2   DW 3F48H
         Y    DW  ?
DATASG   ENDS
;----------------------------
;              3- Define code segment
;              ------------------------
CODESG   SEGMENT  'CODE'
         ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
MAIN     PROC  FAR
         MOV  AX,DATASG
         MOV  DS,AX

         MOV CX,16
         MOV  SI,4
         .
         .
         .
         CALL  SUB1
         MOV  DH,CL
         ADD  CX,DX
         CALL  SUB2
         MOV  AX,4C00H
         INT  21H
MAIN     ENDP
;----------------------------
SUB1     PROC  NEAR
         MOV  AH,00
         MOV  AL,1
         RET
SUB1     ENDP
;----------------------------
SUB2     PROC NEAR
         MOV  AH,07
         LEA  DX,X1
         MOV DH,0AH
         RET
SUB2     ENDP
;-----------------------------
         END  MAIN
```

برنامه های نمونه :

مثال : برنامه ای بنویسید که اعداد 40 , 50 , 100 را با هم جمع نموده و نتیجه را در یک متغیر از حافظه تحت عنوان NATIJE ذخیره نماید .

```
PAGE 100,150
TITLE 'EXAM_1.ASM'
;------------------------
;              1- Define stack segment
;             -------------------------
STACKSG  SEGMENT STACK 'STACK'
         DW  32 DUP (?)
STACKSG  ENDS
;------------------------
;              2-Define data segment
;              ---------------------------
DATASG   SEGMENT  'DATA'
         X1   DB 100
         X2   DB  40
         X3   DB  50
         NATIJE   DB  ?
DATASG   ENDS
;----------------------------
;              3- Define code segment
;              ------------------------
CODESG   SEGMENT  'CODE'
         ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
MAIN     PROC  FAR
         MOV  AX,DATASG
         MOV  DS,AX
         MOV  AL,X1
         ADD  AL,X2
         ADD  AL,X3
         MOV  NATIJE,AL
         MOV  AX,4C00H
         INT  21H
MAIN     ENDP
;----------------------------
         END  MAIN
```

ساختار ساده شده : این ساختار فقط یک سگمنت دیتا و یک سگمنت کد دارد بنابراین نمی توان آنرا با برنامه های دیگر پیوند کرد و اکثراً جهت برنامه های کوچک مناسب می باشد . برنامه زیر مثال قبل به شیوه ساده شده می باشد .

```
PAGE 100,150
TITLE 'EXAM_2.ASM'
.MODEL SMALL
.STACK
         DW   32 DUP (?)
.DATA
        X1   DB 100
        X2   DB  40
        X3   DB  50
        NATIJE   DB   ?
.CODE
MAIN    PROC  FAR
        MOV   AX,@DATA
        MOV   DS,AX
        MOV   AL,X1
        ADD   AL,X2
        ADD   AL,X3
        MOV   NATIJE,AL
        MOV   AX,4C00H
        INT   21H
MAIN    ENDP
        END   MAIN
```

راهنمای اسمبلر می باشد که بطور خودکار دستور ASSUME ایجاد می کند

مثال : برنامه ای بنویسید که دو عدد ۴۰ و ۸۰ را در یکدیگر ضرب کرده و نتیجه در یک متغیر به نام RESULT ذخیره شود .

```
PAGE 70,120
TITLE 'EXAM_3.ASM'
.MODEL SMALL
.STACK
         DB   128 DUP (?)
.DATA
        DATA1    DB   40
        DATA2    DB   80
        RESULT   DW    ?
.CODE
MAIN    PROC   FAR
        MOV   AX,@DATA
        MOV   DS,AX
        MOV   AL,DATA1
        MUL   DATA2
        MOV   RESULT,AX
        MOV   AX,4C00H
        INT   21H
MAIN    ENDP
        END   MAIN
```

مثال : برنامه ای بنویسید که :

الف) در ثبات AL عدد 10 قرار گیرد .

ب) مقدار متغیر NUMBER=20 را در ثبات BL قرار دهد .

ج) ثبات BL را با AL جمع کند و نتیجه را در ثبات AL قرار دهد .

د) ثبات AL را در BL ضرب کند .

ه) ثبات BX را از AX کسر نماید .

و) نتیجه را بر ثبات BL تقسیم کند . خارج قسمت را در RESULT1 و باقیمانده را در RESULT2 ذخیره نماید .

```
PAGE 80,60
TITLE 'EXAM_4.ASM'
.MODEL SMALL
.STACK
         DD  100H DUP (?)
.DATA
        NUMBER   DB  20
        RESULT1  DB   ?
        RESULT2  DB   ?
.CODE
START   PROC  FAR
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  AL,10
        MOV  BL,NUMBER
        ADD  AL,BL
        MUL  BL
        SUB  BX,AX
        MOV  AX,BX
        DIV  BL
        MOV  RESULT1,AL
        MOV  RESULT2,AH
        MOV  AX,4C00H
        INT  21H
START   ENDP
        END  START
```

سوال : به نظر شما در متغیر RESULT1 و RESULT2 چه عددی ذخیره می شود ؟

تمرین مهم : برنامه ای بنویسید که مقدار متوسط اعداد 120 و 96 را محاسبه نماید .

مثال : برنامه ای بنویسید که محتویات خانه های حافظه را از آدرس 8000H الی 800FH را با هم جمع نموده و با فرض اینکه مجموع از هشت بیت بیشتر نمی شود نتیجه کل را در یک متغیر به نام DATAK ذخیره نماید .

```
PAGE 70,120
TITLE 'EXAM_5.ASM'
.MODEL SMALL
.STACK
         DQ  10H DUP (?)
.DATA
        DATA1   DB  8000H
        DATAK   DB   ?

.CODE
START   PROC  FAR
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  DL,16
        SUB  CX,CX
        MOV  BX,DATA1
Back:   ADD  CL,[BX]
        INC  BX
        DEC  DL
        JNZ  back
        MOV  DATAK,CL
        MOV  AH,4CH
        INT  21H
START   ENDP
        END  START
```

توضیحات برنامه :

تمرین : برنامه ای بنویسید که محتویات خانه های حافظه را از آدرس 8000H الی 800FH را با هم جمع نموده و نتیجه کل را در یک متغیر به نام DATAK ذخیره نماید . وجود رقم نقلی در جمع در نظر گرفته شود .

مثال : برنامه ای بنویسید که یک بلوک از حافظه به نام DISK_IN و به تعداد 20 بایت و آفست 100H را به محتویات یک بلوک دیگر از حافظه به نام DISK_OUT به آفست 200H منتقل نماید .

```
PAGE 60,110
TITLE 'COPY.ASM'
.MODEL SMALL
.STACK
.DATA
        ORG  100H
        DISK_IN   DB  20 DUP(?)
        ORG  200H
        DISK_OUT  DB  20 DUP(?)

.CODE
MAIN    PROC  FAR
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  CX,20
        LEA  SI,DISK_IN
        LEA  DI,DISK_OUT
Back :  MOV  AL,[SI]
        MOV  [DI],AL
        INC  SI
        INC  DI
        LOOP  back
        MOV  AX,4C00H
        INT  21H
MAIN    ENDP
        END  MAIN
```

توضیحات برنامه :

مثال : برنامه ای بنویسید که از بین اعداد ۲۲ و ۴۶ و ۱۸ و ۹۴ و ۱۱ و ۵۵ بزرگترین عدد را پیدا کرده و آنرا در یک متغیر از حافظه تحت عنوان MAX ذخیره نماید .

```
PAGE 60,130
TITLE 'MAXIMUM.ASM'
.MODEL SMALL
.STACK
        DW 50H DUP (?)
.DATA
        DATA  DB  22,46,18,94,11,55
        MAX   DB  ?
.CODE
MAIN    PROC  FAR
        MOV   AX,@DATA
        MOV   DS,AX
        MOV   CX,6
        MOV   BL,DATA
        SUB   AH,AH
Back :  CMP   BL,AH
        JS   next
        MOV   AH,BL
Next :  INC   DATA
        LOOP   back
        MOV   MAX,AH
        MOV   AX,4C00H
        INT   21H
MAIN    ENDP
        END   MAIN
```

توضیحات برنامه :

تمرین : برنامه ای بنویسید که از بین محتویات آدرس های 700 الی 800 بزرگترین عدد را پیدا کرده و آنرا در یک متغیر از حافظه تحت عنوان MAX ذخیره نماید .

مثال : برنامه ای بنویسید که بتواند تعداد یک ها را در یک بایت (مثلاً 95H) پیدا کرده و آنرا ذخیره کند .

```
PAGE 60,100
TITLE 'ONES.ASM'
.MODEL SMALL
.STACK
        DB 64 DUP (?)
.DATA
        DATA    DB  95H
        COUNT   DB  ?
.CODE
START   PROC  FAR
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  DL,8
        SUB  BL,BL
        MOV  AL,DATA
Again : ROL  AL,1
        JNC  NEXT
        INC BL
NEXT :  DEC  DL
        JNZ  again
        MOV  COUNT,BL
        MOV  AX,4C00H
        INT  21H
START   ENDP
        END  START
```

توضیحات برنامه :

تمرین : در برنامه زیر مقداری خطای تایپی و منطقی وجود دارد . خطا ها را به همراه شماره خط مشخص و اصلاح کنید . این برنامه چهار کلمه را جمع کرده و نتیجه را ذخیره می کند .

```
1     STSEG    SEGMENT
2              DB    32 DUP (?)
3     STSEG    END
4     ; -----------------
5     DTSEG    SEGMENT
6              DATA     DW    234DH,DE6H,3BC7H,566AH
7              ORG    10H
8              SUM    DW    ?
9     DTSG     ENDS
10    ; -------------------
11    CDSEG    SEGMENT
12    START:   PROC    FAR
13    ASSUME    CS:CDSEG,DS:DTSEG,SS:STSEG
14             MOV    AX , DTSEG
15             MOV DS,AX
16             MOV    CX,04
17             MOV   BX,0
18             MOV   DI,OFFSET DATA
19    LOOP1:   ADD    BX , [DI]
20             INC   DI
21             DEC   BX
22             JNZ    LOOP1
23             MOV    SI,OFFSET RESULT
24             MOV   [SI],BX
25             MOV    AX,4C00H
26             INT 21H
27    CDSEG :  ENDS
28    START    ENDP
29             END   STRT
```

تمرینات فصل ۶ :

۱- برنامه ای بنویسید که یک عدد را دریافت و فاکتوریل آنرا محاسبه و ذخیره کند .

۲- برنامه ای بنویسید که دو عدد مثبت M و N را دریافت و M^N را محاسبه و ذخیره کند .

۳- برنامه ای بنویسید که یک عدد مثبت X را دریافت کرده و زوج بودن آنرا مشخص کند .

۴- برنامه ای بنویسید که یک عدد مثبت X را دریافت کرده و اول بودن آنرا مشخص کند .

۵- برنامه ای بنویسید که یک عدد مثبت X را دریافت کرده و زوج بودن و اول بودن آنرا مشخص کند .

۶- برنامه ای بنویسید که یک کاراکتر را دریافت کرده اگر حرف کوچک بود آنرا به حرف بزرگ تبدیل کند .

۷- برنامه ای بنویسید که عبارت $Y=2X^2+3X-5$ را محاسبه کند .

۸- برنامه ای بنویسید که اعداد ۴۷ و ۳۶ و ۱۵ و ۸۸ و ۵۹ را بصورت صعودی مرتب کرده و ذخیره کند .

۹- برنامه ای بنویسید که در متن MY NAME is ali حروف کوچک انگلیسی را به حروف بزرگ تبدیل کند .

۱۰- برنامه ای بنویسید که درجه حرارت ۱۰ روز را در متغیر TEMP ذخیره کند اگر درجه حرارت روزی به ۴۰ درجه برسد پیغام IT IS TOO HOT را روی مانیتور نمایش دهد .

۱۱- برنامه ای بنویسید که محتویات آدرس های 8000H الی 80FFH را تست نموده و اگر خانه ای با عدد صفر پر شده باشد به سیستم عامل برگردد .

۱۲- برنامه ای بنویسید که تعداد اعداد زوج را در 100 خانه از حافظه از آفست 50H پیدا و ذخیره نماید .

۱۳- برنامه ای بنویسید که اگر بیت پنجم متغیر X برابر یک بود ثبات DX یک واحد اضافه گردد و اگر صفر بود از ثبات BX یک واحد کم شود .

۱۴- برنامه ای بنویسید که یک رشته سه رقمی را بگیرد و یک واحد به آن اضافه کند و حاصل را نمایش دهد .

۱۵- برنامه ای بنویسید که دو عدد سه رقمی را دریافت کند و با هم جمع و نتیجه را نمایش دهد .

۱۶- برنامه ای بنویسید که یک عدد را دریافت و روز هفته معادل آنرا چاپ کند .

۱۷- برنامه ای بنویسید که معدل را بصورت عددی خوانده و بصورت حرفی (A-F) نشان دهد .

۱۸- برنامه ای بنویسید که یک عدد مثبت N را دریافت و اعداد زوج کمتر از آن را نمایش دهد .

۱۹- برنامه ای بنویسید که یک عدد مثبت N را بگیرد و در مبنای ۲ و ۸ و ۱۶ نمایش دهد .

۲۰- برنامه ای بنویسید که یک عدد مثبت N را بگیرد و لگاریتم در پایه دو آنرا نمایش دهد .

فصل ۷

عملیات پردازش رشته ای

طول رشته می تواند تا 64KB باشد .

دستورات رشته ای

MOVS(MOVSB,MOVSW,MOVSD)	جابجایی
CMPS(CMPSB,CMPSW,CMPSD)	مقایسه
LODS(LODSB,LODSW,LODSD)	بارگذاری
SCAS(SCASB,SCASW,SCASD)	جستجو
STOS(STOSB,STOSW,STOSD)	ذخیره

در بعضی از دستورالعمل های پردازش رشته ای استفاده از ES ضروری می باشد :

```
EXSEG SEGMENT PARA 'EXTRA'
.
.
EXSEG ENDS
```

مقدار فلگ DF مشخص کننده این است که عمل جابجایی از اولین عنصر به طرف آخرین عنصر می باشد یا بلعکس . چنانچه DF برابر با صفر باشد عمل جابجایی از اولین عنصر به طرف آخرین عنصر انجام می شود .

مثال : دستورالعمل هایی بنویسید که رشته ای را که در آدرس SOURCE_STR قرار دارد را به DEST_STR منتقل کند .

```
MOV SI,OFFSET SOURCE_STR
MOV DI,OFFSET DEST_STR
CLD
MOVSB
```

در برنامه فوق دستور MOVSB باعث می شود که یک عنصر از رشته مبدا که در آدرس DS:SI قرار دارد به آدرس ES:SI کپی شود و مقدار DI و SI یک واحد افزایش یابد .

جهت انجام انتقال سایر عناصر بایستی تعداد رشته را در ثبات CX قرار داد و از پیشوند REP استفاده نمود .

```
MOV SI,OFFSET SOURCE_STR
MOV DI,OFFSET DEST_STR
CLD
MOV CX,9
REP MOVSB
```

پیشوند REP تا مادامیکه مقدار CX مخالف صفر باشد باعث تکرار دستور MOVSB می گردد .

دستور STOS : این دستور باعث می گردد که یک بایت یا یک کلمه را از ثبات AL یا AX به یک عنصر رشته مقصد منتقل نماید . مقداری که در رشته قرار می گیرد چنانچه از نوع بایت باشد در AL و اگر از نوع کلمه باشد در AX قرار داده می شود .

مثال : دستوراتی بنویسید که ۲۰ عنصر اول رشته ای به نام TEST را برابر کاراکتر @ قرار دهد .

```
TEST DB 20 DUP(?)
MOV CX,20
MOV AL,'@'
MOV DI,OFFSET TEST
CLD
REP STOSB
```

تمرین : دستوراتی بنویسید که صد کلمه اول رشته ای به نام AZMAYESH را معادل صفر قرار دهد .

دستور LODS : این دستور یک عنصر رشته مبدا را در AL یا AX قرار می دهد . دستورات زیر باعث می شود که اولین عنصر رشته SOURCE_STR در ثبات AL ذخیره شود .

```
LEA SI,SOURCE_STR
LODSB
```

دستور CMPS : این دستور دو رشته مبدا و مقصد را با هم مقایسه می نماید (عنصر رشته مبدا را از عنصر متناظر رشته مقصد کم می کند)

رشته مبدا را بایستی در data segment تعریف نمود و آدرس شروع آنرا در SI قرار داد . دستورات زیر دو رشته را با هم مقایسه می کنند :

```
MOV SI,OFFSET SOURCE_STR
MOV DI,OFFSET DEST_STR
CMPSB
```

نکته : در این دستور می تواند از پیشوندهای REPE(repeat equal) و REPZ(repeat zero) استفاده نمود . شرط تکرار این پیشوندها cx<>0 و ZF=1 است . (همچنین REPNE , REPNZ)

دستور SCAS : از این دستور برای جستجوی یک رشته جهت وجود داشتن یا نداشتن یک عنصر رشته ای معین بکار می رود . رشته مورد نظر بایستی در data segment تعریف شده و آدرس شروع آن در DI قرار گیرد . عنصر مورد جستجو چنانچه از نوع بایت باشد در AL و چنانچه از نوع کلمه باشد در AX ذخیره می شود .

مثال : دستوراتی بنویسید که در رشته STR به جستجوی کاراکتر * بپردازد .

```
STR DB 50 DUP(?)
MOV AL,'*'
MOV CX,50
LEA DI,STR
CLD
REPNE SCASB
```

تمرین : برنامه ای بنویسید که رشته ای را دریافت کند و آنرا بصورت معکوس نشان دهد .

ساختار برنامه های رشته ای :

انتقال محتویات سگمنت دیتا به ثبات DS

انتقال محتویات سگمنت دیتا به ثبات ES

تعریف CX برای تعداد عملیات

مشخص کردن نوع حرکت (صعودی یا نزولی) با DF

مشخص کردن مرجع در SI

مشخص کردن مقصد در DI

REP

مثال : برنامه ای بنویسید که رشته This is a test را از آفست SOURCE در حافظه به آدرس DESTINATION به آفست 100H انتقال دهد .

