

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

# زبان ماشین و اسمبلی



دکتر داریوش زین العابدینی

انجمن آیاپیر پاسخگوی سوالات شما عزیزان در زمینه های تخصصی کامپیوتر می باشد.

## « زبان ماشین و اسمبلی »

### منابع :

- 1- برنامه نویسی سازمان اسمبلی - مرجع کامل از 8086 تا پنتیوم ، تألیف (جعفر نژاد قمی)
- 2- زمان ماشین و اسمبلی و کاربرد آن در کامپیوترهای شخصی - تألیف: دکتر حسن سیدرضی
- 3- برنامه نویسی بازبان اسمبلی - ویرایش پنجم - پتیرایبل - دلواری و سالخورده
- 4- کتاب آموزش اسمبلی برای کامپیوترهای شخصی - پیتر نورتن و جان سوچا-ترجمه ادیک باغداساریان.

### اهداف درس :

- آشنایی با زبان اسمبلی کامپیوترهای PC
- نحوه ارتباط مستقیم برنامه ها با سیستم عامل
- برنامه نویسی سخت افزار (hardware programming)
- پیش نیاز : آشنایی با یک زبان سطح بالا ساخت یافته (C یا پاسکال)

## مقدمات :

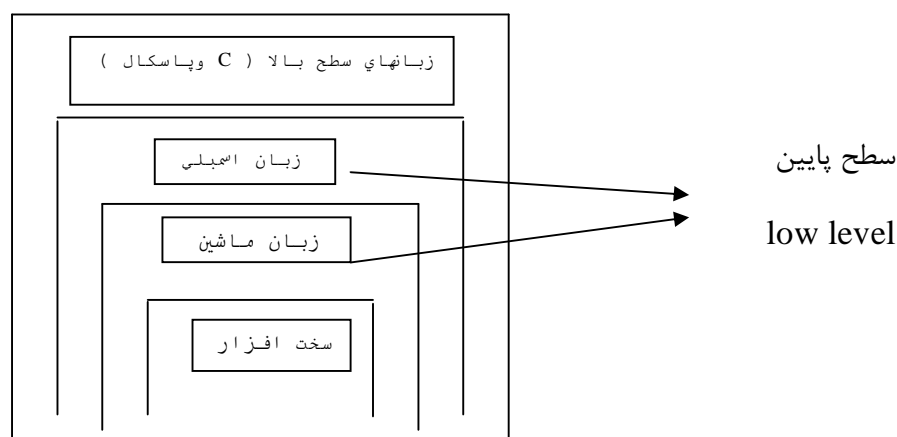
## با زبان اسمبلی :

- دید روشن از سخت افزار و نرم افزار بدست می آید .
- نحوه ارتباط سیستم عامل و برنامه های کاربردی و نیز نحوه ارتباط مستقیم سیستم عامل و سخت افزار .
- نحوه برنامه سازی به زبان ماشین (دستورات قابل فهم برای CPU) و ارتباط آن با سخت افزار تعیین می شوند .

## زبان ماشین :

تنها زبان مناسب قابل فهم برای سخت افزار که مجموعه ای از کدهای 1,0 است .

هر خانواده زبان ماشین خاص در دارد . (..., X86, Apple)

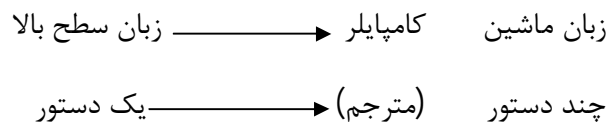


زبان اسمبلی مشابه زبان ماشین است با این تفاوت که کدها با استفاده از سمبل یابی قابل فهم و معنی دار نوشته می شوند .

**نکته مهم :** تناظر یک به یک بین دستورات زبان اسمبلی و زبان ماشین وجود دارد .

تناظر یک به یک بین دستورات زبان ماشین و زبان سطح بالا وجود ندارد و هر دستور زبان سطح بالا معادل چند دستور زبان ماشین ها است .

زبان ماشین	اسمبلر	زبان اسمبلی
یک دستور		یک دستور



نکته: از آنجا که هر CPU زبان ماشین مخصوص به خود را دارد هر CPU زبان اسمبلی مخصوص به خود را نیز دارد.

- برنامه زبان اسمبلی معمولاً سریعتر و کم حجم تر از زبانهای سطح بالا هستند و امکان استفاده از تمام امکانات سخت افزاری وجود دارد.

- برخلاف زبانهای سطح بالا در زبان اسمبلی محدودیت های کمتری اعمال می شود و جزئیات بیشتری به عهده برنامه نویس گذاشته می شود.

مجموعه دستوراتی که یک CPU (CPU Instruction set) می تواند اجرا کند و برای آن شناخته شده است. در خانواده Intel دستورات down ward-compatible هستند. یعنی دستوراتی که در 8086 قابل اجراست در 80286 و 80386 نیز دقیقاً با همان شکل قابل اجراست

#### سیستم اعداد:

$$N = (a_{n-1} a_{n-2} \dots a_1 a_0 a_{-1} a_{-2} \dots a_{-m})_b$$

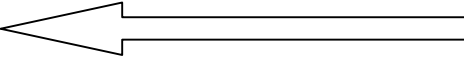
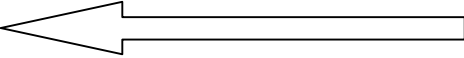
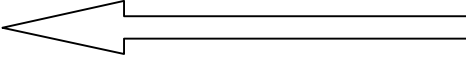
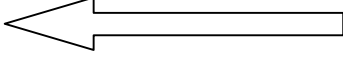
$$N = a_{n-1} b^{n-1} + \dots + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \dots + a_{-m} b^{-m}$$

$$N = \sum_{k=-m}^{n-1} a_k b^k \quad 0 \leq a_k \leq b-1$$

n : تعداد ارقام صحیح

m : تعداد ارقام اعشاری

$a_z, a_1, a_0$  و ... ضرایب

$\sum_{k=-m}^{n-1} a_k (10)^k$		دهدی
$\sum_{k=-m}^{n-1} a_k (2)^k$		دودی
$\sum_{k=-m}^{n-1} a_k (8)^k$		هشت تایی (اوکتال)
$\sum_{k=-m}^{n-1} a_k (16)^k$		شانزده تایی (هگزادسیمال)

در مبنای 16 از 10 تا 15 با معادل A تا F استفاده می شود

تبدیل مبنایها:  $(11001)_2 = 25$

دودی به دهدی و دهدی به دودی (تقسیم متوالی)

عدد اعشاری: برای تبدیل عدد اعشاری مبنای 10 به 2 دو قسمت صحیح و اعشاری را جداگانه به مبنای 2 تبدیل می کنیم. برای تبدیل قسمت صحیح از تقسیم متوالی بر 2 و برای تبدیل قسمت اعشاری از ضرب متوالی در 2 استفاده می شود. در این حالت قسمت اعشار در 2 ضرب شده، قسمت صحیح حاصل، نگهداری می شود و این روند ادامه می یابد تا قسمت اعشار به صفر برسد.

$$(12/25)_{10} \rightarrow (?)_2 \Rightarrow \begin{cases} (12)_{10} = (1100)_2 \\ (0/25)_{10} = (0/01)_2 \end{cases} \Rightarrow (12/25)_{10} = (1100/01)_2$$

$$0/25 \times 2 = 0/5 \rightarrow \text{قسمت صحیح} = 0 \Rightarrow (0/01)_2$$

$$0/5 \times 2 = 1 \rightarrow \text{قسمت صحیح} = 1$$

$$(1110/01)_2 = (?)_{10} \Rightarrow \begin{cases} 1110 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ \quad \quad \quad = 8 + 4 + 2 + 0 = 14 \Rightarrow 14/25 \\ (0/01) = 0 \times 2^{-1} + 1 \times 2^{-2} = 0 + \frac{1}{4} = 0/25 \end{cases}$$

## تبدیل مبنای دو به هشت و بالعکس :

هر رقم مبنای هشت معادل سه رقم مبنای دو است . برای تبدیل مبنای دو به هشت از سمت راست ، سه رقم سه جدا کرده و معادل مبنای هشت آن را قرار می دهیم ، در صورت لزوم به

تعداد لازم صفر در سمت چپ عدد یا جلوی ممیز اضافه می کنیم

$$(11001)_2 = (?)_8$$

$$\begin{array}{c} 011001 \\ \hline 3 \quad 1 \end{array} = (31)_8$$

$$(10011/1101)_2 =$$

$$\begin{array}{c} 010011/110100 \\ \hline 3 \quad 1 \end{array} = (23/64)_8$$

تبدیل مبنای دو به شانزده و بالعکس : مشابه مبنای 8 است فقط به جای 3 رقم ، 4 رقم جدا

می کنیم .

$$(01111101/0110) = (7D/6)_{16}$$

$$(f25/03)_{16} = (11100100101/0000011)$$

## محاسبات در مبنای 2 و 16 :

مانند مبنای 10 است اما به جای ده بر یک ، دو بر یک داریم :

$$\begin{cases} 0+1=1 \\ 1+0=1 \\ 0+0=0 \\ 1+1=10 \end{cases}$$

$$\begin{array}{r} 111 \\ 11111 \\ 11110+ \\ \hline 111101 \end{array}$$

$$\begin{array}{r} 1100 \\ 10 \times \\ \hline 0000 \\ 1100 \\ \hline 11000 \end{array}$$

$$\begin{array}{r} 10010 \\ 1001- \\ \hline 1001 \end{array}$$

$$\begin{aligned} 0-0 &= 0 \\ 1-1 &= 0 \\ 1-0 &= 1 \\ 0-1 &= \end{aligned}$$

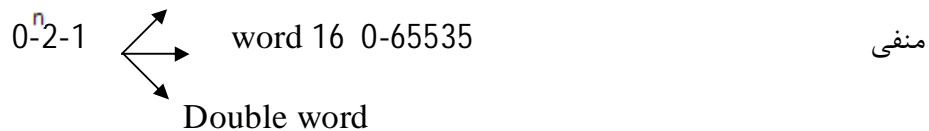
تعریف :

باید از رقم قبلی قرض کرد .

نگهداری اعداد صحیح :

مثبت - به صورت مبنای 2 و دو قسمت 1-بیت علامت 2- مقدار عدد

byte 8 و 0-255



معمولاً طول هر خانه حافظه توانی از 2 (16 یا 32 یا 64 و ...) است .

بیت علامت برای اعداد مثبت صفر می باشد .

۷ ۶ ۵ ۴ ۳ ۲ ۱ ۰

۰	۰	۰	۱	۰	۰	۱	۱
---	---	---	---	---	---	---	---

نمایش 19 در حافظه 8 بیتی

اعداد صحیح منفی: سه روش نگهداری :

1- روش علامت و مقدار - مانند اعداد مثبت فقط بین علامت مقدار یک می گیرد .

2- روش متمم 1

3- روش متمم 2

روش علامت و مقدار:

$$\begin{array}{r} -19 \\ \hline 10010011 \end{array} \leftarrow -19$$

معایب روش علامت و مقدار :

1- دو صفر جداگانه مثبت و منفی داریم .  

$$\begin{cases} 00000 \\ 10000 \end{cases}$$

2- عمل تفریق مدار جداگانه نیاز دارد .

اگر طول M فرض شود بزرگترین و کوچکترین اعداد در این روش :

1-  $(2^{m-1} - 1)$  بزرگترین

2-  $(2^{m-1} - 1)$  - کوچکترین

**روش متمم 1:** در این روش نمایش مثبت عدد را بدست آورده و سپس تمام ارقام را از یک کم نموده یا عبارت دیگر معکوس می کنیم . در حافظه 8 بیتی اشکال دوم روش علامت مقدار حل شد ولی هنوز مشکل اول پابرجاست .

00000 صفر مثبت

11111 صفر منفی

**روش متمم 2:** هر دو عیب روش اول را حل می کند - دارای مراحل زیر است :

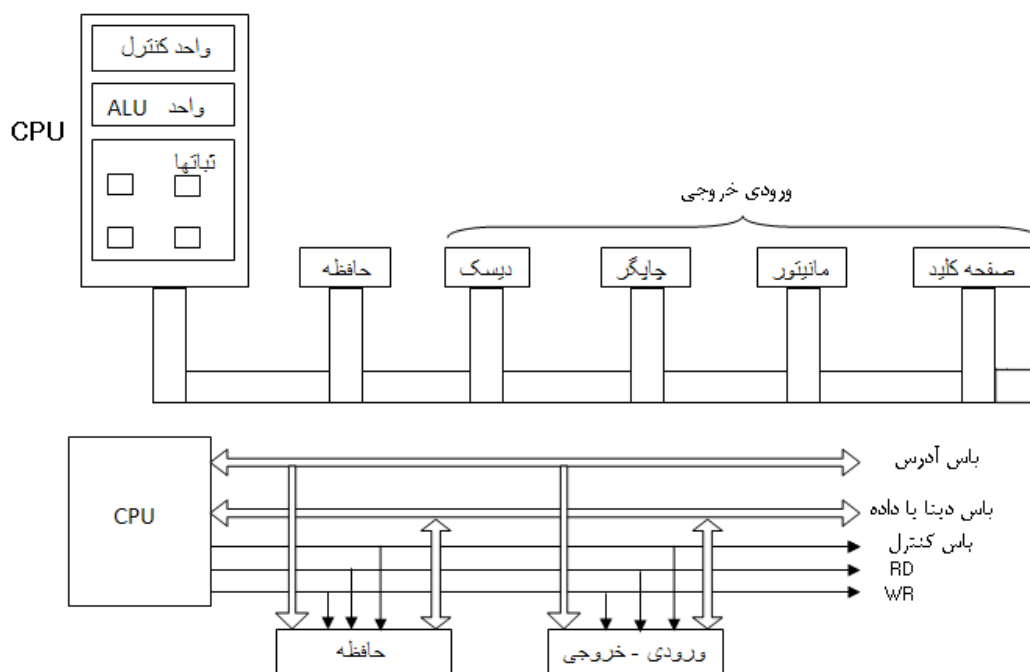
1- نمایش مثبت عدد 2- بد

$-19 \Rightarrow 00010011 \Rightarrow 11101100 \Rightarrow 11101101 \Rightarrow -19$

$$\{-2^{m-1}, 2^{m-1} - 1\}$$

محدوده ←

ساختار کامپیوتر : هر کامپیوتر از واحدهای ورودی-خروجی ، حافظه ALU ، باس یا گذرگاه و واحد کنترل تشکیل شده است ALU ، واحد کنترل و ثبات ها CPU نامیده می شود و وظیفه به اجرای دستورات را بر عهده دارد .





باس یا گذرگاه : برای برقراری ارتباط مداوم بین پردازنده ، ورودی-خروجی و حافظه نیاز به سیمهای بسیار زیادی می باشند که غیر عملی است راه حل عملی آن است که سیمهای ارتباطی بین تعدادی از وسایل مشترک باشند که این سیمهای مشترک را باس یا گذرگاه می نامند .

سه نوع باس وجود دارد :

1- باس آدرس که پردازنده آدرس دستگاههای ورودی-خروجی و یا حافظه را روی آن قرار می دهد .

2- باس داده که اطلاعات از طریق آن بین حافظه و دستگاههای ورودی-خروجی و CPU انتقال می یابد .

3- باس کنترل که شامل فرمانهای کنترلی مانند RD برای خواندن اطلاعات از ورودی-خروجی و انتقال به CPU یا فرمان WR برای نوشتن روی ورودی-خروجی یا حافظه .

**ثباتها :** در داخل پردازنده ، حافظه های سریعی به نام ثباتها وجود دارند بدلیل آنکه دستیابی به ثباتها سریعتر از دستیابی به حافظه است ، دستوراتی که فقط از ثباتها استفاده می کنند بسیار سریعتر از دستوراتی که از حافظه استفاده می کنند اجرا می شوند .

