جزوه درس

زبان ماشین و برنامه سازی سیستم

تهیه کننده

علی چوداری خسروشاهی

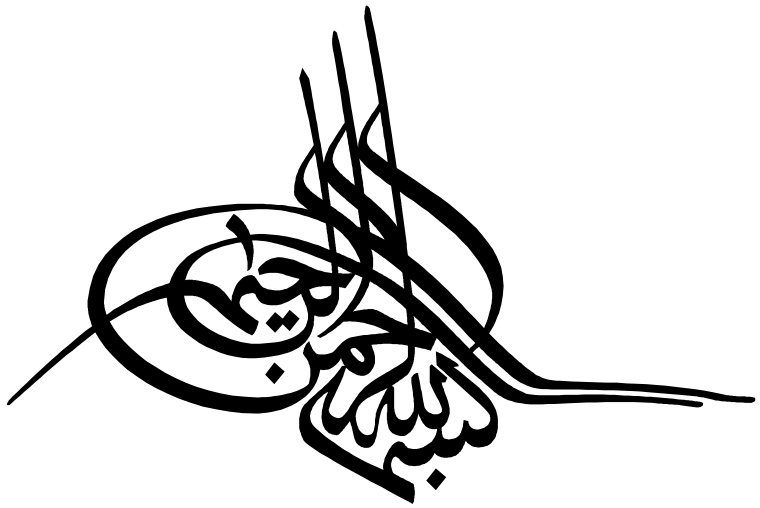
مرجع

عنوان اصلی: IBM PC Assembely Language and Programing

عنوان فارسی : برنامه نویسی و زبان اسمبلی کامپیوترهای شخصی

نویسنده : Peter Abel

ویژه دانشجویان کارشناسی کامپیوتر – نرم افزار



بنام خدا

پیشگفتار

ما تلاش کرده ایم در این جزوه تمام سرفصلهای درس پوشش داده و مطالب مفیدی را در اختیار شما قرار دهیم تا شما با خواندن آن کلیاتی را از درس بدست آورید. در مورد این جزوه یادآوری این نکته مهم، ضروری است که این جزوه جهت کمک به دانشجویان در کاهش یادداشت برداری، ترسیم شکلها، برنامه ها و مثالها در هنگام تدریس بوده است. بنابر این در کنار این جزوه، هر دانشجو مطابق با ذوق و سلیقه خود، جزوه ای دست نویس خواهد داشت که حاوی یادداشت ها به منظور تکمیل و تفهیم این جزوه است.

این جزوه، ادعای جایگزینی مراجع اصلی این درس را ندارد بلکه تنها خلاصه ای از مطالب مهم از مراجع درس است. در گردآوری این جزوه از مرجع زیر استفاده شده است:

عنوان اصلی: **IBM PC Assembely Language and Programing**

عنوان فارسی : **برنامه نویسی و زبان اسمبلی کامپیوترهای شخصی**

نویسنده : *Peter Abel*

در اینجا لازم است از تلاش تمام دانشجویانی که اینجانب را در تدوین این جزوه یاری نموده اند کمال تشکر را داشته باشم. همچنین در این جزوه احتمال اشتباه وجود دارد، به این جهت در مطالعه مطالب جزوه دقت کافی را داشته باشید.

از دانشجویان عزیز تقاضا می‌شود در مطالعه مطالب آن دقت کافی را داشته باشند و وجود هرگونه ایراد ، و همچنین نظرات و پیشنهادات خود را به آدرس پست الکترونیکی اینجانب اطلاع دهند.

Akhosroshahi@IAUT.ac.ir
Akhosroshahi@Gmail.com

باتشکر

علی چوداری خسروشاهی

فهرست مطالب

فصل اوّل

فصل دوم

فصل سوم

فصل چهارم

فصل پنجم

فصل ششم

فصل هفتم

فصل هشتم

فصل نهم

فصل دهم

فصل یازدهم

فصل اوّل

۱- مبانی اعداد

اعداد مبنای ۱۶ و ۱۰ و ۸ و ۲ از جمله پرکاربردترین مبانی عددی می‌باشند . هریک از این مبانی از ارقام زیر تشکیل شده‌اند :

مبنا	ارقام تشکلیل دهنده
2 (binary)	0, 1
8 (Octet)	0, 1, 2, 3, 4, 5, 6, 7
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

از بین این مبناها مبنای ۱۰ بیشتر استفاده می‌شود. برای کار کردن با این مبانی عددی ما نیاز خواهیم داشت که این مبانی را به همدیگر تبدیل کنیم. بعنوان مثال عدد ۵۸ در مبنای ۱۰ برابر ۱۱۱۰۱۰ در مبنای ۲ ، ۷۲ در مبنای ۸ و 3A در مبنای ۱۶ می باشد.

۱-۱- ارزش ارقام

در زمان قرار گرفتن در یک عدد، هر رقم دارای دو ارزش متفاوت است:

ارزش مطلق

ارزش مکانی

ارزش مکانی هر رقم برابر است با ارزش مطلق ضرب در مبنا به توان شماره مکان ، به طوریکه شماره مکان از سمت راست ترین رقم و با مکان صفر شروع میشود

مثال: ارزش مکانی رقم ۲ را مشخص کنید ؟

$(627)_8$

ارزش مکانی ۲ برابر است با :$2 \times 8^1 = 16$

ارزش مکانی هر کدام از ارقام عدد زیر را بدست آورید :

$(A2C)_{16}$

ارزش مکانی C برابر است با :$12 \times 16^0 = 12$

ارزش مکانی ۲ برابر است با :$2 \times 16^1 = 32$

ارزش مکانی A برابر است با :$10 \times 16^2 = 2560$

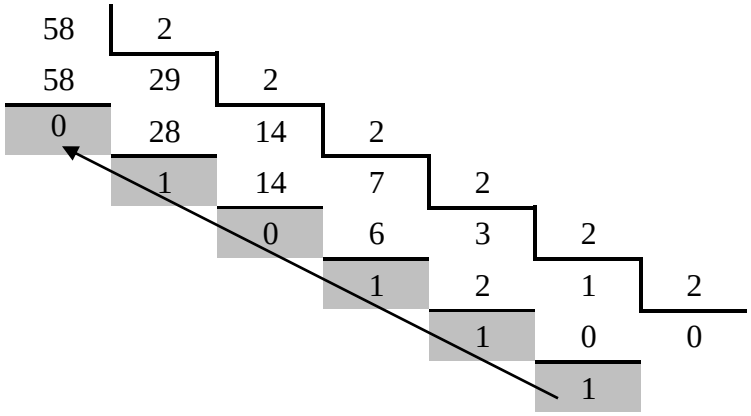
۱-۲- تبدیلات مبنا

۱-۲-۱- تبدیل مبنای ۱۰ به مبنای ۲

برای این منظور روشهای مختلفی وجود دارد یکی از این روشها روش تقسیم بر ۲ می‌باشد. عدد مورد نظر بر ۲ تقسیم می‌کنیم تا خارج قسمت و باقیمانده مشخص شود. باز این عملیات بر روی خارج قسمت تکرار می‌شود و عملیات را تا

زمانی که خارج قسمت برابر صفر شود تکرار می‌دهیم. با کنار هم قرار دادن کلیه باقی‌مانده‌ها از آخرین باقی‌مانده تا اولـین باقی‌مانده عدد در مبنای ۲ حاصل خواهد شد.

مثال :



$$(58)_{10} = (111010)_2$$

راه حل دیگر :

برای این منظور می‌توانیم از توانهای ۲ نیز استفاده کرد. توانهای ۲ به ترتیب عبارتند از :

$$\ldots, 2^4, 2^3, 2^2, 2^1, 2^0$$

یا به عبارت دیگر می‌توان بصورت زیر درنظر گرفته شود :

$$\ldots, 16, 8, 4, 2, 1$$

اگر بخواهیم یک عدد در مبنای ۱۰ را به مبنای ۲ تبدیل کنیم، از بین این توانها سراغ بزرگترین توان کوچکتر مساوی عدد مورد نظر می‌گردیم و از آن (عدد مورد نظر) کم می‌کنیم. باز همین عملیات بر روی حاصل تفریق انجام داده تا زمانی ادامه می‌دهیم که به صفر برسیم. نهایتاً در زیر اعدادی که در تفریقات شرکت داشتند 1 و در زیر بقیه 0 قرار می‌دهیم.

$$\begin{array}{r} 58 \\ \mathbf{-32} \\ \hline 26 \\ \mathbf{-16} \\ \hline 10 \\ \mathbf{-8} \\ \hline 2 \\ \mathbf{-2} \\ \hline 0 \end{array}$$

$$\begin{array}{cccccc} {\scriptstyle 32} & {\scriptstyle 16} & {\scriptstyle 8} & {\scriptstyle 4} & {\scriptstyle 2} & {\scriptstyle 1} \\ 1 & 1 & 1 & 0 & 1 & 0 \end{array}$$

$$(58)_{10} = (111010)_2$$

۲-۲-۱- تبدیل اعداد اعشاری مبنای ۱۰ به مبنای ۲

اعداد را به دو قسمت صحیح و کسری تقسیم می کنیم و هر قسمت را جداگانه تبدیل می کنیم :

تبدیل قسمت صحیح به مبنای ۲: با تقسیمات متوالی بر ۲ و گرد آوری باقیمانده ها به عنوان رقم های مبنای ۲.