```
PAGE 60,100
TITLE 'STRING_1.ASM'
.MODEL SMALL
.STACK
        DW 32 DUP (?)
.DATA
        SOURCE DB "THIS IS A TEST"
        ORG 100H
        DESTINATION DB 14 DUP(?)
.CODE
START   PROC  FAR
        MOV  AX,@DATA
        MOV  DS,AX
        MOV  ES,AX
        MOV  CX,7 ( MOV CX,14)
        CLD
        LEA  SI,SOURCE
        LEA  DI,DESTINATION
        REP MOVSW (REP MOVSB)
        MOV  AX,4C00H
        INT  21H
START   ENDP
        END  START
```

برنامه نویسی سیستم

هدف : ارائه دانش تکمیلی در زبان ماشین و اسمبلی و آشنایی با نحوه بکارگیری امکانات کامپیوتر

مهمترین قسمتها :

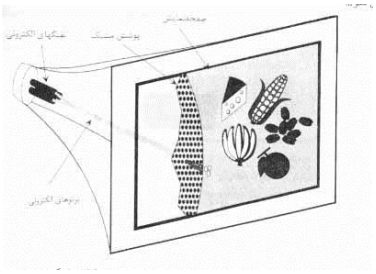
- پرشهای شرطی و غیر شرطی
- مدهای آدرس دهی
- زیر برنامه ها
- محاسبات دودویی
- دستورات منطقی و بیتی و شیفت و چرخش
- برنامه های اجرای COM
- برنامه های مقیم حافظه TSR
- امکانات کامپیوتر و مدیریت آنها مانند مانیتور ، صفحه کلید ، تولید صوت ، ماوس ، مدیریت دیسک و فایل ، چاپ و ...
- آشنایی با پردازنده ها و مادربرد
- آشنایی با نحوه طراحی کارتهای کامپیوتری AGP , ISA , EISA , PCI
- آشنایی با برنامه ریزی پورت موازی
- آشنایی با مودم و تکنولوژی DSL
- برنامه ریزی پورت سریال و آشنایی با USB
- گذری بر پردازنده های 80286 به بعد و آدرس دهی حفاظت شده و برنامه نویسی مد حفاظت شده

فصل ۸

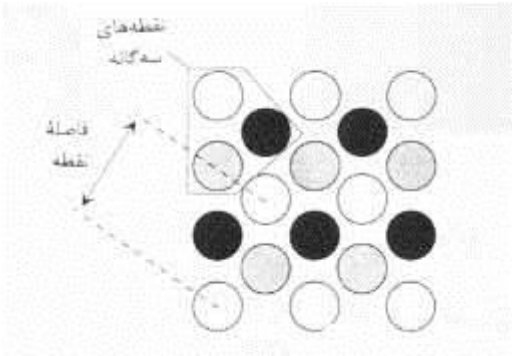
مقدمه مانیتور و کارت های گرافیکی

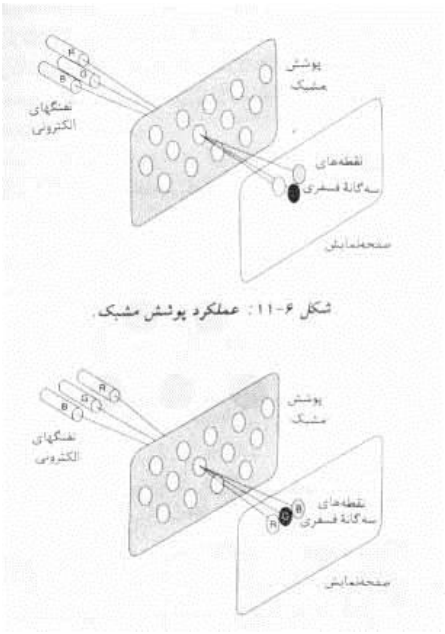
نوع اتصال دهنده : کانکتور D-15 ویدیویی

لامپ تصویر با صفحات انحراف دهنده



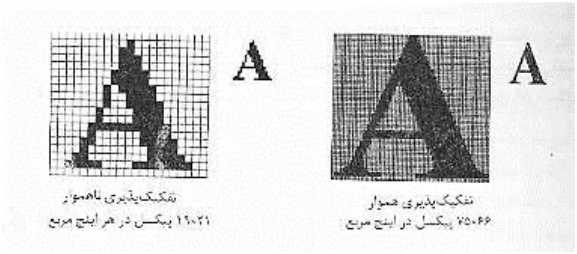
الکترونها توسط صفحات انحراف دهنده الکترومغناطیسی در جهت های افقی و عمودی هدایت می شوند . با برخورد این الکترونها به سطح مقابل مانیتور که توسط فسفر پوشانده شده است موجب درخشش آنها و در نتیجه ایجاد تصویر می شود . در مانیتورهای رنگی هر نقطه فسفری از سه رنگ ساخته شده اند در اینگونه مانیتورها سه نوار متفاوت برای حمل سه شعاع الکترونی لازم است (مانیتورهای قدیمی) اما در مانیتورهای جدیدتر تنها یک نوار هر سه رنگ را ایجاد می کند . صفحه مانیتور دارای گروهی خطوط افقی نزدیک به هم است که raster نامیده می شود . هر خط شامل صدها نقطه موسوم به پیکسل است که خود شامل سه نقطه فسفری قابل روشن شدن است .





بسته به رزولوشن تصویر موجود هر پیکسل ممکن است چندین نقطه سه گانه را شامل شود .

رزولوشن مانیتور با دو عدد نشان داده می شود اولین عدد تعداد پیکسل های موجود در خطوط عمودی و دومی افقی است مثلاً برای یک مانیتور با توجه به کارت گرافیک تعداد پیکسل ها 720*350 ذکر شده است یعنی در هر خط ۷۲۰ پیکسل و در هر صفحه ۳۵۰ خط وجود دارد که در کل ۲۵۲۰۰۰ پیکسل را دارا خواهد بود . تعداد پیکسل ها عامل مهمی در ارزیابی رزولوشن می باشد . رزولوشن مهم ترین مشخصه مانیتور است .



اندازه صفحه نمایش مانیتور : بر حسب قطر صفحه نمایش لامپ تصویر

گام نقطه : عبارتست از فاصله بین پیکسل های مجاور یکدیگر بر حسب میلی متر که هر چقدر کمتر باشد وضوح تصویر بالاتر است . (این عامل همانند چاپگرهای لیزری بصورت تعداد نقاط در اینچ مربع مثلاً 300dpi داده می شود) همانطور که در اشکال فوق دیدیم گام نقطه در مانیتورهای رنگی بصورت فاصله بین دو نقطه همرنگ توصیف می شود .

$$(\text{اندازه قطر تصویر})^2 = (\text{گام نقطه} * \text{تعداد پیکسل های عمودی})^2 + (\text{گام نقطه} * \text{تعداد پیکسل های افقی})^2$$

رابطه فوق فیثاغورثی است که چون اندازه گام بر حسب میلی متر است بنابراین اندازه تصویر بدست آمده را بایستی در 0.039 ضرب تا بر حسب اینچ بدست آید .

مثال : سازنده ای رزولوشن مانیتوری را 1024*768 با گام 0.28 اعلام کرده است . اندازه قطر تصویر روی صفحه نمایش را محاسبه کنید . (اندازه قطر تصویر از اندازه قطر لامپ تصویر کمی کوچکتر است)

$$(\text{اندازه قطر تصویر})^2 = (1024*0.28\text{mm})^2 + (768*0.28\text{mm})^2$$

$$\text{اندازه قطر (اینچ)} = 358\text{mm}*0.039 = 13.99 \Longrightarrow 14 \text{ inch}$$

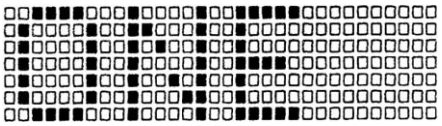
RAM نمایش تصویر :

تبادل اطلاعات بین مادربورد و مانیتور از طریق کارت گرافیک صورت می گیرد . اطلاعاتی که روی مانیتور نمایش داده می شود در حافظه ای به نام RAM نمایش تصویر VDR که بافر تصویر هم خوانده می شود ذخیره می گردد .

جهت نمایش اطلاعات ابتدا بایستی CPU آنرا در RAM تصویر ذخیره و سپس بوسیله یک کنترلگر اطلاعات از RAM تصویر خوانده و آنرا به سیگنالهای لازم برای نمایش روی صفحه تبدیل کند . این کنترلگرها نوعی پردازنده محسوب می شوند و چون خصوص کارهای تصویری طراحی شده اند کارهای متعلق به تصویر را بسیار کاراتر از پردازنده های همچون خانواده 80X86 انجام می دهند .

RAM تصویر بایستی توسط هر دو پردازنده (CPU و کنترلگر) قابل دسترس باشد . (استفاده از RAMهای دو پورتی البته کنترلگر تصویر اولویت بالاتری دارد) از 1MB حافظه قابل آدرس دهی ازآدرس A0000H الی BFFFFH برای RAM تصویرکنار گذاشته شده است . (128KB)

مانیتور در دوحالت متن و گرافیک قرار می گیرد . در حالت متن حروف (مجموعه ای از پیکسل ها در شکل زیر) و در حالت گرافیک پیکسل ها در اختیار کاربر قرار می گیرد . این پیکسل های عمودی و افقی بصورت گروهی هایی که آنها را جعبه کاراکتر می خوانند در می آیند . در شکل زیر کلمه ONE از سه جعبه کاراکتر تشکیل شده است . ابعاد جعبه کاراکتر از کارتی به کارت دیگر متفاوت است . جهت بدست اوردن کاراکترهای زیباتر اندازه جعبه کاراکتر باید افزایش یابد که بیانگر تعداد پیکسل های بیشتر خواهد بود .



وقتی مانیتور روشن می شود به طور پیش فرض در حالت متن قرار می گیرد . به عنوان مثال اگر یک مانیتور حداکثر 80×25 حرف داشته باشد (مجموعاً ۲۰۰۰ حرف) و هر حرف 8×8 پیکسل اشغال کند بنابراین مانیتور فوق حداکثر 640×200 پیکسل خواهد داشت یعنی مجموعاً ۱۲۸۰۰۰ پیکسل برای ۲۰۰۰ حرف وجود دارد .

بررسی چند کارت آداپتور ویدئویی

کارت گرافیکی رنگی CGA : این کارت قادر به تهیه هر دو مد متن و گرافیک بود . جعبه کاراکتر آن 8*8 بود وضوح متن به خوبی MDA نبود (دارای جعبه کاراکتر 14*9) با حداکثر ۸۰ ستون و ۲۵ سطر رزولوشنی برابر 640*200 ایجاد می کرد

80*8=640 , 25*8=200

Color Graphic Adaptor

مدهای تصویری برای CGA

AL	پیکسل	کاراکتر	جعبه کاراکتر	گرافیک/متن	رنگ	صفحات بافر	شروع
00H	320 x 200	40 x 25	8 x 8	متن	16*	8	B8000h
01H	320 x 200	40 x 25	8 x 8	متن	16	8	B8000h
02H	640 x 200	80 x 25	8 x 8	متن	16*	4	B8000h
03H	640 x 200	80 x 25	8 x 8	متن	16	4	B8000h
04H	320 x 200	40 x 25	8 x 8	گرافیک	4	1	B8000h
05H	320 x 200	40 x 25	8 x 8	گرافیک	4*	1	B8000h
06H	640 x 200	80 x 25	8 x 8	گرافیک	2	1	B8000h

* روشن‌تر شدن رنگ خاموش

RAM تصویر CGA : از آدرس B8000Hشروع و تا 16KB ادامه می یابد . با این وجود چون پیاده سازی 16KB به کمک RAM استاتیک گران تمام می شود از DRAMاستفاده می شود . جهت نمایش یک حرف ۲ بایت حافظه مورد نیاز است . یک بایت جهت کد اسکی و یک بایت جهت صفت حرف شامل رنگ و زمینه . در CGA آدرس های زوج کاراکتر مورد نمایش و آدرس های فرد صفت را ذخیره می کنند .

کاراکتر سطر ۱ ستون ۱	B8000
صفت برای کاراکتر سطر ۱ ستون ۱	B8001
کاراکتر سطر ۱ ستون ۲	B8002
صفت برای کاراکتر سطر ۱ ستون ۲	B8003
.....	.....
کاراکتر سطر ۲۵ ستون ۸۰	B87CE
صفت برای کاراکتر سطر ۲۵ ستون ۸۰	B87CF

در یک متن صفحه کامل 80*25=2000 حرف تشکیل شده است . این ۲۰۰۰ حرف 4KB حافظه مصرف می کند (2KB برای کاراکتر و 2KB برای صفت) بنابراین با 16KB می توان ۴ صفحه از متن را در هر زمان نگهداشت . در هر زمان یک صفحه قابل رویت است و هر کس می تواند به هر یک از سه صفحه دیگر بدون تاخیر سوئیچ کند . صفحه ای که در هر زمان در هر حال نمایش است را صفحه فعال گویند .

سوال : با محاسبه نشان دهید که مد 00 کارت CGA دارای ۸ صفحه فعال است .

بایت صفت حرف به شکل زیر می باشد :

		رنگ زمینه	رنگ حروف
صفت	BL	R G B	I R G B
شماره بیت	7	6 5 4	3 2 1 0

منظور از I در رنگ حرف Intensity به معنی شدت نور است و منظور از BL تنظیم چشمک زدن یا نزدن حرف می باشد (اگر این بیت یک باشد حرف چشمک زن و در صورتی که صفر باشد حرف غیر چشمک زن است)

اگر هر سه اشعه RGB فعال باشد (111) رنگ سفید و اگر هر سه غیر فعال باشد (000) رنگ سیاه بدست می آید . جدول زیر کلیه رنگ هایی که یک حرف می تواند داشته باشد را نشان می دهد :

IRGB	رنگ	IRGB	رنگ
1000	خاکستری	0000	سیاه
1001	آبی کمرنگ	0001	آبی
1010	سبز روشن	0010	سبز
1011	آبی آسمانی کمرنگ	0011	آبی آسمانی[1]
1100	قرمز روشن	0100	قرمز
1101	زرشگی روشن	0101	زرشگی[2]
1110	زرد	0110	قهوه‌ای
1111	سفید براق	0111	سفید

جدول زیر نیز برخی از رنگ های زمینه جهت حروف را نشان می دهد :

بایت صفت	ترکیب رنگ حروف	ترکیب رنگ زمینه	رنگ حروف	رنگ زمینه
عدد هگزا معادل	I R G B	BLRGB		
07	0 1 1 1	0 0 0 0	سفید	سیاه
01	0 0 0 1	0 0 0 0	آبی	سیاه
14	0 1 0 0	0 0 0 1	قرمز	آبی
23	0 0 1 1	0 0 1 0	آبی آسمانی	سبز
7D	1 1 0 1	0 1 1 1	زرشگی روشن	سفید
A8	1 0 0 0	1 0 1 0	خاکستری چشمک زن	سبز

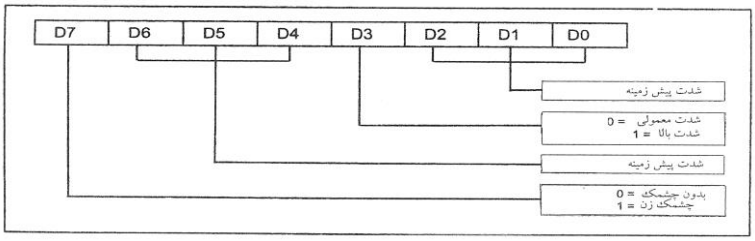
همانطور که گفتیم جهت نمایش یک حرف دو بایت مورد نیاز است که یک بایت کد اسکی می باشد . اما برخی کد های اسکی حرفی را نمایش نمی دهند که به آنها کدهای تنظیم مطلب گویند . کد اسکی 0DH باعث می شود مکان نما به آغاز سطر بازگردد(Carriage Return) و یا کد اسکی 0AH مکان نما را به سطر بعدی می برد(Line Feed) و کد اسکی 09H مکان نما را به ستون بعدی می برد .

دهدهی	اسکی (هگزا)	علامت
13	0D	CR
10	0A	LF
09	09	TAB

کارت گرافیکی تک رنگ MDA: Monochrome Display Adaptor

آدرس شروع	صفحات بافر	رنگ	متن/گرافیک	جعبه کاراکتر	کاراکترها	پیکسل ها	AL
B0000H	8	MONO	متن	9*14	80*25	720*350	07H

در این کارت گرافیک نیز آدرس های زوج کاراکتر و آدرس های فرد صفت را ذخیره می کنند .



بایت خصیصه در MDA

مثلاً کد 07H زمینه سیاه پیش زمینه معمولی بدون چشمک زدن را نشان می دهد .

صفت	مقدار
نشان داده نشود	00H
زیر خط	01H
پیش زمینه سفید معمولی بر روی پس زمینه سیاه	07H
پر رنگ و زیر خط	09H
پر رنگ	0FH
حالت معکوس (پیش زمینه سیاه روی پس زمینه سفید)	70H
چشمک زدن در حالت معمولی	87H
چشمک زدن در حالت پر رنگ	8FH
چشمک زدن در حالت معکوس	F0H

کارت گرافیکی رنگی پیشرفته EGA: این کارت بهترین مشخصات MDA و CGA را ارائه می دهد . این بورد می تواند 256KB حافظه را دارا باشد ولی فقط 128KB از A000H الی BFFFFH را بکار می برد . Enhanced Graphics Adaptor

مدهای تصویری برای EGA

AL	پیکسل ها	کاراکتر	جعبه کاراکتر	متن/گرافیک	رنگ	صفحات بافر	شروع
00H	320 x 350	40 x 25	8 x 14	متن	16*	8	B8000h
01H	320 x 350	40 x 25	8 x 14	متن	16	8	B8000h
02H	640 x 350	80 x 25	8 x 14	متن	16*	8	B8000h
03H	640 x 350	80 x 25	8 x 14	متن	16	8	B8000h
04H	320 x 200	40 x 25	8 x 8	گرافیک	4	1	B8000h
05H	320 x 200	40 x 25	8 x 8	گرافیک	4*	1	B8000h
06H	640 x 200	80 x 25	8 x 8	گرافیک	2	1	B8000h
07H	720 x 350	80 x 25	9 x 14	گرافیک	تک رنگ	4	B0000h
08H - 0CH						استفاده نشده برای PC, PS	
0DH	320 x 200	40 x 25	8 x 8	گرافیک	16	2/4	A0000h
0EH	640 x 200	80 x 25	8 x 8	گرافیک	16	1/2	A0000h
0FH	640 x 350	80 x 25	9 x 14	گرافیک	Mono	1	A0000h
10H	640 x 350	80 x 25	8 x 14	گرافیک	16	2	A0000h

* روشن تر شدن رنگ خاموش

! **توجه:** مدهای 08 ، 09 و 0A فقط بوسیله IBM PC Jr استفاده شده؛ 0B و 0C بوسیله EGA Video BIOS بکار رفته و در دسترس نیست.

آرائه گرافیکی چند رنگ MCGA: این سیستم دارای وضوح اصلاح شده و انتخاب رنگ بیشتر نسبت به CGA است و تا ۶۴ سطح روشنایی در مانیتورهای تک رنگ و ۲۶۲۱۴۴ رنگ در مانیتورهای رنگی که ۲۵۶ رنگ آن قابل انتخاب است را پشتیبانی می کند .

مدهای تصویری برای MCGA

AL	پیکسل ها	کاراکترها	جعبه کاراکتر	متن/گرافیک	رنگ	صفحات بافر	شروع
00H	320 x 400	40 x 25	8 x 16	متن	16*	8	B8000h
01H	320 x 400	40 x 25	8 x 16	متن	16	8	B8000h
02H	640 x 400	80 x 25	8 x 16	متن	16*	8	B8000h
03H	640 x 400	80 x 25	8 x 16	متن	16	8	B8000h
04H	320 x 200	40 x 25	8 x 8	گرافیک	4	1	B8000h
05H	320 x 200	40 x 25	8 x 8	گرافیک	4*	1	B8000h
06H	640 x 200	80 x 25	8 x 8	گرافیک	2	1	B8000h
11H	640 x 480	80 x 30	8 x 16	گرافیک	2	1	A0000h
13H	320 x 200	40 x 25	8 x 8	گرافیک	256	1	A0000h

* روشن تر شدن رنگ خاموش

آرائه گرافیکی تصویری VGA: این کارت قادر به پشتیبانی همه مدل های قبلی است . روی این بورد تا 1MB حافظه DRAM نصب شده است. Video Graphics Array

مدهای تصویری برای VGA

AL	پیکسل ها	کاراکترها	جعبه کاراکتر	متن/گرافیک	رنگ	صفحات بافر	شروع
00H	360 x 400	40 x 25	9 x 16	متن	16*	8	B8000h
01H	360 x 400	40 x 25	9 x 16	متن	16	8	B8000h
02H	720 x 400	80 x 25	9 x 16	متن	16*	8	B8000h
03H	720 x 400	80 x 25	9 x 16	متن	16	8	B8000h
04H	320 x 200	40 x 25	8 x 8	گرافیک	4	1	B8000h
05H	320 x 200	40 x 25	8 x 8	گرافیک	4*	1	B8000h
06H	640 x 200	80 x 25	8 x 8	گرافیک	2	1	B8000h
07H	720 x 400	80 x 25	9 x 16	متن	تک رنگ	8	B0000h
08H - 0CH							برای PC,PS استفاده نشده است
0DH	320 x 200	40 x 25	8 x 8	گرافیک	16	8	A0000h
0EH	640 x 200	80 x 25	8 x 8	گرافیک	16	4	A0000h
0FH	640 x 350	80 x 25	8 x 14	گرافیک	تک رنگ	2	A0000h
10H	640 x 350	80 x 25	8 x 14	گرافیک	16	2	A0000h
11H	640 x 480	80 x 30	8 x 16	گرافیک	2	1	A0000h
12H	640 x 480	80 x 30	8 x 16	گرافیک	16	1	A0000h
13H	320 x 200	40 x 25	8 x 8	گرافیک	256	1	A0000h

کارتهای گرافیک امروزی SVGA و AGP و PCI-Express

نکته : کارتهای on borad دارای هیچ حافظه تصویری نیستند .

نکاتی پیرامون مد گرافیک :

مشخصه هر پیکسل شامل مکان پیکسل و صفت پیکسل است . این دو مشخصه بایستی در RAM تصویر ذخیره شود . تعداد رنگ های نمایش داده شده در هر زمان همیشه 2^n است که n تعداد بیت های رنگ است مثلاً اگر ۴ بیت برای رنگ پیکسل در نظر گرفته شود در هر لحظه ۱۶ رنگ قابل نمایش است .

کارت گرافیک CGA : در مد گرافیک تعداد رنگ های پشتیبانی شده بستگی به وضوح تصویر دارد مثلاً با مراجعه به جدول مد این کارت می بینم که در مد گرافیک دو وضوح مختلف وجود دارد 320*200 و 640*200

وضوح متوسط 320*200 :در این مد جمعاً 64000 پیکسل وجود دارد (320*200=64000)

تعداد رنگها از تقسیم تعداد کل بیتهای RAM تصویر کارت بر تعدادکل پیکسل ها بدست می آید .

مثلاً RAM تصویر کارت CGA دارای 16KB بود بنابراین 16000*8bit=128000bit

$$\frac{128000\ bit}{64000\ pixel} = 2\ bit$$

همانطور که دیده می شود ۲ بیت برای رنگ بدست آمده که میشود ۴ رنگ متفاوت .

وضوح بالا 640*200 :در این مد جمعاً 128000 پیکسل وجود دارد (640*200=128000)

تصویر کارت CGA دارای 16KB بود بنابراین RAM 16000*8bit=128000bit

$$\frac{128000\ bit}{128000\ pixel} = 1\ bit$$

همانطور که دیده می شود 1 بیت برای رنگ بدست آمده که میشود ۲ رنگ متفاوت .

نتیجه اینکه در قبال مقدار ثابتی از حافظه تصویر با افزایش وضوح تعداد رنگ های پشتیبانی شده کاهش می یابد .

کاربردهای زبان اسمبلی در نمایش اطلاعات بر روی مانیتور در مد متن و گرافیک (INT 10H و INT 21H)

00H	Set Video Mode	تنظیم مد ویدئو
01H	Set Cursor size	تنظیم اندازه مکان نما
02H	Set Cursor Position	تنظیم محل استقرار مکان نما
03H	Return Cursor status	بازگرداندن وضعیت مکان نما
05H	Select Active Page	انتخاب صفحه فعال
06H	Scroll UP Screen	حرکت طوماری صفحه‌تصویر ببالا
07H	Scroll Down Screen	حرکت طوماری صفحه‌تصویر بپائین
08H	Read Character/Attribute	خواندن کاراکتر / صفت
09H	Display Character/Attribute	نمایش کاراکتر/صفت
0AH	Display Character	نمایش کاراکتر
0BH	Set Color Palette	تنظیم جعبه رنگ
0CH	Write Pixel dot	نوشتن پیکسل
0DH	Reed Pixel dot	خواندن پیکسل
0EH	Write in teletype mode	نوشتن در وضعیت تله تایپ
0FH	Get Current Video mode	اخذ مُد جاری ویدئو
10H	Access Palette Registers	دستیابی به ثباتهای مجموعه رنگها
11H	Access Character generator	دستیابی به تولیدکننده کاراکترها
12H	Select Alternative Routine	انتخاب روال جایگزین
13H	Display Character String	نمایش رشته کاراکتری
1BH	Return Video information	بازگرداندن اطلاعات ویدئو

❖ کاربرد سرویس های دستور INT 10H (سرویس های BIOS) برای حالت متن مانیتور

سرویس 00H دستور INT 10H : تغییر حالت یا مد مانیتور (دقت برحسب پیکسل = رزولوشن)

AH=00H

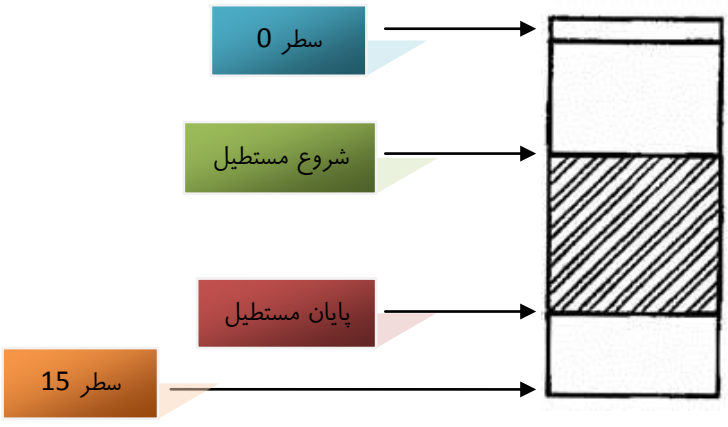
AL= شماره مد

INT 10H

حالت های تک رنگ حالت متن می باشند . اگر بخواهیم مانیتور حروف را بزرگتر بنویسد از مدی استفاده می کنیم که تعداد حروف کمتری داشته باشد مثلاً مد 00

سرویس 01H دستور INT 10H : تعیین اندازه مکان نما

مکان نما در حالت عادی به شکل یک خط تیره (▬) است اما می توان آنرا به شکل یک مستطیل (▮) حداکثر تا ۱۵ سطر تنظیم کرد



AH=01H

CH=شروع مستطیل

CL=پایان مستطیل

INT 10H

سرویس 02H دستور INT 10H : تغییر محل مکان نما

AH=02H

شماره سطر (مختصات Y) =DH

شماره ستون (مختصات X) =DL

شماره صفحه فعال =BH

INT 10H

برخی محل های مکان نما :

محل مکان‌نما برای یک حرف در مانیتور	به صورت دهدهی		به فرم هگزا	
	سطر	ستون	سطر	ستون
گوشه سمت چپ بالا	00	00	00H	00H
گوشه سمت راست بالا	00	79	00H	4FH
وسط مانیتور	12	39,40	0CH	27H,28H
گوشه سمت چپ پائین	24	00	18H	00H
گوشه سمت راست پائین	24	79	18H	4FH

مثال : زیر برنامه ای بنویسید که به ترتیب :

الف) مد مانیتور را به مد ۷ تغییر دهد .

ب) مستطیل مکان نما را از سطر ۵ تا سطر ۱۲ تنظیم کند .

ج) مکان نما را در مرکز صفحه نمایش مستقر سازد .