شرکت اینتل : 16 بیتی 8086,8088,80286

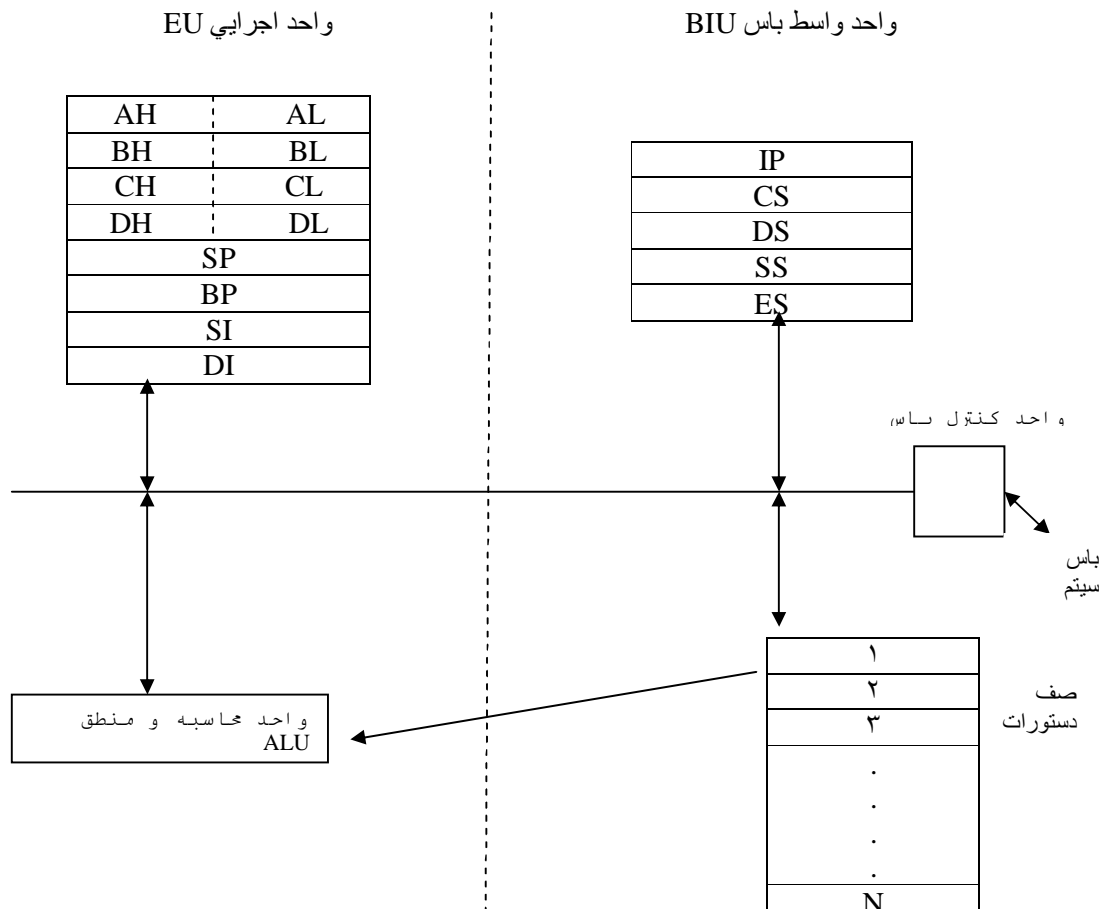
32 بیتی 80386,80486 و پنتیوم

**ساختار داخلی پروسسور :**

CPU از دو قسمت واحد اجرایی [Execution Unit (EU)] و واحد واسط باس [Bus] [INterface unit (BIU)] تشکیل شده است. واحد EU مسئول اجرای دستورات است که از ALU و تعدادی ثبات تشکیل شده است .

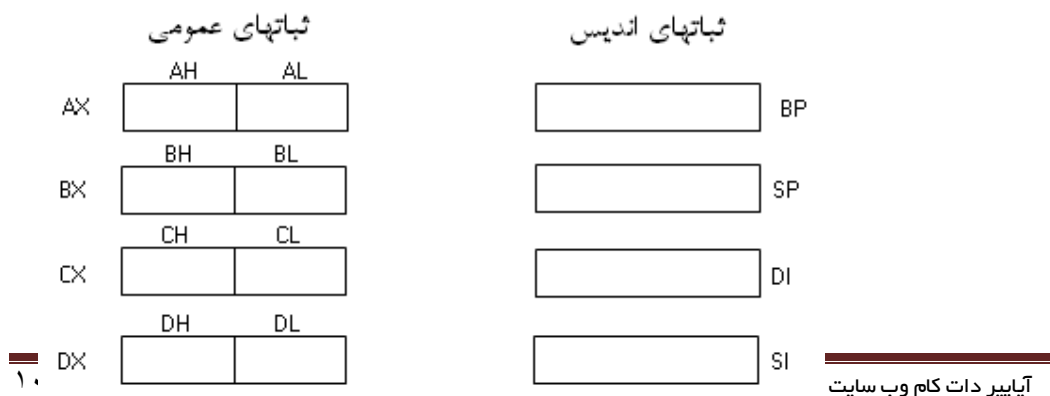
واحد واسط باس (BIU) شامل واحد مدیریت کنترل باس ، ثبات های سگمنت و صف دستورات است . BIU همواره یک سری از دستورات را از قبل ، از حافظه خوانده در صفحه دستورات قرار

می دهد Prefetih هر لحظه واحد اجرایی بخوهد دستور را اجرا کند بلافاصله از صف دستورات ، دستور را می گیرد و منتظر خواندن دستور نمی شود.



### ثباتهای پردازنده های 16بیتی :

به چند دسته تقسیم می شوند ثباتهای عمومی ، ثباتهای سگمنت ، ثباتهای اندیس ، ثباتهای وضعیت و کنترلی .



ثباتهای وضعیت و کنترلی		ثباتهای سگمنت	
AX	IP	<input type="text"/>	CS
BX	Flogs	<input type="text"/>	DS
		<input type="text"/>	SS
		<input type="text"/>	ES

### ثبات های عمومی :

ثبات Ax : در اعمال ورودی و خروجی و محاسبات استفاده می شود .

	۱۵	۸	۷	۰
AX	<input type="text"/>		<input type="text"/>	
	AH		AL	

دو بخش بالا و پایین دارد .

ثبات Bx : به عنوان اندیس در توسعه آدرس و محاسبات بکار رفته به آن ثبات پایه هم

	۱۵	۸	۷	۰
BX	<input type="text"/>		<input type="text"/>	
	BH		BL	

می گویند

ثبات Cx : به آن ثبات شمارنده گفته شده و برای کنترل تعداد دفعات حلقه تکرار و محاسبات

	۱۵	۸	۷	۰
CX	<input type="text"/>		<input type="text"/>	
	CH		CL	

استفاده می شود .

ثبات Dx : به آن ثبات داده ها گفته شده و در اعمال ضرب و تقسیم با اعداد بزرگ بکار می

رود .

	۱۵	۸	۷	۰
DX	<input type="text"/>		<input type="text"/>	
	DH		DL	

سگمنت یا قطعه : ناحیه ای از حافظه است که آدرس شروع آن بر 16 قابل قسمت می باشد . اندازه هر سگمنت می تواند تا 64k باشد . چهار نوع سگمنت مختلف وجود دارد :

1-سگمنت کد Code segment

2- سگمنت داده ها – Data segment

3-سگمنت پشته Stack segment

4-سگمنت اضافی Extra segment

**سگمنت کد :** دستورات زبان ماشین در این سگمنت قرار می گیرند اگر برنامه بزرگتر از 64k باشد چند سگمنت کد می توانیم داشته باشیم – آدرس ابتدای سگمنت توسط ثبات Cs تعیین می شود

**سگمنت داده ها :** مقدار متغیرهای برنامه در آن قرار می گیرند آدرس ابتدای سگمنت توسط DS مشخص میشود .

سگمنت پشته: حاوی آدرس های برگشت از زیر برنامه ها است . در فراخوانی زیربرنامه ها استفاده می شود . ثبات SS ، آدرس ابتدای سگمنت را مشخص می کند .

**سگمنت اضافی :** برای انجام عملیات بر روی رشته ها استفاده می شود . آدرس ابتدای آن توسط Es تعیین می گردد .

آدرس شروع هر سگمنت معمولاً از محل هایی از حافظه که سمت راست آدرس آنها صفر است شروع می شود مانند:

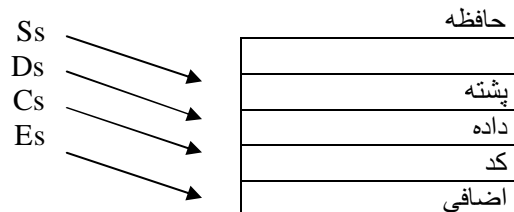
```

(
00000H
00010H
00020H
00030H

```

یعنی آدرس شروع هر سگمنت از حافظه نسبت به سگمنت بعدی حداقل به اندازه 16 بایت فاصله دارد . چون همواره اولین رقم سمت راست صفر است برای صرفه جویی در سخت افزار این صفر در

ثبات ها ذخیره نمی شود و فقط چهار رقم هگزا با ارزش در Es,Cs,Ds,SS ذخیره می گردد و در هنگام استفاده بوسیله سخت افزار یک صفر در جلوی آنها قرار داده می شود .

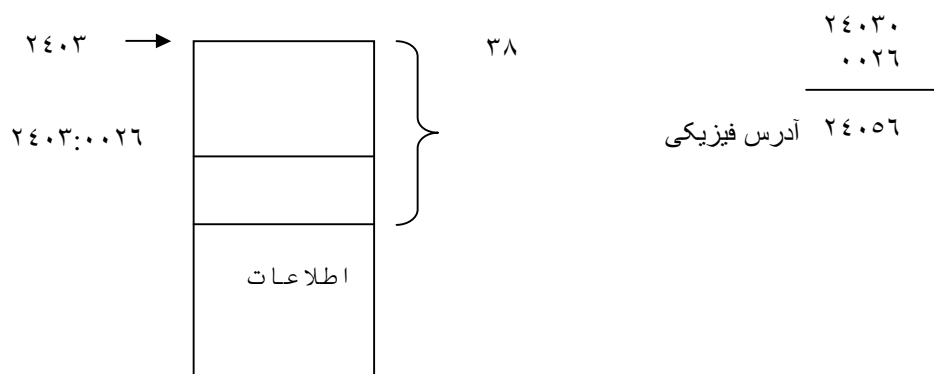


دلیل: ثباتها 16 بیتی و حافظه 2 خانه ای (1 meg) با این روش به جای 64k خانه، 1 meg آدرس دهی می شود .

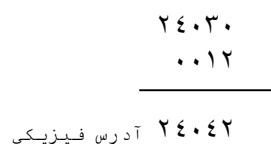
### آفست سگمنت :

در یک برنامه اسمبلی تمام محلهای حافظه نسبت به آدرس ابتدای سگمنت مشخص می شوند . این فاصله آفست آدرس گفته می شود و بین 0000H تا ffffH می باشد.

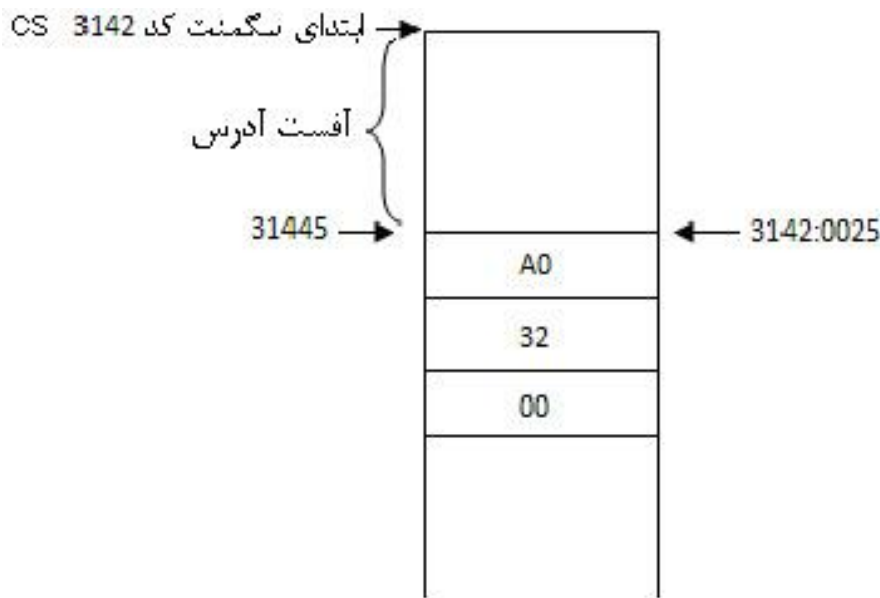
به عنوان مثال 24030H و سگمنت داده یعنی Ds=2403H اگر آفست 26H (بایت 38) یعنی فاصله این اطلاعات از ابتدای سگمنت داده 38 بایت است . آدرس منطقی به صورت : 2403 0026 می باشد . آدرس فیزیکی می شود :



MOV AL,[0012H] به معنای آن است که محتویات خانه 2403:0012 را به AL منتقل کن.



مجموعه ثباتهای IP : CS را آدرس منطقی دستور می نامند .  
 بعنوان مثال  $CS : IP 3142 : 0025 \equiv$  بدین معناست که آدرس شروع سگمنت کد 31420 و  
 آفست آدرس 0025 می باشد . اگر در این آدرس دستور `MOV AX,[0032H]` را داشته باشد  
 نحوه ذخیره سازی اطلاعات در حافظه به صورت زیر است :

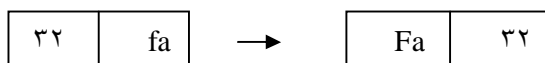


$$\begin{array}{r} 31420 \\ + 0025 \\ \hline 31445 \end{array} \quad \text{آدرس واقعی}$$

← A0 00 32 معادل زبان ماشین

بدین معنی است که در هنگام ذخیره اعداد بایت با ارزش در مکان بارزتر ذخیره می شود .

(HI-Address در HI-byte)



ثبات 2131 حافظه 2130

`MOV Dx, 8642` → `BA4286`

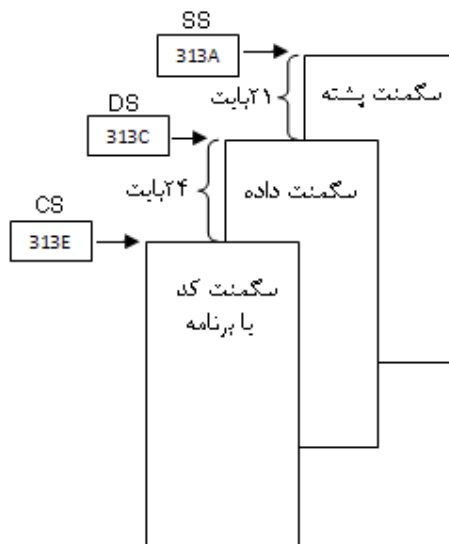
نحوه قرار گرفتن سگمنت پشته ، داده و کد در حافظه :

آدرس شروع هر سگمنت در حافظه ، نسبت به سگمنت بعدی ، حداقل باید 16 بایت فاصله داشته باشد . در این حالت امکان دارد قسمتی از سگمنتهای 64k بایتی روی هم بیافتد اما می توان این سگمنتها را در نقاط مختلف حافظه و مجزا اختیار کرد.

بعنوان مثال اگر SS=313AH باشد و در پشته 21 بایت رزو کنیم ، سگمنت داده باید در 2 سگمنت بعد (بدلیل اینکه هر سگمنت 16 بایت است) قرار گیرد .

و اگر در سگمنت داده ، 24 بایت تعریف کنیم . سگمنت کد نیز در 313c و سگمنت بعدی قرار خواهد گرفت .

$$\begin{array}{r}
 Ss=313A \\
 \quad \quad 2 + \\
 \hline
 Ds=313c \\
 \quad \quad 2 + \\
 \hline
 Cs=313E
 \end{array}$$



برنامه DEBUG.COM

دیبگ برنامه ای است که در سیستم عامل گنجانده شده است تا به برنامه نویس اجازه نظارت بر برنامه را برای رفع عیب بدهد. این برنامه برای بررسی و تغییر محتویات حافظه، ورود و اجرای برنامه ها و توقف اجرا در نقاط معین برای واری و تغییر داده مورد استفاده قرار می گیرد.

### ورود و خروج از دیباگ : C:\>debug

پس از تایپ Debug و زدن enter نشانه - در خط بعد ظاهر می شود. اکنون دیباگ منتظر تایپ فرمانی از جانب شماست. تمامی فرمانهای دیباگ را می توان به صورت کوچک یا بزرگ تایپ نمود برای خروج از دیباگ فرمان Q را تایپ می کنیم.

### بررسی و تغییر محتوای ثبات ها :

فرمان ثبات (R) اجازه بررسی و تغییر محتوای ثبات های درونی CPU را می دهد. این فرمان دارای ترکیب زیر است :

< نام ثبات > R

این فرمان اگر نام ثبات خاصی برده نشود محتویات همه ثبات ها را نمایش می دهد و در صورت ذکر نام فقط محتویات ثبات نام برده شده را نمایش می دهد.

خروجی فرمان R به صورت زیر خواهد بود :

C:\>Debug

-r

↵

Ax=0000 Bx=000 Cx=0000 Dx=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0AEA Es=0AEA SS=0AEA IP=0100 NV UP DI PL NZ NA PO NC

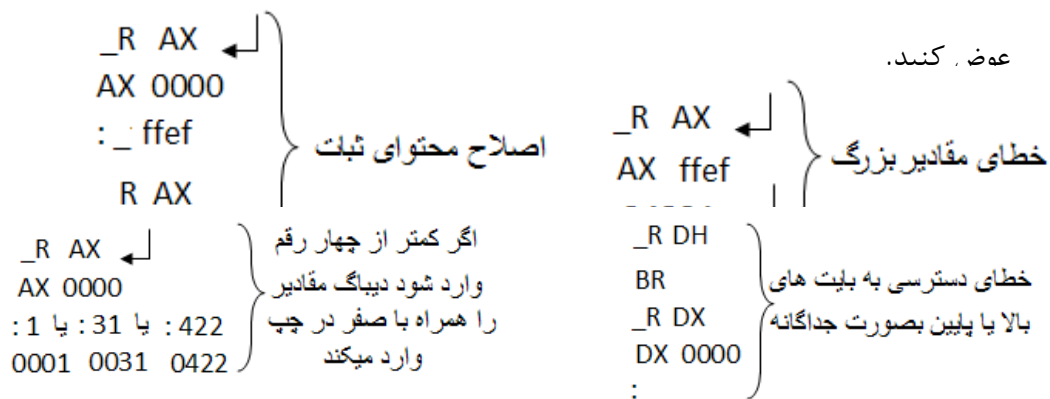
OAEA : 0100 B80100 MOV Ax , 0001

دیبگ سه خط اطلاعات را در این حالت می دهد. خط اول محتوای ثباتهای همه منظوره و اندیس را می دهد. خط دوم محتوای ثبات های قطعه، مقدار جاری ثبات دستورالعمل و بیت های ثبات پرچم را نشان می دهد. خط سوم دستوری را که با IP : CS به آن اشاره می شود نشان می دهد.