تبدیل قسمت کسری به مبنای ۲ : با ضرب متوالی در ۲ و گرد آوری اعداد صحیح تولید شده به عنوان رقم های مبنای جدید ۲. عملیات ضرب را تا جایی ادامه می دهیم که دیگر رقم اعشار باقی نماند.

مثال: عدد 41.6875 در مبنای ۱۰ می‌باشد. آنرا به مبنای ۲ تبدیل کنید.

حل: ابتدا قسمت اعشار را از صحیح جدا کرده عمل تبدیل را برای هر یک جداگانه انجام می‌دهیم.

صحیح = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

اعشار = 0.6875

0.6875 x 2 = 1.3750

0.7500 x 2 = 1.5000

0.5000 x 2 = 1.0000

$(0.6875)_{10} = (0.1011)_2$

$$(41.6875)_{10} = (101001.1011)_2$$

۳-۲-۱- تبدیل اعداد مبنای ۲ به مبنای ۱۰

برای این منظور می‌توانیم همانند حالت قبل عمل کنیم. یعنی اول وزن ارقام را به ترتیب از راست به چپ مشخص می‌کنیم. نهایتاً مجموع وزن ارقام ۱، عدد معادل را در مبنای ۱۰ مشخص خواهد کرد.

مثال : معادل عدد $(11001011)_2$ در مبنای ۱۰ چیست؟

$$\begin{pmatrix} {\scriptstyle 128} & {\scriptstyle 64} & {\scriptstyle 32} & {\scriptstyle 16} & {\scriptstyle 8} & {\scriptstyle 4} & {\scriptstyle 2} & {\scriptstyle 1} \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}_2 = (128+64+8+2+1)_{10} = (203)_{10}$$

۴-۲-۱- تبدیل عدد مبنای ۲ به مبنای ۸

برای تبدیل اعداد مبنای ۲ به اعداد مبنای ۸ کل ارقام موجود را با شروع از سمت راست ترین رقم به گروههای سه تایی تقسیم خواهیم کرد. اگر تعداد ارقام سمت چپ ترین گروه کمتر از سه باشد به سمت چپ آن به تعداد مورد نیاز صفر اضافه می‌کنیم. نهایتاً با نوشتن معادل هریک از گروه‌ها در مبنای ۸ کل عدد در مبنای ۲ تبدیل به عدد مبنای ۸ خواهد شد.

مثال: عدد $(101000110)_2$ را به مبنای 8 تبدیل کنید؟

$$\left(\underbrace{101}_{5},\underbrace{000}_{0},\underbrace{110}_{6}\right)_2 = (506)_8$$

۵-۲-۱- تبدیل عدد مبنای ۸ به مبنای ۲

برای این منظور هریک از ارقام عدد مبنای ۸ به مبنای ۲ تبدیل کرده و در یک میدان سه رقمی می‌نویسیم (اگر طول عدد کمتر از ۳ باشد به تعداد مورد نیاز صفر در سمت چپ اضافه می‌کنیم). با کنار هم قرار دادن این گروه‌ها عدد معدل در مبنای ۲ حاصل خواهد شد.

مثال : عدد $(517)_8$ را به مبنای ۲ تبدیل کنید؟

$$(\underset{101}{5}\ \underset{001}{1}\ \underset{111}{7})_8 = (101001111)_2$$

۶-۲-۱- تبدیل مبنای ۲ به مبنای ۱۶

نحوه تبدیل اعداد مبنای ۲ به مبنای ۱۶ همانند روش تبدیل اعداد مبنای ۲ به ۸ می‌باشد، با این تفاوت که بجای میدان سه رقمی ، میدان ۴ رقمی خواهیم داشت.

مثال : عدد $(101001110)_2$ را به مبنای ۱۶ تبدیل کنید؟

$$\left(\underbrace{0001}_{1}\underbrace{0100}_{4}\underbrace{1110}_{E}\right)_2 = (14E)_{16}$$

۷-۲-۱- تبدیل مبنای ۱۶ به مبنای ۲

نحوه تبدیل اعداد مبنای ۱۶ به مبنای ۲ باز همانند روش تبدیل اعداد مبنای ۸ به ۲ می‌باشد با این تفاوت که بجای میدان سه رقمی ، از میدان ۴ رقمی استفاده خواهیم کرد.

مثال : عدد $(2A5)_{16}$ را به مبنای ۲ تبدیل کنید؟

$$\left(\underset{0010}{2}\ \underset{1010}{A}\ \underset{0110}{5}\right)_{16} = (001010100110)_2$$

۸-۲-۱- تبدیل همه مبناها به ۱۰

برای تبدیل همه مبنا ها کافیست مجموع ارزش مکانی ارقام عدد را محاسبه کنیم

مثال: عدد $(A2C)_{16}$ را به مبنای ۱۰ تبدیل کنید؟

$(A2C)_{16}= 12+32+2560=(2604)_{10}$

۹-۲-۱- سایر تبدیلات

به غیر از تبدیلاتی که گفته شد تبدیلات دیگری نیز می‌تواند وجود داشته باشد. برای اینگونه تبدیلات راحت ترین روش استفاده از تبدیلات بالا می‌باشد. جدول زیر این تبدیلات و راه حلهای آنها را نشان می‌دهد.

تبدیل (مبنا)	راه حل
۱۰ به ۸	تبدیل ۱۰ به ۲ سپس ۲ به ۸
۸ به ۱۰	تبدیل ۸ به ۲ سپس ۲ به ۱۰
۱۰ به ۱۶	تبدیل ۱۰ به ۲ سپس ۲ به ۱۶
۱۶ به ۱۰	تبدیل ۱۶ به ۲ سپس ۲ به ۱۰

۱۶ به ۸	تبدیل ۱۶ به ۲ سپس ۲ به ۸
۸ به ۱۶	تبدیل ۸ به ۲ سپس ۲ به ۱۶

عدد $(AC2)_{16}$ را به مبنای ۸ تبدیل نمایید

حل) ابتدا عدد را به مبنای ۲ می بریم (AC2)16=(1010,1100,0010)2

سپس تبدیل به ۸ را انجام میدهیم

=(101,011,000,010)2=(5302)8

۳-۱- مکمل اعداد

دو نوع مکمل برای هر عدد در مبنای Rوجود دارد: مکمل R و مکمل(R-1) . در مکمل (R-1) ، هر رقم را از مقدار (R-1) کسر می‌کنیم. برای مثال مکمل ۹عدد $۸۳۵_{۱۰}$ برابر است با $۱۶۴_{۱۰}$ و مکمل ۱عدد $۱۰۱۰_{۲}$برابر است با $۰۱۰۱_{۲}$

برای یافتن مکمل Rیک عدد ابتدا مکمل R-1 آن عدد را محاسبه کرده سپس مقدار۱را با آن جمع می کنیم.

مکمل ۱۰عدد $۸۳۵_{۱۰}$ برابر است با $۱۶۴_{۱۰}+۱=۱۶۵_{۱۰}$

مکمل ۲عدد $۱۰۱۰_{۲}$برابر است با $۰۱۰۱_{۲}+۱ = ۰۱۱۰_{۲}$

۱-۳-۱- مکمل ۲

برای محاسبه مکمّل ۲ ، کل بیت ها را نقیض کرده (1 ها را به 0 و 0 ها را به 1 تبدیل می‌کنیم) و نهایتاً این عدد را با 1 جمع خواهیم کرد.

۲-۳-۱- روش ساده محاسبه مکمّل ۲ :

از سمت راست‌ترین بیت شروع کرده و به سمت چپ حرکت می‌کنیم تا به اولین 1 برسیم، این بیت ها را عیناً نوشته و بقیه بیت‌ها را نقیض خواهیم کرد.

مثال : عدد علامت‌دار $(10010111)_2$ برابر است با $(-105)_{10}$.

$$\left(\underset{-}{1}\underbrace{0010111}_{1101001}\right)_2 = (-1101001)_2 = (-105)_{10}$$

روش محاسبه مکمل ۲ عدد 0010111 در مثال بالا:

$$0010111 \xrightarrow{\text{نقیض}} 1101000 \xrightarrow{+1} 1101001$$

مثال : عدد علامت‌دار $(11110100)_2$ برابر است با $(-12)_{10}$.

$$\left(\underset{-}{1}\underbrace{1110100}_{0001100}\right)_2 = (-0001100)_2 = (-12)_{10}$$

روش محاسبه مکمل ۲ عدد 1110100 در مثال بالا:

$$1110100 \xrightarrow{\text{تقیض}} 0001011 \xrightarrow{+1} 0001100$$

۳-۳-۱- نکات مهم

مکمل دو در مبنای دو معادل به قرینه کردن عدد در مبنای ده است همچنین جمع هر عدد با مکملش برابر با صفر خواهد بود
در زمان محاسبه مکمل باید به بیت های صفر بی اهمیت توجه شود برای مثال اعداد زیر با هم مساوی نیستند :

۱۰۱۰
۰۱۰۱۰
۰۰۱۰۱۰
۰۰۰۰۱۰۱۰

زیرا مکمل های یکسانی ندارند.
در زمان محاسبه مکمل تعداد بیتهای واقعی عدد اهمیت دارد. چون بیت های بی اهمیت به ۱ مبدل خواهند شد.
مثال: مکمل ۲ عدد ۱۰۱۰ را بدست آورید
راه حل غلط: ۰۱۱۰
روش صحیح : عدد چند بیتی است ؟ ۸ بیتی .