```
MOV  AH,00H
MOV  AL,07H
INT 10H
;-------------
MOV  AH,01H
MOV  CH,05H
```

```
MOV  CL,0CH
INT 10H
;--------------
MOV  AH,02H
MOV  BH,00
MOV  DH,0CH
MOV  DL,27H
INT 10H
```

مثال : برنامه ای بنویسید که با بکار بردن سرویس 02H دستور INT 10H مکان نما را به ترتیب در موقعیتهای $A\begin{cases}3\\4\end{cases}$ و $B\begin{cases}5\\8\end{cases}$ و $C\begin{cases}10\\13\end{cases}$ و $D\begin{cases}6\\20\end{cases}$ قرار دهد.

```
PAGE 50,40
TITLE 'SET_4.ASM'
.MODEL SMALL
.STACK DW 32DUP(0)
.DATA
     ROW  DB  3,5,10,6
     CAL  DB  4,8,13,20
.CODE
MAIN PROC FAR
     MOV AX,@DATA
     MOV DS,AX
     MOV CX,4
     LEA SI,ROW
     LEA DI,CAL
BACK:SUB BH,BH
     MOV AH,02
     MOV DH,DI
```

```
        MOV DL,SI
        INT 10H
        INC SI
        INC DI
        LOOP BACK
        MOV AX,4C00H
        INT 21H
MAIN ENDP
        END MAIN
```

سرویس 03H دستور INT 10H : بررسی موقعیت و اندازه مکان نما

با کمک این سرویس می توان مختصات مکان نما (محل فعلی آن) و همینطور اندازه آنرا بدست آورد .

```
AH=03H
INT 10H
```

پس از اجرای دستورات فوق نتایج زیر حاصل می شود :

محل ذخیره شماره سطر=DH

محل ذخیره شماره ستون=DL

محل ذخیره شروع مستطیل=CH

محل ذخیره پایان مستطیل=CL

مثال : زیر برنامه ای بنویسید که با بکار بردن سرویس 03H محل و اندازه مکان نما خوانده شود و با بکار بردن سرویس 02H مکان نما به ستون بعدی رود .

```
MOV BH,00
MOV AH,03H
INT 10H
MOV AH,02H
INC DL
INT 10H
```

سرویس 05H دستور INT 10H : انتخاب صفحه فعال برای نمایش اطلاعات روی مانیتور

جهت فعال کردن هرکدام از صفحه فعال ها از دستورات زیر استفاده می کنیم

```
AH=05H
AL= شماره صفحه فعال (0,1,2,3)
INT 10H
```

سرویس 06H دستور INT 10H : پاک کردن و چرخش اطلاعات مانیتور رو به بالا

AH=06H

AL= تعداد خطوطی که باید پاک شود یا چرخش یابد (00 همه صفحه را پاک می کند)

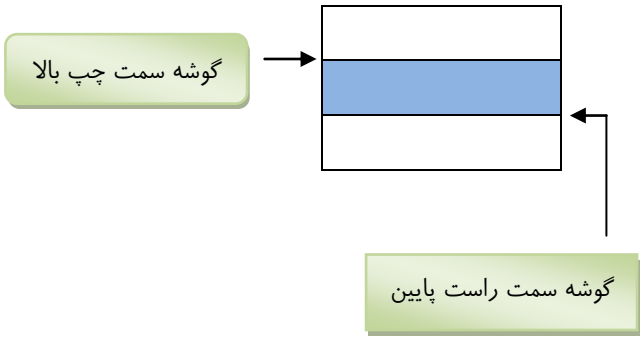
BH= رنگ زمینه و حروف

CL= شماره ستون گوشه سمت چپ بالا

CH= شماره سطر گوشه سمت چپ بالا

DL= شماره ستون گوشه سمت راست پایین

DH= شماره سطر گوشه سمت راست پایین



مثال : زیر برنامه ای بنویسید که مانیتور را از گوشه سمت چپ به مختصات (0,0) به تعداد 15 سطر پاک کند . از رنگ سیاه با زمینه سفید استفاده شود .

```
MOV AH,06H
MOV AL,0FH
MOV CX,0000
MOV DX,0F4F
MOV BH,70H
INT 10H
```

سرویس 07H دستور INT 10H : پاک کردن و چرخش اطلاعات مانیتور رو به پایین

- این سرویس مانند سرویس 06H می باشد .

سرویس 08H دستور INT 10H : خواندن حرف و رنگ مربوط به آن در محل فعلی مکان نما

```
AH=08H
INT 10H
```

با اجرای این دستور کد اسکی حرف در ثبات AL و شماره رنگ آن (بایت صفت) در ثبات AH ذخیره می شود .

سرویس 09H دستور INT 10H : نوشتن یک یا تعدادی حرف و رنگ مربوط به آنها در محل مکان نما بدون تغییر محل مکان نما

```
AH=09H
AL= کد اسکی حرف
BL= کد رنگ
BH= شماره صفحه فعال
CX= تعداد دفعاتی که حرف بایستی تکرار شود
INT 10H
```

مثال :زیربرنامه ای بنویسید که حرف Z را با رنگ قرمز و زمینه آبی در محل فعلی مکان نما 100 بار پشت سرهم بنویسد .

```
MOV AH,09H
MOV AL,'Z'
MOV BL,14H
MOV BH,00
MOV CX,100
INT 10H
```

سرویس 0AH دستور INT 10H : نوشتن یک یا تعدادی حرف در محل مکان نما بدون تغییر رنگ و بدون تغییر مکان نما

این سرویس مانند سرویس قبلی است اما از رنگ فعلی تنظیم شده استفاده می کند .

AH=0AH

AL= کد اسکی

BH= شماره صفحه فعال

CX= تعداد دفعات تکرار حرف

INT 10H

مثال : زیر برنامه ای بنویسید که ۵۰ عدد کاراکتر ▲ را با رنگ قهوه ای و زمینه آبی روی مانیتور رسم کند .

```
MOV AH,09H
MOV AL,1EH
MOV BL,16H
MOV CX,50
INT 10H
```

مثال : زیر برنامه ای بنویسید که ابتداء صفحه نمایش را پاک کند . سپس حرف D را بارنگ فعلی نشان دهد .

```
MOV AH,06H
MOV BH,00
MOV AL,00
MOV CX,0000
MOV DX,184FH
INT 10H
;------------
MOV AH,0AH
MOV BH,00
MOV AL,44H
MOV CX,1
INT 10H
```

سرویس 0EH دستور INT 10H : نوشتن یک حرف روی مانیتور و تغییر محل خودکار مکان نما

AH=0EH

AL=کد اسکی

BL= کد رنگ

BH= شماره صفحه مانیتور

INT 10H

مثال : حرف R را روی مانیتور با رنگ دلخواه نمایش دهید .

```
MOV AH,0EH
MOV AL,'R'
MOV BL,07H
MOV BH,00
INT 10H
```

نکته : کدهای خاص این سرویس عبارتند از

BEEP:07H

BACKSPACE :08H

0AH : یک خط فاصله LF

0DH : بازگشت به آغاز سطر CR

مثال : زیر برنامه ای بنویسید که بلند گوی کامپیوتر را روشن کند .

```
MOV AH,0EH
MOV AL,07H
INT 10H
```

سرویس 0FH دستور INT 10H : تعیین حالت یا مد مانیتور

AH=0FH

INT 10H

با اجرای این دستورات نتایج زیر بدست می آید :

محل ذخیره شماره مد=AL

محل ذخیرهتعداد حروف مانیتور در ی سطر =AH

محل ذخیره شماره صفحه=BH

❖ کاربرد سرویس های دستور INT 21H برای حالت متن مانیتور

سرویس 02H دستور INT 21H : نمایش یک حرف روی مانیتور و تغییر محل خودکار مکان نما

```
AH=02H
DL= کد اسکی
INT 21H
```

نکته : کدهای خاص این سرویس

LF:0AH

CR:0DH

سرویس 09H دستور INT 21H : نمایش یک رشته اطلاعات یا یک پیغام روی مانیتور

```
AH=09H
DX= آفست شروع رشته
INT 21H
```

نکته : در تعریف رشته مورد نمایش علامت '$' در پایان رشته لازم است در این صورت رشته از چپ به راست روی مانیتور نوشته می شود و نمایش حروف وقتی به علامت فوق می رسد متوقف می شود .

مثال : برنامه ای بنویسید که پیغام TEST را روی مانیتور نمایش دهد (با استفاده از سرویس 02 دستور INT 21H)

```
        .DATA
PEG   DB   'TEST'
        .CODE
```

```
        .
        .
        MOV AH,02
        MOV CX,4
        LEA DI,PEG
BACK:   MOV DL,[DI]
        INT 21H
        INC DI
        LOOP BACK
        .
        .
```

مثال : برنامه ای بنویسید که ابتدا مکن نما به خط بعدی رود و سپس به آغاز همان سطر بازگردد و سرانجام پیغام ENTER YOUR NAME روی مانیتور نمایش یابد .

```
        .DATA
  MSG   DB   0AH,0DH,'ENTER YOUR NAME','$'
        .CODE
        .
        .
        MOV AH,09H
        LEA DX,MSG
        INT 21H
        .
        .
```

مثال : برنامه ای بنویسید که

الف) مانیتور را پاک کند (از رنگ قهوه ای با زمینه آبی استفاده شود)

ب) مانیتور را در حالت 25×40 تنظیم کند .

ج) مکان نما را به سطر ۱۰ ستون ۲۰ ببرد .

د) پیغام های زیر را روی مانیتور نمایش دهد :

```
MY NAME IS :
WHAT IS YOUR NAME :
```

```
PAGE 100,120
TITLE 'TEST_2.ASM'
.MODEL SMALL
.STAK DW 32 DUP(0)
.DATA
MESSAGE DB 'MY NAME IS :'
          'WHAT IS YOUR NAME :' ,'$'
.CODE
START PROC FAR
      MOV AX,@DATA
      MOV DS,AX
;----------------------
      MOV AH,07H
      MOV AL,00
      MOV BH,16H
      MOV CX,0000
      MOV DX,184FH
      INT 10H
;-------------------------
      MOV AH,00
      MOV AL,1
```

```
        INT 10H
;-------------------------
        MOV AH,02
        MOV BH,00
        MOV DH,10
        MOV DL,20
        INT 10H
;---------------------------
        MOV BH,09H
        LEA DX,MESSAGE
        INT 21H
;-------------------------
        MOV AH,4CH
        INT 21H
START ENDP
        END START
```

مثال : برنامه ای بنویسید که ابتدا مانیتور را پاک کند . سپس مکان نما را در مختصات x=30 و y=40 قرار دهد . سپس عبارت good را روی مانیتور نمایش و سرانجام مکان نما را در سطر بعدی قرار دهد .

پاک کردن مانیتور CLEAR
قرار دادن مکان نما در نقطه مورد نظر SETCUR1
نوشتن عبارت WRITE
مکان نما در سطر بعدی SETCUR2

```
PAGE 70,100
TITLE 'TEST_B.ASM'
.MODEL SMALL
.STACK DW 128 DUP(?)
```

```
.DATA
MS DB 'GOOD BYE','$'
.CODE
MAIN PROC FAR
MOV AX,@DATA
MOV DS,AX
MOV ES,AX
CALL CLEAR
CALL SETCUR1
CALL WRITE
CALL SETCUR2
MOV AX,4C00H
INT 21H
MAIN ENDP
;-----------------
CLEAR PROC NEAR
MOV AX,600H
MOV BH,00
MOV CX,0000
MOV DX,184FH
INT 10H
RET
CLEAR ENDP
;------------------
SETCUR1 PROC NEAR
MOV AH,02
MOV DX,281EH
INT 10H
RET
```

```
SETCUR1 ENDP
;--------------------
WRITE PROC NEAR
MOV AH,09H
LEA DX,MS
INT 21H
RET
WRITE ENDP
;------------------
SETCUR2 PROC NEAR
MOV AH,02
MOV DL,0AH
INT 21H
RET
SETCUR2 ENDP
;--------------------
END MAIN
```

رسم خطوط با استفاده از سرویس 09H دستور INT 10H و INT 21H

کدهای اسکی 80H الی FFH جهت حروف و علامات خاصی بکار می روند .

جدول (۱۲-۶) نمایش کُدهای اسکی خطها و علامات

کد اسکی	طرز نمایش روی مانیتور
C4H	─
B3H	│
DAH	┌
BFH	┐
C0H	└
D9H	┘
C3H	├
B4H	┤
C2H	┬
C1H	┴
C5H	┼
B0H	▒
B1H	▒
B2H	▓
DBH	▓
D6H	╓
B7H	╖
D3H	╙
BDH	╜
C7H	╟
B6H	╢
D2H	╥
D0H	╨
D7H	╫
CDH	═
BAH	║
C9H	╔
BBH	╗
C8H	╚
BCH	╝
CCH	╠
B9H	╣
CBH	╦
CAH	╩
CEH	╬
DFH	▀
DDH	▌
DEH	▐
DCH	▄
D5H	╒
B8H	╕
D4H	╘
BEH	╛
C6H	╞
B5H	╡
D1H	╤
CFH	╧
D8H	╪

مثال : زیر برنامه ای بنویسید که بتواند یک خط افقی با ۲۰ خط تیره به رنگ زرد با زمینه سفیدرسم کند . کد اسکی خط تیره را C4H درنظر بگیرید .

```
MOV AH,09H

MOV AL,0C4H

MOV BL,0EH

MOV BH,00

MOV CX,20

INT 10H
```

مثال : مساله قبل را با دستور INT 21H تکرار کنید .

```
.DATA

LINE DB 20 DUP ('-'),'$'

.

.

.CODE

.

.

MOV AH,09H

LEA DX,LINE

INT 21H
```

مثال : برنامه ای بنویسید که بتواند یک مستطیل با طول ۲۵ کاراکتر و عرض ۸ کاراکتر رسم کند . این مستطیل از سمت چپ ۱۰ کاراکتر با لبه مانیتور فاصله دارد . کد 0DH مکان نما را به سطر بعدی و 0AH مکان نما را به آغاز همان سطر باز می گرداند .

```
.DATA
MOST   DB   10   DUP(' '),0DAH,25 DUP(0C4H),0BFH,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0B3H,25 DUP(' '),0B3H,0DH,0AH
       DB   10   DUP(' '),0C0H,25 DUP(0C4H),0D9H,0DH,0AH,'$'
```

تمرینات شکل با نوشتن در داخل آنها

❖ کاربرد سرویس های دستور INT 10H (سرویس های BIOS) برای حالت گرافیک مانیتور

در مد گرافیک مکان نما حذف می گردد

تعداد پیکسل ها بسته به وضوح و کارت گرافیک برای انواع مانیتورها متفاوت است .

سرویس 00 دستور int 10H: قرار دادن مانیتور در حالت گرافیک

AH=00

شماره مد=AL

INT 10H

سرویس 0C دستور int 10H: روشن کردن یک پیکسل

AH=0CH

شماره رنگ=AL

شماره ستون (X)=CX

شماره سطر (Y)=DX

شماره صفحه فعال=BH

INT 10H

رنگ پیکسل	شماره رنگ	
	باینری	هگزا
سیاه	0000	0
آبی	0001	1
سبز	0010	2
آبی آسمانی	0011	3
قرمز	0100	4
زرشکی	0101	5
قهوه‌ای	0110	6
سفید	0111	7
خاکستری	1000	8
آبی کمرنگ	1001	9
سبز روشن	1010	0A
آبی روشن	1011	0B
قرمز روشن	1100	0C
زرشکی روشن	1101	0D
زرد	1110	0E
سفید براق	1111	0F

نکته : در مد سیاه و سفید جهت روشن کردن یک پیکسل از مد فوق با AL=1 پیکسل روشن و با AL=0 پیکسل خاموش می شود .

مثال : پیکسل به مختصات $\begin{cases}20\\30\end{cases}$ را با رنگ آبی کم رنگ روشن کنید .

```
MOV AH,0CH
MOV AL,09H
MOV CX,20
MOV DX,30
MOV BH,00H
INT 10H
```

مثال : زیر برنامه ای بنویسید که در مد گرافیک بتواند یک خط افقی از مختصات $\begin{cases}15\\30\end{cases}$ و به طول ۱۰۰ پیکسل با رنگ قهوه ای رسم کند .

```
       MOV CX,15
       MOV DX,30
BACK : MOV AH,0C
       MOV AL,06
       INT 10H
       INC CX
       CMP CX,115
       JNZ BACK
```

مثال : زیر برنامه ای بنویسید که در مد گرافیک بتواند یک خط عمودی از مختصات $\begin{cases}70\\120\end{cases}$ الی $\begin{cases}70\\220\end{cases}$ با رنگ قرمز روشن رسم کند .