در هنگام ورود به دیباگ محتویات ثباتهای همه منظوره صفر شده و بیت های پرچم پاک شده و تمامی ثبات های قطعه دارای مقدار یکسان تعیین شده توسط سیستم عامل می شوند . هنگامی که برنامه اسمبلی در دیباگ بار شود ثبات های قطعه بر طبق پارامترهای برنامه تنظیم می شوند.

اگر نام ثبات در فرمان R ذکر شود محتویات ثبات نمایش داده شده و می توانید مقدار ثبات را



مقدمه ای بر برنامه نویسی اسمبلی :

یک برنامه زمان اسمبلی شامل مجموعه ای از خطوط دستورات زمان اسمبلی است . هر دستور شامل نمادی است که بطور اختیاری با یک یا دو عملوند دنبال می شود . عملوندها داده هایی هستند که باید دستکاری شوند نمادها همان فرمانهای CPU هستند که بر روی داده عمل می کنند . برای ترجمه برنامه های زبان اسمبلی 8086 به زبان ماشین ، اسمبلرهای بسیاری در دسترسند از معروفترین آنها MASM ساخت میکروسافت و TASA . ساخت بورلند است . از برنامه دیباگ که همراه سیستم عامل DOS ارائه می شود نیز می توان برای اسمبل کردن استفاده نمود .

## دستور MOV :

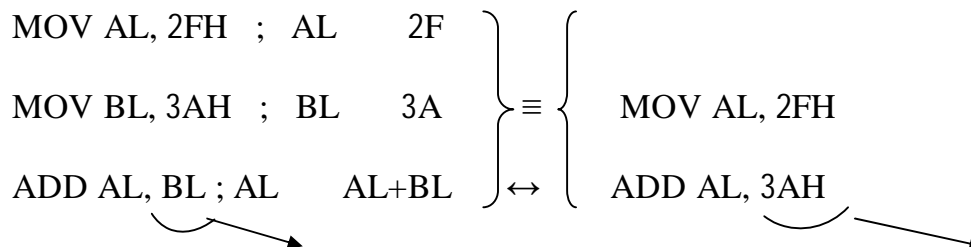
این دستور داده را از یک مکان به مکان دیگر کپی می کند . دارای قالب زیر است :

MOV destination , Source

کپی عملوند مبدأ به مقصد :



این دستور عملوندهای مبدأ و مقصد را با هم جمع کرده و فامیل را در مقصد قرار می دهد.



به این عملوند ، عملوند فوری می گوییم . عملوند مبدأ می تواند ثبات و یا یک داده فوری باشد اما عملوند مقصد باید ثبات باشد .

### کد کردن و اجرای برنامه ها در دیباگ :

حال به چگونگی ورود دستورات عملها به زبان اسمبلی در دیباگ می پردازیم .

### فرمان اسمبل کردن A :

این فرمان برای ورود دستورات عملهای اسمبلی به حافظه بکار می رود قالب آن به صورت روبرو است:

<آدرس شروع> A

آدرس شروع می تواند به صورت افسست تنها و یا قطعه کد و افسست داده شود . یعنی نتیجه مشابهی در برخواهند داشت .

A 200 ← A OAEF:100

در ادامه دیباگ منتظر ورود دستورات اسمبلی خواهد ماند . همزمان با ورود دستورات دیباگ دستورات را به کد ماشین تبدیل می کند . در صورت غلط وارد کردن دستور ، دیباگ پیام خطا اعلام می کند و دوباره منتظر ورود دستور می ماند . با تبدیل صحیح هر دستور به کد ماشین ، افسست به مکان بعدی اصلاح می شود .

نکته مهم: پیش فرض اعداد در دیباگ مبنای شانزده می باشد اما پیش فرض اعداد در اسمبلرها (TASM=MASM) مبنای ده می باشد. برای استفاده از اعداد مبنای شانزده در اسمبلرها باید به دنبال این اعداد H ذکر گردد.

-A 100

132F : 0100 MOV Bx,3

132F : 0103 MOV Ax,4

132F : 0106 MOV Cx,5

132F : 0109 Add Ax , Cx

132F : 010B Add Ax,Bx

132F : 010D INT 3

132F : 010E

تفاوت نوشتن یک دستور در دیباگ و اسمبلر در زیر آمده است :

MOV Ax,AB4F ⇒ دیباگ

MOV Ax,AB4F H ⇒ اسمبلر

نکته: آدرسهای صفر تا 100H (256 خانه اول) برای DOS ذخیره شده است و کاربر نمی تواند از آن استفاده کند. بنابراین برای دستور اسمبلر A باید از 100 به بعد شروع کرد.

**فرمان عکس اسمبلر U:** تبدیل از زمان ماشین به دستور اسمبلی

این فرمان کد ماشین و معادل زمان اسمبلی دستور را نمایش می دهد. این دستور عکس دستور اسمبلر A را انجام می دهد. دستور A دستورات عمل های اسمبلی را از کار برگرفته به زبان ماشین تبدیل و در حافظه ذخیره می کند. دستور U زبان ماشین ذخیره شده در حافظه را دریافت، و به دستور اسمبلی تبدیل و نمایش می دهد. این فرمان دارای دو قاب به صورت زیر است:

$$\left\{ \begin{array}{l} \text{> آدرس پایان < > آدرس شروع < -U} \\ \text{تعداد بایت } \rightarrow \text{LD (شروع) 100 -U} \\ \text{< تعداد بایت L > < آدرس شروع < -U} \end{array} \right.$$
 دستور 1 فرمان می دهد از آدرس شروع CS : 0100 تا CS: 010D تبدیل به اسمبلی کند .

دستور 2 فرمان می دهد از آدرس شروع CS : 0100 به تعداد D بایت را تبدیل به اسمبلی کند .

103D :0100 BB0300 MOV Bx,0003

103D : 0103 B80400 MOV Ax,0004

103D : 0106 B90500 MOV Cx,0005

103D : 0109 01C8 ADD Ax,CX

103D : 010B 01D8 ADD Ax,Bx

103D : 010D CC INT 3

فرمان U بدون پارامتر به معنای تبدیل 32 بایت از IP : CS می باشد . فرمان U بعدی موجب می شود 32 بایت جدید از ادامه 32 بایت قبلی تبدیل شود با این روش می توان محتویات یک فایل بزرگ را مشاهده کرد .

**فرمان اجرا (G) :** این فرمان با دیباگ دستور می دهد تا دستور عملهای بین دو آدرس را اجرا کند ، قالب آن به صورت زیر است :

G <آدرس پایان > <آدرس شروع=>

این دستور را به چند صورت می توان بکار برد :

حالت 1- بدون دادن آدرس- دیباگ در این حالت شروع به اجرای دستورات از IP : CS نموده و تا رسیدن به نقطه توقفی مانند INT3 به اجرا ادامه می دهد . در این حالت مهم نیست چه تعداد نقطه توقف داریم دیباگ در اولین نقطه توقف متوقف می شود.

در این حالت 010D و 0AEf به معنای آن است که دستور بعدی برای اجرا INT3 است .

-R

Ax=0000 Bx=0000

DS=0AEF Es=0000 SS=0000 CS = 0000 IP=0100

0AEF = 0100 BC=300 MOV Bx,0003

-g

Ax=000C Bx=0003 Cx=0005 Px=0000

Ds=000C Bs=0003 SS=0005 IP=010D

0Aef=010D CC INT 3

حالت 2 به شکل آدرس شروع = G

در این حالت دیباگ از آدرس شروع ، اجرا کرده تا رسیدن به نقطه توقف ادامه می دهد.

103 D : 010D CC INT3 -G=100 یا G=0Aef - 0100

G=100 106

حالت 3 شکل آدرس پایان آدرس شروع = G

در این حالت فقط دستورات بین دو آدرس اجرا می شود . Ax=004 Bx=003 Cx=0000

حالت 4 شکل آدرس G

-G 109

Ax=0004 Bx=0003 Cx=0005

103D:0109 , 01CS ADD AX,CX

در این حالت فقط آدرس پایان داده شده و آدرس شروع داده نشده است. دیباگ بطور پیش فرض

مقدار CS:IP را بعنوان آدرس شروع بکار می برد .

همانطور که قبلاً گفته شده هر برنامه اسمبلی می تواند از سه سگمنت (قطعه) تشکیل شود .

- 1- قطعه کد : این سگمنت حاوی دستورات زبان اسمبلی می باشد که این دستورات وظایف برنامه را انجام می دهند .
  - 2- قطعه داده : از این قطعه برای ذخیره اطلاعاتی که باید بوسیله دستورات قطعه کد استفاده شود ، بکار می رود .
  - 3- قطعه پشته : از این قطعه برای ذخیره اطلاعات موقت استفاده می شود .
- بعنوان مثال برنامه ای برای جمع 5 بایت داده به صورت زیر داریم :

```
MOV AL,00H
```

```
Add AL,36H
```

```
Add AL,2CH
```

```
Add AL,3FH
```

```
Add AL,9CH
```

```
Add AL,1BH
```

مشکل این برنامه آن است که دستورات و داده ها با هم مختلط شده اند در نتیجه اگر قصد داشته باشید یکی از پنج عددی که قصد جمع کردن دارید را عوض کنید بدین معنی که داده عوض شود باید کل کد را جستجو نموده و داده مورد نظر را عوض کنید . بهمین دلیل بهتر است داده ها را در قطعه داده ذخیره کنیم تا دسترس و تغییر آنها آسانتر باشد .

بعنوان مثال تفاوت مکان (آفست) قطعه داده 300H می باشد و DS نیز حاوی آدرس شروع این قطعه می باشد . برنامه بهبود داده شده به صورت زیر است :

```
DS:0300=36 MOV AL,0
```

```
DS:0301=2C ADD AL,[0300]
```

DS:0302=3F    ADD AL,[0301]

DS:0303=9C    ADD AL,[0302]

DS:0304=1B    ADD AL,[0303]

ADD AL,[0304]

آدرس افست درون کروش است . کروش به معنی آدرس داده و نه خود داده می باشند. حال اگر قصد داشته باشیم داده را به جای تفاوت مکان (افست) 300 در افست دیگری ذخیره کنیم برنامه باید اصلاح شود : در این حالت باید ثباتی برای نگهداری آدرس افست استفاده کنیم. 8086 فقط اجازه استفاده از ثباتهای DI,SI,BX را بعنوان ثبات تفاوت مکان برای قطعه داده می دهد .

*BX*

*SI    DS    ,    IP ← CS*

*DI*

دستور "INC BX" معادل دستور "ADD BX,1" می باشد .

برنامه اصلاح می شود و به صورت زیر است

در این حالت اگر قصد تغییر افست را داشته باشیم فقط کافی است یک خط را بصورت جزئی تغییر دهیم . در این برنامه حلقه می توانست استفاده شود .

MOV AL,0

MOV Bx,0300H

ADD AL,[Bx]

INC Bx

ADD AL,[Bx]

INC Bx

ADD AL,[Bx]



INC Bx

ADD AL,[Bx]

همانطوری که می دانیم کامپیوترهای 8086 از قرارداد HI-byte در HI-address استفاده می کنند . این قانون برای داده های 16 بیت نیز صادق است . در این حالت بایت بالارزش بالاتر به مکان بالاتر قطعه داده رفته و بایت کم ارزشتر به مکان پایینتر قطعه داده می رود . یعنی :

MOV Dx, 3BCA

MOV [1700],Bx

DS:1700=CA

DS:1701=3B

فرمان ردیابی T : با این فرمان می توان به هر تعداد دلخواه دستور را اجرا نموده و علاوه بر آن تأثیر برنامه روی ثبات ها را ردگیری کرد :

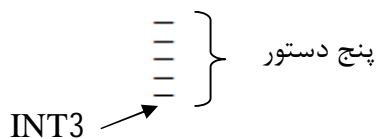
<تعداد دستورالعمل ها > <آدرس شروع = T>

به عنوان مثال 5 T=100 بدین معنی است که از آدرس 100 به تعداد 5 دستور را اجرا کن . اگر تعداد دستورالعملها داده نشود مقدار پیش فرض یک در نظر گرفته می شود . اگر آدرس شروع داده نشود CS:IP در نظر گرفته می شود .

تفاوت این فرمان با فرمان اجرا G در آن است که فرمان T پس از اجرای هر دستور محتوای ثبات ها را نشان می دهد در صورتی که فرمان اجرا G محتوای ثباتها را تا پایان برنامه نشان نمی دهد . فرمان T در برنامه اجازه می دهد تا آنچه را که بوسیله یک دستور از برنامه اتفاق می افتد مشاهده کنید . فرمان T بدون پارامتر باعث اجرای فقط یک دستور خواهد شد .

اگر سرعت رد شدن دستورات زیاد بود با Ctrl+Numlock می توان توقف نمود . و با فشار دوباره کلید ادامه کار انجام شود .

T=100 5



### روشهای آدرس دهی 8086 :

دستورات مختلف برای انجام ، باید عمل خود را بر روی داده ها انجام دهند . روشهای مختلف برای دسترسی به عملوندها (داده ها) وجود دارد . به این روشها ، روشهای آدرس دهی گفته می شود . در واقع روش آدرس دهی روشی است که برنامه نویس به cpu محل برداشتن عملوند (داده) را نشان می دهد. در 8086 هفت روش آدرس مختلف وجود دارد:

- 1- ثباتی
- 2- فوری
- 3- مستقیم
- 4- غیرمستقیم ثباتی
- 5- نسبی پایه
- 6- نسبی اندیس
- 7- نسبی اندیس دار پایه

1-روش آدرس دهی ثباتی : در این روش از ثبات ها برای نگهداری داده یا همان عملوندها استفاده می شود . بهمین دلیل نیاز به دستیابی به حافظه نداریم در نتیجه دستوراتی که از این روش آدرس دهی استفاده می کنند نسبتاً سریع هستند .

MOV Ax,Bx

MOV ES,Cx

ADD AL,DL

2-روش آدرس دهی فوری: در این روش عملوند مبدا یک مقدار ثابت است. همانطور که می دانیم بعد از تبدیل به زبان ماشین، عملوند بلافاصله بعد از کد دستور در حافظه ذخیره می شود بهمین دلیل CPU دسترسی سریع به عملوند دارد و اجرای دستور سریع صورت می گیرد. همانطور که گفته شد محدودیت این دستور در قرار دادن داده فوری درون ثباتهای قطعه و ثبات پرچم است.

MOV Bx,0003 → BB0300, MOV CX,0005 → B90500

در دو روش بالا عملوند یا در داخل CPU یا همراه دستور است و دسترسی آن با سهولت و سریع صورت می گیرد در روشهای بعدی داده اغلب خارج CPU و در جایی درون حافظه قرار داد. دسترسی به داده ها به سهولت دو دستور بالا نمی باشد.

3-روش آدرس دهی مستقیم: در این روش به جای داده آدرس داده بلافاصله بعد از کد دستور می آید. خود داده در مکانی دیگر در حافظه است. برعکس روش فوری که خود عملوند همراه دستور است. در این روش آدرس عملوند همراه دستور است. همانطور که قبلاً گفته شد آدرس موجود در دستور، آدرس تفاوت مکان (افست) در قطعه داده است. آدرس فیزیکی را می توان با ترکیب آدرس افست و DS بدست آورد

انتقال محتویات DS:1300 درون CL: MOV CL,[1300]

در این حالت اگر گروه موجود نبود خطا تولید می شد زیرا CL هشت بیتی و 1300 16 بیتی است این دستور باعث می شود محتوای آدرس DS:1300 درون CL قرار گیرد. این دو دستور باعث می شود که مقدار H 32 درون آدرس 1244:2342 حافظه ذخیره گردد.

MOV BL,32H

MOV [2342],BL

DS=1244      12440

2342

14782

آدرس فیزیکی

4-روش آدرس دهی غیر مستقیم ثباتی : در این روش آدرس مکان حافظه ای که عملوند در

آن است بوسیله ثبات نگهداری می شود . نکته قابل توجه آن است که در این روش فقط ثباتهای

SI,DI,BX را می توان بعنوان نگهداری کننده آدرس عملوند (اشاره گر) استفاده نمود.