۰۰۰۰۱۰۱۰
۱۱۱۱۰۱۱۰

۴-۱- اعداد علامت دار

در حالت طبیعی سیستم اعداد انتخاب شده برای نمایش اعداد باید بتواند هم اعداد بدون علامت و هم اعداد علامت دار را نمایش دهد. سه روش زیر برای نمایش اعداد علامت دار وجود دارد:

۱. نمایش بصورت اندازه-علامت.
۲. نمایش بصورت مکمل یک.
۳. نمایش بصورت مکمل دو.

در سیستمهای کامپیتری از روش سوم استفاده می شود در یک عدد n بیتی علامتدار با ارزشترین بیت، بیت علامت است

۱-۴-۱- روش نمایش اعداد علامت دار بصورت مکمل دو

ابتدا روی بیت علامت ۱ قرار می دهیم
سپس روی بقیه بیتها مکمل ۲ عدد را قرار می دهیم

مثال:

$$(-1101001)_2 = \left(\underset{-}{1}\underbrace{0010111}_{1101001}\right)_2$$

۲-۴-۱- اعداد علامت‌دار در سیستم مبنای ۲

برای ایجاد اعداد علامت‌دار و ذخیره‌سازی آن روی حافظه روشهای مختلفی وجود دارد. در سیستم کامپیوتر به این صورت عمل می‌شود که از سمت چپ‌ترین بیت (با ارزشترین بیت) بعنوان بیت علامت استفاده می‌شود. این بیت می‌تواند دو

حالت 0 یا 1 را داشته باشد. اگر این بیت صفر باشد به این مفهوم می‌باشد که عدد مثبت است و اگر یک باشد به مفهوم می‌باشد که عدد منفی است.

اگر عدد مثبت باشد (بیت علامت برابر صفر) معدل عدد در مبنای ۲ روی بقیه بیتها (بقیه بیتها به غیر از بیت علامت) قرار می‌گیرد. برای بدست آوردن عدد در مبنای ۱۰ می‌توان به راحتی آنرا از مبنای ۲ به ۱۰ تبدیل کرد.

مثال: عدد علامت‌دار $(00010111)_2$ برابر است با $(+23)_{10}$.

$$\left(\underset{+}{\underline{0}}\underset{23}{\underline{0010111}}\right)_2 = (+23)_{10}$$

۳-۴-۱- تبدیل اعداد باینری منفی به دهدهی:

اگر عدد منفی باشد(بیت علامت برابر یک) مکمّل ۲ عدد مورد نظر در مبنای ۲ روی بقیه بیتها (به غیر از بیت علامت) قرار می‌گیرد. برای مشخص شدن عدد در مبنای ۱۰ ابتدا مکمل ۲ کلیه بیتها به غیر از بیت علامت را بدست آورده و آنرا به مبنای ۱۰ تبدیل می‌کنیم و یک علامت منفی به جواب اضافه می کنیم.

- مثال : عدد علامت دار $(10100100)_2$ را مبنای ۱۰ تبدیل کنید؟

10100100
01011100 = 92 مکمل ۲ → ضرب در منفی = -92

۴-۴-۱- روش میانبر تبدیل اعداد باینری منفی به دهدهی:

با توجه به تعداد بیت های عدد آخرین بیت را با علامت منفی در نظر می گیریم:

-128 64 32 16 8 4 2 1

- ***مثال*** : عدد علامت دار $(10001001)_2$ را مبنای ۱۰ تبدیل کنید؟

-128 64 32 16 8 4 2 1
1 0 0 0 1 0 0 1 → -128 + 8 + 1 = -119

۵-۴-۱- تبدیل معکوس

برای تبدیل یک عدد منفی دهدهی به مبنای دو هیچ روش مستقیمی وجود ندارد

ابتدا عدد را به صورت مثبت به مبنای دو تبدیل کرده ، سپس مکمل آنرا محاسبه می کنیم

مثال : عدد 85- را به صورت عددی ۸ بیتی و علامتدار در مبنای دو نمایش دهید.

ابتدا ۸۵ را به مبنای دو می بریم

128 64 32 16 8 4 2 1
0 1 0 1 0 1 0 1 = 85
1 0 1 0 1 0 1 0 = -85 مکمل ۲ :

۵-۱- جمع در مبنای ۲

شکل زیر حالتهای مختلف جمع دو بیت را نشان می‌دهد. برای جمع هر بیت یک از بیتهای عدد مبنای دو با حالتهای زیر برخورد خواهیم کرد.

$$\begin{array}{r} 0 \quad 0 \quad 1 \quad 1 \\ +\ \underline{0} + \underline{1} + \underline{0} + \underline{1} \\ 00 \quad 01 \quad 01 \quad 10 \end{array}$$

در حاصل این جمع ها رقم سمت راست حاصل جمع و رقم سمت چپ بعوان رقم نقلی مرحله بعد می‌باشد. اگر رقم نقلی مرحله قبل را نیز دخالت دهیم حتماً در محاسبه مجموع در روی هر رقم یکی از موارد زیر را خواهیم داشت.

رقم نقلی مرحله قبل	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

باز در شکل نیز رقم سمت راست عدد حاصل مشخص کننده حاصل جمع و رقم سمت چپ بعوان رقم نقلی مرحله بعد می‌باشد. با این مفروضات می‌توان نتیجه جمع اعداد مبنای ۲ را محاسبه نمود.

مثال : حاصل جمع عدد 11110001 با عدد 00110110 در مبنای ۲ برابر است با

$$\begin{array}{r} 11110001 \\ +\ 00110110 \\ \hline 100100111 \end{array}$$

مثالهای زیر نمونه هایی را از عمل جمع در مبنای دو نشان می‌دهد.

$$\begin{array}{r} 10000101 \\ +\ 00101111 \\ \hline 10110100 \end{array} \qquad \begin{array}{r} 10011011 \\ +\ 00101000 \\ \hline 11000011 \end{array} \qquad \begin{array}{r} 11100111 \\ +\ 10001101 \\ \hline 101110100 \end{array}$$

۶-۱- تفریق اعداد در مبنای ۲

با توجه به رقم قرضی مرحله قبل می‌توانیم یکی از حالات زیر را داشته باشید.

رقم قرضی مرحله قبل	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	- 0	- 1	- 0	- 1	- 0	- 1	- 0	- 1
	0 0	1 1	0 1	0 0	1 1	1 0	0 0	1 1

مثالهای زیر نمونه هایی را از عمل تفریق در مبنای دو می‌باشند.

$$\begin{array}{r} 10000101 \\ -\ 00101111 \\ \hline 01010110 \end{array} \qquad \begin{array}{r} 10011011 \\ -\ 00101000 \\ \hline 01110011 \end{array} \qquad \begin{array}{r} 11100111 \\ -\ 10001101 \\ \hline 01011010 \end{array}$$

در روی سیستم کامپیوتری مدار جمع کننده وجود دارد اما مدار تفریق کننده وجود ندارد. برای محاسبه حاصل تفریق می‌توان از مدار جمع‌کننده بصورت زیر استفاده کرد :

$$\mathbf{A - B = A + (-B)}$$

با توجه به این رابطه جواب تفریقهای مثال قبل را می‌توان با جمع بصورت زیر شبیه سازی کرد.

$$\begin{array}{r} 01100111 \\ -\ 00001101 \\ \hline 01011010 \end{array} \xrightarrow{\text{حالت معادل}} \begin{array}{r} 01100111 \\ +\ 11110011 \\ \hline 101011010 \end{array} \qquad \begin{array}{r} 103 \\ -\ +13 \\ \hline 90 \end{array} \xrightarrow{\text{یا}} \begin{array}{r} 103 \\ +\ -13 \\ \hline 90 \end{array}$$

سمت چپ ترین بیت رقم نقلی نهایی می‌باشد که در حاصل دخالت داده نمی‌شود (دور ریخته می‌شود).

$$\begin{array}{r} 11010011 \\ -\ 00101000 \\ \hline 10101011 \end{array} \xrightarrow{\text{حالت معادل}} \begin{array}{r} 11010011 \\ +\ 11011000 \\ \hline 110101011 \end{array} \qquad \begin{array}{r} -45 \\ -\ +40 \\ \hline -85 \end{array} \xrightarrow{\text{یا}} \begin{array}{r} -45 \\ +\ -40 \\ \hline -85 \end{array}$$

$$\begin{array}{r} 00000101 \\ -\ 00101111 \\ \hline 11010110 \end{array} \xrightarrow{\text{حالت معادل}} \begin{array}{r} 00000101 \\ +\ 11010001 \\ \hline 11010110 \end{array} \qquad \begin{array}{r} +5 \\ -\ +47 \\ \hline -42 \end{array} \xrightarrow{\text{یا}} \begin{array}{r} +5 \\ +\ -47 \\ \hline -42 \end{array}$$

۱-۷- بیت نقلی

در تفریق اعداد علامتدار ایجاد بیت نقلی لزوماً به معنی بروز خطا نیست معمولا این بیت دور ریخته میشود. تنها حالت خطا زمانی است که نتیجه در ۸ بیت قابل ذخیره نباشد.