```
       MOV CX,70
       MOV DX,120
BACK : MOV AH,0C
       MOV AL,0C
       INT 10H
       INC DX
       CMP DX,220
       JNZ BACK
```

مثال : زیر برنامه ای بنویسید که در مد گرافیک بتواند ده خط افقی از مختصات $\begin{cases}15\\40\end{cases}$ و به طول ۲۰ پیکسل زیر هم با رنگ قهوه ای رسم کند

```
        MOV CX,15
        MOV DX,40
BACK :  MOV AH,0C
        MOV AL,06
        INT 10H
        INC CX
        CMP CX,35
        JNZ BACK
        MOV CX,15
        SHL DX,1
        CMP DX,60
        JNZ BACK
```

توضیح برنامه :

جهت رسم یک خط افقی کافیست پیکسل های ستونهای یک ردیف را روشن کنیم (INC CX)

انتهای یک خط به طول ۲۰ پیکسل که از ستون ۱۵ شروع شده است ستون ۳۵ می باشد .

فاصله خطوط موازی از هم به فاصله دو پیکسل دو پیکسل می باشد دستور SHL DX,1 ثبات DX را در عدد ۲ ضرب می کند .

تمرین : رسم خط ۴۵ درجه با طول و مختصات اختیاری

سرویس 0BH دستور INT 10H : تغییر صفحه نمایش به رنگ آبی

مثال : زیر برنامه ای بنویسید که مانیتور با کارت گرافیکی CGA را در مد گرافیک با وضوح بالا تنظیم کند . سپس تمام صفحه را آبی کند .

```
MOV AH,00
MOV AL,06     ;640*200 High resolution
INT 10H
;----------------
```

```
MOV AH,0BH
MOV BH,00
MOV BL,01
INT 10H
```

سرویس 0DH دستور INT 10H : خواندن مشخصات یک پیکسل در مد گرافیک

این سرویس دقیقاً عکس سرویس 0CH می باشد .

AH=0DH

BH=شماره صفحه فعال

CX=ستون

DX=سطر

INT 10H

با اجرای دستورات فوق شماره رنگ پیکسل مورد نظر در ثبات AL ذخیره می گردد .

تمرین ۱ : تابع AH=03 از وقفه INT 10H بکار رفته است . پس از آن نتایج DH=05 و DL=34 بدست آمده است . این به چه معناست ؟

تمرین ۲ : هدف از برنامه زیر برای مانیتور تک رنگ چیست ؟

```
MOV AH,02
MOV BH,00
MOV DX,0000
INT 10H
MOV AH,09
MOV BH,00
MOV AL,2AH
MOV CX,80
MOV BL,0F0H
INT 10H
```

تمرین ۳ : برنامه زیر کل صفحه نمایش را در کارت CGA پاک می کند . خطا ها را اصلاح کنید .

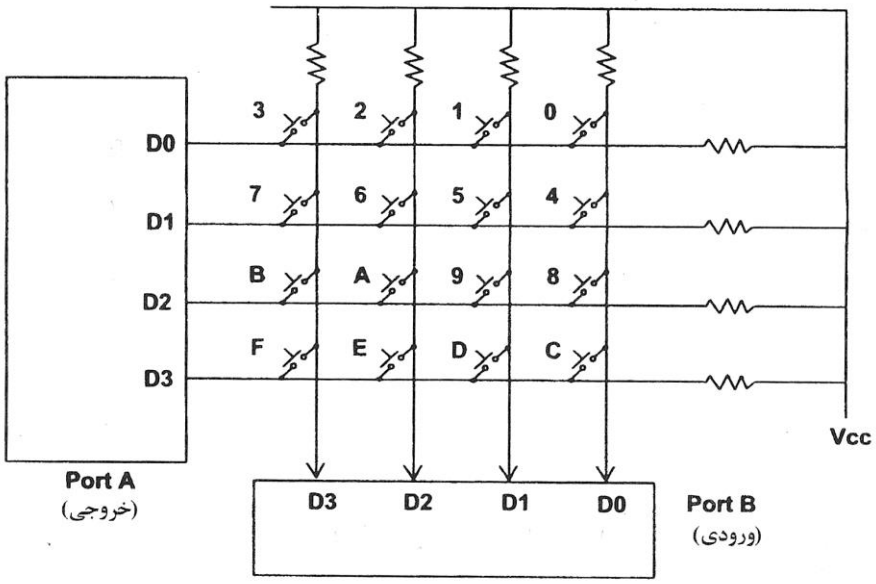
```
MOV AX,0600H
MOV BH,07
MOV CX,0000
MOV DX,184F
INT 10H
```

تمرین ۴ :

کاربردهای زبان اسمبلی جهت عملیت بر روی صفحه کلید

مقدمه سخت افزاری

اسکن و شناسایی کلید : شکل زیر یک صفحه کلید ماتریسی 4*4 می باشد که به دو پورت متصل است را نشان می دهد . ریزپردازنده صفحه کلید را دائماً برای تشخیص و شناسایی کلید فشرده شده اسکن می کند .



برای تشخیص کلید فشرده شده ریزپردازنده همه سطرها را صفر کرده و سپس ستون ها را می خواند . اگر داده خوانده شده از ستون D3-D0=1111 باشد کلیدی فشرده نشده است .

اگر یکی از بیتهای ستون حاوی صفر باشد مثلاً D3-D0=1101 کلیدی در ستون D1 فشرده شده است . پس از تشخیص فشردن کلید ریزپردازنده وارد فرایند شناسایی کلید می شود . ریزپردازنده ابتدا با صفر کردن سطر D0 و یک کردن مابقی سطرها و سپس خواندن ستون ها به دنبال پیدا کردن کلید فشرده شده در سطر می گردد . اگر داده خوانده شده تماماً یک باشد کلید در آن سطر فشرده نشده و فرایند به سطر بعدی منتقل می شود . این عمل تا شناسایی سطری که کلید در آن فشرده شده ادامه می یابد . پس از شناسایی سطر یافتن ستونی که کلید به آن متعلق است کار بعدی می باشد .

مثال : در هر یک از حالات زیر کلید فشرده شده را بیابید . (کد های زیر کد اسکن می باشند)

الف : D3-D0=1110 برای سطر و D3-D0=1011 برای ستون

ب : D3-D0=1101 برای سطر و D3-D0=0111 برای ستون

جواب : کلید ۲ و کلید ۷

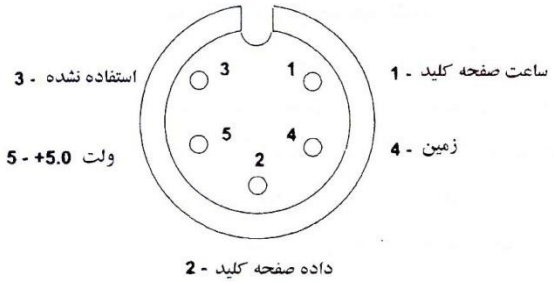
در IBM PC از یک میکروکنترولر (Intel 8042) برای تشخیص و شناسایی کلیدها استفاده می شود . در IBM PC فشردن کلید یک کد اسکن و رها کردن کلید کد اسکن دیگری دارد (کد رها کردن همیشه 128 + کد اسکن کلید فشرده شده است) مثلاً اگر کلیدی دارای کد اسکن 06 باشد و فشرده شود کد قطع آن برابر 06+80H=86H خواهد شد .

صفحه کلیدهای امروزی دارای 101 یا 104 کلید هستند :

الف – کلیدهای A-Z و ارقام 9-0 و علاماتی مانند ؟ > # % $

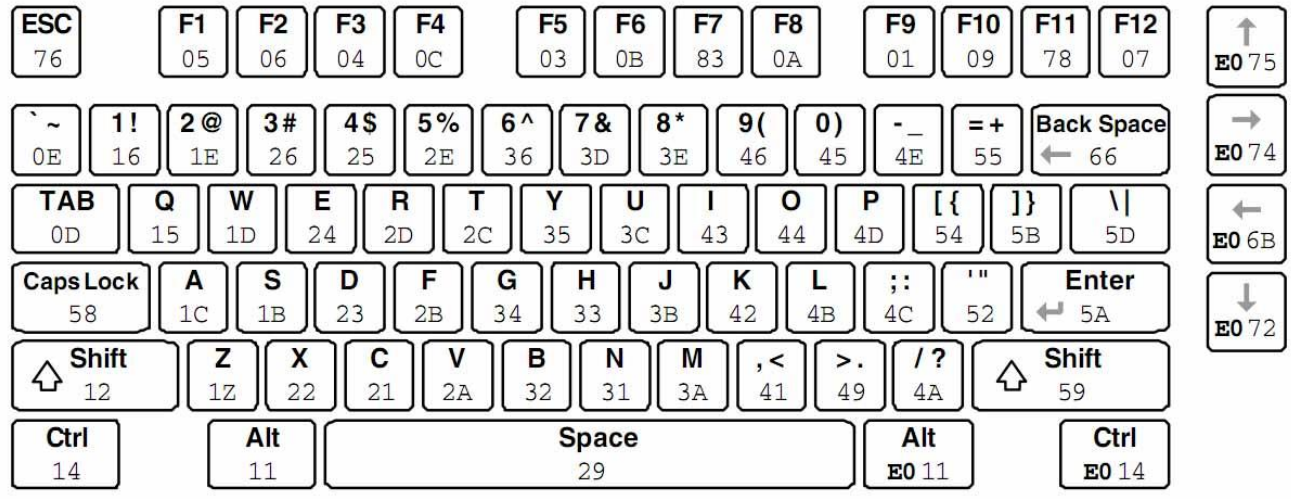
ب – کلیدهای تابع F1 - F12 بعلاوه Ctrl Shift Alt و کلیدهای Page UP و...

ج – کلیدهای عملیاتی Enter, Esc و ....



رخداد فشردن کلید در IBM PC :

پس از فشردن کلید کد اسکن آن توسط میکروکنترولر صفحه کلید تولید می شود . همزمان بطور خودکار دستور INT 9 فعال می شود . این وقفه کد اسکن کلید فشرده شده را خوانده و کد اسکی لازم را تولید می کند و سپس کد اسکی و کد اسکن را در مکانی از حافظه به نام بافر صفحه کلید که در محدوده داده BIOS قرار دارد ذخیره می کند . سپس می توان بوسیله دستورات INT 16H و INT 21H به این دو کد دست یافت .



Keyboard Scan Codes

توابع مربوط به INT 16H	
خواندن یک حرف بدون نمایش	00H
بررسی فشار دادن کلید	01H
گزارش وضعیت برخی کلیدها	02H
تنظیم نرخ کارکترهای تایپ (سرغت تایپ)	03H
نوشتن در بافر صفحه کلید	05H
خواندن کاراکتر از صفحه کلید	10H
تعیین وجود کاراکتردر بافر	11H
باز گرداندن وضعیت کلیدهای شیفت صفحه کلید	12H

توابع مربوط به INT 21H	
خواندن یک حرف بدون نمایش	01H
تشخیص فشار دادن کلید	06H
ورودی مستقیم از صفحه کلید بدون نمایش آن	07H
ورودی از صفحه کلید بدون نمایش آن	08H
ورودی از بافر صفحه کلید	0AH
تست وضعیت صفحه کلید	0BH
پاک کردن بافر صفحه کلید واجرای عمل ورودی	0CH

سرویس های دستور INT 21H مربوط به صفحه کلید :

سرویس 01H دستور INT 21H : خواندن یک حرف از صفحه کلید و نمایش آن روی مانیتور (حساس به ctrl+break)

```
AH=01H
INT 21H
```

پس از اجرای دستورات فوق کامپیوتر منتظر می ماند تا کلیدی فشار داده شود . به محض فشار کلیدی کد اسکی آن در ثبات AL ذخیره می شود و همزمان حرف مربوطه روی مانیتور نمایش داده می شود . این سرویس با فشار ctrl+break از دستور فوق جلوگیری می کند .

سرویس 06H دستور INT 21H : تشخیص فشار دادن کلید

از این سرویس می توان برای تشخیص اینکه آیا کلیدی فشار داده شده است یا خیر استفاده نمود (بدون نمایش حرف)

```
AH=06H
DL=FFH
INT 21H
```

پس از اجرای دستورات فوق اگر کلیدی فشار داده شده باشد کد اسکی آن در ثبات AL ذخیره می شود (ZF=0) ولی اگر کلیدی فشار داده نشده باشد AL=0 (ZF=1) می شود .

سرویس 07H دستور INT 21H : خواندن یک حرف از صفحه کلید بدون نمایش آن روی مانیتور

این سرویس مانند سرویس 01 می باشد اما حرف را نشان نمی دهد معمولاً از این سرویس جهت وارد کردن Password که نباید روی مانیتور نمایش داده شود استفاده می شود . پس از اجرای دستورات زیر کد اسکی کلید فشار داده شده در ثبات AL ذخیره می شود .

```
AH=07H
INT 21H
```

مثال : برنامه ای بنویسید که ابتدا مانیتور را پاک کند سپس مکان نما را در مرکز صفحه نمایش قرار دهد سپس پیغام This is a test of the display را روی مانیتور نمایش دهد سپس اگر کلید M روی کیبر فشرده شود ابتدا * را با استفاده از سرویس 02 دستور INT 21H نمایش دهد سپس * را با استفاده از سرویس 0EH دستور INT 10H نمایش دهد و سرانجام کامپیوتر بوق بزند .

```
.DATA
DATAS DB 'This is a test of display','$'

.CODE

;----------------
MOV AH,06
MOV CX,0000
MOV DX,184F
INT 10H
;-----------------
MOV AH,02
MOV BH,00
MOV DH,12
MOV DL,39
INT 10H
;-----------------
MOV AH,09H
LEA DX,DATAS
INT 21H
;-----------------
MOV AH,07H
```

```
INT 21H
CMP AL,'M'
JNZ QUIT ; end of program
;-------------------
MOV AH,02
MOV DL,'*'
INT 21H
;----------------------
MOV AH,0EH
MOV AL,'*'
INT 10H
;------------------
MOV AH,0EH
MOV AL,07H
INT 10H ; beep
```

مثال :برنامه ای بنویسید که ابتداء مانیتور را پاک کند سپس پیغام HELLO را روی مانیتور نمایش دهد سپس اگر کلید C فشرده شود ابتداء حرف D را با استفاده از سرویس 02 دستور INT 21H نمایش دهد سپس حرف K را با استفاده از سرویس 0EH دستور INT 10H نمایش دهد در غیر اینصورت به سیستم عامل برگردد .

```
PAGE 70,100
TITLE 'salam.ASM'
.MODEL SMALL
.STACK DW 64 DUP(?)
.DATA
SM DB 'HELLO,'$'
.CODE
MAIN PROC FAR
MOV AX,@DATA
MOV DS,AX
```

```
MOV ES,AX
CALL CLR MON
CALL WRT MES
CALL C DETECT
MOV AX,4C00H
INT 21H
MAIN ENDP
;-----------------
CLR MON PROC NEAR
MOV AX,600H
MOV BH,00
MOV CX,0000
MOV DX,184FH
INT 10H
RET
CLR MON ENDP
;-------------------
WRT MES PROC NEAR
MOV AH,09
LEA DX,SM
INT 21H
RET
WRT MES ENDP
;-------------------
C DETECT PROC NEAR
MOV AH,07
INT 21H
CMP AL,43H
JNZ QUIT
```

```
CALL DISPLAY
QUIT : RET
C DETECT ENDP
;--------------------
DISPLAY PROC NEAR
MOV AH,02
MOV DL,'D'
INT 21H
MOV AH,0EH
MOV AL,'K'
INT 10H
RET
DISPLAY ENDP
;--------------------
END MAIN
```

سرویس 0AH دستور INT 21H : خواندن یک رشته اطلاعات از صفحه کلید و قرار دادن آن در محلی در بافر حافظه و نشان دادن آن روی مانیتور

با فشار دادن کلیدهای صفحه کلید (مثلاً تایپ) کدهای اسکی آن ها در محلی از حافظه که در سگمنت داده تعریف می شود قرار می گیرد و حرف های نظیر کلیدهای فشار داده شده نیز بر روی مانیتور نشان داده می شود

AH= 0AH

DX= آفست رشته اطلاعات

INT 21H

سرویس 0BH دستور INT 21H : تست بافر صفحه کلید (حساس به Ctrl+Break)

AH=0BH

INT 21H

اگر در بافر صفحه کلید حرفی وجود داشته باشد آنگاه AL=FFH در غیر اینصورت AL=0

سرویس 0CH دستور INT 21H : پاک کردن بافر صفحه کلید و فراخوانی یک سرویس صفحه کلید

این سرویس با سرویس های 0AH , 07 , 06 , 01 دستور INT 21H کار می کند

AH=0CH

AL = شماره سرویس

DX=آفست آدرس بافر ورودی

INT 21H

با اجرای این دستور بافر صفحه کلید پاک و سرویس وقفه ای که شماره آن در AL قرار دارد اجرا می شود .

سرویس های دستور INT 16H مربوط به صفحه کلید

سرویس 00H دستور INT 16H : خواندن یک حرف از صفحه کلید بدون نمایش آن بر روی مانیتور

AH=00

INT 16H

اگر در بافر حرفی باشد کد اسکی آن در AL و کد اسکن آن در AH ذخیره می شود . اگر در بافر حرفی نباشد منتظر فشار کلید می ماند . جهت کلیدهای F1-F12 که کد اسکی ندارند AL=0 است .

سرویس 01H دستور INT 16H : بررسی فشار دادن کلید کیبرد

AH=01

INT 16H

اگر کلیدی فشار داده شود ZF=0 می شود در غیر اینصورت ZF=1

می توان بعد از این سرویس از سرویس 00 استفاده کرد که اگر کلیدی فشار داده شده باشد آنرا پیدا کند .

سرویس 02H دستور INT 16H : گزارش وضعیت بعضی کلیدها (قبل از اجرا کلید بایستی فشار داده شود)

```
AH=02
INT 16H
```

در ثبات AL کدی قرار می گیرد که بیتهای آن بصورت زیر است :

بیت ۰	اگر یک باشد Right+Shift فشار داده شده
بیت ۱	اگر یک باشد Left+Shift فشار داده شده
بیت ۲	اگر یک باشد Ctrl فشار داده شده
بیت ۳	اگر یک باشد Alt فشار داده شده
بیت ۴	اگر یک باشد Scroll Lock فشار داده شده
بیت ۵	اگر یک باشد Num LOCK فشار داده شده
بیت ۶	اگر یک باشد Caps Lock فشار داده شده
بیت ۷	اگر یک باشد Insert فشار داده شده

مثال : برنامه ای بنویسید که تست کند آیا کلید Caps Lock فشار داده شده است یا خیر ؟ اگر کلید موردنظر فشار داده شده باشد به برچسب next پرش انجام شود .

```
MOV AH,02
INT 16H
TEST AL,00000010B
JNZ next
```

سرویس 11H دستور INT 16H : مشخص می کند آیا حرفی در بافر صفحه کلید وجود دارد یا خیر .

```
AH=11H
INT 16H
```

اگر حرفی در بافر وجود نداشته باشد ZF=1 ولی اگر حرفی وجود داشته باشد ZF=0

مثال : زیربرنامه ای بنویسید که تست کند آیا کلید فشار داده شده حرف B است یا خیر ؟

```
MOV AH,00
INT 16H
CMP AL,'B'
JZ NEXT
```

مثال : برنامه ای بنویسید که ابتدا تست کند آیا کلیدی فشار داده شده است یا خیر ؟ اگر کلید فشار داده شده باشد ابتدا پیغام ENTER YOUR PASSWORD را روی مانیتور نمایش دهد و اگر کلیدی فشار داده نشده باشد از برنامه خارج شود .