محتویات خانه DS:BX حافظه درون CL قرار می گیرد      MOV CL,(BX);

محتویات خانه DS:SI حافظه درون CL قرار می گیرد      MOV CL,(SI),

محتویات خانه DS:DI حافظه از AL درون آن قرار می گیرد      MOV (DI),AL;

محتوای BX در مکانهای DS:SI و DS:SI+1 منتقل می شود HI-      MOV (SI),BX;

Byte در HI-Address رعایت می گردد

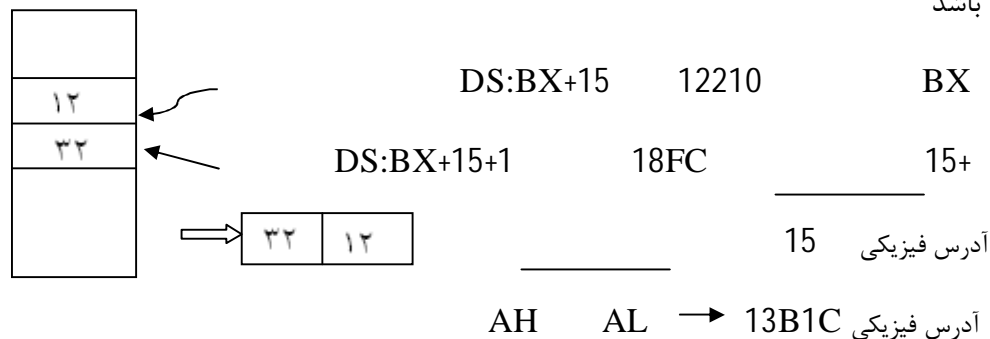
5-روش آدرس دهی نسبی پایه : در این روش مفهومی به نام آدرس مؤثر داریم .

در این روش از ، ثبات های پایه Bx یا BP و مقدار جابجایی برای محاسبه آدرس مؤثر استفاده می

شود . قطعه پیش فرض آدرس فیزیکی برای DS,Bx و برای BP ، SS می باشد .

در دستور MOV Ax,[Bx]+15 آدرس مؤثر Bx+15 می باشد . آدرس فیزیکی صورت DS0

می باشد



این دستور موجب می شود محتویات خانه های  $DS:BX+15$  و  $DS:BX+15+1$  درون  $AX$  قرار گیرد. آدرس پایین به  $AL$  و آدرس بالا درون  $AH$  قرار می گیرد.

`MOV Ax,[Bx+15]`

`MOV Ax,15[Bx]`

$$\begin{array}{r} SS\phi \\ BP \\ 6 \\ \hline \text{آدرس فیزیکی} \end{array}$$
 یکی معادل  $BP+6 = \text{آدرس مؤثر}$   
`MOV CL,[BP]+6`  
`MOV CL,[BP+6]`  
`MOV CL,6 [BP]`

6-روش آدرس دهی نسبی اندیس دار: آدرس فیزیکی در این روش مشابه روش آدرس دهی نسبی پایه است با این تفاوت که تباتهای  $SI,DI$  آدرس تفاوت مکان را در خود نگهداری می کنند.

`MOV Dx,[SI]+20` ; آدرس فیزیکی =  $DS\ 0 + SI+20$

`MOV AL,[DI]+32` ; آدرس فیزیکی =  $DS\ 0 + DI+32$

قطعه پیش فرض در این روش  $DS$  است.

روش آدرس دهی نسبی اندیس دار پایه: این روش از ترکیب دو روش پایه و اندیس دار بوجود می آید. این روش دارای یک ثابت پایه و یک ثابت اندیس است ابتدا ثابت پایه سپس ثابت اندیس ذکر میگردد. قطعه پیش فرض برای  $DS, BX$  و برای  $SS, BP$  است.

`MOV AL,[BX][SI]+16;` آدرس فیزیکی =  $DS0+BX+SI+16$

`MOV AL,[BX+SI+16];` معادل

`MOV AH,[BX][DI]+5;` آدرس فیزیکی =  $DS0+BX+DI+5$

`MOV AH,[BX+DI+5];` معادل

`MOV BL,[BP][DI]+6;` آدرس فیزیکی =  $SS\ 0 + BP+DI+6$

`MOV BH,[BP][SI]+30;` آدرس فیزیکی =  $SS\ 0 + BP+SI+30$

خلاصه روشهای آدرس دهی به صورت زیر است :

روش آدرس دهی	عملوند	قطعه پیش فرض
ثباتی	reg	-
فوری	data	-
مستقیم	[offset]	DS
غیر مستقیم ثباتی	[Bx]	DS
	[SI]	DS
	[DI]	DS
نسبی پایه	[Bx]+disp	DS
	[BP]+disp	SS
نسبی اندیس دار	[DI]+disp	DS
	[SI]+disp	DS
نسبی اندیس دار پایه	[Bx][SI]+disp	DS
	[Bx][DI]+disp	DS
	[BP][SI]+disp	SS
	[BP][DI]+disp	SS

ثبات های تفاوت ممکن برای قطعات مختلف بطور پیش فرض معین می باشد که به صورت زیر است :

نام ثبات : CS DS ES SS

SP,BP      SI,DI,Bx      SI,DI,Bx      IP : ثبات تغییر مکان

می توان قطعه پیش فرض را لغو کرده و ثبات قطعه دیگری را استفاده نمود برای انجام این کار

باید نام قطعه را در دستور ذکر کنیم .

اعداد در دیباگ در مبنای 16 و در اسمبلر در مبنای 10 می باشند .

بعنوان مثال در دستور MOV AL,[BP] قطعه پیش فرض SS:BP می باشد برای حذف آن

دستور را به صورت MOV AL,CS : [BP] می توان نوشت در این صورت عملوند موجود در

خانه CS : BP به جای خانه SS : BP درون AL می رود.

	قطعه به کار رفته	قطعه پیش فرض
MOV SS:[BX][DI]+۳۰,AX	SS:BX+DI+۳۰	DS:BX+DI+۳۰
MOV Dx,ES:[BP]+20	ES:BP+20	SS:BP+20

**دستکاری داده در دیباگ :** سه فرمان برای بررسی و تغییر محتویات حافظه در دیباگ وجود

دارند که عبارتند از :

F : پر کردن یک بلوک ، از حافظه بوسیله داده ای که داده می شود.

D : فرمان تخلیه یا همان نمایش محتوای حافظه روی صفحه نمایش.

E : فرمان وارد کردن داده که محتوای حافظه را تغییر می دهد.

فرمان پر کردن Fill برای پر کردن حافظه با داده ای از طرف کاربر استفاده می شود .

دارای قالب زیر است :

<داده> <آدرس پایان> <آدرس شروع> F

<داده> <تعداد بایت L> <آدرس شروع> F

معمولاً از این فرمان برای پر کردن قطعه داده استفاده می شود در این حالت آدرس شروع و پایان ، آدرس های تفاوت مکان در قطعه داده است . برای قطعات دیگر باید نام ثابت قطعه قبل از تفاوت مکان ذکر شود .

از DS:100 تا DS:10F را با FF پر می کند .

-F 100 10F FF

از CS:100 تا CS:1FF (256 بایت) با 20 پر می کند

-F CS:100 1FF 20

داده رشته 00FF است - 20H بایت (32 بایت) داده 00FF L20 100 -F با شروع از DS:100 با 00FF پر می شود .

فرمان Dump تخلیه D برای بررسی محتویات حافظه :

قالب آن به شکل زیر است :

<آدرس پایان> <آدرس شروع> -D

<تعداد بایت L> <آدرس شروع> -D

فرمان D می تواند با آدرس شروع و پایان و همچنین آدرس شروع و تعداد بایت ها در مبنای 16 بکار رود . در هر دو صورت محتویات حافظه نمایش داده می شود .

فرمان D تنها باعث می شود، دیباگ 128 بایت متوالی از DS:100 را نشان دهد با هر بار زدن D 128 بایت بعدی نشان داده می شود .

-F 100 14F 20	112A:0100	20 20 .... 20	10F
-F 150 19F 00	112A:0110	20 20 .... 20	11F
	⋮		
-D 100 19F	112A:0140	20 20 ... 20	14f
	112A:0150	00 00 .....00	15F



⋮  
112A:0190      00 00 .....00      19F

از فرمان Dump می توان برای دیدن صرفاً زبان ماشین موجود در قطعه کد استفاده نمود در این حالت وظیفه مشابه دستور U است ولی در دستور Dump صرفاً کد ماشین نمایش داده می شود و معادل دستور اسمبلی نمایش داده نمی شود .

-U      100    11E

1232:    0100    B057   MOV   AL,57

1232:    0102    B686   MOV   DH,86

1232:    0104    B272   MOV   DL,72

          :    0106

⋮

-D   CS:100   11F

1231:0100   B0    57   B6   86   B2   72....OW6,2r.R.Q.G39

1232:0110   01    D9   .....y.5t..ce.t.sv

در انتها کاراکترهای اسکی معادل کد بایت نمایش داده می شود اگر محتوای یک بایت کد اسکی نباشد قابل نمایش نبوده و با "." نشان داده می شود .

**فرمان ورود E برای وارد کردن داده به حافظه :**

این فرمان برخلاف فرمان F برای پر کردن حافظه با یک نوع داده بکار می رفت می تواند برای پر کردن حافظه با لیستی از داده های مختلف بکار رود . دارای قالب زیر است :

<لیست داده> <آدرس>-E

<آدرس>-E

برای تغییر آدرس

-E 100 'John Snith '

-D 100 10f

113D:0100 4A 6F 68 6E 20 53 6E<sup>۱۰۶</sup> 69 74 68 20 20 20 john snith

در این دستور داده اسکن در علامت ' قرار دارد . اگر فرمان E بدون داده و فقط با آدرس داده شود دیباگ فرض خواهد کرد که مایل به بررسی آن بایت و احتمالاً تغییر آن هستید. سپس چهار انتخاب دارید :

1- با دادن مقدار جدید جایگزین مقدار قبلی می شود :

-E 106

113D:0106 6E - 6D

-D 100 10F

113D:0100 4A 6 F.....6D .....John Smith

2- زدن enter به معنی عدم تمایل برای تغییر داده است . و برای رفتن به خانه بعدی از Space و برای رفتن به خانه قبلی از خط تیره استفاده می شود .

3- کلید فاصله باعث می شود بایت در حال نمایش را بدون تغییر گذاشته و بایت بعدی برای تغییر نشان داده میشود .

-E 100

113D:0100 4A, 6f, 68, 6E, 20, 53, 6E, 6D

-D 100 10F

113D:0100 4A, 6f, 68, 6E, 20, 53, 6E, 6D, 69.....John Smith

4-وارد کردن علامت منفی "-" باعث میشود بایت در حال نمایش را رها کرده و بایت قبلی را نشان دهد .

-E 107

113D:010 7 69.-

113D:0106 6E. 6D

از فرمان E برای وارد کردن داده های عددی هم استفاده می شود .

-E 100 32 24 B4 02 3F

### بررسی و تغییر ثبات پرچم

کامپیوترهای شخصی دارای ثبات پرچم FR (Flag Register) شانزده بیتی می باشد این رجیستر وضعیت فعلی پردازنده را مشخص می کند شش بیت پرچم Of.Pf.Af,Sf,Zf,Cf را پرچم های شرطی می نامند زیرا در نتیجه اجرای دستورات محاسباتی یک یا صفر می شوند . سه بیت پرچم IF و TF و DF پرچم های کنترل می باشند زیرا برای کنترل عملیات دستورات استفاده می گردند .

### 1-بیت CF-Carry Flag

این بیت حاوی رقم نقلی است و هنگامی  $CF=1$  می شود که در محاسبات یک بیت نقلی ایجاد شود .

MOV Ax,FFFF

Add Ax,1

### 2-بیت تشخیص صفر-Zf- Zero Flag

این بیت هنگامی یک می شود که نتیجه عملیات محاسباتی یا منطقی برابر صفر شود .

MOV Ax,3

Add Ax,FFFD

### 3-بیت پرچم علامت – Sign Flag -SF

بعد از عملیات محاسباتی و منطقی مقدار بیت علامت یا همان پر ارزش ترین بیت روی بیت پرچم علامت کپی می شود اگر نتیجه علامت منفی باشد این بیت برابر یک وگرنه برابر صفر . لذا بیت پرچم علامت ، علامت نتیجه آخرین محاسبات را نشان می دهد .

### 4-بیت پرچم نقلی کمکی -AF -Auxiliary carry flag

چنانچه در محاسبات از چهارمین بیت ، بیت نقلی به بیت بعدی ایجاد شود AF یک می گردد .

### 5-بیت پرچم توازن – Parity flag-PF

بعد از عملیات محاسباتی یا منطقی ، بایت کم ارزش بررسی می گردد ، اگر تعداد یک ها زوج باشد  $Pf=1$  اگر فرد باشد  $fPF=$  می گردد . (توازن فرد)

### 6-بیت پرچم سرریز -of-overflow Flag (Cout clg = $\pm 1$ )

این بیت هنگامی یک می شود که نتیجه عملیات اعداد جبری خارج از مقدار مجاز باشد در نتیجه سرریز رخ می دهد .

### 7-بیت فعال کردن وقفه -If-Interrupt flag

اگر این بیت برابر یک باشد ، سیستم به وقفه های خارجی پاسخ می دهد وگرنه وقفه ها را نادیده می گردد .

### 8-بیت پرچم Trap Flag-TF یا Trace flag

Trace به معنی قدم به قدم است. چنانچه این بیت برابر با یک باشد ، اجرای برنامه به صورت دستور به دستور انجام می شود . این عمل برای پیدا کردن اشتباه در برنامه وسیله مناسبی می باشد .

### 9-یک پرچم Direction flag-DF

این بیت برای کنترل جهت عملیات بعضی دستورات خاص بکار می رود . اگر این بیت برابر با یک باشد ، عمل مقایسه یا شیفت از سمت راست به چپ و گرنه از چپ به راست انجام می شود .

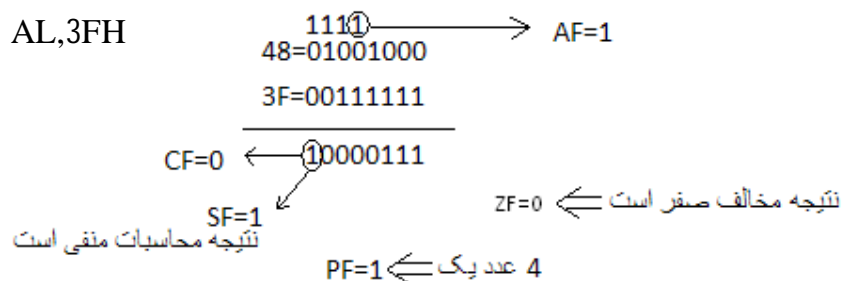
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

نکته : تمام دستورات روی بیت های پرچم اثر نمی گذارند . به عنوان مثال دستور MOV ، فقط اطلاعات را منتقل می کند و روی بیت های پرچم اثر نمی گذارد . اما دستورات محاسباتی و منطقی مانند ..... و SUB,ADD روی بیت های پرچم اثر می گذارند.

MOV AL,48H

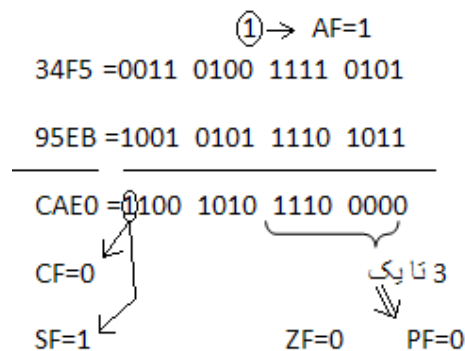
مثال

ADD AL,3FH



MOV Bx,34F5H

Add Bx,95EBH



پرچم	برقراری پرچم (=1)	پاک شدن پرچم $f(=)$
پرچم سرریز OF	سرریز OV	بدون سرریز NV
پرچم جهت DF	پایین DN	بالا UP
پرچم وقفه IF	وقفه فعال EI	وقفه غیر فعال DI
پرچم علامت SF	منفی NG	مثبت PL
پرچم صفر ZF	صفر ZR	غیر صفر NZ
پرچم نقلی کمکی AF	نقل کمکی AC	بدون نقل کمکی NA
پرچم توازن PF	توازن زوج PE	توازن فرد PO
پرچم نقلی CF	نقلی CY	بدون نقلی NC

در هنگام ورود به دیباگ اگر همه بیت های ثبات پرچم پاک یا صفر باشند این ثبات به صورت ریز دیده می شود.

NV UP DI PL NZ NA PO NC

اگر همه پرچم ها در یک قرار داشته باشند ثبات به صورت ریز دیده می شود :

OV DN EI NG ZR AC PE CY

-تغییر محتوای ثبات پرچم

-R F

NV UP DI PL NZ NA PO NC-DN OV NG

-R F

OV DN DI NC NZ NA PO NC

MOV BX , AAAA

ADD BX,5556

INT 3

.....BX=0000

AAAA  
5556  
-----  
10000  
CF=1      ZF=1

NV UP DI PL ZR AC PE CY

برنامه نویسی اسمبلی :

در این بخش اجزا یک برنامه ساده به زبان اسمبلی که قرار است با اسمبلر ، اسمبل گردد ، بحث خواهد شد .