۱-۸- کد ASCII

برای ذخیره‌سازی کاراکترها از کد گذاری استفاده می‌شود. یکی از مشهورترین روشهای کد گذاری ، روش کد گذاری ASCII می‌باشد. در این روش به منظور کد گذاری از ۸ بیت (۱ بایت) استفاده می‌شود و می‌تواند $2^8 = 256$ کاراکتر مختلف را کدگذاری کند. بعنوان مثال کد کاراکتر A در ASCII برابر ۶۲ می‌باشد.

'A'= 62

۱-۸-۱- کاراکترهای قابل چاپ

کراکترهای قابل چاپ دارای کدهای 32 تا 126 می باشند.

کاراکتر	کد اسکی
0 تا 9	48 تا 57
A تا Z	65تا90
a تا z	97 تا 122

۲-۸-۱- کاراکترهای کنترلی

کرکترهای کنترلی دارای کدهای 0 تا 31 می باشند

کاراکتر	کد اسکی
ESC	27
CR	10
LF	13

فصل دوم

۲- قسمت های یک سیستم کامپیوتری

۲-۱- واحد اجرایی[1] و واحد رابط گذرگاه[2]

پردازشگر از نظر منطقی به دو بخش تقسیم شده است. واحد اجرایی و واحد رابط گذرگاه. وظیفه EU اجرای دستورات دریافتی از BIU می‌باشد. واحد اجرایی شامل قسمتهای محاسباتی و واحد منطقی (ALU) و واحد کنترل (CU) وتعدادی ثبات است. این قسمتهای اساسی به منظور اجرای دستورات و عملیات محاسباتی و منطقی فراهم شده‌اند.

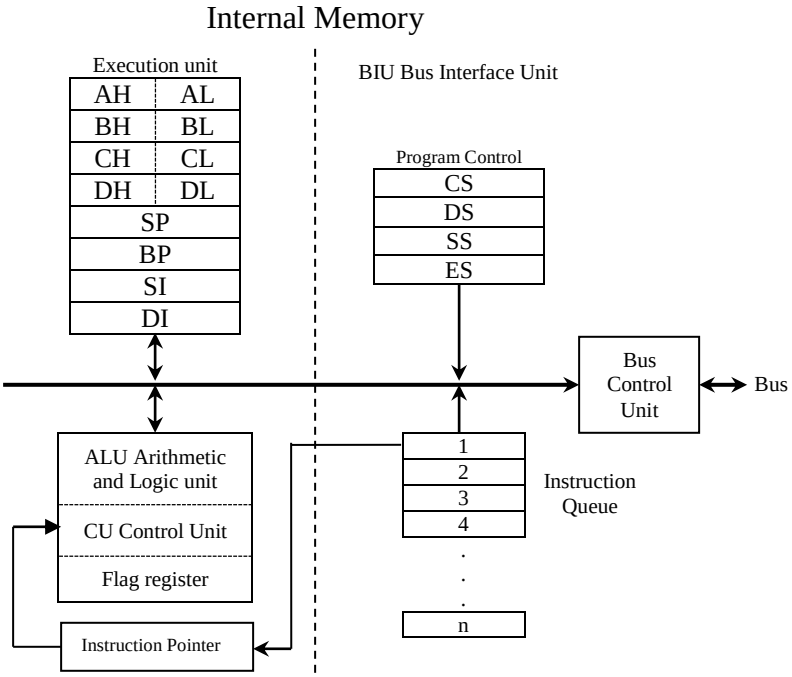
مهمترین عمل BIU ، مدیریت واحد کنترل گذرگاه و ثبات‌های سگمنت و صف دستورالعمل می‌باشد. BIU گذرگاه‌هایی که داده را به EU ، حافظه و دستگاه‌های ورودی _ خروجی جانبی ارسال می‌کنند، کنترل می‌کند و ثباتهای سگمنت آدرسهای حافظه را کنترل می‌کنند.

عمل دیگری از BIU دستیابی به دستورالعمل می باشد، زیرا تمام دستورالعمل‌های برنامه در حافظه قرار دارد. BIU باید دستورالعمل‌ها را از حافظه خوانده و در یک **صف دستورالعمل** قرار دهد که در اندازه ، بسته به نوع پردازشگر متفاوتند. این اصول BIU را قادر می‌سازد تا ضمن پیش‌بینی دستورالعمل‌ها را واکشی نموده بطوریکه همواره صفی از دستورالعملها برای اجرا آماده باشد.

اگرچه BIU و EU با هم بطور موازی عمل می‌کنند، همواره BIU یک قدم جلوتر از EU است. اگر BIU نیازمند دستیابی به داده‌ها در حافظه یا یک دستگاه ورودی ، خروجی باشد EU آن را آگاه می‌کند. همچنین EU دستورالعمل‌های ماشینی را از صف دستورالعمل BIU مطالبه می‌کند.

بالاترین دستورالعمل صف دستورالعمل قابل اجرای جاری می‌باشد. مادامیکه EU در حال اجرای یک دستورالعمل است BIU دستورالعمل بعدی را از حافظه واکشی می‌کند. این واکشی با اجرا در یک زمان اتفاق می‌افتد و سرعت پردازش افزایش می‌یابد.

[1] EU (Execution Unit)
[2] BIU (Bus Interface Unit)



شکل ۱ : واحد اجرا و واحد رابط گذرگاه

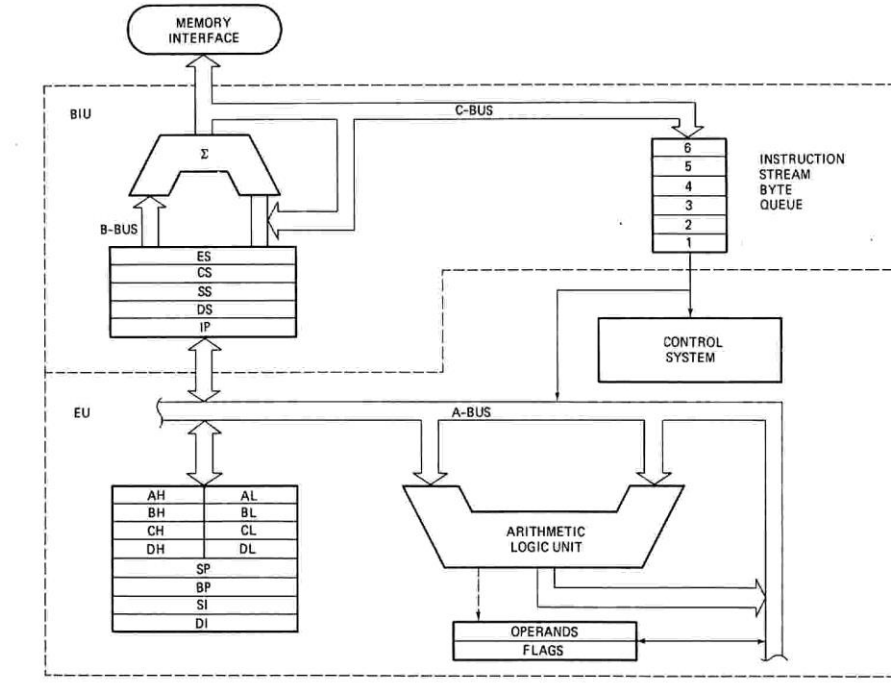


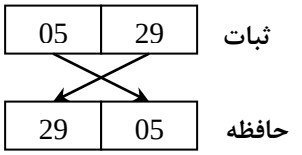
FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

شکل ۲ : واحد اجرا و واحد رابط گذرگاه

۲-۲- آدرس‌دهی داده ها در حافظه

حافظه اصلی یک مجموعه منطقی از مکان‌هایی است که هرکدام می‌تواند یک بایت دستورالعمل‌ها یا داده‌ها را ذخیره نماید. هر بایت حافظه اصلی دارای یک برچسب عددی به نام آدرس می‌باشد.