```
        MOV AH,01
        INT 16H
        JZ NEXT
        LEA DX,PEG
        MOV AH,09
        INT 21H
NEXT:   MOV AX,4C00H
```

مثال :زیر برنامه ای بنویسید که با فشار کلید Alt بلندگوی کامپیوتر روشن شود و با فشار کلید Ctrl بلندگو خاموش شود .

دستور خروج باعث قطع بلندگو می شود (تفاوت دستور TEST و CMP)

```
BACK1 : MOV AH,02
        INT 16H
        CMP AL,03H
        JNZ BACK1
;--------------------
BACK2 : MOV AH,0EH
        MOV AL,07
        INT 10H
;--------------------
        MOV AH,02
        INT 16H
        CMP AL,02H
        JNZ BACK2
        MOV AX,4C00H
```

```
INT 21H
```

تمرین : برنامه ای بنویسید که با فشار یک کلید ، حرف مورد نظر بصورت زیر روی مانیتور نمایش داده شود .

حرف فشار داده شده : KEY

تمرین : برنامه ای بنویسید که ابتدا صفحه نمایش را پاک کند سپس مکان نما را در سطر ۱۵ و ستون ۲۰ قرار دهد سپس عبارت WHAT IS YOUR NAME ? را نمایش دهد و سرانجام پاسخی را از صفحه کلید دریافت و آنرا در سطر ۱۷و ستون ۲۰ نمایش دهد .

نکته : به کار بردن سرویس 0AH دستور INT 21H می خواهیم فضایی تا ۵۰ کاراکتر برای رشته در نظر بگیریم

```
USE_STR DB 51,51 DUP(?)
.
LEA DX,USE_STR
MOV AH,0AH
INT 21H
```

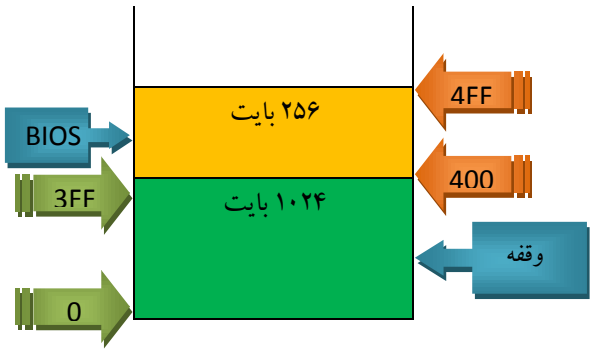
تمرین : برنامه ای بنویسید که جدول ضرب را چاپ کند .

فصل ۹

وقفه ها INTERRUPTS

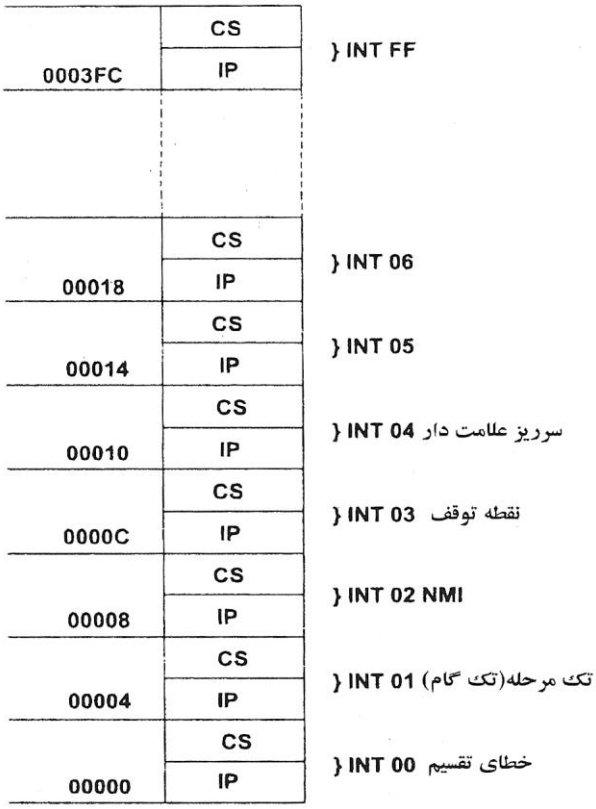
هر گاه وسایل جانبی کامپیوتر نیاز به کمک داشته باشند این نیاز خود را از طریق سیگنالهایی به نام وقفه به پردازنده اعلام می کنند . هنگامی که یک وسیله جانبی اقدام به ارسال وقفه می کند ، شماره شناسایی خود را که Type Code می باشد را نیز ارسال می کند . ریزپردازنده از این type code استفاده نموده و از آدرس محاسبه شده آدرس دیگری را می خواند . این آدرس جدید بردار وقفه نام دارد که در حقیقت آدرس برنامه ای است که کار آن وقفه را برعهده دارد . برنامه های سرویس دهنده وقفه در تراشه ROM ذخیره شده اند . در بسیاری از این کامپیوترها ROM به BIOS معروف است . اطلاعاتی که در این ناحیه حافظه قرار می گیرد عبارتست از آدرس پورت های سری و موازی ، اندازه حافظه ، تعداد درایو های نصب شده ، مد کاری مانیتور ، اندازه مکان نما و ... که سیستم عامل یا کاربر می تواند در موقع لزوم از آنها استفاده کند .

در کامپیوتر ها 1024 بایت اول حافظه (0 to 3FFH) اختصاص به جدولی به نام جدول بردار وقفه دارد . (از آدرس 400H به اندازه ۲۵۶ بایت برای اطلاعات BIOS درنظر گرفته شده است) این جدول با آدرس های ۳۲ بیتی به روتین های سرویس وقفه در کامپیوتر اشاره می کند . در مجموع ۲۵۶ وقفه مختلف وجود دارد که از میان 256 وقفه موجود ۳۲ تای اول یعنی از ۰ الی ۳۱ (0 to 1FH) بوسیله Intel رزرو گردیده است و کامپیوتر آنرا برای خودش درنظر می گیرد و ۳۲ وقفه بعدی (20H to 3FH) برای استفاده در سیستم عامل DOS درنظر گرفته شده و مابقی برای سایر موارد استفاده می گردد .نمونه هایی از وقفه های رزرو شده :



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

وقفه را می توان از طریق دستورالعمل های وقفه و یا توسط تجهیزات خارجی فعال نمود . وقتی ریزپردازنده یک وقفه دریافت می کند شماره وقفه را در ۴ ضرب کرده تا آدرس بردار وقفه در جدول را بدست آورد . (هر ۴ بایت یا ۳۲ بیت مربوط به یک وقفه ۱۶ بیت IP و ۱۶ بیت CS) سپس محتوای آدرس بدست آمده را در ثبات IP و ثبات CS قرار می دهد و شروع به اجرای دستورالعمل ها در آن آدرس می نماید .



مثال : آدرس وقفه INT 21H را پیدا کنید ؟

21H --→ 33 33*4=132

آدرس های ۱۳۲ و ۱۳۳ مربوط به IP و آدرس های ۱۳۴ و ۱۳۵ مربوط به CS است .

به طور کلی عملیاتی که در هنگام اجرای وقفه صورت می گیرد به شرح زیر است :

۱- برنامه اصلی کاربر متوقف شده و در آن وقفه صورت می گیرد .

۲- محتویات ثبات های CS, IP, FR در پشته ذخیره و عملیات خواسته شده انجام می شود .

۳- زیربرنامه سرویس وقفه ایجاد و اجرا می شود .

۴- مقادیر اولیه ثبات های CS, IP, FR از پشته دریافت و اجرای برنامه اصلی دنبال می شود .

وقفه های خارجی : پایه های ۱۷ و ۱۸ پردازنده ۸۰۸۶ مربوط به NMI و INTR است .

خط وقفه NMI (NON MASKABLE INTERRUPT) جهت تشخیص اشتباهات دستگاههای I/O و عملیات با اهمیت و فوری در نظر گرفته شده است .

خط وقفه (INTERRUPT REQUEST) INTR نیز از طریق آی سی کنترل کننده وقفه (8259A) توسط تایمر ، صفحه کلید ، پورت سری و موازی ، هارد دیسک و ... فعال می شود .

وقفه های نرم افزاری (داخلی) : این وقفه ها از طریق دستور INT N یا توسط سیستم عامل و یا هنگام تقسیم بر صفر یا ایجاد سر ریز و... فعال می شوند .

دستورالعمل های وقفه :

INT

INTO

IRET

- دستور INT N

N عددی بین 0 الی 255 است . این دستور مقدار IF و TF را صفر می کند . محتوای ثباتهای FR , IP , CS وارد پشته می شوند و آدرس بردار وقفه محاسبه می شود . (شماره وقفه ضرب در ۴) جهت اجرا محتوای دو بایت اول بردار وقفه در ثبات IP و محتوای دو بایت دوم آنرا در ثبات CS ذخیره می کند . این دستور شبیه به دستور CALL عمل می کند .

- دستور INTO

این دستور یک وقفه شرطی است و مخفف کلمات Interrupt Of Overflow می باشد . این دستور فاقد عملوند است و در صورتی که OF=1 شود ایجاد وقفه می کند . این دستور وقفه نوع ۴ را فعال و بیتهای IF و TF را صفر می کند .

- دستور IRET

این دستور نیز فاقد عملوند بوده و مخفف Interrupt Return است . این دستور برای وقفه شبیه به RET عمل می کند و باعث می شود که سه مقدار شانزده بیتی از پشته خارج شده و به ترتیب در ثباتهای IP , CS , FR شود. در انتهای روتین وقفه این دستور قرار می گیرد (جهت بازگشت به برنامه اصلی کاربر و ادامه آن)

برخی از انواع وقفه های داخلی :

الف) وقفه شماره صفر (تقسیم بر صفر) هنگامی که CPU در شرایطی قرار می گیرد که نتواند کاری انجام دهد این وقفه فعال می شود . یکی از این حالتها تقسیم بر صفر است . در شرایط زیر کامپیوتر قفل می شود :

```
MOV AX,70
SUB BL,BL
DIV BL
```

نکته : البته زمانی که جواب تقسیم (خارج قسمت) عددی بزرگ باشد نیز این حالت رخ می دهد .

```
MOV AX,0FFFFH
MOV BL,2
DIV BL
```

ب) وقفه شماره یک : اجرای یک دستور یک دستور(اگر TF=1 شود این وقفه فعال می شود بیت نهم FR)

دستورات زیر جهت یک کردن TF :

```
PUSHF
POP AX
OR AX,0000000100000000B
PUSH AX
POPF
```

دستور POP AX محتویات پرچم از حافظه پشته را در ثبات AX قرار می دهد .

دستورات زیر جهت صفر کردن TF :

```
PUSHF
POP AX
AND AX,1111111011111111B
PUSH AX
POPF
```

ج) وقفه شماره ۳ : توقف برنامه INT 3

شبیه به دستور MOV AX,4C00H

INT 21H

د) وقفه شماره ۴ : همان دستور INTO

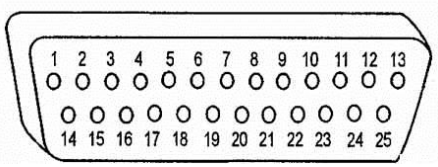
```
MOV AL,DATA1
MOV BL,DATA2
ADD AL,BL
INTO
MOV SUM,AL
```

در دستورات فوق اگر در خط سوم سرریز بوجود آید برنامه به INT 4 مراجعه و دستورات آن قسمت را اجرا می کند . اما اگر سر ریز بوجود نیاید دستور MOV SUM,AL اجرا خواهد شد .

فصل ۱۰

پورت موازی

از پورت موازی جهت اتصال چاپگر ، اسکنر ، هارد دیسک های خارجی ، کارت شبکه و... استفاده می شود . این پورت توسط شرکت IBM و به منظور اتصال یک چاپگر به کامپیوتر طراحی شده بود . زمانی که IBM در اندیشه طراحی و ارائه کامپیوترهای شخصی بود ضرورت استفاده از چاپگرهای شرکت centronicskdc نیز احساس گردید .



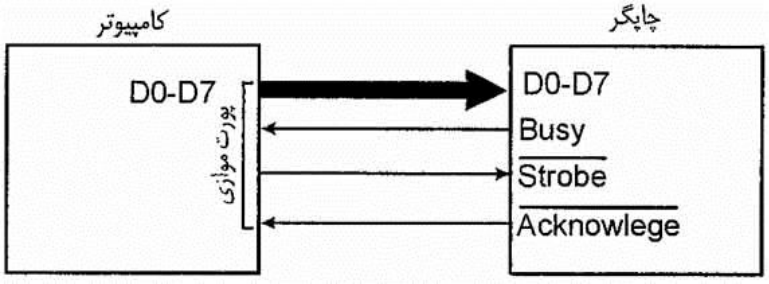
کانکتور DB-25

شماره پایه	نام پایه‌ها		
1	Strobe		سیگنال خروجی *
2	D0	بیت صفر داده	پایه‌های داده خروجی
3	D1	بیت ۱ داده	
4	D2	بیت ۲ داده	
5	D3	بیت ۳ داده	
6	D4	بیت ۴ داده	
7	D5	بیت ۵ داده	
8	D6	بیت ۶ داده	
9	D7	بیت ۷ داده	
10	Acknowledge		سیگنال تصدیق (ورودی)**
11	Busy		سیگنال مشغول (ورودی)
12	Out of Paper		کاغذ نیست (ورودی)
13	Select		چاپگر انتخاب شده (ورودی)
14	Auto Feed		کاغذ به‌طور خودکار بیاید (خروجی)
15	Error		اشتباه (ورودی)
16	Initialize Printer		مقدار اولیه دادن به چاپگر (خروجی)
17	Select Input		انتخاب ورودی
18 تا 23	زمین		

از آنجایی که این پورت اطلاعات هشت بیتی را به صورت موازی برای دستگاهها ارسال می کند به پورت موازی مشهور است . پورت موازی استاندارد قادر است اطلاعات هشت بیتی را با نرخ ۵۰ تا ۱۰۰ کیلو بایت در هر ثانیه ارسال کند .همانطور که از جدول فوق مشاهده می گردد جهت کاهش نویز هر سیگنال دیتا یک زمین جدا و مخصوص به خود دارد

مراحل ارتباط بین کامپیوتر و چاپگر :

۱- کامپیوتر جهت ارسال اطلاعات به چاپگر ابتدا سیگنال busy را بررسی می کند اگر این سیگنال صفر بود یعنی چاپگر اتصال دارد و آماده دریافت اطلاعات است (not busy) در غیر اینصورت چاپگر آماده نیست و کامپیوتر بایستی منتظر بماند .



۲- در صورتی که چاپگر آماده بود (busy=0) کامپیوتر یک بایت داده را روی پایه های D0 الی D7 قرار می دهد .

۳- کامپیوتر سیگنال strobe را فعال می کند که چاپگر اطلاعات را بگیرد .

۴- فعال سازی چاپگر موجب می شود که سیگنال busy=1 شود و این به معنی انتظار برای کامپیوتر است .

۵- بعد از اینکه چاپگر اطلاعات را دریافت کرد سیگنال تائید acknowledge را برای کامپیوتر فعال می کند که آمادگی خود را برای در یافت اطلاعات جدید به کامپیوتر اعلام کند . اگر چاپگر به دلایلی کار نکند سیگنال تائید را به کامپیوتر نخواهد فرستاد لذا پس از مدتی (حدود ۲۰ثانیه) کامپیوتر سیگنال strobe را غیر فعال کرده و اعلام اشتباه time out خواهد کرد .

دستور وقفه INT 17H عموماً جهت کنترل و برنامه ریزی چاپگر به کار می رود . معمولاً کامپیوتر می تواند چهار پورت موازی داشته باشد (LPT1 , LPT2 , LPT3 , LPT4) که چاپگرها به آنها متصل می شوند و دستور وقفه مورد نظر آدرس این پورت ها را با شماره های 3 , 2 , 1 , 0 می شناسد .

سرویس 00 دستور INT 17H : ارسال و چاپ یک حرف به چاپگر

```
AH=00
DX  =  شماره دستگاه چاپگر
AL  =  کد اسکی حرف
INT 17H
```

بعد از اجرای وقفه مورد نظر محتوای ثبات AH وضعیت چاپگر را نشان می دهد :

- بیت صفر خطای time out اشکال در ارسال اطلاعات (در صورتی که یک باشد)
- بیت ۱و۲ بدون استفاده
- بیت ۳ خطای انتقال اطلاعات I/O
- بیت ۴ فعال بودن چاپگر (انتخاب چاپگر)
- بیت ۵ نبود کاغذ

- بیت ۶ نشان دهنده ارسال سیگنال تائید acknowledge
- بیت ۷ نشان دهنده busy بودن چاپگر(busy=0 , not busy=1)

مثال : زیر برنامه زیر حرف A را جهت چاپ به پرینتر ارسال کرده و پرینتر آنرا چاپ می کند :

```
MOV AH,0
MOV DX,0
MOV AL,41H
INT 17H
```

سرویس 01 دستور INT 17H : مقدار اولیه دادن به پورت چاپگر و تعیین آدرس پورت موازی

این دستور یک پورت را انتخاب کرده چاپگر را متناسب با تنظیمات آن برای چاپ داده ارزشدهی اولیه می کند معمولاً این عمل برای اغلب چاپگرها وقتی که کامپیوتر روشن می شودبه طور خودکار صورت می گیرد .