هر برنامه اسمبلی شامل مجموعه ای از دستورات زبان اسمبلی و عباراتی دیگر به نام رهنمون ها می باشد . رهنمون ها که شبه دستور هم خوانده می شوند در هنگام ترجمه، به اسمبلر راهنمایی کرده و جهت می دهند . رهنمون ها به زبان ماشین ترجمه نمی شوند . اسمبلر از شبه دستورات برای سازمان دهی برنامه استفاده می کند .

هر دستور زبان اسمبلی دارای چهار قسمت مختلف است :

[توضیحات ؛] [عملوندها] نماد [برچسب]

برچسب اجازه می دهد تا با یک نام به خطی ارجاع داده شود . برچسب از 31 کاراکتر تجاوز نمی کند .

اگر برچسب به دستور اشاره کند باید دارای دو نقطه باشد ولی برای رهنمونها نیازی به دو نقطه نیست. توضیحات ممکن است در انتهای یک خط بوده و یا یک خط را تشکیل دهند اسمبلر توضیحات را نادیده می گیرد ولی برای برنامه نویس مفیدند . استفاده از توضیحات برای خواندن و درک برنامه پیشنهاد می شود

تعریف قطعات برنامه :

یک برنامه شامل حداقل سه قطعه پشته ، داده و کد است . رهنمون SEGMENT شروع سمگنت و رهنمون ENDS پایان قطعه را اعلام می کند .

## Label SEGMENT

## Label Ends

نام سگمنت باید یک شناسه مجاز باشد .

1- با رقم و نقطه و @ شروع نشود .

2- حداکثر 31 کاراکتر

3- ترکیب حروف و ارقام و علائمی مثل ؟ ، - ، @ ، \$

قالب کلی یک برنامه اسمبلی را می توان به صورت زیر در نظر گرفت :

تعریف سگمنت پشته

تعریف سگمنت داده

Segment نام سگمنت کد

Proc far نام برنامه

⋮

End p نام برنامه

Ends نام سگمنت کد

end نام برنامه

سوال: برنامه ای به زبان اسمبلی بنویسید که دو بایت اول DS را با هم جمع نموده و حاصل را در

بایت بعدی قرار دهد.

STSEG SEGMENT

DB 64 DUP (?) رزرو 64 بایت حافظه

STSEG ENDS

.....;

DTSEG SEGMENT



```
DATA1    DB    52 H } تخصیص مقادیر و ذخیره سازی در حافظه
DATA2    DB    29 H }
```

```
SUM      DB    ?    مقدار بعدا مشاهده می شود
```

```
;.....
```

```
CDSEG     SEGMENT
```

```
MAIN      PROC FAR
```

```
          ASSUME CS:CDSEG , DS:DTSEG , SS:STSEG
```

```
          MOV     AX , DTSEG
```

```
          MOV     DS , AX
```

```
          MOV     AL , DATA1
```

```
          MOV     BL , DATA2
```

```
          ADD     AL , BL
```

```
          MOV     SUM , AL
```

```
          MOV     AH , 4C H
```

```
          INT     21 H
```

```
MAIN      END
```

```
CDSEG     ENDS
```

```
          END     MAIN
```

رویه (Procedure) گروهی از دستورات است که برای انجام عمل خاصی در نظر گرفته شده اند .  
 قطعه کد ممکن است یک یا چند رویه باشند .

PROC رهنمون PROC می تواند  
 FAR نام رویه

از نوع FAR یا NEAR  
 ENDP نام رویه

باشد . در DOS نقطه ورود به برنامه کاربر باید از نوع FAR باشد .

در ادامه باید هر سگمنت را به ثبات آن سگمنت مربوط کنیم یعنی سگمنت کد را به ثبات سگمنت کد ، سگمنت داده را به ثبات سگمنت داده و الی آخر . اینکار بدین دلیل لازم است که در یک برنامه به زبان اسمبلی ممکن است چندین قطعه داده و یا اضافی یا غیره وجود داشته باشد ولی هر بار فقط یکی از آنها بوسیله CPU آدرس دهی می شود . زیرا ثباتهای قطعه در هر لحظه فقط یک مقدار دارند . بعنوان مثال در دستور MOV AL,[Bx] دقیقاً باید معلوم باشد Bx آفست نسبت به کدام یک از قطعات داده است .

پس از تحویل کنترل از DOS به برنامه کاربر ، CS,SS توسط سیستم عامل مقدار دهی مناسبی شده اند و دارای مقادیر صحیحی هستند ولی مقادیر DS و ES باید بوسیله برنامه مقدار دهی شوند .

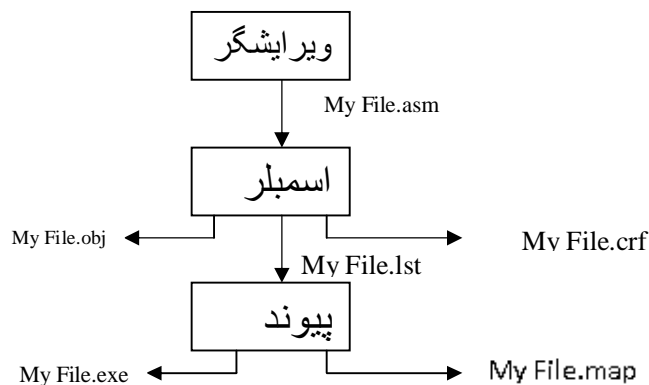
MOV Ax,DTSEG

MOV DS,Ax

در ادامه دستورات اصلی قرار دارند . برنامه غیر از دستورات اصلی را پوسته برنامه اسمبلی می نامند.

MOV AH,4CH هدف بازگشت کنترل به سیستم عامل است .  
 INT 21H صورت عدم استفاده از این دستورات کامپیوتر قفل می کند.

سه خط آخر رویه و سگمنت قطعه را پایان می دهند .



### اسمبل ، پیوند و اجرای برنامه های اسمبلی:

برنامه های اسمبلی تا زمانی که با زبان ماشین ترجمه نشوند قابل اجرا نیستند . برای ترجمه می توان از برنامه MASM یا TASM استفاده نمود .

### 1- اسمبل کردن برنامه با ماکرواسمبلر :

Ex.Asm ← MASM

برنامه ماکرواسمبلر نام برنامه اسمبلی کاربر را با : Source Filename [Asm] پسوند Asm سؤال می کند .

در ادامه نام برنامه مقصد یا ترجمه شده را Ex.obj : object filename [Ex.obj] سؤال می کند . اگر نام برنامه مقصد و مبدأ یکی باشد فقط کافی است enter بزنیم .

Source listing [NUL.Lst] : Ex.LST

در ادامه در مورد فایل لیست سؤال می کند . پیش فرض آن است که برنامه لیست لازم نیست . با زدن enter فایل لیست ایجاد نمی شود در صورت دادن نام ، فایل لیست ایجاد می شود .

فایل لیست شامل اصل برنامه ، ترجمه آن و آدرس های متناظر هر دستور است در صورتی که برنامه اشتباهی داشته باشند در فایل لیست اشتباهات توسط اسمبلر مشخص می گردد و در صورت وجود اشتباه فایل obj تولید نمی گردد .

می توان دستور MASM را به صورت زیر نیز نوشت .

MASM EX.ASM , EX.OBJ , EX.LST

و یا در صورت یکسان بودن نامها :

MASM EX.ASM, ,

ماکرو اسمبلر فایل دیگری با پسوند CRF می تواند تولید کند . این فایل ، لیست سمبل ها ، برچسب ها و شماره سطرهایی که آنها در فایل LIST تعریف شده اند را میدهد . با استفاده از برنامه ای به نام CREF می توان آن را تبدیل به کد اسکی نمود تا قابل دیدن روی مانیتور باشد . این فایلها کمتر مورد استفاده قرار می گیرند و معمولاً Enter زده می شود : (ارجاع متقابل)

Cross-reference [ NUL.CRF] :

C:>Cref Ex.crf Ex.asc

تبدیل به فایل اسکی

### پیوند برنامه – LINK :

اسمبلر کدهای عملیات ، عملوندها و آدرس های تفاوت مکان را در فایل مقصد "obj" ایجاد می کند . نوع آماده برای اجرا از یک برنامه ، بوسیله برنامه LINK تولید می شود که پسوند "exe" را داراست . برنامه LINK فایل را طوری ایجاد می کند که بوسیله DOS قابل باز شدن و اجرا شدن می باشد .

LINK.EXE

LINK Ex.obj,Ex.Exe,Ex.MAP

پیام آخر مربوط به فایلهای کتابخانه ای است که هر کدام کار خاصی را انجام می دهند.

Object MOdUls [.OBJ] : Ex.obj

Run File [Ex.exe] : Ex.exe

List File [Nul.Map] : Ex. MAP

چون احتمال دارد بیتی از یک قطعه کد یا داده داشته باشیم لازم است بدانیم هر یک در کجا مستقر است و برای هر کدام چند بایت بکار برده شده است. این عمل بوسیله فایل Map صورت می گیرد.

فایل MAP نام هر قطعه، نقطه شروع، نقطه پایان و سایر بایت ها را مشخص می نماید.

Start	Stop	Length	Name	Class
00000H	00063H	00064H	STACKSG	STACK
00070H	00072H	00003H	DATASG	DATA
00080H	0009AH	0001BH	CODCSG	CODE

آدرس نقطه ورود سیستم Program entry poINT at 0008:0000

آدرس کد سگمنت – اولین دستور در آفست صفر کد سگمنت است.

### : Title

برای خواناتر شدن فایل lst. در هنگام چاپ دو رهنمون PAGE, TITLE وجود دارد. نقش

PAGE تعیین تعداد خطوط هر قطعه و تعداد کاراکترهای هر خط است.

PAGE [lines], [columns]

PAGE 60,132

پیش فرض 66 خط و 80 کاراکتر است.

هنگامی که خروجی بیش از یک صفحه باشد می توان به اسمبلر دستور داد تا عنوان برنامه را در

بالای هر صفحه چاپ کند. شبه دستور TITLE چنین کاری انجام می دهد.

برنامه ای برای جمع 5 بایت داده 1F,15,12,25 و 2B و سپس ذخیره حاصل جمع:

PAGE 60,132

TITLE PRG.EXE POURPOSE:ADDS 5 BYTES OF DATA

STSEG SEGMENT

DB 32 DUP(?)

STSEG ENDS

; .....

DTSEG SEGMENT

DATA\_IN DB 25 H,12 H,15 H,1F H,2B H

SUM DB ?

DTSEG ENDS

; .....

CDSEG SEGMENT

MAIN PROC FAR

ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG

MOV Ax,DTSEG

MOV DS,Ax

MOV Cx , 05 تنظیم شمارنده

MOV Bx,Offset DATA\_In

MOV Al,0

AGAIN :ADD AL,[BX]

```
INC  BX
DEC  CX
JNZ  AGAIN
MOV  SUM,AL
MOV  AH,4CH
INT  21H
MAIN ENDP
CDSEG ENDS
END MAIN
```

پس از اسمبل شدن برنامه و پیوند آن می توان با دیباگ نتایج را مشاهده کرد.

```
C:\>debug prg.exe
```

```
-U CS:0 19
```

```
1053:0000 B86510 MOV  AX , 1052
1053:0003 8ED8  MOV  DS , AX
1053:0005 B90500 MOV  CX , 0005
1053:0008 BB0000 MOV  BX , 0000
1053:000D 0207  ADD  AL , [BX]
1053:000F 43  IWC  BX
1053:0010 49  DEC  CX
1053:0011 75FA  JNZ  000D
```

```
1053:0013  A20500  MOV  [۰۰۰۰], AL
```

```
1053:0016  B44C    MOV  AH, 4C
```

```
1053:0018  CD21    INT  21
```

```
-D 1052:OF
```

```
1052:0000  25 12 15 1F 2B 00 00 00 00 00
```

```
-G
```

```
Program 52 terminated normally
```

```
-D
```

برنامه ای بنویسید که 6 بایت داده را از مکان های حافظه ای با تفاوت مکان 0010H به مکان های حافظه دیگری با تفاوت مکان 0028H کپی کند .

```
TITLE PRG(EXE)
```

```
PAGE 60 و 132
```

```
STSEG SEGMENT
```

```
DB 32 DUP(?)
```

```
STSEG ENDS
```

```
;.....
```

```
DTSEG SEGMENT
```

```
ORG 10 H
```

```
DATA-IN DB 25H,4FH,85H,1FH,2BH و 0C4H
```

```
ORG 28H
```



COPY DB 6 DUP (?)

DTSEG ENDS

;.....

CDSEG SEGMENT

MAIN PROC FAR

ASSUME CS:CD SEG,DS : DTSEG , SS:ST SEG

MOV AX,DTSEG

MOV DS, AX

MOV SI, OFFSET DATA-IN

MOV DI, OFFSET COPY

MOV CX, 06H

MOV LOOP: MOV AL,[SI]

MOV [DI], AL

INC SI

INC DI

DEC CX

JNZ MOV-LOOP

MOV AH,4CH

INT 21H

MAIN ENDP

CDSEG ENDS

رهنمون ORG : این رهنمون می تواند برای تنظیم آدرس های تفاوت مکان اقلام داده مورد نیاز هستند قرار گیرد . استفاده از این رهنمون برنامه نویس می تواند آدرس های تفاوت مکان را تخصیص دهد هر گاه اولین رقم اعداد A تا F باشد صفر قبل از عدد قرار داده می شود . C4 بصورت 0C4.

پس از اسمبل و پیوند زدن برنامه با استفاده از دیباگ می توان آن را اجرا نمود .

```
C:\>debug prg.exe
```

```
-U CS: 0 1
```

```
1069: 0000 B86610 MOV Ax,1066
```

```
-d 1066: 0 2f
```

```
1066:0000 00 .....
```

```
1066:0010 25 4F 85 1F 2B C4.....
```

```
1066: 0020 00 00 .....
```

```
-g
```

```
Program terminated normally
```

```
-d 1066: 0 2f
```

```
1066: 0000 00 00 .....
```

```
1066: 0010 25 4F 85 1F 2B C400 00 .....
```

```
1066 : 0020 00 00 00 00 00 00 00 00 -25 4F 85 9F 2B C4 .....
```

```
-g
```

دستورات انتقال کنترل :

اگر کنترل برنامه به مکانی در داخل قطعه کد جاری انتقال یابد ، به آن پرش NEAR گوئیم ، گاهی اوقات به آن درون قطعه ای هم می گویند . در پرش IP,NEAR تغییر و CS دست نخورده باقی می ماند زیرا هنوز کنترل در داخل قطعه کد جاری است .

در پرش FAR ، چون کنترل به خارج قطعه کد جاری تحویل می گردد ، هر دو قسمت IP,CS تغییر می کنند .

### پرش :

-شرطی

-غیر شرطی

پرش های شرطی: در این نوع پرشها کنترل به مکان جدید براساس برقراری شرط انجام می شود . به عنوان مثال در JNZ label در صورتی که  $zf=0$  پرش صورت گرفته و در صورتی که  $ZF=1$  باشد پرش صورت نگرفته و دستور بعد انجام می شود .

پرشهای شرطی از نوع پرش کوتاه می باشد .

در پرش کوتاه آدرس بایت هدف در محدوده 128- تا 127+ بایت از IP می باشد . پرش شرطی دستوری دو بایت است یکی بایت مربوط به کد عمل پرش و بایت دوم مقداری بین 00 تا ff می باشد . در پرش به عقب بایت دوم متمم 2 مقدار جابجایی است . برای محاسبه آدرس پرش بایت دوم به IP اضافه می شود .

در رو به جلو بایت دوم (هدف) با IP جمع می شود .

1067:0000 B86610 MOV Ax,1066

1067:0003 8ED8 MOV DS,Ax

1067:0005 B90500 MOV Cx,0005

1067: 0008 BB0000 MOV Bx,0000

1067: 000D 0207 ADD AL,[Bx] 6 AGAIN

1067: 000F 43 INC Bx

1067: 0010 49 DEC Cx

1067:0011 75FA JNZ 000D ↔ JNZ AGAIN

1067: 0013 A20500 MOV [0005],AL

1067: 0016 B44C MOV AH,4C

1067: 0018 CD21 INT 21

(FA همان متمم 2 عدد 6- است)

IP=0013+FA=000D

6- قدیم IP+FA0=IP قدیم IP= جدید

**پرش های بدون شرط :** در این نوع پرش ، کنترل را بدون هر گونه شرطی به مکانی خاص محول

می کند . سه نوع مختلف دارد :

Short Jump -1

NEAR Jump -2

FAR Jump -3

1- دارای قالب Jmp short label است . آدرس معرف در محدوده 128- تا 127+ نسبت به Ip

است . اگر پرش رو به عقب باشد عملوند متمم 2 است – کد عمل EB و عملوند یک بایتی در

محدوده 00 تا ff است .