سیستم داده ها را در حافظه به ترتیب بایت معکوس ذخیره سازی می کند، یعنی بایت با ارزش کم در آدرس کم و بایت با ارزش بیشتر در آدرس بالاتر ذخیره سازی می‌شود. وقتی پردازشگر اطلاعات را از حافظه می‌خواند به ترتیب بایت معکوس عمل می‌کند. شکل زیر نمونه‌ای از این حالت را نشان می‌دهد.



آدرس هر بایت از حافظه اصلی را می توان با سگمنت حاوی بایت مزبور و به دنبال آن افستی که از ابتدای سگمنت یاد شده در نظر گرفته می شود، آدرس دهی کرد.

۲-۳- آدرسدهی و سگمنتها

برای دسترسی به داده ها روی RAM باید دو آدرس مشخص شود :

۱. آدرس سگمنت (Segment)
۲. آدرس آفست (Offset)

سگمنتها بخش هایی با اندازه 64KB هستندآدرس شروع همه سگمنت ها به صفر ختم میشود . این صفر را خذف می کنیم.

آدرس سگمنت ۱۶ بیتی می شود. آفست فاصله از ابتدای سگمنت است.

آدرس segment ناحیه ای است که از مرز پاراگراف شروع می شود، یعنی آدرسی می باشد که بر ۱۶ قابل قسمت است.

بدست آوردن آدرس واقعی:

مرحله اول : آدرس واقعی سگمنت را تولید می کنیم واضافه کردن صفری که حذف شده بود.

مرحله دوم : جمع آدرس شروع سگمنت با آفست

سه سگمنت اصلی عبارتند از : Code Segment ، Data Segment و Stack Segment .

۲-۳-۱- Code Segment

این سگمنت حاوری دستور العمل های اجرایی برنامه می باشد. ثبات CS برای اشاره به این سگمنت استفاده می‌شود.

۲-۳-۲- Data Segment

این سگمنت حاوی داده های برنامه می باشد و ثبات DS برای آدرس دهی آن استفاده می شود.

Stack Segment -۲-۳-۳

این سگمنت حاوی آدرس های بازگشت به سیستم عامل یا تابع فراخوانی کننده می باشد. ثبات سگمنت SS برای این منظور استفاده می شود

به غیر از آدرس سگمنت آدرس آفست را هم خواهیم داشت این آدرس اختلاف مکان بایت یا سلول مورد نظر شما را از ابتدای سگمنت مشخص می کند مجموع آدرس سگمنت و آفست یک سلول از حافظه را مشخص خواهد کرد.

در روی پردازشگر ثبات های مختلفی وجود دارد که جهت کنترل دستوراتی که اجرا می شوند، دستیابی به آدرس های حافظه و قابلیت فراهم سازی محاسبات استفاده می شود. این ثباتها بر اساس ویژگی هایی که دارند به گروههای مختلفی تقسیم می شوند .

۲-۴- اجزاء یک برنامه اسمبلی

هر برنامه اسمبلی از بخشهای زیر تشکیل می شود

۱. Code
۲. Data
۳. Stack
۴. Extra

هر یک از این بخشها باید روی RAM قرار گیرند

۲-۵- ثباتها

در روی پردازنده انواع مختلفی از ثباتها وجود دارد که عبارتند از:

۱. ثباتهای سگمنت
۲. ثباتهای اشاره‌گر
۳. ثباتهای عمومی
۴. ثباتهای شاخص
۵. ثبات پرچم

۲-۵-۱- ثباتهای segment

طول ثباتهای سگمنت برابر ۱۶ بیت می‌باشد و به ناحیه از حافظه اشاره می کند که به عنوان یک سگمنت شناخته می شود این ثباتها عبارتند از:

ثبات CS : ثبات CS حاوی آدرس شروع سگمنت کد می‌باشد آفست دستورالعمل در ثبات IP قرار می‌گیرد یعنی CS +IP مشخص کننده آدرس دستور العمل بعدی است که باید اجرا شود.

ثبات DS : ثبات DS حاوی آدرس شروع سگمنت داده می باشد.

ثبات SS : ثبات SS ثبات حاوی آدرس سگمنت پشته می باشد این آدرس به اضافه آدرس موجود در SP عبارت بالای پشته را نشان می دهد

ثبات ES : ثبات ES ثبات برای برخی از کاربردهای دیگر مانند اعمال رشته ای استفاده می شود.

ثبات GS و FS : این ثباتها سگمنت ثباتهای اضافی هستند که در پردازشگرهای 80386 و به بعد قرار گرفته اند.

۲-۵-۲- ثباتهای اشاره گر

سه ثبات اشاره گر وجود دارد که عبارتند از IP ، SP و BP .

اینها ثباتها ۱۶ بیتی می باشند ثبات IP حاوی آدرس آفست دستورالعمل بعدی جهت اجرا می باشد. ثبات SP حاوی آدرس آفست عضو بالای پشته می باشد و ثبات BP ثبات اشاره گر پایه می باشد که ارجاع به پارامترها، شامل داده ها و آدرسهایی که از یک برنامه بر پشته عبور می‌کنند را ساده می‌نماید.

۲-۵-۳- ثباتهای عمومی

این ثباتها کاربردهای عمومی دارند این ثبات ها ۱۶ بیتی بوده که به دو بخش ۸ بیت کم ارزش و ۸ بیت پر ارزش تقسیم می شوند . ۸ بیت کم ارزش با L و ۸ بیت پرارزش با H مشخص می‌شوند. به عنوان مثال ثبات ۱۶ بیتی AX شامل قسمتهای AH (۸ بیت پرارزش) و AL (۸ بیت کم ارزش) می‌باشد. این ثبات ها عبارتند از :

ثبات AX : به ثبات انباشت گر (accumulator) معروف است و برای اعمال ورودی خروجی و برخی اعمال رشته ای و حسابی استفاده می شود. برای برخی از اعمال مانند ضرب و تقسیم و انتقال از ثبات AX استفاده می شود.

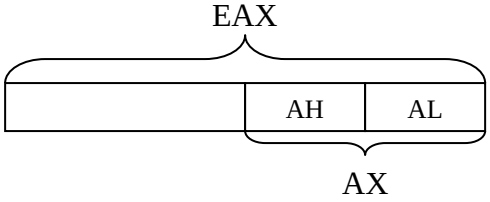
ثبات BX : به ثبات پایه (Base) معروف است و به عنوان یک شاخص (Index) برای توسعه دادن آدرس دهی استفاده می شود و این ثبات به منظور محاسبه نیز استفاده می شود.

ثبات CX : (C مخفف Counter یا شمارش معروف است) از این ثبات می توان به منظور کنترل کردن تعداد دفعات اجرای یک حلقه استفاده کرد و همچنین این ثبات به منظور محاسبه نیز استفاده می شود.

ثبات DX : این ثبات به ثبات داده (Data) نیز معروف است برخی از اعمال ورودی و خروجی و اعمالی مانند ضرب و تقسیم از این ثبات استفاده می کنند.

در روی پردازش گرهای 80386 و به بعد این ثبات ها را گسترش داده و از ۱۶ بایت به ۳۲ بیت تبدیل کردند، و همچنین نام های جدید EAX و EBX و ECX و EDX را برای آنها نهادند.

به عنوان مثال برای ثبات ۳۲بیتی EAX، ۱۶ بیت کم ارزش آن ثبات AX می باشد. باز خود AX به AH و AL تقسیم می شود. پس تغییر ثبات به عنوان مثال AH باعث تغییر ثبات های AX و EAX خواهد شد.



۲-۵-۴- ثباتهای شاخص

دو ثبات DI و SI برای آدرس دهی شاخص و استفاده در جمع و تفریق قابل دسترسی می باشند.

ثبات SI به عنوان ثبات شاخص مبدا شناخته می شود و **ثبات DI** به عنوان ثبات مقصد شناخته می شود که برای اعمال رشته ای مورد استفاده قرار می گیرند. در این مورد ثبات SI با ثبات DS و ثبات DI با ثبات ES مرتبط می باشند .

۵-۵-۲- ثبات پرچم

ثبات پرچم یک ثبات ۱۶ بیتی می باشد که از این ثبات ۱۶ بیتی برخی از بیتها برای منظور های خاصی استفاده می شوند پس از انجام برخی از دستورالعمل ها از جمله دستورات مقایسه و محاسبه وضعیت ثبات پرچم تغییر پیدا می کند. این پرچم ها عبارتند از :

OF - Overflow flag (سرریز) مشخص کننده سرریز اتفاق افتاده بر روی پر ارزش ترین بیت یک محاسبه می باشد.

DF = Direct flag (جهت) تعیین کننده جهت چپ و راست برای انتقال یا مقایسه داده های رشته ای می باشد.

IF = Interrupt flag تعیین می کند که یک وقفه ناتوان است.

TF = Trap flag به پردازش گر اجازه می دهد تا دستورات برنامه را به صورت تک مرحله ای اجرا کند.

SF = Sign flag نشان دهنده علامت نتیجه محاسبه می باشد (0 مثبت و 1 منفی).