```
AH=01
DX=( می توان هر عددی گذاشت ) شماره پورت یادستگاه چاپگر
INT 17H
```

با اجرای دستورات فوق چاپگر آماده کار می شود و وضعیت آن در ثبات AH مطابق سرویس 00 ذخیره می شود .

سرویس 02 دستور INT 17H : خواندن وضعیت چاپگر (تست چاپگر)

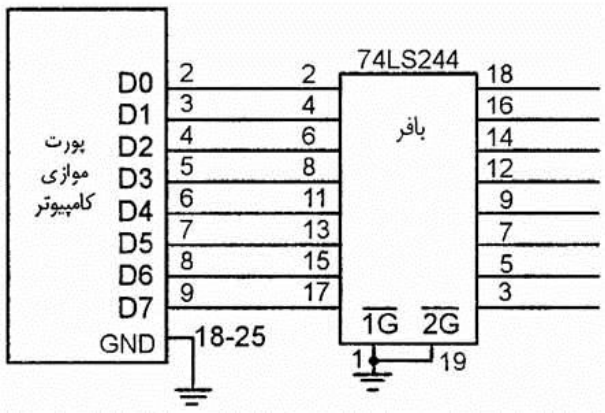
```
AH=02
DX=شماره دستگاه چاپگر
INT 17H
```

با اجرای دستورات فوق محتوای ثبات AH وضعیت چاپگر را نشان می دهد . (مطابق قبل)

نکاتی پیرامون time out : این اصطلاح به معنی ان است که پورت چاپگر نصب شده ولی آماده چاپ نیست . BIOS پس از تشخیص نصب چاپگر مرتباً برای مدت 20 ثانیه سعی در آماده یافتن آن برای چاپ داده می نماید . اگر باز هم چاپگر آماده نبود PC از آن صرفنظر می کند (Time out) و پیامی برای اعلام آن نمایش می دهد . مقدار زمانی که برای دریافت پاسخ از چاپگر در نظر گرفته شده در محدوده داده bios از 0040:0078 الی 0040:007B قرار دارد . مکان 0040:0078 زمان time out برای LPT1 و 0040:0079 زمان Time out برای LPT2 و ... هستند . در زمان بوت این مکان ها با 20 مقدار دهی اولیه می شود .

پورت موازی جهت ارسال اطلاعات

از پورت موازی می توان جهت ارسال اطلاعات جهت کارهای صنعتی استفاده نمود .



در این صورت کامپیوتر اطلاعات را به جای چاپگر به یک بافر می فرستد که می توان خروجی بافر را برای فرمان مدارهای منطقی ، میکروپروسسورها ، میکروکنترلرها و خلاصه ارتباط کامپیوتر با خارج استفاده نمود . جهت ارسال اطلاعات به پورت موازی بایستی آدرس آن را در اختیار داشت . این آدرس ها در ناحیه BIOS برطبق جدول زیر قرار دارند .

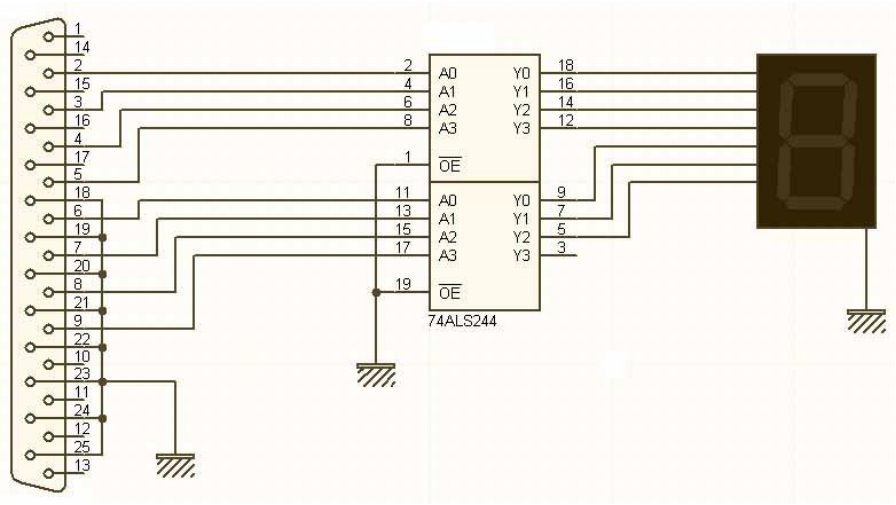
پورت‌های موازی	آدرس‌های ناحیه داده BIOS
LPT1	0040:0008-0040:0009
LPT2	0040:000A-0040:000B
LPT3	0040:000C-0040:000D
LPT4	0040:000E-0040:000F

زمانی که کامپیوتر روشن می شود برنامه بوتینگ کامپیوتر بررسی می کند که کدام پورت موازی در کامپیوتر نصب شده است . مثلاً وقتی کامپیوتر فقط پورت LPT2 را دارا باشد آدرس آنرا از ناحیه 0040:000Aو0040:000B حافظه برداشته و روی مانیتور می نویسد . حال با دستورات زیر و با کمک ثبات AL می توان اطلاعات را به پورت موازی ارسال نمود در این حالت سیگنال های کنترل Strobe و acknowledge تولید نمی شوند :

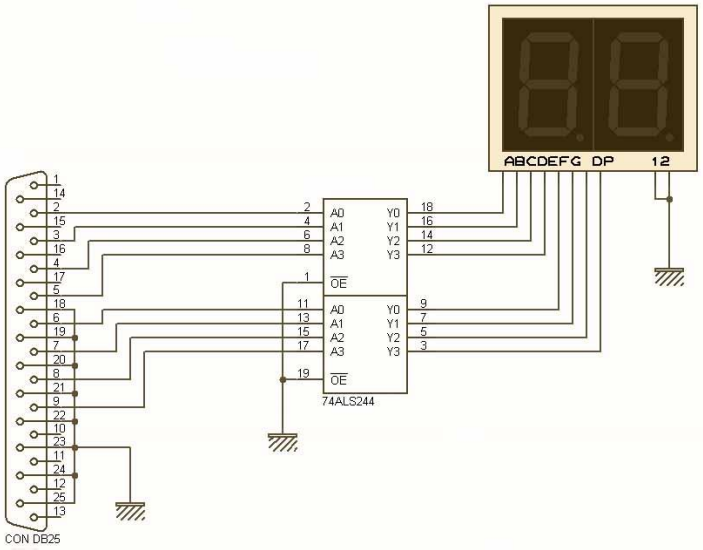
```
MOV AL,15H
MOV DX,3BCH
OUT DX,AL
```

با دستورات فوق عدد 15H به پورت موازی به آدرس 3BCH منتقل می شود .

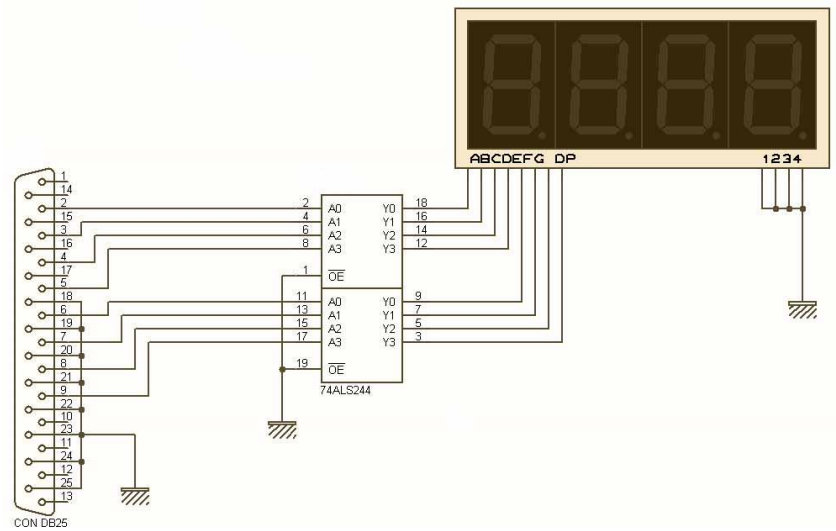
مثال : بر طبق سخت افزار زیر برنامه ای به زبان اسمبلی بنویسید که برای مدتی مشخص اعداد 0 الی 9 را بر روی 7-segment کاتد مشترک نمایش دهد .



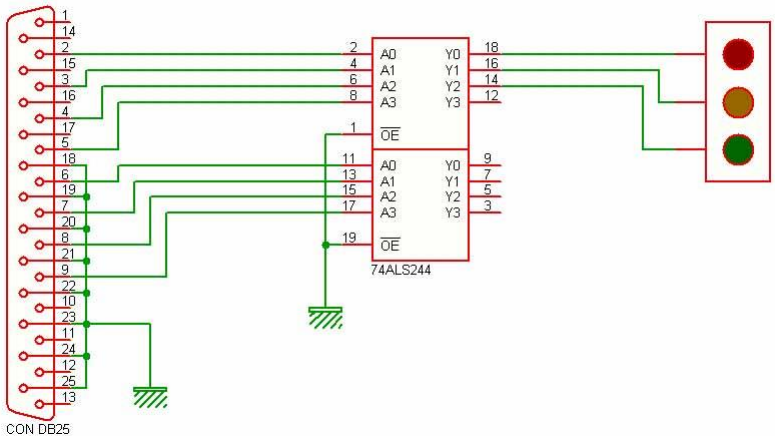
تمرین : بر طبق سخت افزار زیر برنامه ای به زبان اسمبلی بنویسید که عدد 7 . 5 را بر روی 7-segment کاتد مشترک نمایش دهد .



تمرین : بر طبق سخت افزار زیر برنامه ای به زبان اسمبلی بنویسید که عبارت ErOr را بر روی چهار عدد 7-segment کاتد مشترک نمایش دهد .



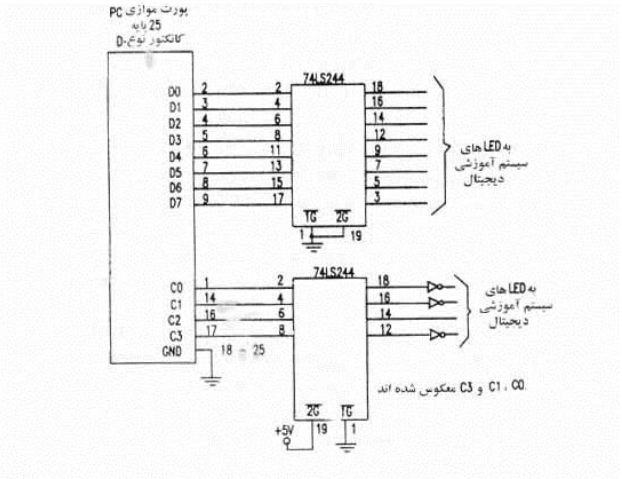
تمرین : برطبق شماتیک زیر برنامه ای بنویسید که چراغ راهنمایی و رانندگی را شبیه سازی کند .

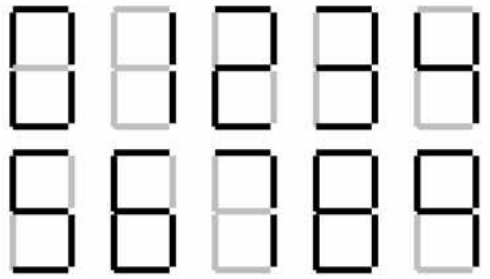


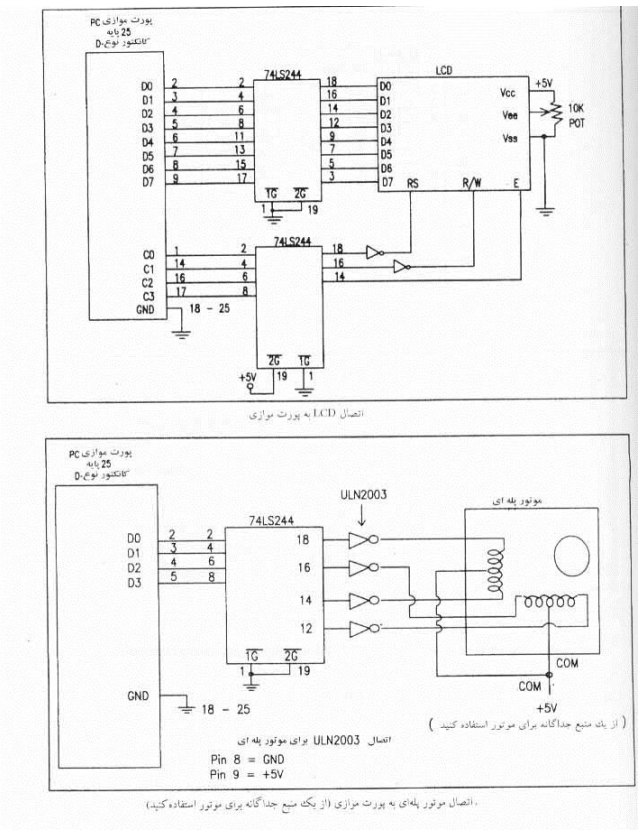
تمرین : برنامه ای بنویسید که با فشار کلید R بر روی کیبرد هشت عدد LED که به پورت موازی کامپیوتر متصل شده اند به صورت یک در میان روشن و خاموش شوند و با فشار کلید P حالت روش و خاموش بودن عوض شود و با فشار کلید Y همه LEDها روشن و با فشار کلید Z همه خاموش شوند .

تمرین : برنامه ای بنویسید که با فشار کلید enter موتوری dc که به پورت موازی کامپیوتر متصل شده در جهت عقربه های ساعت شروع به چرخش نماید و با فشار مجدد همان کلید در خلاف جهت عقربه های ساعت بچرخد .

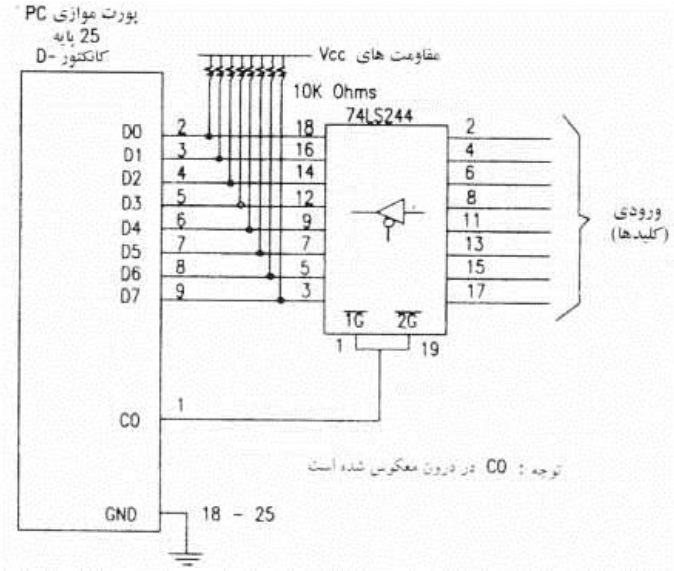
سوال : آیا می توانید یک step motor را که به پورت موازی کامپیوتر متصل شده در جهت عقربه های ساعت بصورت نیم پله به چرخش در آورید . شماتیک سخت افزاری آنرا رسم کنید .







بافر کردن پورت داده LPT به عنوان ورودی با استفاده از مقاومت های بالاکش



PRINTER PORT ADDRESSES	
PRINTER PORT	BASE ADDRESS
LPT1	378H OR 3BC
LPT2	278H OR 378
LPT3	278H

فصل ۱۱

برنامه ریزی ماوس

BIOS-IBM PC و DOS اولیه ماوس را پشتیبانی نمی کردند . به همین علت وقفه INT 33H بخشی از BIOS و یا DOS نیست . این وقفه بخشی از نرم افزار راه انداز است که به هنگام بوت PC نصب می شود .

تغییر محل ماوس با واحدی به نام MIKEYS (میکی یا مایکی) اندازه گیری می شود (حساسیت ماوس) این واحد به تعداد پیکسل های جابجا شده در هر اینچ از مسیر گفته می شود . ماوسی که بتواند به ازاء هر اینچ مکان نما را به اندازه ۲۰۰ پیکسل جابجا کند دارای حساسیت ۲۰۰ میکی است . ماوسی که حساسیت بالایی دارد نسبت به ماوسی که حساسیت پایینی دارد فقط در تعداد تعداد پیکسل های جابجا شده در طی یک مسیر مشابه با یکدیگر اختلاف دارند . حساسیت بالا در واقع نشاندهنده این است که با اعمال حرکتی اندک به ماوس مکان نمای آن فاصله بیشتری را طی خواهد کرد .

سرویس 00 دستور INT 33H : فعال کردن و پیدا کردن ماوس

```
AX=00
INT 33H
```

با اجرای دستورات فوق اگر ماوس وجود داشته باشد AX=FFFFH و ثبات BX تعداد دکمه های ماوس را نشان می دهد در غیر اینصورت AX=0 خواهد شد .

سرویس 01 دستور INT 33H : فعال یا غیرفعال کردن مکان نمای ماوس

مکان نما ظاهر می شود

```
MOV AX,01
INT 33H
```

مکان نما پنهان می شود

```
MOV AX,02
INT 33H
```

مثال : زیر برنامه ای بنویسید که ابتدا تست کند که آیا ماوس فعال است یا خیر ؟ در صورتی که ماوس فعال باشد تعداد کلیدهای ماوس را در متغیر KEY ذخیره کرده و سرانجام ماوس را روی صفحه مانیتور ظاهر کند .

```
Again : MOV AX,00
        INT 33H
        CMP AX,0
        JZ again
        MOV KEY,BX
        MOV AX,1
        INT 33H
```

سرویس 03 دستور INT 33H : تعیین محل مکان نمای ماوس

AX=03H

INT 33H

با اجرای دستورات فوق :

CX : محل ذخیره مختصات افقی بر حسب پیکسل

DX : محل ذخیره مختصات عمودی بر حسب پیکسل

BX : محل ذخیره وضعیت دکمه های ماوس (بیت ۰ و ۱ و ۲ ثبات مذکور نشان دهنده کلیک چپ و راست و وسط ماوس می باشد اگر یک باشد کلیک فشار داده شده است)

مکان نما در مد متن مانیتور به شکل مستطیل و در مد گرافیک به شکل یک فلش یا پیکان می باشد .

سرویس 04 دستور INT 33H : قرار دادن مکان نمای ماوس در یک موقعیت مشخص

AX=04

CX= مختصات افقی بر حسب پیکسل

DX= مختصات عمودی بر حسب پیکسل

INT 33H

زیر برنامه ای بنویسید که مکان نمای ماوس را در سطر ۱۰ ستون ۲۰ قرار دهد .

```
MOV AX,04
MOV CX,160    ;8*20
MOV DX,80     ;8*10
INT 33H
```

سرویس 05 دستور INT 33H : تشخیص وضعیت کلیدهای فشار داده شده ماوس

AX=05

BX=0 or 1 or 2

INT 33H

نتیجه دستورات فوق به صورت زیر است :

- بیت صفر ثبات AX اگر یک باشد یعنی کلیک چپ فشار داده شده است .
- بیت یک ثبات AX اگر یک باشد یعنی کلیک راست فشار داده شده است .
- بیت دو ثبات AX اگر یک باشد یعنی کلیک وسط فشار داده شده است .
- ثبات CX محل ذخیره مختصات عمودی و ثبات DX مختصات افقی مکان نما برای آخرین باری که کلیک ماوس فشار داده شده است می باشد .
- ثبات BX محل ذخیره تعداد دفعات کلیک فشار داده شده می باشد .

سرویس 06 دستور INT 33H : تشخیص وضعیت کلیک های فشار داده نشده

این سرویس مانند سرویس قبلی می باشد .

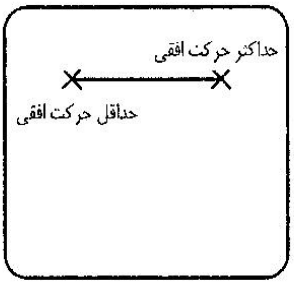
سرویس 07 دستور INT 33H : محدود کردن حرکت افقی مکان نمای ماوس

AX=07

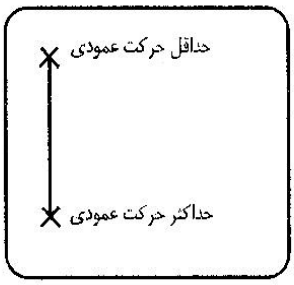
مختصات حداقل حرکت افقی =CX

مختصات حداکثر حرکت افقی =DX

INT 33H



محدود کردن حرکت افقی مکان‌نمای ماوس، در مانیتور



محدود کردن حرکت عمودی مکان‌نمای ماوس، در مانیتور

سرویس 08 دستور INT 33H : محدود کردن حرکت عمودی مکان نمای ماوس

AX=08

مختصات حداقل حرکت عمودی =CX

مختصات حداکثر حرکت عمودی =DX

INT 33H

سرویس 10 دستور INT 33H : محدود کردن حرکت مکان نمای ماوس در یک محدوده

AX=10

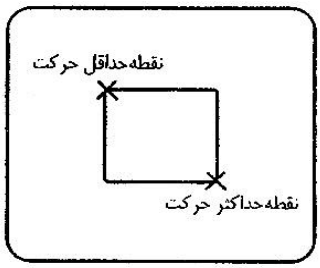
مختصات حداقل حرکت افقی =CX

مختصات حداقل حرکت عمودی =DX

مختصات حداکثر حرکت افقی (ستون) =SI

مختصات حداکثر حرکت عمودی (سطر) =DI

INT 33H



حرکت مکان‌نما در یک محدوده مخصوص مانیتور

مثال : زیر برنامه ای بنویسید که ابتدا ماوس را ظاهر کند . سپس محل مکان نمای ماوس را تعیین کند (مختصات سطر را در متغیر RAW1 و ستون را در CAL1 قرار دهد) سپس تست نماید که آیا کلیک وسط ماوس فشار داده شده است یا خیر ؟ در صورتی که این کلیک فشار داده شده باشد مختصات مکان نما را برحسب حرف (مد متن) تعیین کند و آنها را در RAW2 و CAL2 ذخیره کند و در صورتی که این کلیک فشار داده نشده باشد از برنامه خارج شود .

```
;----------
MOV AX,01
INT 33H
;-----------
```

```
MOV AX,03
INT 33H
MOV RAW1,CX
MOV CAL1,DX
TEST BX,0004H
JZ NEXT
MOV AX,CX
MOV BL,8
DIV BL
MOV ROW2,AL
MOV AX,DX
DIV BL
MOV CAL2,AL
NEXT :
MOV AX,4C00H
INT 21H
```

زیر برنامه ای بنویسید که مشخص کند آیا کلیک سمت چپ ماوس فشار داده شده است یا خیر ؟ اگر فشار داده شود بلندگو روشن شود .

```
MOV AX,05
MOV BX,0
INT 33H
;-----------
TEST AX,0
JZ NEXT
MOV AH,0EH
MOV AL,07
INT 10H
NEXT :
```

فصل ۱۲

برنامه ریزی دیسک

وقتی فایلی ایجاد می شود DOS یک شماره شانزده بیتی به آن اختصاص می دهد که به آن شناسه فایل گویند (FILE HANDLE)

سرویس 3CH دستور INT 21H : ایجاد فایل

جهت ایجاد و دستیابی فایل دو مرحله لازم است :

۱- ابتدا یک رشته ASCIIZ را در سگمت دیتا تعریف می کنیم (پسوند فایل بایستی .ASC باشد) که شامل نام فایل ، نام درایو و مسیر آن باشد و در انتها 00 قرار می دهیم .
۲- سپس در سگمنت کد از سرویس ذکر شده استفاده شود .

- اگر DOS بتواند فایل را ایجاد کند شناسه فایل ساخته شده و CF=0 می شود .
- اگر DOS نتواند فایل را ایجاد کند کد خطاایجاد خواهد شد و CF=1 می شود .

```
AH=3CH
CX= نوع فایل
DX= آفست رشته تعریف شده
INT 21H
```

- نوع فایل 0 برای NORMAL و 1 برای READ ONLY و 2 برای HIDDEN و 4 برای سیستمی DOS و ...
- با اجرای دستورات فوق شناسه فایل به ثبات AX منتقل می شود .

مثال : فایلی پنهان در مسیر MY DOCUMENT ایجاد نمایید . سپس شناسه آنرا به متغیر HANDLE_NEW منتقل کنید .

```
.DATA
NEW FOLDER DB 'C:\MY DOCUMENT\NEW FOLDER.ASC',00

.CODE

MOV AH,3CH
MOV CX,2
LEA DX,NEW FOLDER
```

```
INT 21H
MOV HANDLE_NEW,AX
```

سرویس 40H دستور INT 21H : ذخیره اطلاعات در فایل

AH=40H

CX= تعداد بایت هایی که می خواهیم ذخیره کنیم

BX= شناسه فایل

DX= آفست اطلاعاتی که می خواهیم در فایل ذخیره شود

INT 21H

- اگر CF=0 شود عملیات نوشتن و ذخیره موفقیت آمیز بوده است و AX برابر تعداد بایت های ذخیره شده می شود .
- اگر CF=1 شود عملیات نوشتن و ذخیره موفقیت آمیز نبوده است و AX برابر کد خطا می شود .

مثال : رشته THIS IS A TEST زیر را در پوشه NEW FOLDER ایجاد شده در مثال قبل ذخیره کنید .

```
.DATA
TEST_A DB 'THIS IS A TEST$'

.CODE

MOV AH,40H
MOV BX,HANDLE_NEW ; FROM LAST EXAMMPLE
MOV CX,14
LEA DX,TEST_A
INT 21H
```

نکته : از دستور فوق جهت چاپ کاراکترها توسط چاپگر نیز استفاده می شود بطوری که CX حاوی تعداد کاراکترهایی که بایستی چاپ شود و DX حاوی آدرس داده هایی که بایستی چاپ شود می باشد .

مثال : رشته THIS IS A TEST را چاپ کنید .