NEAR Jump -2

در این حالت کد عمل E9 و پرش در قطعه کد جاری انجام می شود آدرس هدف می تواند با هر

یک از روش های مستقیم ثبات و غیرمستقیم ثبات مشخص شود

الف - jump مستقیم : مشابه پرش کوتاه است با این تفاوت که ادرس هدف می تواند در محدوده +32767 تا -32768- نسبت به IP باشد

ب - jump غیر مستقیم ثبات : ادرس پرش در یک ثبات قرار می گیرد IP مقدار BX را اختیار میکند

پ - jump غیر مستقیم حافظه ایی : ادرس هدف محتوای دو مکان از حافظه است که به وسیله ثبات به آن اشاره می شود مثلاً در JMP[DI] درون IP محتوای حافظه های اشاره شده با DI و DI+1 جایگزین میگردد.

### FAR JUMP-3

دارای قالب JUMP PTRLEABLE است این پرش به خارج قطعه کد جاری صورت می گیرد و به این معنی است که هم IP و هم CS با مقادیر جدید جایگزین می گردند

پشته:

پشته بخشی از حافظه RAM است که به وسیله CPU برای ذخیره موقت اطلاعات استفاده می شود به دلیل محدود بودن تعداد ثبات ها CPU به این ناحیه ذخیره سازی نیاز دارد

پشته بخشی از RAM است و باید به وسیله ثبات هایی به آن اشاره شود دو ثبات اصلی دستیابی پشته عبارتند از ثبات SS (قطعه پشته) و ثبات SP (اشاره گر پشته) تمامی ثبات های درون CPU به جز ثبات های قطعه و SP قابل ذخیره سازی در پشته و بازیابی آن به CPU است.

قرار دادن در پشته را PUSH (درج) و بار کردن محتوای پشته POP (بازیافت) میگویند.

شرایط پرش به سه گروه دسته بندی می شود :

1- مقادیر پرچم

2- مقایسه اعداد بدون علامت

3- مقایسه اعداد علامت دار

1- پرش شرطی condition j که در آن شرط به مقدار پرچم اشاره دارد . وضعیت پرچمها قبل از پرش در تصمیم گیری بکار می رود .

JC Jump Carry Jump Of CF=1

fJNC Jump No Carry Jump of CF=

JP Jump Parity Jump of PF=1

JNP Jump No Parity Jump of PF= f

JZ Jump Zero Jump of ZF=1

JNZ Jump No Zero Jump of ZF= f

JS Jump Sign Jump of SF=1

JNS Jump No Sign Jump of SF= f

Jo Jump Overflow Jump of OF=1

fJNO Jump No Overflow Jump of OF=

برای AF دستورالعمل پرش شرطی وجود ندارد .

- پرش شرطی "J Condition" که در آن شرط منوط به مقایسه اعداد بی علامت است. پس از

اجرای دستور مقایسه CMP dest,source , CF و ZF نتیجه مقایسه را مشخص می کنند :

SUB dest,Source ولی عمل انجام نمی شود و فقط روی فلاگ اثر دارد .

CF ZF

مبدأ > مقصد f f

مبدأ = مقصد 1 f

مبدأ < مقصد 1 f

f JA Jump Above Jump IF CF= and fZF=

JAE Jump Above or Equal Jump if CF=0

JB Jump Below Jump if CF=1

JBE Jump Below or Equal Jump if CF=1 or ZF=1

JE Jump Equal Jump if ZF=1

JNE Jump Not Equal Jump if ZF=

3-در حالت مقایسه اعداد علامت دار ، هر چند که همان دستورالعمل CMP dest,source بکار رفته است . پرچمهای بکار رفته برای نتیجه به قرار زیرند :

مبدأ > مقصد  $ZF=$  یا  $OF=SF$

مبدأ = مقصد  $ZF=1$

مبدأ < مقصد  $SF \neq OF$

JG Jump Greater Jump if  $ZF=0$  Or  $OF=SF$

JGE Jump Great or Equal Jump if  $OF=SF$

JL Jump Less Jump if  $OF \neq SF$

JLE Jump Less Or Equal Jump if  $ZF=1$  Or  $Of SF \neq$

JE Jump if Equal Jump Of  $ZF=1$

پرش شرطی دیگری هم وجود دارد : JCXZ ; Jump if Cx is Zero

ثبات اشاره گر پشته SP به مکان حافظه جاری بکار رفته در بالای پشته اشاره می کند و به محض Push داده کاهش می یابد . برعکس هنگام بار POP کردن داده این اشاره گر افزایش می یابد . دستورات POP,Push روی کل ثبات اثر می گذارند . یعنی دستوراتی مانند Push AL یا Push AH وجود ندارد .

SS:1230	96	Push DX
Push Ax SS:1231	5F	SP=1230
SS:1232	C2	Push DI
Push DI SS:1233	85	SP=1232
SS : 1234	86	Push Ax
Push Dx SS:1235	27	SP=1234
SS : 1236	30	Start

SP=1236

Push و POP باید با هم مساوی باشند . در ابتدا

POP CX	23	SS:18FA	POP CX
	14	:18FB	SP=18FC POP DX
POP DX	6B	: 18FC	SP=18FC
	2C	: 18FD	C2=1423
POP BX	91	: 18FE	SP=18FE
	F6	: 18FF	DX=2C68
	20	: 1900	SP=1900
		SP=18FA	BX=F691

عبارت فراخوانی (CALL) :

از این عبارت برای فراخوانی زیر برنامه ها استفاده می شود . اگر آدرس هدف در قطعه جاری باشد فراخوانی NEAR و اگر خارج قطعه CS جاری می باشد فراخوانی از نوع FAR است .



در هنگام فراخوانی ، بطور خودکار آدرس دستور بعد از فراخوانی در پشته ذخیره می شود . در فراخوانی NEAR فقط IP در پشته ذخیره می شود . و در فراخوانی FAR هر دو مقدار IP,CS ذخیره می شود .

پس از پایان اجرای زیر روال ، برای انتقال کنترل به محل فراخوانی ، آخرین دستور زیر روال باید RET باشد .

این دستور مقدار درون پشته را برای کنترل اجرا POP می کند .

1302:0100 MOV AL,2

1302:0102 Call SUB1

$$\text{IP} \begin{cases} 02 \text{ FFFC} \\ 01 \text{ FFFD} \end{cases}$$

FFFE

1302:0105 MOV AL,4

**تعریف زیر روال ها :**

اگر پس از PROC ذکری از FAR نشود ، پیش فرض NEAR است . یعنی درون قطعه کد جاری می باشد . در صورت FAR بودن خارج قطعه کد جاری است .

; .....

CODESEG SEGMENT

MAIN PROC FAR

ASSUME ...

```
MOV      AX,...
MOV      DS,AX
Call     SUBR1
Call     SUBR2
Call     SUBR3
MOV      AH,4CH
INT      21H
```

```
MAIN     END P
```

```
;.....
```

```
SUBR1 PROC
```

```
    .
```

```
    .
```

```
    .
```

```
RET
```

```
SUBR1 ENDP
```

```
;.....
```

```
SUBR2 PROC
```

```
    .
```

```
    .
```

```
RET
```

```
SUBR2 ENDP
```

```
RET
```

```
;.....
```

```
SUBR3 PROC
```

```
.
```

```
.
```

```
RET
```

```
SUBR3 ENDP
```

```
;.....
```

```
CODESEG ENDS
```

```
END MAIN
```

### انواع تعریف داده

DB (تعریف بایت)

اجازه تخصیص حافظه با سایر بایت را می دهد برای تخصیص کد اسکی نیز بکار می رود اسمبلر کد اسکی را بطور خودکار به اعداد یا کاراکترها اختصاص می دهد .

```
DATA3 DB 1000 1111 B
```

```
DATA4 DB 13
```

```
DATA4 DB 14H
```

```
DATA5 DB 'MY Home'  $\Rightarrow$  4D 79 20 6E 61 6D 65
```

DUP (کپی) : برای کپی کردن تعداد مفروضی کاراکتر بکار می رود .

```
DATA1 DB 01H , 01H, 01H,01H,01H,01H
```

معادل ⇓

DATA1 DB 6 DUP (01H)

پر کردن شش خانه 01H

**DW** (تعریف کلمه) : برای اختصاص دو بایت از حافظه در هر زمان می باشد .

DATA1 DW 1954

DATA2 DW 253FH

DATA3 DW 1001010101111111B

**EQU** (برابر گرفتن) :

این رهنمون هیچ مکان حافظه ای را ذخیره نمی کند . صرفاً یک مقدار ثابت را به برجسب نسبت می دهد . و می توان بجای مقدار ثابت از برجسب استفاده نمود . (مشابه ثابت در زمانهای برنامه سازی است).

Count EQU 30

-

-

-

Counter1 DB Count

-

-

-

Counter 2 DB Count

**DD** (تعریف جفت کلمه) :

4 بایت -

**DQ** (تعریف چهار کلمه) :

تخصیص 8 بایت از حافظه

**DT** (تعریف ده بایت) کاربرد آن در BCD است .

**مدلهای حافظه**

Huge ,LARGE, COMPACT , MEDIUM , SMALL , TINY

مدل **SMALL** : در این مدل حداکثر 64K بایت از حافظه را برای کد و به همان مقدار هم برای داده بکار می برد .

مدل **MEDIUM** : در این مدل ، داده می تواند در همان 64K قرار گیرد ولی کد می تواند از 64K تجاوز کند .

مدل **COMPACT** : برخلاف مدل Medium ، داده می تواند بیش از 64K بایت باشد ، کد نمی تواند .

مدل **LARGE** : ترکیب دو مدل قبل می باشد . این مدل اجازه می دهد کد و داده از 64K تجاوز کنند ولی هیچ مجموعه از داده ها (مثل آرایه) نباید از 64K بیشتر شود.

مدل **HUGE** : هر دو کد و داده می توانند از 64k بایت بیشتر باشند . یک مجموعه از داده ها (آرایه) هم می تواند از 64k بایت بیشتر باشد .

مدل **TINY** : جمع کل حافظه برای کد و داده باید در 64k بایت جای گیرد . این مدل برای فایل‌های COM استفاده می شود . این مدل نمی تواند با تعریف قطعه ساده شده مورد استفاده قرار گیرد .

تعریف قطعه ساده شده : این قالب سه رهنمون ساده را استفاده می کند :

.CODE

.DATA

.STACK

در این روش ، بکارگیری رهنمونهای SEGMENT و END لازم نیست . و ترتیب قطعات در این حالت اهمیت ندارد .

.STACK

.CODE

.Data

TITLE PRG.EXE

PAGE 60,132

.MODEL SMALL

تعریف قطعه پشته

. STACK 32

.DATA

DATA1 DW 3342 H,2E3BH , 4C5FH,3244H

ORG 10H

DW 2

.CODE

ASSUME نیست و نبودن پروسیجر

```

START  MOV  Ax,@ DATA

        MOV  DS,Ax

        MOV  Cx,04

        MOV  DI,OFFSET DATA1

        SUB   Bx,Bx           Bx صفر کردن

```

```

ADD-LP : ADD Bx,[DI]

        INC DI

        INC DI

        DEC Cx

        JNZ ADD-LP

        MOV SI,OFFSET SUM

        MOV [SI] , Bx

        MOV AH , 4 CH

        INT  21 H

        END START

```

**فایل‌های COM :** در مواقعی که مجبوریم فایل خیلی فشرده ای داشته باشیم از COM استفاده می شود . برای محدود کردن سایز فایل به 64K بایت لازم است تا داده را در داخل قطعه کد تعریف کنیم . یعنی فایل COM فاقد قطعه داده جداگانه است .

فایل COM	فایل EXE
حداکثر اندازه 64K بایت	سایز نامحدود

تعریف قطعه پشته	عدم تعریف قطعه پشته
تعریف قطعه داده	تعریف قطعه داده در قطعه کد
کد و داده می توانند در هر تفاوت مکانی قرار گیرند .	کد و داده باید از تفاوت مکان 0100 شروع شوند .
فایل بزرگ (حافظه زیادی می برد)	فایل کوچکتر (حافظه کمتری مصرف می کند)

1-TITLE PRG.COM

2-PAGE 60.132

3-CODSG SEGMENT

4- ORG 100H

5- ASSUME CS:CODSG,DS:CODSG,ES:CODSG

6- ;---- CODE AREA

7-PROC CODE PROC NEAR

8- MOV AX,DATA1

9- ADD Ax,DATA2

10- MOV SUM,AX

11- MOV AH,4CH

12- INT 21H

13-PROC CODE ENDP

14- ;----- DATA Area



15-DATA1 DW 2456

16-DATA2 DW 3672

17-SUM DW ?

18- ;-----

19-CODSG ENDS

20-END PROGCODE

در این قالب اگر ابتدا کد و سپس داده وارد شود زمان اسمبل طولانی تر می شود . بنابراین پیشنهاد می شود تا داده اول و سپس کد وارد شود و برنامه بوسیله دستور **Jump** از ناحیه داده پرش کند .

START : JMP PROGCODE

; .... DATA Area

DATA1 DW 2390

DATA2 DW 3456

Sum DW ?

;..... CODE Area

PROGCODE : MOV Ax,DATA1

ADD Ax,DATA 2

MOV Sum,Ax

MOV AH,4CH

INT 21H

; .....

CODSG ENDS

END START

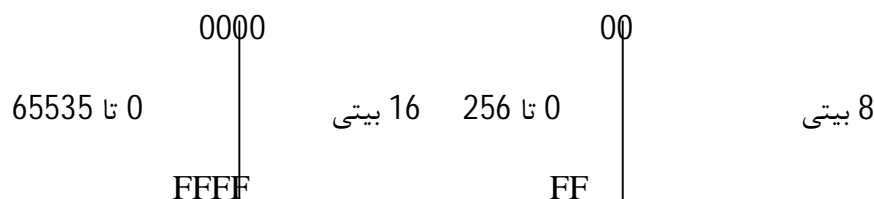
تبدیل از Exe به COM :

ابتدا باید فایل exe را به قالب استاندارد com در آورد سپس اسمبل و پیوند زده شود. در ادامه با استفاده از فایل کمکی EXE 2BIN تبدیل به فایل COM صورت می گیرد .

Exe 2BIN PRG PRG.COM

دستورات محاسباتی و منطقی :

این دستورات اعداد بدون علامت مدنظر هستند . در اعداد بدون علامت هیچ بیتی به عنوان علامت مثبت و منفی در نظر گرفته نمی شود .



جمع اعداد بدون علامت :

ADD Dest , Source  $\rightarrow$  Dest=Dest+Source

- عملوند مقصد می تواند ثبات یا در حافظه باشد .
- عملوند مقصد می تواند ثبات ، در حافظه یا عملوند فوری باشد .
- دستورات حافظه به حافظه در زبان 80X86 مجاز نمی باشد .

ADC Dest , Source  $\rightarrow$  Dest = Dest+Source +CF

با استفاده از دستور ADC می توان کری را ذخیره نمود در مواقعی که حاصلجمع بزرگتر از حداکثر مقدار مجاز ثباتها می باشد با دستور ADC کری ذخیره می شود .

برنامه ای بنویسید که جمع کل پنج کلمه داده را محاسبه نماید. (داده ها دلخواه هستند).

TITLE PRG

PAGE 60 , 132

STSEG SEGMENT

. DB 67 DUP (?)

STSEG ENDS

;.....

DTSEG SEGMENT

Count EQU 5f

DATA DW داده 5 و ..... و داده 2 و داده 1

ORG 0010H

DW 2 DUP (3)

CDSEG SEGMENT

MAIN PROC FAR

ASSUME CS:CDSEG, DS:DTSEG,SS:STSEG

MOV Ax,DTSEG

MOV DS,Ax

MOV CX,Count

MOV SI,offset DATA

ff MOV Ax,

MOV Bx,Ax

```

BACK    : ADD      Ax,[SI]

          ADC      Bx,    f

          INC      SI

          INC      SI

          DEC      CX

          JNZ      BACK

          MOV      Sum,Ax  ذخیره حاصلجمع

          MOV      Sum+2,Bx  ذخیره کری

          MOV      AH,4CH

          INT21

MAIN     ENDP

CDSEG    ENDS

          END      MAIN

برنامه ای بنویسید که دو عدد چند کلمه ای را با هم جمع نماید و نتیجه را ذخیره کند.

DATA1=367FC25963CBH    DATA2=23FA324633CFH

TITLE    PRG.exe

PAGE     f6,132

STSEG    SEGMENT

          DB 64 DUP (?)

STSEG    ENDS

; .....

```

DTSEG        SEGMENT

DATA1       DQ

ORG...10H

DATA2       DQ .....

ORG 0020H

DATA3       DQ ?

DTSEG       ENDS

; .....

CDSEG       SEGMENT

MAIN        PROC FAR

ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG

MOV Ax,DTSEG

MOV DS,Ax

CLC ; Clear Carry Flag

MOV SI,Offset DATA1

MOV ; DI , Offset DATA2

MOV Bx , Offset DATA3

بدلیل آنکه ممکن است کپی از حلقه قبل خارج باشد به جای 04,03 گذاشتیم .

MOV Cx,04

BACK        : MOV Ax,[SI]

ADC Ax,[DI]

```

MOV    [BX],AX
INC    SI
INC    SI
INC    SI
INC    DI
INC    DI
INC    Bx
INC    Bx
LOOP   BACK
MOV    AH,4CH
INT    21.H

MAIN    ENDP
CDSEG   ENDS

END     MAIN

```

-DQ داده ای به بزرگی 8 بایت تعریف می کند .