ZF = Zero flag نتیجه عمل محاسباتی یا منطقی را نشان می دهد 1 برای نتیجه صفر و 0 برای نتیجه غیر صفر.

AF = (رقم نقلی کمکی) حاوی رقم نقلی خارج شده از بیت ۳ در داده های ۸ بیتی در عمل حسابی می باشد.

PF= Parity flag تعیین کننده توازن فرد یا توازن زوج می باشد.

CF = Cary flag نشان دهنده رقم نقلی خارج شده از بیت بالا مرتبه اعمال حسابی می باشد همچنین حاوی آخرین بیت در اعمال جابجایی یا چرخش می باشد.

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

فصل سوم

۳- موارد ضروری برای کد نویسی در زبان اسمبلی

۳-۱- توضیحات برنامه (comment)

توضیحات برنامه در زبان اسمبلی با ";" آغاز می‌شود، یعنی در یک سطر کل کاراکتر های بعد ";" به عنوان توضیح در نظر گرفته می شود و بر اجرای برنامه هیچ تاثیری نخواهند داشت.

۳-۲- کلمات رزرو شده

در زبان اسمبلی مانند سایر زبان ها برخی از کلمات به عنوان کلمات رزرو شده می باشند. از این کلمات برای کاربرد های دیگر نمی توان استفاده کرد. این کلمات عبارتند از:

دستورات، مانند MOV , ADD اعمالی هستند که کامپیوتر می تواند اجرا کند.

پیش پردازنده ها، مانند SEGMENTیا END ، که برای دادن اطلاعات به اسمبلر استفاده می‌شود.

عملگرها، مانند FAR و SIZE که شما در عبارات از آن استفاده می‌کنید.

سمبولهای از پیش تعیین شده، مانند Data@ و Model@ که اطلاعاتی را به برنامه شما بر می‌گردانند.

استفاده نادرست از کلمات رزرو شده باعث می شود اسمبلر پیغام خطایی را تولید کند.

۳-۳- شناسه‌ها

یک شناسه نامی است که به یک عنصر در برنامه داده می شود که نیاز به رجوع به آن وجود دارد. دو نوع از شناسه ها نام و بر چسب هستند.

۱. **نام** اشاره به آدرسی از عنصر داده دارد. مانند counter در `counter DB 0`

۲. **برچسب** اشاره به آدرس دستور، رویه یا سگمنت خاصی دارد. مانند main در `main PROC FAR`

۳-۳-۱- قوانین تعریف شناسه

۱. برای تعریف یک شناسه می توانیم از کاراکترهای زیر استفاده کرد:

نوع	کاراکترهای مجاز
حروف الفبا	A تا Z و a تا z
ارقام	0 تا 9 (اولین کاراکتر نباید باشد)
کاراکترهای خاص	?, _, $, @, .

۲. شروع شناسه باید یا حروف الفبا و یا کاراکترهای خاص به غیر از " . " (نقطه) باشد.

۳. نمی توان از کلمات رزرو شده به عنوان شناسه استفاده کرد. همچنین از نام ثبات ها برای نام گذاری شناسه نمی توان استفاده کرد.

۳-۴- دستورات

شکل زیر نمونه ای از دستورات زبان اسمبلی را نشان می دهد. دستورات زبان اسمبلی شامل چهار بخش می باشد که هر یک از بخش ها می تواند با استفاده از کاراکتر فاصله از یکدیگر جدا شوند. بخش اول مشخص کننده شناسه بخش دوم دستورالعمل بخش سوم عملوند و بخش چهارم توضیحات می باشد.

نوشتن بخش ۱و بخش۴ اختیاری می باشد و بخش ۳ وابسته به بخش ۲ می باشد یعنی اینکه ممکن است یک دستورالعمل دو عملوند یک عملوند داشته باشد و یا اصلا عملوند نداشته باشد.

[identifier:] operation [operand(s)] [;comment]

به عنوان مثال:

```
P30:    ADD    AX , BX     ;ADD Ax with Bx
```

۳-۵- پیش‌پردازنده‌ها

۳-۵-۱- پیش پردازنده PAGE و TITLE

پیش پردازنده page: در شروع برنامه حداکثر تعداد خط در یک صفحه و حداکثر ستون در یک سطر را مشخص می‌کند. شکل کلی این پیش‌پردازنده بصورت زیر می‌باشد.

PAGE [length],[width]

به عنوان مثال Page 60,132 صفحه ای با ۶۰ سطر و ۱۳۲ ستون فراهم می کند. در یک صفحه تعداد خط ها می تواند از ۱۰ تا ۲۵۵ باشد و همچنین تعداد ستون ها (تعداد کاراکتر ها در هر خط) می تواند از ۶۰ تا ۱۳۲ باشد. اگر در یک برنامه پیش پردازنده page نوشته نشود کامپایلر مقدار پیش فرض آن یعنی page 50,80 را در نظر خواهد گرفت.

پیش پردازنده title : برای اینکه بتوان برای برنامه عنوان رامشخص نمود ازپیش پردازنده title استفاده میکنیم.

۳-۵-۲- پیش پردازنده segment

از این پیش پردازنده برای تعریف segment می توان استفاده کرد. حداکثر طول یک segment در حالت طبیعی می تواند ۶۴ کیلو بایت باشد.

پیش پردازنده SEGMENT شروع سگمنت و FNDS پایان آن را مشخص می‌کند. شکل کلی این پیش پردازنده بصورت زیر می‌باشد:

NAME	OPERATION	OPERAND	COMMENT
name	SEGMENT	[operations]	; begin segment
	.		
	.		
	.		
name	ENDS		; end segment

۶-۳- برنامه EXE

هر برنامه EXE از سگمنتهای زیر تشکیل می شود.

۱. STACK SEGMENT
۲. DATA SEGMENT
۳. EXTRA SEGMENT
۴. CODE SEGMENT

برای تعریف این سگمنتها می توان از پیش پردازنده SEGMENT استفاده کرد.

شکل زیر قالب کلی یک برنامه EXE را که با استفاده از پیش‌پردازنده SEGMENT تعریف شده است نشان می‌دهد.

```
1                Page  60,132
2  Title         A04ASMl      segments   for an . EXE program
3  ;……
4  STACKSG   SEGMENT   PARA   STACK ‘stack’
5                …
6  STACKSG   ENDS
7  ;……
8  DATASEG   SEGMENT   PARA   ‘data’
9                …
10 DATASEG   ENDS
11 ;……
12 CODESEG   SEGMENT   PARA    ‘code’
13 MAIN      PROC      FAR
14               ASSUME    SS: STACKSG, DS:DATASEG, CS:CODESEG
15               MOV       AX, DATASG     ; set address of data
16               MOV       DS, AX         ; segment in DS
17               …
18               MOV       AX, 4C00H      ; End processing
19               INT       21H
20 MAIN      ENDP                         ; End of procedure
21 CODESEG   ENDS                         ; End of segment
22               END       MAIN           ; End of program
```

قالب کلی یک برنامه EXE

برنامه بالا چهار چوب کلی برنامه های EXE را نشان می دهد. برنامه‌های اسمبلی نوشته شده باید دقیقاً این چهار چوب را رعایت کنند. در این مثال STACKSG مشخص کننده سگمنت پشته می باشد. در این بخش باید طول پشته را مشخص کنیم. DATASG مشخص کننده سگمنت داده می باشد، در این بخش باید متغیر های برنامه را تعریف کنیم. CODESG مشخص کننده سگمنت کد می باشد، در این بخش باید کد های برنامه را بنویسیم. داخل این بخش حتما باید یک تابع اصلی داشته باشیم. در این مثال تابع اصلی با نام MAIN مشخص شده است. MAIN PROC FAR شروع تابع MAIN را مشخص می‌کند. شروع اصلی برنامه باید دستور ASSUME باشد. این دستور هدف از هر سگمنت را مشخص می کند. شکل کلی این دستور به صورت زیر می‌باشد:

ASSUME SS:stackname , DS: datasegment , CS: codesegment , …

برای مقدار دهی ثبات DS دو دستور MOV نوشته شده است چون امکان انتقال مستقیم آدرس Datasg بر روی DS وجود ندارد. ابتدا با استفاده از دستورMOV آدرس Datasg بر روی AX منتقل شده و سپس از روی AX به DS انتقال پیدا می کند. بعد از مقدار دهی DS می توانیم کد های برنامه خود را بنویسیم. نهایتا پس از اتمام کار برای خروج از برنامه باید تابع 4CH از وقفه شماره 21H را فراخوانی کنیم.

```
MOV   AX, 4C00H
INT   21H
```

پیش پردازندهENDP انتهای تابع را مشخص می کند. در این مثالMAIN ENDP انتهای تابع MAIN را مشخص می کند. عبارت END پایان برنامه را مشخص می کند. بعد از END می توان هیچ عبارتی نوشته نشود، در این صورت برنامه کامپایلر می شود اما اجرا نخواهد شد. عبارت بعد از END تابع اصلی برنامه را مشخص می کند. END MAIN مشخص می کند که MAIN تابع اصلی برنامه می باشد. برای خروج از برنامه تابع 4Ch وقفه 21h را استفاده می‌کنیم شکل کلی فراخوانی این وقفه بصورت زیر خواهد بود:

```
MOV   AH , 4CH
MOV   AL , retcode
INT   21H
```

کد خروج باید در ثبات AL قرار گیرد(کد صفر برای خروج نرمال از برنامه می باشد). پس اگر بخواهیم به حالت نرمال خارج شویم همچنان که در مثال دیده می‌شود عمل خواهیم کرد.