```
.DATA
HEADING DB 'THIS IS A TEST',0AH,0DH

.CODE

MOV AH,40H
MOV BX,HANDLE_NEW
MOV CX,14
LEA DX,HEADING
INT 21H
```

سرویس 3EH دستور INT 21H : بستن فایل

```
AH=3EH
BX= شناسه فایل
INT 21H
```

سرویس 3FH دستور INT 21H : خواندن فایل

```
AH=3FH
BX= شناسه فایل
CX= تعداد بایتهایی که بایستی خوانده شود
DX=آفست آدرس اطلاعاتی که بایستی خوانده شود
INT 21H
```

- اگر CF=0 شود عملیات نوشتن و ذخیره موفقیت آمیز بوده است و AX برابر تعداد بایت های خوانده شده می شود .
- اگر CF=1 شود عملیات نوشتن و ذخیره موفقیت آمیز نبوده است و AX برابر کد خطا می شود .

سرویس 3DH دستور INT 21H : باز کردن فایل

AH=3DH

AL= حالت

DX=آفست آدرس فایلی که می خواهیم باز شود

INT 21H

- ❖ اگر CF=0 شود عملیات نوشتن و ذخیره موفقیت آمیز بوده است و AX برابر شناسه فایل می شود .
- ❖ اگر CF=1 شود عملیات نوشتن و ذخیره موفقیت آمیز نبوده است و AX برابر کد خطا می شود .

➢ حالت 0 باز کردن فایل برای خواندن و حالت 1 باز کردن فایل برای نوشتن و حالت 2 باز کردن فایل برای نوشتن - خواندن

مثال : اطلاعات TEST_A مثال قبل را بخوانید .

```
MOV AH,3FH
MOV BX,HANDEL_NEW
MOV CX,14
MOV DX,TEST_A
INT 21H
```

مثال : زیر برنامه ای بنویسید که فایل NEW FOLDER برنامه قبل را برای نوشتن باز کند .

```
MOV AH,3DH
MOV AL,1
MOV DX,OFFSET_NEW FOLDER ; or lea dx,new folder
INT 21H
```

فصل ۱۳

روشهای تولید صوت و بررسی و دستیابی به ساعت و تاریخ

همانطور که می دانید کامپیوتر دارای پالس هایی از ساعت می باشد جهت عملکرد ریزپردازنده ولی برخی قسمتها مانند تاریخ ، بوق ، صدا و .... به فرکانس کمتری نیاز دارند . لذا سازنده در کامپیوترهای شخصی با هر فرکانسی ، فرکانس ثابت 1.19MHz را تولید کرده و در ورودی یک تایمر 8254 به نام PIT قرار داده است .

این تایمر دارای ۳ شمارنده ۱۶ بیتی می باشد که هر کدام می توانند فرکانس ثابت ورودی را بر اعداد ۱ الی ۶۵۵۳۶ نمایند و در خروجی OUT خود قرار دهند .

فصل ۱۴

برنامه نویسی ۳۲ بیتی جهت کامپیوترهای 486 و 386

راهنماهای اسمبلر 486 . و 586 . و ....

پردازنه های 386 و 486 می توانند در دو مد واقعی و حفاظت شده کار کنند . در مد واقعی همه آنها همانند ماشین های 8086/286 عمل می کنند با این استثنا که ثباتها بصورت ۳۲ بیت در دسترس هستند . (حداکثر ۱ مگا بایت حافظه قابل دسترس) اما در مد حفاظت شده می توانند حداکثر تا ۴ گیگا بایت از حافظه را دستیابی کنند ولی نیاز به یک سیستم عامل پیچیده دارند که سیستم عامل های UNIX و OS/2 IBM قابلیت مد حفاظت شده را دارا می باشند . (امروزه ویندوز مد حفاظت شده است) توان واقعی این پردازنده ها هنگامی ارائه می شوند که در مد حفاظت شده کار کنند .

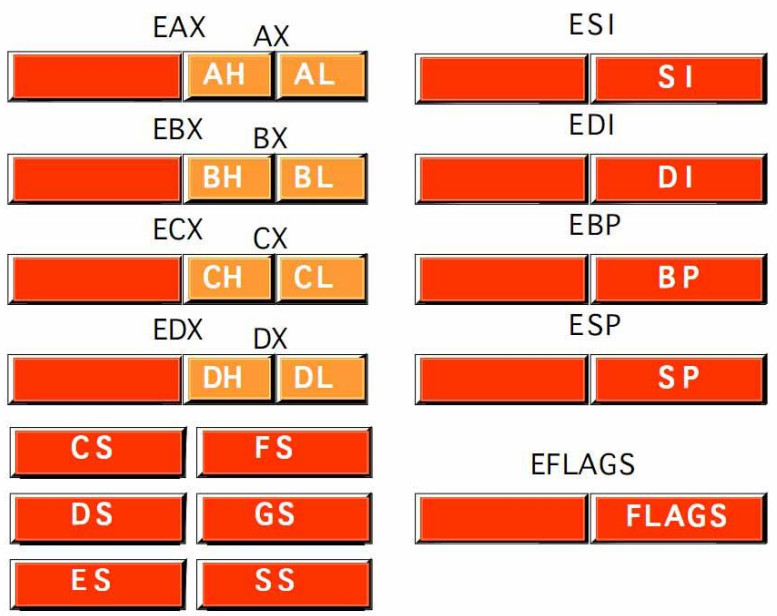
چند کاربره به سیستمی گفته می شود که قادر به پشتیبانی بیش از یک صفحه کلید را در هر زمان داشته باشد .

چند کاره به سیستمی گفته می شود که قادر به اجرای بیش از یک برنامه را در یک زمان داشته باشد (مد حفاظت شده)

مد واقعی : قابلیت اجرای یک برنامه در یک زمان

ثبات ها

نام ثبات	بیت	گروه
EAX,EBX,ECX,EDX AX,BX,CX,DX AH,AL.BH,BL,CH,CL,DH,DL	32 16 8	همه منظوره
ESP,EBP SP,BP	32 16	اشاره گر
ESI,EDI SI,DI	32 16	اندیس
CS,DS,SS,ES FS,GS	16	قطعه
EIP	32	دستور
EFR	32	پرچم
CR0,CR1,CR2,CR3	32	کنترل



از مجمع شش ثبات قطعه در مجموع ، دستیابی به 384KB فضای قطعه فراهم می شود . (فضای هر قطعه همچنان جهت سازگاری 64KB در نظر گرفته می شود)

گرچه ثبات پرچم و IP به ۳۲ بایت توسعه یافته اند اما در مد واقعی تنها ۱۶ بیت پایین در دسترس هستند . جهت دستیابی به همه ۳۲ بیت پردازنده بایستی به مد حفاظت شده سوئیچ شود .

بیت 0 ثبات CR0 همان بیت فعال ساز حفاظت است (تنها این بیت در مد واقعی در دسترس است . سایر ثبات های کنترل در مد حفاظت در دسترس می باشند .) وقتی کامپیوتر روشن می شود به طور خودکار بیت 0 ثبات CR0 برابر صفر قرارگرفته و مد واقعی انتخاب می شود جهت رفتن به مد حفاظت این بیت بایستی یک شود .

ثبات های AX, CX, DX در پردازنده های ۱۶ بیتی نمی توانستند نقش اشاره گر را داشته باشند (فقط ثبات BX, SI, DI می توانستند) مثلاً نوشتن دستوری مانند MOV CL, [AX] تولید خطا می کرد . این قابلیت در پردازنده های 386/486 کاملاً مجاز است :

```
MOV AX,[ECX]
ADD SI,[EDX]
OR EBX,[EAX]+20
```

محاسبه آدرس فیزیکی در مد واقعی همانند 86/286 است .

ثبات SS قطعه پیشفرض برای ESP, EBP است .

ثبات CS قطعه پیشفرض برای EIP است .

ثبات DS قطعه پیشفرض برای سایر ثبات ها است .

سعی کنید آدرس فیزیکی را از روی آدرس منطقی (FS:ECX) 12E0:00000120 را بدست آورید .

دستور MOV RESULT,EAX بایت اول ثبات EAX (ثبات AL) در محل RESULT حافظه و بایت دوم (ثبات AH) در محل RESULT+1 و بایت سوم در محل RESULT+2 و بالاخره بایت چهارم در محل RESULT+3 حافظه ذخیره می شود .

مثال : برنامه ای بنویسید که با استفاده از ثباتهای ۳۲ بیتی مقادیر 100000 و 200000 و 300000 را با یکدیگر جمع نماید و حاصل را در result ذخیره نماید .

```
PAGE 50,100
TITLE 'EX_32.ASM'
.MODEL SMALL
        .586
        .STACK 64
        .DATA
RESULT DD ?
        .CODE
MAIN PROC FAR
        MOV AX,@DATA
        MOV DS,AX
        MOV EAX,0
        ADD EAX,100000
        ADD EAX,200000
        ADD EAX,300000
        MOV RESULT,EAX

        MOV AX,4C00H
        INT 21H
MAIN ENDP
        END MAIN
```

مثال : برنامه ای بنویسید که دو عدد اسکی '2' و '5' را بصورت ۳۲ بیتی با هم جمع و روی مانیتور نمایش دهد .

```
PAGE 70,100
TITLE 'EX_32.ASM'
.MODEL SMALL
        .586
        .STACK 64
        .DATA
RESULT DD ?
        .CODE
MAIN PROC FAR
        MOV AX,@DATA
        MOV DS,AX
        MOV EAX,00000032H
        MOV EBX,00000034H
        ADD EDX,EBX
        ;SHOW IN MONITOR
        SUB DL,30H
        MOV AH,02H
        INT 21H

        MOV AX,4C00H
        INT 21H
MAIN ENDP
        END MAIN
```

تمرین : محتوای ثباتهای خواسته شده را بدست آورید .

```
MOV EAX,9823F4B6H          AL  ,  AH  ,  AX  ,  EAX
MOV ESI,120000H            SI  ,  ESI
```

تمرین : نتیجه دستورات زیر چیست .

```
MOV EBX,9FE35DH
XOR EBX,0F0F0F0H
```

فصل ۱۵

برنامه های TSR (TERMINATE AND STAY RESIDENT)

فلسفه اقامت در حافظه آن است که پس از اجرای برنامه ، برنامه مورد نظر در حافظه باقی مانده و سیستم عامل آنرا در حافظه نگه می دارد که در صورت نیاز مجدد به آن نیاز به فراخوانی مجدداز هارد دیسک نباشد.ازآنجا که برنامه در حافظه قرار دارد سریعتر اجرا خواهد شد (بخشی از حافظه نیز اشغال می شود) برنامه ساعت در ویندوز و ماشین حساب و ... از اینگونه برنامه ها هستند .

روش مقیم کردن برنامه در حافظه

کافیست در انتهای برنامه به جای دستورات خروج و بازگشت به سیستم عامل از سرویس 31H دستور INT 21H استفاده شود به شرطی که اندازه برنامه کاربر برحسب تعداد پاراگراف (هر پاراگراف شانزده بایت) قبلاً در ثبات DX قرار داده شود .

```
AH=31H
DX= تعداد پاراگراف
INT 21H
```

چگونگی فراخوانی برنامه های TSR

برای انجام این عمل آدرس برنامه TSR را جایگزین آدرس وقفه سخت افزاری می نماییم . بدین ترتیب هر موقع که وقفه مورد نظر فعال شود به جای روتین وقفه ، برنامه TSR اجرا می شود .

برای اینکه وقفه سخت افزاری نیز دچار اشکال نشود آدرس آن در محل دیگری از حافظه ذخیره می گردد که در انتهای برنامه TSR به آن مراجعه می شود .

وقفه های سخت افزاری INT 09 مربوط به فشار دادن کلید کیبرد و یا INT 08 وقفه تایمر بکار برده می شود .

روش جایگزین کردن آدرس برنامه TSR با آدرس بردار وقفه

۱. سرویس 25H دستور INT 21H : جایگزینی آدرس برنامه TSR با آدرس وقفه

```
AH=25H
AL= شماره وقفه
DX= آفست آدرس برنامه مقیم در حافظه
INT 21H
```

با اجرای دستورات فوق آدرس برنامه TSR در محل بردار وقفه قرار می گیرد .

۲. سرویس 35H دستور INT 21H : پیدا کردن آدرس وقفه (که می خواهیم TSR را جایگزین آن کنیم)

```
AH=35H
AL= شماره وقفه
INT 21H
```

با اجرای دستورات فوق آدرس سگمت کد روتین وقفه در ES و آدرس تفاوت مکان آن در BX ذخیره می شود . یعنی آدرس وقفه در ES:BX قرار می گیرد . CS:IP

مثال : زیر برنامه ای بنویسید که ابتدا برنامه TEST را در محل آدرس دستور وقفه INT 09 قرار دهد و سپس آدرس منطقی این وقفه را پیدا و در متغیرهای X و Y حافظه ذخیره نماید .

```
MOV AH,25H
MOV AL,09H
MOV DX,OFFSET TEST ; or LEA DX,TEST
INT 21H
;------------------
MOV AH,35H
MOV AL,09H
INT 21H
MOV X,ES
MOV Y,BX
```

نکته مهم : برنامه های TSR باید به صورت COM نوشته شود .

یادآوری برنامه COM :

نوشتن این روتین جهت برنامه هایی که کمتر از 64KB حافظه مصرف می کنند بکار می رود (برنامه های EXE بصورت استاندارد به هر اندازه ای می توانند باشند)

۱- فقط یک سگمنت کد برای متغیرها و دستورات تعریف می شود (سگمنت کد و داده یکسان است)
۲- سگمنت پشته بصورت مجزا تعریف نمی شود (سیستم عامل بطور خودکار از انتهای سگمنت کد برای پشته استفاده می کند)
۳- در شبه دستور ASSUME تمام ثبات های قطعه را به سگمنت کد نسبت می دهیم
۴- دستورات و اطلاعات از آدرس 100H شروع می شوند .

۵- برنامه حتماً بایستی با یک دستور (JMP) شروع شود .

۶- تمام متغیرها معمولاً بعد از اولین دستور (JMP) نوشته می شوند .

۷- تمام روال ها با صفت NEAR تعریف می شوند .

۸- مقابل دستور END برچسب اولین دستور برنامه نوشته می شود .

```
        PAGE  60 , 132
TITLE   A07COMI COM program to move and add
CODESG  SEGMENT PARA  'Code'
        ASSUME CS:CODESG, DS:CODESG, SS:CODESG, ES:CODESG
        ORG 100H                ;Start at end of PSP
BEGIN:  JMP A10MAIN             ;Jump past data
;-------------------------------------------------
FLDD    DW   175                ;Data definitions
FLDE    DW   150
FLDF    DW   ?
;-------------------------------------------------
A10MAIN PROC NEAR
        MOV  AX , FLDD          ;move 0175 to AX
        ADD  AX , FLDE          ;add 0150 to AX
        MOV  FLDF , AX          ;store sum in FLDF
        MOV  AX , 4C00H         ;End processing
        INT  21H
A10MAIN ENDP
CODESG  ENDS
        END  BEGIN
```

نمونه مثالی از یک برنامه COM با پیش‌پردازنده های سگمنت

```
        PAGE  60 , 132
TITLE   A07COM2 COM program to move and add
        .MODEL  SMALL
        .CODE
        ORG 100H                ;Start at end of PSP
BEGIN:  JMP A10MAIN             ;Jump past data
;-------------------------------------------------
FLDD    DW   175                ;Data definitions
FLDE    DW   150
FLDF    DW   ?
;-------------------------------------------------
A10MAIN PROC NEAR
        MOV  AX , FLDD          ;move 0175 to AX
        ADD  AX , FLDE          ;add 0150 to AX
        MOV  FLDF , AX          ;store sum in FLDF
        MOV  AX , 4C00H         ;End processing
        INT  21H
A10MAIN ENDP
        END  BEGIN
```

نمونه مثالی از یک برنامه COM با پیش پردازنده های segment ساده شده

مثال : برنامه ای بنویسید که هر لحظه کلیدهای Alt و F7 با هم فشار داده شوند بلندگوی کامپیوتر برای مدتی روشن شود و این برنامه در حافظه مقیم شود .

```
      PAGE 110,100
TITLE 'alt_f7.asm'
      .MODEL SMALL
.CODE
;-----------------
       ORG 100H
START: JMP MAIN
```

```
;-----------------
OLDINT9 DD ?
;-----------------PART1-------
KEY PROC NEAR
       PUSH AX
       MOV AH,02
       INT 16H
       TEST AL,00001000B
       JZ QUIT
       IN AL,60H    ; GET SCAN CODE
       CMP AL,41H   ; SEE IF IT IS F7
       JNZ QUIT
       MOV CX,00FFH
AGAIN: MOV AH,0EH
       MOV AL,07
       INT 10H
       LOOP AGAIN
QUIT:  POP AX
       JMP CS:OLDINT9
KEY    ENDP
;-----------------PART2-----
MAIN PROC NEAR
       MOV AH,35H
       MOV AL,09H
       INT 21H
       MOV WORD PTR OLDINT9,BX
       MOV WORD PTR OLDINT9+2,ES
;-------------------------
       MOV AH,25H
```

```
        MOV AL,09H
        LEA DX,KEY
        INT 21H
;---------------------------
        MOV DX,(OFFSET MAIN-OFFSET CODSEG)
        ADD DX,15
        MOV CL,4
        SHR DX,CL

        MOV AH,31H
        INT 21H
MAIN    ENDP
        END START
```

توضیحات برخی از قسمت های برنامه :

قسمت ? OLDINT9 DD آدرس وقفه INT 09H را ذخیره می کند .

برنامه TSR بایستی بصورت زیر نوشته شود :

```
KEY PROC NEAR
        PUSH AX

        برنامه TSR

        POP AX
        JMP CS:OLD INT
KEY     ENDP
```

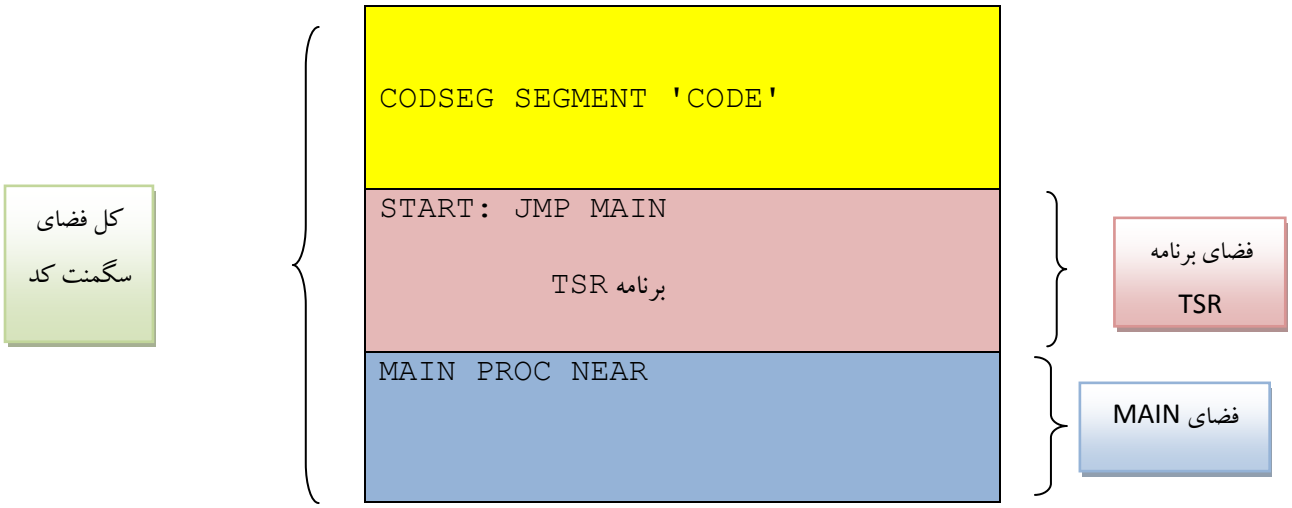
در قسمت PART2 ابتدا آدرس وقفه ۹ در OLDINT9 ذخیره می شود و سپس آدرس برنامه TSR جایگزین آن می شود .

یادآوری :

```
TOTAL DW 0F25BH

MOV AX,TOTAL                ; AX=F25BH
MOV AL,BYTE PTR TOTAL       ; AL=5BH
MOV AH,BYTE PTR TOTAL+1     ;AH=F2H
```

همانطور که قبلاً اشاره شد برای اقامت در حافظه باید از سرویس 31H دستور INT 21H استفاده نمود فقط بایستی تعداد پاراگراف محاسبه و در DX قرار گیرد . معمولاً اختلاف آدرس CODSEG و MAIN اندازه برنامه TSR را بر حسب بایت تعیین می کند



که این اختلاف را می توان بصورت تفریق دو آدرس به شکل زیر در DX ذخیره کرد :

```
MOV DX,(OFFSET MAIN-OFFSET CODSEG)
```

از آنجایی که هر پاراگراف شانزده بایت است اکنون محتوای ثبات DX را برعدد ۱۶ تقسم می کنیم تا جواب بر حسب پاراگراف بدست آید بدین منظور از دستورات زیر استفاده می کنیم :