-در جمع اعداد چند بایتی یا چند کلمه ای دستور ADC همیشه استفاده می شود زیرا رقم نقلی باید به بایت بالاتر بعدی در تکرار بعدی اضافه شود .

-قبل از اجرای دستور ADC پرچم نقلی باید صفر شود  $CF=0$  تا در اولین تکرار اضافه نگردد .

CLC

- \*\*\*\*\* LOOP جایگزین دستورات  $DEC\ Cx$  شده است .

JNZ \*\*\*\*\*

با این دستور Cx به طور خودکار یک واحد کاهش می یابد در صورت صفر نبودن پرش می کند .

در صورت صفر بودن Cx دستورالعمل بعدی اجرا می شود

تعریف اعداد بدون علامت SUB dest , source ; dest-source

تعریف با روش متمم 2 صورت می گیرد . می توان مراحل زیر را در نظر گرفت :

1- متمم دو مفروق یا همان عملوند مبدأ را بدست آورد .

$$\begin{array}{r} 00011011 \quad 1B \\ 00100010 \quad - \quad 22 \\ \hline \end{array}$$

2- حاصل را با مفروق منه یا همان مقصد جمع نمود .

$$\begin{array}{r} 00\overset{1}{0}\overset{1}{1}\overset{1}{1}\overset{1}{0}11 \\ 11011110 \quad + \\ \hline F9 \leftarrow 11111001 \end{array}$$

3- رقم نقلی معکوس می شود .

بعد از اجزای دستور SUB اگر  $fCF=1$  باشد نتیجه عملیات مثبتی است ؛ اگر  $CF=1$  باشد

نتیجه منفی است و متمم 2 نتیجه در مقصد قرار می گیرد . با استفاده از دستورات NOT و INC

می توان حاصل نهایی را بدست آورد .

DATA1	DB	4CH	} قطعه داده
DATA2	DB	6EH	
DATA3	DB	?	

این برنامه دو عدد را از هم کم کرده و در صورت منفی بودن حاصل نهایی را محاسبه می کند .

MOV DH,DATA1

SUB DH,DATA2

JNC NEXT

NOT DH

INC DH

NEXT:MOV DATA3,DH

### تفریق با قرض SBB :

این دستور برای اعداد چند بایتی استفاده می شود و رقم قرضی عملوند مرتبه قبل را بکار می گیرد

پرچم نقلی SBB Des,Source ; Des=Des-Source-

اگر پرچم نقلی صفر باشد مانند دستور SUB عمل می کند اگر پرچم نقلی یک باشد این دستور یک واحد از نتیجه کم می کند .

رهنمون PTR : از این رهنمون برای مشخص کردن سائز عملوند هنگامی که با سائز تعریف شده تفاوت داشته باشد استفاده می گردد . بعنوان مثال PTR WORD به معنی استفاده از عملوند کلمه ای است هر چند داده به صورت جفت کلمه تعریف شده باشد .

DATA-A DD 32FA234CH

DATA-B DD 2832FC BA H

RESULT DD ?

MOV Ax,WORD PTR DATA-A ; Ax=234C

SUB Ax,WORD PTR DATA-B ; SUB FCBA

MOV WORD PTR RESULT, Ax ; ذخیره حاصل

MOV Ax,WORD PTR DATA-A + 2 ; Ax=32FA

SBB Ax,WORD PTR DATA-B+2,SUB 0412 با رقم قرض



ذخیره حاصل 23 ; Ax, PTR RESULT+2 MOV WORD

- بعد از آنکه  $234C_{FCBA}$  - جام شد . CF=1 می شود . در هنگام اجرای SBB

$$\begin{array}{r} 32FA \\ - 2832 \\ \hline -1 \end{array}$$

انجام خواهد شد .

ضرب و تقسیم اعداد بدون علامت :

ضرب اعداد بدون علامت :

1-بایت ضرب در بایت

2-کلمه ضرب در کلمه

3-بایت ضرب در کلمه

1-بایت X بایت : یکی از عملوندها در ثبات AL بوده و دومین عملوند می تواند در ثبات و یا در

حافظه باشد . نتیجه در ثبات Ax است .

RESULT DW ?

MOV AL,32H

MOV CL,40H

MUL CL

MOV RESULT,Ax

عملوند دوم در ثبات

DATA1 DB 32H MOV AL,DATA1 MOV AL,DATA1

DATA2 DB 40H MOV CL,DATA2 MOV DATA2

RESULT DW ? MUL CL MOV RESULT,Ax

آدرس دهی مستقیم MOV RESULT,Ax

آدرس دهی ثباتی

MOV AL,DATA1

MOV SI,Offset DATA2

PRT آدرس دهی غیر مستقیم ثباتی

با استفاده از شبه دستور Byte اندازه عملوند مشخص می گردد .

MUL Byte PTR [SI]

2-کلمه X کلمه : اولین عملوند در ثبات Ax قرار گرفته و عملوند دوم در ثبات یا در حافظه قرار

می گیرد .

نتیجه در مکانهای Dx,Ax قرار می گیرد . کلمه پایین تر در ثبات Ax و بالاتر در ثبات Dx قرار

می گیرد .

DATA1 DW 3278H

DATA2 DW 4C22H

RESULT OW 2 DUP(?)

MOV Ax,DATA1

MUL DATA2

MOV RESULT , Ax بایت پایین حاصلضرب

MOV RESULT+2 , DX بایت بالای حاصلضرب

3-بایت X کلمه : شبه ضرب کلمه در کلمه می باشد با این تفاوت که AL حاوی بایت عملوند بوده

و AH باید صفر شود .

DATA1 DB 3CH

DATA2    DW 13B2H

RESULT    DW 2 DUP (?)

MOV AL, DATA1

SUB AH,AH

MUL DATA2

MOV Bx,offset RESULT

MOV [Bx],Ax

MOV [Bx]+2,Dx

نتیجه	عملوند دوم	عملوند اول	ضرب
Ax	ثبات یا حافظه	AL	بایت X بایت
Dx-Ax	ثبات یا حافظه	Ax	کلمه X کلمه
Dx-Ax	ثبات یا حافظه	بایت AL, fAH=	بایت X کلمه

تقسیم اعداد بدون علامت :

1- بایت تقسیم بر بایت

2- کلمه تقسیم بر کلمه

3- کلمه تقسیم بر بایت

4- کلمه مضاعف تقسیم بر کلمه

در کامپیوترهای سازگار با 8086 در صورتی که مقسوم علیه صفر باشد یا خارج قسمت بزرگتر از ثبات اختصاص یافته باشد پیام divide error نمایش خواهد یافت.

بایت در بایت : در این حالت باید مقسوم در ثبات AL قرار گیرد و ثبات AH صفر شود مقسوم علیه می تواند عملوند فوری باشد ولی می تواند درون ثبات یا حافظه باشد. خارج قسمت در AL و باقیمانده در ثبات AH قرار خواهد گرفت .

DATA1 DB 102

DATA2 DB 20

QOUT DB خارج قسمت ?

REMAIN DB باقیمانده ?

MOV AL,DATA1

SUB AH,AH

DIV 10 غیر مجاز

MOV AL,DATA1 AL=12

SUB AH,AH AH<==0 AC 05

DIV DATA2 Ax مدت آدرس دهی مستقیم

AH |\_\_20 è {AL<=05 , AH<=02

MOV QOUT,AL

MOV REMAIN,AH

MOV AL,DATA1

SUM AH,AH

MOV BH,DATA2

DIV BH AX |\_\_BH =>{AL 05 , AH 02

MOV QoUT , AL

MOV REMAIN , DH

MOV AL,DATA1

SUB AH,AH مد آدرس دهی ثباتی غیر مستقیم

MOV Bx,Offset DATA2

DIV Byte PTR [Bx] AX DATA2

MOV QOUT , AL

MOV REMAIN , AH

کلمه بر کلمه : در این حالت مقسوم در Ax قرار گرفته و Dx باید صفر گردد . مقسوم علیه می تواند در یک ثبات یا حافظه باشد . بعد از اجرای دستور ، خارج قسمت در Ax و باقیمانده در Dx قرار گیرد .

MOV Ax,320 10

SUB Dx,Dx

MOV Bx,100

DIV Bx

MOV QOUT,Ax

MOV REMAIN , Dx

کلمه بر بایت : مقسوم در Ax و مقسوم علیه در ثبات یا حافظه است . بعد از اجرای دستور ، خارج قسمت در ثبات AL و باقیمانده در ثبات AH قرار خواهد گرفت .

MOV Ax,3252

MOV CL,100

DIV CL

MOV QOUT, AL

MOV REMAIN, AH

کلمه مضاعف بر کلمه : مقسوم در ثبات  $Dx, Ax$  می باشد . با ارزش در  $Dx$  و کم ارزش در ثبات

$Ax$  قرار می گیرد . مقسوم علیه در یک ثبات و یا حافظه می باشد . بعد از انجام دستور DIV

خارج قسمت در ثبات  $Ax$  و باقیمانده در  $Dx$  خواهد بود.

DATA1 DD 345607

DATA2 DW 10000

QUOT DW ?

REMAIN DW ?

MOV Ax,WORD PTR DATA1

MOV Dx,WORD PTR DATA1+2

DIV DATA2

MOV QUOT,Ax

MOV REMAIN,Dx

هنگامی که سائز مقسوم علیه یک کلمه باشد بطور خودکار  $Dx:Ax$  بعنوان مقسوم در نظر گرفته

می شود بنابراین در حالت تقسیم کلمه بر کلمه  $Dx$  باید قبلاً صفر گردد .

تقسیم	مقسوم	مقسوم علیه	خارج قسمت	باقیمانده
بایت بر بایت	$AH, f$ و بایت $AL$	ثبات یا حافظه	$AL$	$AH$
کلمه بر کلمه	$DX=0$ و کلمه $AX$	ثبات یا حافظه	$Ax$	$Dx$

AH	AL	ثبات یا حافظه	کلمه Ax=	کلمه بر بایت
Dx	Ax	ثبات یا حافظه	کلمه مضاعف DxAx=	کلمه مضاعف بر کلمه

## دستورات منطقی :

AND dest Cnatcon , Source : دستور AND

MOV CL,2BH

عملوند مبدأ می تواند ثبات ، حافظه یا فوری باشد .  $f \text{ SF} = f, \text{ZF} = f, \text{PF} = f, \text{CF} = \text{OF} = f$

عملوند مقصد می تواند ثبات یا حافظه باشد .  

$$\begin{array}{r} 00101011 \\ 00010000 \\ \hline 00010000 \end{array} \rightarrow 08$$

دستورات AND و OR و XOR

بطور خودکار OF,CF را به صفر تبدیل می نماید . و بیت های SF,ZF,PF را بر طبق نتیجه عملیات مقدار دهی می کند .

از این دستور برای آزمون صفر بودن یک عملوند استفاده می شود .

فقط در صورت صفر بودن DH ,  $\text{ZF}=1$  می شود .

AND CH,CH

JZ \*\*\*\*

دستور OR : عملوندهای مبدأ و مقصد را OR می کند . مبدأ می تواند ثبات ، حافظه یا داده فوری باشد . از دستور OR برای آزمون صفر بودن یک عملوند می توان استفاده نمود .

فقط در صورت صفر بودن BL ,  $\text{ZF}=1$  می شود .

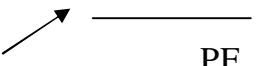
f OR BL,

OR BL,BL

MOV CL,2BH

00101011  
00010000  
00101011

$\text{SF} = \text{ZF} = f \quad \text{CF} = \text{OF} = f$

OR CL, f CH  PF PF= f  
2FH

MOV DH,54H XOR dest,Sour : دستور XOR

XOR DH,78H تأثیر پرچمها مانند OR,AND است .

$S2 \Rightarrow F=PF=f, CF=OF=f$

```

  ۰۰۱۰ ۱۰۱۱
  ۰۰۰۰ ۱۱۰۰
  -----
  ۰۰۱۰ ۱۱۱۱

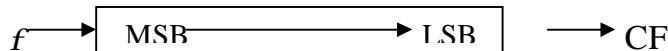
```

شیفت : دو نوع شیفت ریاضی و منطقی وجود دارد شیفت منطقی برای عملوندهای بدون علامت و ریاضی برای عملوندهای علامت دار استفاده می شود .

شیفت منطقی : دارای دو نوع شیفت به راست SHR و شیفت به چپ SHL است .

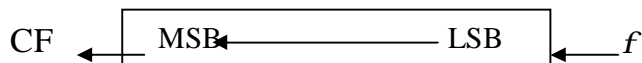
SHR : عملوند بیت به بیت به راست نقل مکان می یابد در هر نقل مکان بیت LSB به پرچم

نقلی منتقل شده و MSB با صفر پر می گردد .



SHL : این عمل عکس SHR است . پس از هر نقل مکان LSB با صفر پر می شود و MSB

درون CF می رود .



تعداد دفعاتی یا بیتی که عملوند نقل مکان داده می شود اگر یکبار باشد مستقیماً در دستور ذکر

می گردد و اگر بیش از یکبار باشد از ثبات CL استفاده می شود .

MOV AL,3BH MOV AX,0004 | یک دستور صرفه جویی شده است .

MOV CL,3 SHR BX,1

SHR AL,CL



MOV	DL,12	⇔	MOV DL,12
MOV	CL,2		SHL DL,1
SHL	DL,CL		SHL,DL,1

CMP des,Source : COMPARE

دستور مقایسه در واقع یک تفریق است با این تفاوت که در آن مقادیر عملوندها تغییر نمی کنند  
 پرچم ها مشابه اجرای SUB تغییر می کند . گرچه همه پرچمها نتیجه را منعکس می کنند ولی  
 فقط CF و ZF بکار می روند .

مقایسه عملوندها	CF	ZF
Des>Source	f	f
Des=Source	f	1
Des<Source	1	f

برای حالات بزرگتر یا کوچکتر CF تست می شود برای مساوی ZF چک می شود .

برنامه ای برای چک کردن وجود 99 در متغیر TEMP

```

TEMP    DB ?

MOV     AL,TEMP

CMP     AL,99

JZ      HOT

MOV     Bx,Z
  
```

برنامه ای برای پیدا کردن ماکزیمم نمرات در بین 5 نمره :

```
TITLE    PRG.EXE

PACE     66.132

STSEG    SEGMENT

          DB 64 DUP(?)

;.....

DTSEG    SEGMENT

GRADES   DB  23.68.33.62.92

          ORG 0008

HIGHEST  DB

DTSEG    ENDS

;.....

CDSEG    SEGMENT

MAIN     PROC FAR

          ASSUME  CS:CDSEG,DS:DTSEG,SS:STSEG

          MOV     AX,DTSEG

          MOV     DS,AX

          MOV     CX,5

          MOV     BX,OFFSET GRADES

          SUB     AL,AL

AGAIN :   CMP     AL,[BX]
```

```

JA      NEXT      پرش در صورتی AL بزرگتر بود
MOV     AL,[BX]
NEXT :   INC       BX
        LOOP      AGAIN
        MOV       HIGHEST,AL
        MOV       AH,4CH
        INT       21 H
MAIN    ENDP
CDSEG   ENDS
        END MAIN

```

### تبدیل حروف بزرگ و کوچک :

حروف بزرگ و کوچک در جدول اسکی دارای مقادیر زیرند:

مبنای شانزده حرف	مبنای شانزده حرف	مبنای شانزده حرف	مبنای شانزده حرف
A	41	a	61
B	42	b	62
C	43	c	63
Y	59	y	79
Z	5A	z	7A

رابطه ای بین حروف کوچک و بزرگ وجود دارد که به صورت زیر است :

تنها بیتی که تغییر می کند df است 0100 0001 A 41H

برای تبدیل از حروف کوچک به بزرگ باید df ماسک شود .

a        61H     0110 0001

برنامه زیر ابتدا برای تعیین کوچک بودن حرف ، آن را با 61H و 7AH مقایسه می کند و در صورت کوچک بودن حرف را با DFH=1111 1101 AND می کند .

TITLE     PRG.EXE

PAGE     6 0,132

STSEG     SEGMENT                    تبدیل حروف کوچک به بزرگ

DB   64 DUP(?)

STSEG     ENDS

; .....

DATSEG    SEGMENT

DATA1     DB   'mY NAME is Ali'

ORG 0020H

DATA2     DB   14 DUP(?)

DTSEG     ENDS

; .....