۱-۶-۳- پیش پردازنده PROC

سگمنت کد حاوی کدهای اجرایی برنامه می‌باشد. این کدهای اجرایی در قالب یک یا بیشتر از یک رویه می باشند که تعریف می‌شوند.تعریف هر رویه با پیش پردازنده PROC آغاز شده و با ENDP پایان می‌یابد. سگمنتی که فقط شامل یک روال باشد چنین خواهد بود:

NAME	OPERATION	OPERAND	COMMENT
segname	SEGMENT	PARA	
procname	PROC	FAR	; One
	.		; procedure
	.		; within
	.		; the code
procname	ENDP		; segment
segname	ENDS		

نام رویه باید حتما تعریف شده و منحصر به فرد باشد، همچنین نام رویه حتما باید تابع قوانین تعریف شناسه‌های زبان اسمبلی باشد.

۲-۶-۳- حالت محافظت شده

در حالت محافظت شده که تحت پردازش گرهای 80386 و به بعد وجود دارد یک برنامه می تواند تا ۱۶ مگا بایت حافظه را آدرس دهی کند. پیش پردازنده 386. به اسمبلر اجازه پذیرش دستوراتی که در 80386 و به بعد منحصر به فرد هستند می دهد. به عنوان مثال دستوراتی را که از ثبات های عمومی گسترش یافته مانند: EAX , EBX , ECX , EDX استفاده می کنند.

برخی از دستورات وجود دارند که امکان استفاده از آن ها در حالت محافظت شده وجود ندارد ایـن دسـتورات عبارتنـد از: STI، CLI، IN وOUT .

```
        Page    60,132
Title A04ASM1 segments   for an . EXE program
;……….……
STACKSG SEGMENT    PARA   STACK 'stack'
        DW 32 DUP(0)
STACKSG ENDS
;……….……
DATASEG SEGMENT    PARA    'data'
        FLDD   DW   175
        FLDE   DW   150
        FLDF   DW   ?
DATASEG ENDS
;……….……
CODESEG SEGMENT PARA 'code'
MAIN PROC FAR
     ASSUME SS: STACKSG, DS:DATASEG, CS:CODESEG
     MOV  AX, DATASG        ; set address of data
     MOV  DS, AX            ; segment in DS

     MOV  AX, FLDD          ; move 175 to AX
     ADD  AX, FLDE          ; add 150 to AX
     MOV  FLDF, AX          ; store sum in FLDF

     MOV  AX, 4C00H         ; End processing
     INT  21H
MAIN ENDP                   ; End of procedure
CODESEG ENDS                ; End of segment
        END MAIN            ; End of program
```

یک برنامه EXE با سگمنت‌های معمولی

۳-۶-۳- پیش پردازنده سگمنت ساده شده

در زبان اسمبلی پیش پردازندۀ ساده شده ای برای تعریف سگمنت وجود دارد. برای استفاده از آنها باید ابتدا مدل حافظـه را مشخص کنیم. قالب آن چنین است:

.MODEL memory-model

این پیش پردازنده مشخص کننده مدل حافظه مورد استفاده قرار گرفته برای برنامه ها می باشد. مدل حافظه مشـخص کننده تعداد سگمنت کد و داده است. مـدل هـای حافظـه ای کـه مـی توانـد اسـتفاده شـود: Tiny، Small،Medium، Compact یا Large می‌باشند. مدل Tiny در برنامه های Com استفاده می شود و مشخص می کند که بایـد داده هـا و کد برنامه در یک segment تعریف شود.

پیش پردازنده های ساده شده تعریف سگمنت عبارتند از

.STACK [size]
.DATA
.CODE [name]

هر یک از این پیش پردازنده ها می توان برای تعریف سگمنت پشته، داده و کد استفاده شوند. اگر داخل برنامه از ایـن پیش پردازنده ها برای تعریف سگمنت های مورد نظر استفاده شود دیگر نیازی بـه اسـتفاده از عبـارت segment و ends در برنامه ها وجود ندارد و اسمبلر می تواند به راحتی ابتدا و انتهای سگمنت را تشخیص دهد. همچنین در صورت استفاده از این پیش پردازنده ها دیگر نیازی به استفاده از Assume داخل برنامه ها وجود ندارد. اگـر از پـیش پردازنـده DATA.

برای مشخص کردن سگمنت داده استفاده شود استفاده کنیم برای مقدار دهی ثبات DS می توان به صورت زیر عمل کرد:

```
MOV  AX , @Data
MOV  DS , AX
```

شکل زیر برنامه قبل را با استفاده ار پیش پردازنده های ساده شده نشان می‌دهد.

```
        Page    60,132
Title A04ASM1 segments for an .EXE program
;……….
        .MODEL      SMALL
        .STACK        64
        .DATA
          FLDD  DW   175
          FLDE  DW   150
          FLDF  DW   ?
;……….
        .CODE
MAIN    PROC FAR
        MOV AX, @data    ; set address of data
        MOV DS, AX       ; segment in DS

        MOV AX, FLDD     ; move 175 to AX
        ADD AX, FLDE     ; add 150 to AX
        MOV FLDF, AX     ; store sum in FLDF

        MOV AX, 4C00H    ; End processing
        INT 21H
MAIN    ENDP             ; End of procedure
        END MAIN         ; End of program
```

یک برنامه EXE با پیش پردازنده‌های سگمنت ساده شده.

۴-۶-۳- تعریف داده

برای تعریف متغیر ها در زبان اسمبلی می توان از شکل کلی زیر استفاده کرد:

[name]	Dn	expression

در این شکل name مشخص کننده نام متغیر می باشد و نوشتن آن اختیاری می باشد. اگر name نوشته نشود سلول حافظه ای اختصاص داده می شود که نام آن مشخص نیست و می‌توان با استفاده از نام متغییر قبلی یا آفست استفاده کرد. Dn مشخص کننده نوع این سلول می باشد که می تواند یکی از انواع زیر باشد:

نوع	اندازه	توضیحات
DB	1 byte	Byte
DW	2 byte	Word
DD	4 byte	Double word
DF	1 6byte	Far word
DQ	8 byte	Quad word
DT	10 byte	Ten butes

بخش expression مشخص کننده مقداری است که باید داخل این سلول حافظه قرار گیرد. اگر در این بخش علامت سوال قرار داده شود مقدار سلول حافظه نا مشخص خواهد بود. برای تعریف آرایه ها می توان از عبارت Dup در بخش

expression استفاده کرد. قبل از Dup باید تعداد سلول های این آرایه را مشخص کنیم و بعد از Dup داخل پرانتز بـاز و بسته مقادیری که داخل این سلول های حافظه قرار می گیرند مشخص خواهیم کرد. مثال:

```
Arr1  DD  15 Dup(0)
Arr2  DB  20 Dup(?)
```

در مثال زیر نمونه ای از برنامه ای برای تعریف داده های کاراکتری و عددی ارائه شده است:

```
                                    Untitled
                                Page  60,132
                                Title   A04DEFIN(EXE)    Define data directives
                                .MODEL      SMALL
                                .DATA
                                ;       DB  -    Define  Bytes:
                                ;……….
0000  00                        BYTE1   DB   ?              ; Uninitialized
0001  30                        BYTE2   DB   48             ; Decimal constant
0002  30                        BYTE3   DB   30H            ; Hex constant
0003  7A                        BYTE4   DB   01111010B      ; Binary constant
0004  000A[                     BYTE5   DB   10 Dup(0)      ; Ten zeros
          00
                          ]

000E  50 63 20 45 6D 70         BYTE6   DB   'Pc Emporium'; Character string
      6F 72 69 75 6D
0019  31 32 33 34 35            BYTE7   DB   '12345'        ; Number as chars
001E  01 4A 61 6E 02 46         BYTE8   DB   01,'Jan', 02,'Feb',03,'Mar'
      65 62 03 4D 61 72
                                                            ; Table of months

                                ;       DW  -    Define Words
                                ;……….
002A  FFF0                      WORD1   DW   0FFF0H         ; Hex constant
002C  007A                      WORD2   DW   01111010B      ; Binary constant
002E  001E R                    WORD3   DW   BYTE8          ; Address constant
0030  0002 0004 0006 0007       WORD4   DW   2,4,6,7,9      ; Table of 5 constant
      0009
003A  0008[                     WORD5   DW   8 Dup(0)       ; Six zeros
          0000
                          ]


                                ;       DD  -    Define Doublewords
                                ;……….
004A  00000000                  DWORD1  DD   ?              ;Uninitialized
004E  0000A25A                  DWORD2  DD   41562          ; Decimal value
0052  00000018 00000030         DWORD3  DD   24 , 48        ; Two constant
005A  00000001                  DWORD4  DD   BYTE3-BYTE2    ;Difference


                                ;        DQ  -    Define Quawords
                                ;……….
005E  0000000000000000          QWORD1  DQ   0              ; Zero constant
0066  395E000000000000          QWORD2  DQ   05E39H         ; Hex constant
006E  5AA2000000000000          QWORD3  DQ   41562          ;Decimal constant
                                END
```

تعریف رشته های کاراکتری و مقادیر عددی

نکته: اگر عددی که در مبنای ۱۶ مشخص شده با یکی از حروف A….F آغاز شود لازم است به ابتدای عدد 0 افـزوده شود تا اسمبلر تشخیص دهد منظور از این عبارت یک عدد می باشد. بـه عنـوان مثـال بـرای مشـخص کـردن FFF0 در مبنای ۱۶ را باید 0FFF0H نوشت.