```
MOV CL,4
SHR DX,CL
```

اگر محتوای DX را چهار بار به سمت راست شیفت دهیم عمل تقسیم بر ۱۶ انجام می شود (هر شیفت به راست تقسیم بر ۲ است)

اما ممکن است اندازه TSR دقیقاً مضرب ۱۶ نباشد و تعداد پاراگراف در DX کمتر از مقدار واقعی گردد لذا برای اطمینان بیشتر قبل از دستورات فوق از دستور زیر استفاده می شود :

```
ADD DX,15
```

دستور فوق ۱۵ بایت به DX اضافه می کند تا پس از تقسیم تعداد پاراگراف کمتر از مقدار واقعی بدست نیاید .

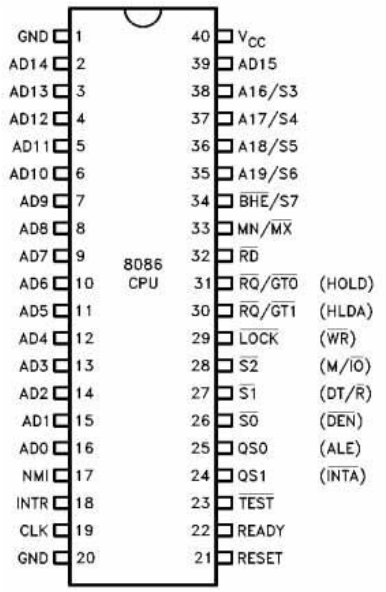
همچنین بجای دستورات زیر :

```
MOV AH,02
INT 16H
TEST AL,00001000B
JZ QUIT
IN AL,60H    ; GET SCAN CODE
CMP AL,41H   ; SEE IF IT IS F7
JNZ QUIT
```

می توان از دستورات زیر استفاده نمود : (با مراجعه به کد اسکن ALT+F7)

```
IN AL,60H
CMP AL,6EH
JNZ QUIT
```

توصیف پایه های ریزپردازنده



۱. ADDRESS BUS
۲. DATA BUS
۳. GND
۴. VCC
۵. WR : هنگامی که بخواهیم اطلاعات موجود دریک پردازنده را در یک حافظه یا یک چیپ I/O بنویسیم این پایه صفر می گردد
۶. RD : این پایه به حافظه یا چیپ I/O می فهماند که پردازنده آماده خواندن اطلاعات است و آنها نیز اطلاعات را روی باس دیتا قرار می دهند .
۷. RESET
۸. INTR : در خواست وقفه برای تقاضای وقفه سخت افزاری است . (دستوراتی برای ممانعت از این وقفه وجود دارد)
۹. NMI : شبیه به INTR است با این تفاوت که هیچ وقت قابل سد کردن نیست . (شبیه به CTRL+ALT+DELET)
۱۰. CLK
۱۱. INTA : سیگنال تصدیق وقفه در پاسخ به ورودی INTR است . این پایه معمولاً برای بارگذاری شماره بردار وقفه در باس داده بکار می رود .
۱۲. ALE : این پایه جهت دی مولتی پلکی کردن آدرس/داده بکار می رود .
۱۳. A16/S3 الی A19/S6 :
۱۴. BHE/S7 :
۱۵. MN/MX :
۱۶. RQ/GT0 (HOLD) :
۱۷. RQ/GT1 (HLDA) :
۱۸. LOCK :
۱۹. S2 (M/IO) :
۲۰. S1 (DT/R) :

۲۱. S0(DEN):

۲۲. QS0,QS1:

۲۳. TEST:

۲۴. READY:

❖ نکات مفیدی پیرامون BIOS

مهمترین وظیفه آن استقرار سیستم عامل در لحظه روشن شدن در حافظه است .

زمانی که کامپیوتر روشن می شود BIOS عملیات متفاوتی را انجام خواهد داد :

- بررسی محتویات CMOS (این قسمت شامل اطلاعات جزئی در رابطه با سیستم بوده و در صورت بروز هرگونه تغییر در سیستم اطلاعات فوق نیز تغییر خواهد کرد .)
- لود کردن درایورهای استاندارد و INTERRUPT HANDLERS (این قسمت نوع خاصی از نرم افزار بوده که به عنوان یک مترجم بین عناصر سخت افزاری و سیستم عامل ایفای نقش می کند . مثلا زمانی که شما کلیدی را بر روی صفح کلید فعال می کنید سیگنال مربوطه برای INTERRUPT HANDLERS صفحه کلید ارسال شده تا از این طریق به پردازنده اعلام گردد که کدامیک از کلیدهای صفحه کلید فعال شده اند .
- مقدار دهی اولیه ثبات ها و مدیریت POWER
- اجرای برنامه POST به منظور اطمینان از صحت عملکرد سخت افزار
- تشخیص درایوی که سیستم می بایست از طریق آن راه اندازی گردد
- مقدار دهی اولیه برنامه مربوط به استقرار سیتم عامل در حافظه

با توجه به اینکه BIOS به طور دائم با سیگنال های ارسالی توسط سخت افزار مواجه است معمولا یک نسخه از آن در RAM تکثیر خواهد شد .

راه اندازی کامپیوتر BOOTING

پس از روشن کردن کامپیوتر BIOS بلافاصله عملیات خود را شروع می کند . در اغلب سیستم ها BIOS در زمان انجام عملیات مربوطه پیام هایی را نیز نمایش می دهد (میزان حافظه ، نوع هارد دیسک و ...). پس از بررسی CMOS و استقرار INTERRUPT HANDLER در حافظه RAM کارت گرافیکی بررسی می شود . اغلب کارت های گرافیکی دارای BIOS اختصاصی بوده که حافظه و پردازنده مربوط به کارت گرافیکی را مقدار دهی اولیه می نماید . درصورتیکه BIOS اختصاصی برای کارت گرافیکی وجود نداشته باشد از درایور استانداردی که در ROM ذخیره شده است استفاده و درایور مربوطه فعال خواهد شد . در ادامه BIOS نوع راه اندازی مجدد REBOOT و یا راه اندازی اولیه را تشخیص خواهد داد . برای تشخیص موضوع فوق از محتویات آدرس 0472 : 0000 حافظه استفاده خواهد شد . درصورتی که در آدرس فوق مقدار 123H موجود باشد به منزله راه اندازی مجدد بوده و برنامه BIOS بررسی صحت عملکرد حافظه را انجام نخواهد داد . در غیر اینصورت یک راه اندازی اولیه تلقی می گردد . در این حالت بررسی صحت عملکرد و سالم بودن حافظه انجام خواهد شد . در ادامه پورت های سریال و USB برای اتصال به صفحه کلید و ماوس بررسی خواهد شد . در مرحله بعد کارت های PCI نصب شده بر روی سیستم بررسی می گردد . در صورتی که در هر یک از مراحل فوق BIOS با اشکالی برخورد نماید با نواختن چند BEEP معنی دار مورد خطا را اعلام خواهد کرد . خطا های اعلام شده اغلب به موارد سخت افزاری مربوط می شود .

پیکر بندی BIOS

برای پیکربندی CMOS در زمان راه اندازی سیستم کلیدهای خاصی را فعال تا زمینه استفاده فراهم گردد (کلید F2 یا DEL یا ...) پس از فعال شدن برنامه پیکربندی با استفاده از مجموعه ای از گزینه می توان اقدام به تغییر پارامترهای مورد نظر کرد . تنظیم تاریخ و زمان ف مشخص نمودن اولویت درایو بوت ، تعریف رمز و ... در این تکنولوژی یک باطری کوچک لیتیوم انرژی لازم برای نگهداریی اطلاعات به مدت چندین سال را فراهم می کند .

ارتقای برنامه BIOS

پس از اخذ فایل (برنامه) مربوطه آنرا بر روی دیسک قرار داده و سیستم را از طریق درایوی که فایل روی آن ذخیره شده راه اندازی کرده در این حالت برنامه موجود بر روی دیسک BIOS قدیمی را پاک و اطلاعات جدید را در BIOS می نویسد .

علل به روز رسانی BIOS

- استفاده از هارد بیش از 8GB
- استفاده از درایور هارد IDE Ultra DMA
- استفاده از بوت کردن سیستم با درایو CD-ROM
- تصحیح خطای سال ۲۰۰۰ و کبیسه
- استفاده از پردازنده های جدید

اگر شما یک سخت افزار جدید نصب کرده اید و حتی دستورات نصب را به درستی انجام داده اید اما نمی توانید با آن کار کنید این خطا ممکن است از BIOS باشد و لازم است به روز رسانی شود .بسیاری از سیستم های قدیمی نیاز به به روز رسانی BIOS دارند تا به طور کامل از ویژگی های PLUG-AND-PLAY استفاده کنند .

جهت به روز رسانی :

سازنده و مدل مادربرد ۲- نسخه فعلی BIOS مادربرد

بعضی گزینه های BIOS:

Viruse warning : اگر این گزینه بر روی بخش راه انداز boot sector باشد کاربر پیغام دریافت می کند پس از دریافت پیغام بایستی یک گزینه ضد ویروس را اجرا کنید . فعال بودن این گزینه موجب می شود که پیغام هشدار را تنها وقتی در مرحله راه اندازی قرار دارید دریافت کنید و نه وقتی ویندوز در حال اجرا شدن است .

در هنگام نصب یک سیستم عامل جدید بایستی این گزینه غیرفعال شود .

Cpu internal cache : این انتخاب فعال بودن حافظه نهان را نشان می دهد .

Processor number feature : شماره سریال پردازنده می تواند توسط نرم افزار مخصوصی خوانده شود .

Boot from lan first : اگر این انتخاب فعال باشد در ان صورت BIOS سعی می کند به جای راه اندازی شدن از طریق درایو موجود بر روی آن سیستم از طریق درایو سرویس دهنده شبکه محلی راه اندازی شود .

Boot sqence : اگر این انتخاب بر روی تنظیمات پیش فرض خود باشد در آن صورت BIOS در ابتدا تلاش می کند که سیستم را از درایو a راه اندازی کند اگر موفق نشد سپس به سراغ درایو c می رود .

Vga boot from : اگر در سیستم شما دو کنترل کننده تصویری وجود داشته باشد با استفاده از این انتخاب می توانید کنترل کننده ای که سیستم باید انتخاب کند را معین کنید .

Boot up numlock status : با این انتخاب شما می توانید تصمیم بگیرید که آیا کلید numlock با روشن شدن سیستم فعال شود یا خیر .

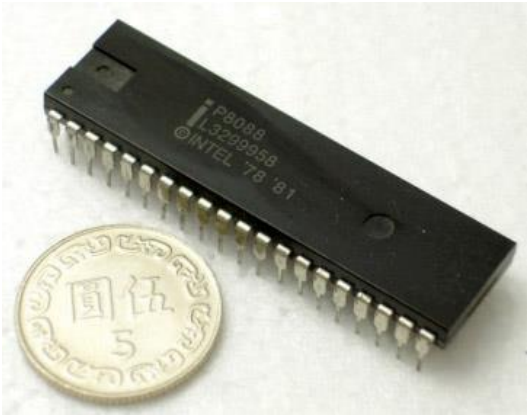
Dual bios : بعضی مادربردها دو BIOS دارند یکی اصلی و دیگری ذخیره است . در برخی موارد ممکن است اطلاعات BIOS از بین برود در این صورت اگر مادر برد دارای دو بایوس باشد نیازی به تعویض بایوس نیست و میتوان از بایوس دوم جهت راه اندازی سیستم و انتقال اطلاعات به بایوس اول استفاده کرد .

عوامل پیدایش ویروسهای کامپیوتری :

۱. گسترش کامپیوترهای شخصی
۲. استفاده از اینترنت و بوجود آمدن اسب تروجان
۳. فلاپی دیسک

DEC	HEX	ASCII	Key	DEC	HEX	ASCII
0	0x00	NUL	Ctrl @	64	0x40	@
1	0x01	SOH	Ctrl A	65	0x41	A
2	0x02	STX	Ctrl B	66	0x42	B
3	0x03	ETX	Ctrl C	67	0x43	C
4	0x04	EOT	Ctrl D	68	0x44	D
5	0x05	ENQ	Ctrl E	69	0x45	E
6	0x06	ACK	Ctrl F	70	0x46	F
7	0x07	BEL	Ctrl G	71	0x47	G
8	0x08	BS	Ctrl H	72	0x48	H
9	0x09	HT	Ctrl I	73	0x49	I
10	0x0A	LF	Ctrl J	74	0x4A	J
11	0x0B	VT	Ctrl K	75	0x4B	K
12	0x0C	FF	Ctrl L	76	0x4C	L
13	0x0D	CR	Ctrl M	77	0x4D	M
14	0x0E	SO	Ctrl N	78	0x4E	N
15	0x0F	SI	Ctrl O	79	0x4F	O
16	0x10	DLE	Ctrl P	80	0x50	P
17	0x11	DC1	Ctrl Q	81	0x51	Q
18	0x12	DC2	Ctrl R	82	0x52	R
19	0x13	DC3	Ctrl S	83	0x53	S
20	0x14	DC4	Ctrl T	84	0x54	T
21	0x15	NAK	Ctrl U	85	0x55	U
22	0x16	SYN	Ctrl V	86	0x56	V
23	0x17	ETB	Ctrl W	87	0x57	W
24	0x18	CAN	Ctrl X	88	0x58	X
25	0x19	EM	Ctrl Y	89	0x59	Y
26	0x1A	SUB	Ctrl Z	90	0x5A	Z
27	0x1B	ESC	Ctrl [	91	0x5B	[
28	0x1C	FS	Ctrl \	92	0x5C	\
29	0x1D	GS	Ctrl]	93	0x5D	]
30	0x1E	RS	Ctrl ^	94	0x5E	^
31	0x1F	US	Ctrl _	95	0x5F	_
32	0x20	SP		96	0x60	`
33	0x21	!		97	0x61	a
34	0x22	"		98	0x62	b
35	0x23	#		99	0x63	c

36	0x24	$		100	0x64	d
37	0x25	%		101	0x65	e
38	0x26	&		102	0x66	f
39	0x27	'		103	0x67	g
40	0x28	(		104	0x68	h
41	0x29	)		105	0x69	i
42	0x2A	*		106	0x6A	j
43	0x2B	+		107	0x6B	k
44	0x2C	,		108	0x6C	l
45	0x2D	-		109	0x6D	m
46	0x2E	.		110	0x6E	n
47	0x2F	/		111	0x6F	o
48	0x30	0		112	0x70	p
49	0x31	1		113	0x71	q
50	0x32	2		114	0x72	r
51	0x33	3		115	0x73	s
52	0x34	4		116	0x74	t
53	0x35	5		117	0x75	u
54	0x36	6		118	0x76	v
55	0x37	7		119	0x77	w
56	0x38	8		120	0x78	x
57	0x39	9		121	0x79	y
58	0x3A	:		122	0x7A	z
59	0x3B	;		123	0x7B	{
60	0x3C	<		124	0x7C	|
61	0x3D	=		125	0x7D	}
62	0x3E	>		126	0x7E	~
63	0x3F	?		127	0x7F	DEL



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

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

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

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

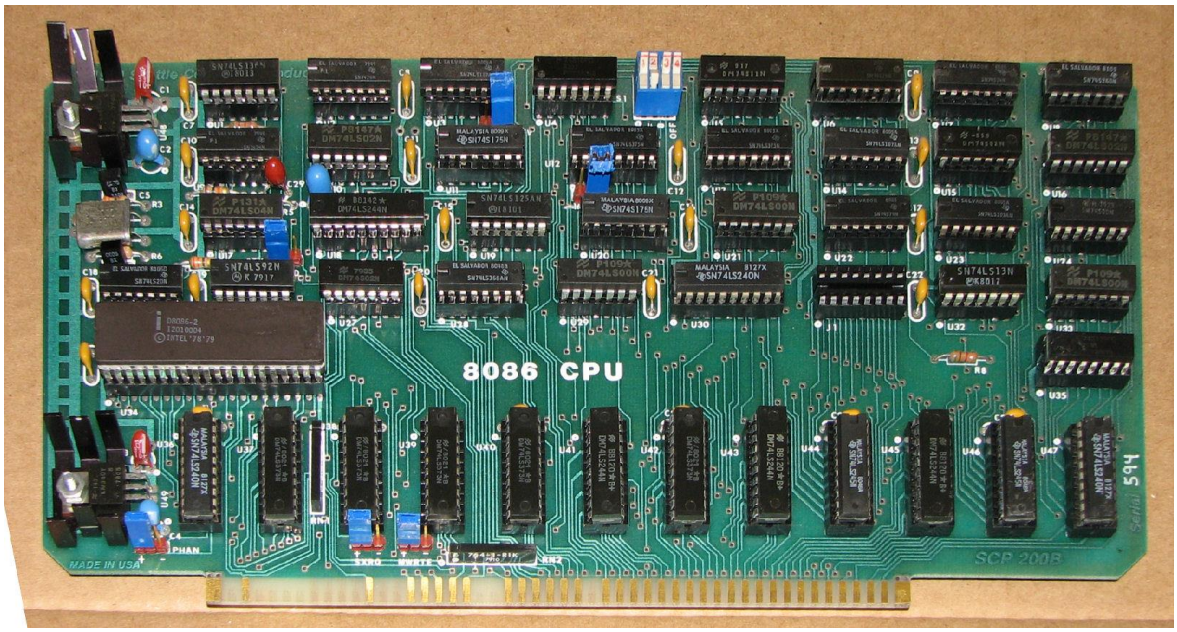
Instruction		Clocks	Bytes
OR	Memory, register	7*	2-4
OR	Register, immediate	3	3-6
OR	Memory, immediate	7*	3-6
OR	Accumulator; immediate	3	2-3
OUT	Immed8, accumulator	3	2
OUT	DX, accumulator	3	1
OUTS	DX,source-string	5	1
OUTS	(rep)DX, source-string	5+4(rep)	1
POP	Register	5	1
POP	Memory	5*	2-4
POPA		19	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

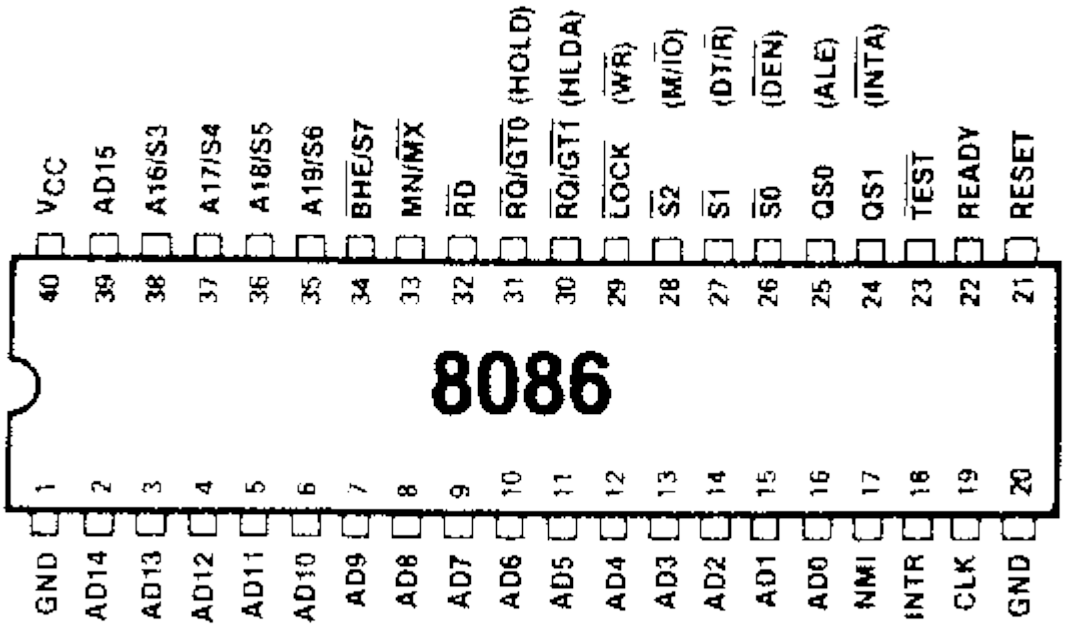
Instruction		Clocks	Bytes
SAL/SHL/SAR/SHR register, 1		2	2
SAL/SHL/SAR/SHR register, CL		5+1/bit	2
SAL/SHL/SAR/SHR memory,1		7*	2-4
SAL/SHL/SAR/SHR memory, CL		8*+1/bit	2-4
SAL/SHL/SAR/SHR reg, count		5+1/bit	3
SAL/SHL/SAR/SHR memory, count		8*+1/bit	3-5
SBB	Register, register	2	2
SBB	Register, memory	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

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


8086 CPU
MADE IN USA


8086
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GT0 (HOLD)
30 RQ/GT1 (HLDA)
29 LOCK (WR)
28 S2 (M/IO)
27 S1 (DT/R)
26 S0 (DEN)
25 QS0 (ALE)
24 QS1 (INTA)
23 TEST
22 READY
21 RESET
1 GND
2 AD14
3 AD13
4 AD12
5 AD11
6 AD10
7 AD9
8 AD8
9 AD7
10 AD6
11 AD5
12 AD4
13 AD3
14 AD2
15 AD1
16 AD0
17 NMI
18 INTR
19 CLK
20 GND