CDSEG     SEGMENT

MAIN       PROC     FAR

ASSUME CS:CDSBG, DS:DTSEG,SS:STSEG

MOV        AX,DTSEG

MOV        DS,AX

MOV        SI,Offset   DATA1

```

MOV     BX,Offset  DATA2
MOV     CX,14
BACK :  MOV     AL,[SI]   گرفتن کاراکتر بعدی
        CMP     AL,61H    'a' کوچکتر از
        JB      Over
        CMP     AL,7AH
        JA      Over     'z' بزرگتر از
        AND     AL,11011111D
Over :   MOV     [BX],AL
        INC     SI
        INC     BX
        LOOP    BACK
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
CDSEG   ENDS
        END     MAIN

```

روش دوم : کم کردن 20 تا از AL

fK6 :

روش IBM BIOS ⇐

```

CMP     AL,'a'
JB      K61

```

```

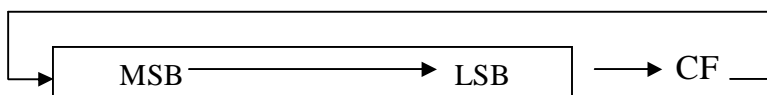
CMP    AL,'z'

JA      K61

SUB     AL,'a'-'A'

```

K61:    MOV



RCR چرخش به راست از طریق نقلی در RCR، بیت LSB به رقم نقلی CF و CF هم به MSB می رود. در واقع CF بصورت بخشی از عملوند عمل می کند.

*f* CLC                    ; CF=

MOV                    BL,32H

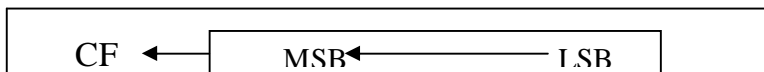
RCR	BL,1	⇒	$\left[ \begin{array}{ll} \text{M.V} & \text{CL},3 \\ \text{RCR} & \text{AL},\text{CL} \end{array} \right]$
RCR	BL,1		
RCR	BL,1		

STC                    ; CF=1

MOV                    BX,32FAH

MOV                    CL,6

RCR                    BX,CL



RCL چرخش به چپ از طریق نقلی :

در این چرخش، MSB به CF و CF به LSB می رود.

CLC

MOV AX,33 RCH

MOV CL,5

RCL AX,CL

نقشه حافظه IBM PC :

8086 دارای 20 بیت آدرس می باشد با 20 بیت اجازه دستیابی به یک مگابایت حافظه وجود دارد .  
 تخصیص حافظه یا نقشه حافظه نحوه تخصیص فضای یک مگابایتی به بخش های مختلف PC را  
 نشان می دهد .

RAM 640K	00000H ,FFFFH
Video D1SPlay RAM 128K	A0000H BFFFFH
ROM 256 K	C0000H FFFFFFH

: BIOS RoM

CPU برنامه های ذخیره شده در حافظه را اجرا می کند به هنگام روشن شدن کامپیوتر باید  
 حافظه ای دائمی برای ذخیره برنامه ای وجود داشته باشد تا به CPU کارهایی را که باید انجام  
 دهد گوشزد کند .

این مجموعه برنامه ها که بوسیله RoM نگهداری می شود BIOS خوانده می شود . BIOS حاوی برنامه تست RAM و دیگر اجزا متصل به CPU است . برنامه های ارتباط با وسایل جانبی مانند صفحه کلید ، مانیتور ، چاپگر و دیسک نیز در آن قرار دارد . تست همه وسایل متصل به PC به هنگام روشن شدن و گزارش هر فعل هم بعهده BIOS است . مثلاً اگر صفحه کلید قبل از روشن شدن کامپیوتر از PC جدا شود، BIOS ، DOS را از دیسک به RAM انتقال داده و سپس کنترل PC را به DOS می دهد .

زیر روالهای بسیار مفیدی از قبل آماده شده و در BIOS سیستم و سیستم عامل DOS وجود دارند . از طریق دستورالعمل INT کاربر می تواند از این زیرروالها استفاده کند . هنگامی که دستور INT اجرا می شود مشابه فراخوانی دور سیستم CS:IP و پرچم ها را در پشته ذخیره کرده و به زیرروال مربوط به وقفه می رود . هر وقفه دارای شماره ای بین 00-FFH است .

256 وقفه مختلف وجود دارد . INT XX ,00-FFH

مراحل زیر برای وقفه اجرا می شود :

- 1- SP دو واحد کسر شده و پرچم ها Push می شوند .
  - 2- SP دو واحد کسر شده و Push CS می شوند .
  - 3- SP دو واحد کسر شده و Push IP می شوند .
  - 4- شماره نوع وقفه در 4 ضرب می شود تا آدرس جدول بردار بدست آید . با شروع از این آدرس 2 بایت اول مقدار IP , 2 بایت بعد مقدار CS روال سرویس وقفه را می دهد .
  - 5- TF,IF صفر می شوند .
- هر وقفه برنامه ای مربوط به خود دارد که به آن روال سرویس وقفه ISR می گویند . وقتی وقفه ای رخ می دهد ، آدرس CS:IP مربوط به ISR از جدول بردار بازیابی می شود .



آدرس جدول بردار وقفه ای که پرش می شود همیشه چهار برابر عدد شماره وقفه است. مثلاً INT03 به حافظه  $4 \times 03 = 12 = 0000CH$  برای بدست آوردن IP,CS پرش خواهد کرد .

1024 بایت اول RAM ( $4 \times 256 = 1024$ ) برای جدول بردار وقفه کنار گذاشته شده و نباید برای کار دیگری بکار رود .

آدرس فیزیکی	INT
00000	INT 00
00004	INT 01
I	I

نکته : وقتی که IBM PC ساخته شد ، طراحان IBM مجبور شدند تا 256 وقفه موجود را با میکروسافت که سازنده سیستم عامل DOS بود هماهنگ نمایند . در نتیجه پرش وقفه ها مربوط به BIOS بوده و برخی دیگر مربوط به DOS می باشد بعنوان مثال INT00 مربوط به BIOS ولی INT 21H مربوط به DOS است .

انواع وقفه ها : دو نوع وقفه سخت افزاری و نرم افزاری وجود دارد .

وقفه سخت افزاری :

8086 دو پایه برای وقفه سخت افزاری کنار گذاشته است . پایه های INTR (تقاضای وقفه) و NMI (وقفه غیر قابل پوشش) - این وقفه از بیرون با قرار گرفتن 5 ولت در پایه های سخت افزاری NMI یا INTR فعال می شود اینتل INT02 را به NMI تخصیص داده است .

وقفه های نرم افزاری :

بدلیل اینکه این وقفه ها در نتیجه اجرای یک دستور و نه سخت افزار بیرونی رخ می دهند به آن نرم افزار می گویند . این وقفه ها با اجرای دستورالعمل INT XX در هر زمان بوسیله یک برنامه می توانند رخ دهند .

نکته : غیر از چهار وقفه :

INT 00	خطای تقسیم
INT 01	تک مرحله که توابع از پیش تعریف شده
INT 03	نقطه توقف یا شکست
INT 04	سرریز عدد علامت دار

دارنده بقیه وقفه ها برای پیاده سازی وقفه های نرم افزاری یا سخت افزاری می توانند بکار روند .  
نکته مهم : می توان جدول بردار وقفه هر کامپیوتر سازگار با IBM را بدست آورد . و آدرس منطقی هر وقفه را جستجو کرد .  
بسته به نسخه DOS ممکن است تفاوت هایی باشد .

C:\> debug

-D 0000:0000

```

0000 : 0000 72 , 30 E3 00 ED08      00 06
      _____
0000 : 0010  IP      CS      IP      CS

```

نکته : در انتهای یک روال سرویس وقفه دستور IRET همه پرچمها ، IP,CS را با مقادیری که قبل از وقفه داشته اند بار می کند بطوری که اجرا از دستور بعد از دستور INT ادامه یابد . معادل RET در دستورالعمل CALL می باشد .

دو عدد از وقفه ها که بطور گسترده ای بکار رفته و قادرند اعمال بسیاری انجام دهند عبارتند از INT21H,INT10H. قبل از تقاضا سرویس بوسیله این دو وقفه ، بسته به تابع مورد تقاضا ، باید در ثبات های معین مقادیر خاصی وارد شود .

برنامه نویس INT 10H از BIOS

زیر روالهای این وقفه در Rom BIOS کامپیوترهای IBMPC سوزانده شده و برای ارسال اطلاعات به صفحه تصویر کامپیوتر مورد استفاده قرار می گیرند . از جمله کاربردهای آن تغییر رنگ کاراکترها رنگ پس زمینه ، پاک کردن صفحه نمایش و تغییر محل مکان نما است . این اعمال با قراردادن یک مقدار خاص در ثبات AH انتخاب می گردند .

صفحه نمایش در حالت متنی :

80 ستون و 25 سطر تقسیم شده است .

پاک کردن صفحه نمایش بکمک تابع 06H :

ثبات های زیر قبل از فراخوانی INT 10H

باید با مقادیر معینی کار شوند :

۰۰ , ۰۰	۰۰ , ۷۹
۱۲ , ۳۹	
مرکز	
۲۴ , ۰۰	۲۴ , ۷۹

MOV AH,06 انتخاب تابع

MOV AL,00

MOV BH,07 ویژگی عادی

MOV CH,00

}

گوشه بالا سمت چپ

MOV CL,00

MOV DH,24

MOV DL,79

گوشه پایین سمت راست

MOV AX,0600H

MOV BH,07

MOV CX,0000

MOV DX,184FH; 24.79 معادل

INT 10H

می توان هر پنجره ای را با هر سایز با تعیین سطر و ستون مناسب انتخاب کرد .

تابع 02 : انتقال مکان نما : مکان مورد نظر در DX قرار می گیرد .

{ DH= سطر  
DL= ستون

RAM تصویر می تواند محتویات بیش از یک صفحه از متن را داشته باشد ولی هر بار تنها یکی از

آنها قابل مشاهده است . صفحه صفر توسط BH=00 انتخاب می گردد.

-برنامه ای بنویسید که صفحه نمایش را پاک نموده و مکان نما را در مرکز صفحه نمایش قرار دهد .

MOV AX,0600H

MOV BX,0F

{ MOV CX,0000

پاک کردن    MOV    DX,184FH

INT    10,H

قرار دادن    MOV    AH,02  
 مکان نما    MOV    BH,00  
               MOV    DL,39  
               MOV    DH,12  
               INT    10 H

تابع 03 : تعیین محل فعلی مکان نما    MOV AH,03

پس از اجرای برنامه فوق ، ثبات های DL,DH    MOV BH,00 صفحه صفر

INT 10H

حاوی شماره سطر و ستون محل های جاری مکان نما و CX اطلاعات مربوط به شکل .

هدف برنامه ریزی بیکسل AH=0CH : 0 : .....

AL=1 بیکسل روش

ثبات CX برای ستون و DX برای سطر BH شماره صفحه بیکسل خاموش f AL=

ترسیم خط از ستون 200 سطر f5 تا ستون 200 سطر f5

MOV CX=100

MOV DX=50

BACK : MOV AH, f CH

1f MOV AL ,

Hf INT 1

INC CX

CMP C2,200

JNZ BACK

تکلیف : برنامه ای بنویسید که کلمه ای را که در مبدأ 10 دیتا سگمنت ذخیره شده است خواند . و آن را به صورت بزرگ در خروجی چاپ کند . (چندین زیربرنامه برای 0 تا F) بسته به عدد زیر برنامه نمایش بزرگ آن عدد فراخوانی شود.

وقفه 21H : این وقفه بوسیله DOS فراهم گشته است . به آن توابع DOS می گویند.

تابع 09 : خروج رشته ای از داده روی مانیتور

برای ارسال یک رشته داده اسکی به مانیتور استفاده می شود .  $9fAH=$  و  $DX$  آفست آدرس داده اسکی مورد نمایش است . (پیش فرض دیتا سگمنت است) انتهای رشته باید علامت \$ باشد .

DATA-ASC DB 'Hello World', '\$'

MOV AH, 9

MOV DX, Offset DATA-ASC INT 21 H

تابع 02 : ارسال یک کاراکتر به مانیتور : DL کاراکتر خروجی مقدار دهی می شود .

2f MOV AH=

MOV DL,'6'

INT 21H

تابع 01 : ورود کاراکتر و نمایش :

این تابع تا ورود یک کاراکتر از صفحه کلید به انتظار می ماند ، سپس آن را به مانیتور ارسال می کند کاراکتر ورودی در AL خواهد بود .

f MOV AH, 1

INT 21H

```
TITLE    Test

PAGE 60,132

STSEG  SEGMENT

        DB  64  DUP(?)

SESEG  ENDS

;.....

DTSEG  SEGMENT

        DATA  DB  'HELLO WORLD','$'

DTSEG  ENDS

;.....

CDSEG  SEGMENT

MAIN PROC  FAR

ASSUME  .....

MOV  AX,DTSEG

MOV  DS,AX

CALL  CLEAR

CALL  CURSOR

CALL  DISPLAY

MOV  AH,4CH

INT  21H

MAIN  ENDP
```

```
;.....  
;SUB PROTIN  CLEAR  
  
CLEAR  PROC  
  
    MOV  AX,0600H  
  
    MOV  BH,07  
  
    MOV  CX,0000  
  
    MOV  DX,184FH  
  
INT    10H  
  
RET  
  
CLEAR  ENDS  
  
;.....  
;SET  CURUSOR  CENTER  
  
CURSOR  PROC  
  
    MOV  AH,02  
  
    MOV  BH,00  
  
    MOV  DH,12  
  
    MOV  DL,39  
  
INT    10H  
  
RET  
  
CURSOR  ENDS  
  
;.....
```



```
;DISPLAY STRING
```

```
DISPLAY PROC
```

```
MOV AH,09
```

```
MOV DX,OFFSET DATA
```

```
INT 21H
```

```
RET
```

```
DISPLAY ENDS
```

```
;.....
```

```
CDSEG ENDS
```

```
END MAIN
```

تابع OAH: دریافت رشته داده از صفحه کلید-به همراه نمایش روی صفحه نمایش دریافت داده از صفحه کلید و ذخیره آن در مکان حافظه ای که از قبل در قطعه داده تعریف شده است. DX,AH=0AH آدرس آفستی که رشته داده در آن ذخیره می گردد. (بافر داده هم می گویند)

اولین بایت بافر برای سباز در نظر گرفته می شود تعداد کاراکترها در بایت دوم و داده وارد شده از بایت سوم به بعد است.

```
ORG 10H
```

```
DATA1 DB 6 ? 6 DUP (FF)
```

تا زمانی که کلید بازگشت Retor زده نشود از INT21H خارج نمی شود. مثلاً Alt زده شود

```
AHf MOV AH,
```

MOV Dx,Offset DATA1

INT 21H

0010 0011 0012 0013 0014 0015 0016

06 03 061H 6D 6A 0D

CR I L A تعداد واقعی کاراکتر سائز ..... کداسکی

مبنای 16 وارد شده

بافر

A16 L16 I CR

اگر بیش از 6 کاراکتر (حداکثر +5 CR) زده شود کامپیوتر بلند گو را بصدا درآورده و مابقی کاراکترها ذخیره نخواهد شد .

-اگر فقط Enter زده شود تعداد واقعی کاراکتر 00 خواهد بود . CR شمرده نمی شود.

تابع 07 : ورود از صفحه کلید بدون نمایش

این تابع یک کاراکتر از ورودی دریافت می کند (بدون نمایش) -پس از اجرا منتظر یک کاراکتر مانده سپس کاراکتر را در AL ذخیره می کند :

MOV AH,07

INT 21H

برنامه نویسی صفحه کلید INT 16 H :

تابع 01 : تست فشردن کلید : اگر کلیدی فشرده شده باشد  $fZF=1$  در غیر اینصورت  $ZF=1$  .

تابع 00 : تشخیص کلید فشرده شده - AL حاوی کد اسکی کلید فشرده شده است.

این تابع باید بلافاصله پس از تابع 01 استفاده شود .

برنامه ریز کد اسکی کاراکتر زنگ 07H را مرتباً به صفحه نمایش می فرستد تا زمانی که کاربر کلیدی را فشار دهد .

```
TITLE .....

.MODEL      SMALL

.STACK

.DATA

MESSAGE    DB 'TO STOP THE BELL SOUND PRESS ANY
KEY$'

.CODE

MAIN        PROC

    MOV      AX,@ DATA

    MOV      DS,AX

    MOV      AH,09

    MOV      DX,Offset MESSAGE    نمایش پیغام

    INT      21H

2 f AGAIN : MOV      AH, تابع ارسال کاراکتر

    MOV      DL,07 کاراکتر زنگ

    INT      21H

    MOV      AH,01 چک کلید فشرده

    INT      16H

    JZ       AGAIN اگر کلیدی فشرده نشده بود
```

```

MOV     AH,4CH
INT     21H

MAIN    ENDP

END

```

تغییر یافته )

```
MESSAGE DB 'TO STOP THE BELL SOUND PRESS Q (or q)
```

```
key$'
```

```

CODE    }
CODE    } نمایش
CODE    }

```

```
AGAIN : AH,COE
```

```

CODE    }
CODE    } فرستادن زنگ
CODE    }

CODE    }
CODE    } یک کلید

```

```
JZ AGAIN
```

```

MOV     AH,00 H }
INT     16H      } در صورت زدن کلید
                  } شست کاراکتر زده شده

```

```
CMP     AL,'Q'
```

```
JE      EXIT     اگر درست بود خارج
                  شود
```

CMP AL,'q'

JE EXIT

JMP AGAIN

EXIT:MOV AH,4CH

INT 21H

MAIN ENDP

تکلیف برنامه ای بنویسید که محتوای یک فایل را روی صفحه نمایش نشان دهد .