نکته: در این شکل بالا روی word3 آدرس آفست byte8 قرار خواهد گرفت، کـه 001E در مبنـای ۱۶ مـی باشـد. برای مشخص کردن این آدرس یک ارجاع به حافظه داریم. به این دلیل در ستون دوم بعد از 001E عبارت R قرار گرفتـه. R مشخص می کند که یک ارجاع به حافظه داریم.

```
002E  001E R                    WORD3   DW   BYTE8
```

اگر متغیر از نوع DQ یا DT داشته باشیم داده ها به صورت بایت معکوس ذخیره سازی خواهد شـد. بـه عنـوان مثـال اگر بخواهیم روی QWORD2 عدد 5E39H را ذخیره سازی کنیم این عدد به ترتیب بایت معکوس قرار خواهـد گرفـت یعنی به صورت: 395E000000000000 H

۵-۶-۳- پیش پردازنده EQU

از این پیش پردازنده می توان برای تعریف ثابت ها استفاده کرد. بـه عنـوان مثـال Factor EQU 12 ثـابتی بـا نـام Factor و مقدار 12 را مشخص می کند.

فصل چهارم

۴- نوشتن برنامه های COM

برنامه‌های COM فقط شامل یک سگمنت خواهد بود و اندازه یک برنامه COM نمی‌تواند بزرگتر از یک سگمنت یا 64k باشد یک برنامه‌ی com فقط شامل سگمنت کد خواهد بود و سگمنت پشته و داده را نخواهد داشت. کاربر مجبور هست در صورتی که برنامه حاوی داده باشد آنها را در سگمنت کد تعریف کند. اولین دستور نوشته شده داخل سگمنت کد 100H org خواهد بود تا اسمبلر چینش برنامه را از آفست 100 شروع کند.

داده‌های برنامه در ابتدای سگمنت کد تعریف می‌شوند برای این که بتوانیم از روی بخش داده‌ای پرش کنیم و به بخش اجرایی برنامه برسیم اولین دستور اجرایی که بعد از 100H org قرار خواهد گرفت دستور JMP می‌باشد.

```
        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

فصل پنجم

۵- دستورات سمبولیک و آدرس دهی

۵-۱- عملوند دستورات

تعداد عملوند دستورات بسته به نوع دستور می‌تواند متفاوت باشد ممکن است دستوری عملوند نداشته باشد، یک عملوند و یا دو عملوند داشته باشد. شکل کلی دستورات در زیر آمده است.

[label:]	عملیات	عملوند ۲ ، عملوند ۱

۵-۱-۱- انواع عملوند ها

- ثبات (Register)
- بلافصل (Immediate)
- حافظه (Memory)

عملوند Register (**ثبات**) می‌تواند یکی از ثباتهای ۸، ۱۶یا ۳۲ بیتی پردازنده باشد. بعنوان مثال در دستور زیر هر دو عملوند از نوع ثبات می باشد.

```
MOV    AX,BX
```

عملوندی که مقدار ثابت یا مقدار یک عبارت را داشته باشد به عنوان عملوند Immediate (**بلافصل**) گفته می شود. در دستور زیر عملوند دوم از این نوع می‌باشد. و عملوند اولی از نوع ثبات می باشد.

```
MOV   AX , 4C00H
```

اگر در یکی از عملوند ها ارجاع به حافظه داشته باشیم آن عملوند از نوع Memory (حافظه) خواهد بود. این عملوند می تواند ارجاع مستقیم به حافظه یا ارجاع غیر مستقیم به حافظه داشته باشد.

عملوند حافظه:

ارجاع مستقیم به حافظه: در این شکل یک عملوند به یک موقعیت مستقیم حافظه رجوع می‌کند.

```
WORDA      DW    0
BYTEA      DB    0
           ...
           MOV   BX , WORDA
           MOV   BYTEA , DL
           MOV   CX , DS:[38B0H]
           INC   BYTE PTR[1B0H]
```

عملوند دوم دستور MOV اولی، عملوند اول دستور MOV دومی، عملوند دوم دستور MOV سومی و همچنین تنها عملوند دستور INC از نوع ارجاع مستقیم به حافظه می باشد. در دو دستور آخر از کروشه بعنوان **شاخص نشانه گذاری** استفاده شده است.

همچنین می توان دستورات MOV زیر را در نظر گرفت که باز عملوند دوم آن ها از نوع ارجاع مستقیم به حافظه می باشد.

```
CODETLB   DB   20 DUP(?)
```

```
...
MOV  CL , CODETBL[3]
MOV  CL , CODETBL+3
```

ارجاع غیر مستقیم به حافظه: می توان با استفاده از ارجاع غیر مستقیم برای آدرس دهی تفاوت مکانی سگمنت استفاده نمود. ثبات هایی که می توان برای این منظور استفاده شوند عبارتند از SI , DI , BX و BP که ثباتهای SI، DI و BX با ثبات DS برای پردازش داده های سگمنت داده به شکل DS:SI ، DS:DI و DS:BX استفاده می‌شوند و ثبات BP با ثبات SS به شکل SS:BP برای دستکاری داده‌های پشته استفاده می‌شود. در مثال زیر نمونه ای از این نوع عملوند ها دیده می شود:

```
ADD    [BX] , 25
ADD    CL , [BX]
```

۵-۲- دستور MOV

این دستور دو عملوند خواهد داشت عملوند اول می تواند از نوع حافظه یا ثبات و عملوند دوم می تواند از نوع بلافصل، حافظه و یا ثبات باشد. این دستور مقدار موجود در عملوند دوم خود را بر روی عملوند اول کپی خواهد کرد. شکل کلی این دستور بصورت زیر خواهد بود:

[label:] MOV Register/Memory , Register/Memory/immediate

در این دستور باید اندازه حافظه عملوند اول و عملوند دوم یکسان باشد. باید توجه داشته باشید که توسط دستورات MOV نمی‌توان حافظه را به حافظه، بلافصل را به ثبات Segment و ثبات Segment را به ثبات Segment انتقال را برای این منظور بیش از یک دستور mov نیاز است. به عنوان مثال برای انتقال بلافصل بر روی یک ثبات Segment می‌توان ابتدا بلافصل را بر روی یک ثبات عمومی انتقال داد و سپس از روی ثبات عمومی بر روی ثبات Segment انتقال داد.

۵-۳- دستور انتقال و پر کردن

اگر اندازه‌ی حافظه‌ی مبداء از مقصد کوچکتر باشد می‌توان از دستورات MOVSX و یا MOVZX استفاده کرد. MOVSX برای داده‌های علامت دار استفاده می‌شود. این دستور می‌تواند یک بایت را روی یک کلمه، یا یک کلمه را روی دو کلمه منتقل کند. این کار را با کپی کردن بیت علامت روی بقیه‌ی بیت‌ها انجام می‌دهد. دستور MOVSX برای داده‌های بدون علامت استفاده می‌شود این دستور بر روی بقیه‌ی بیت‌ها عدد صفر را قرار خواهد داد. شکل کلی این دستور بصورت زیر خواهد بود.

[label:] MOVSX/MOVZX Register/Memory , Register/Memory/immediate

۵-۴- دستور XCHG

دستور XCHG مقدار عملوندهای خود را جابجا می‌کند. ساختار کلی این دستور به صورت زیر می‌باشد:

[label:] XCHG Register/Memory , Register/Memory

در این مثال:

```
XCHG CL,BH
```

دستور XCHG مقدار BH را روی ثبات CL و مقدار ثبات CL را روی ثبات BH کپی خواهد کرد.

۵-۵- دستور LEA

این دستور شامل دو عملوند می‌باشد، عملوند اولی از نوع ثبات و عملوند دومی از نوع Memory خواهد بود توسط این دستور آدرس آفست عملوند دومی بر روی عملوند اولی قرار خواهد گرفت. توسط این دستور می‌توانیم ثبات‌هایی مانند SI ، DI و BX را مقدار دهی کنیم. شکل کلی این دستور بصورت زیر می‌باشد:

[label:] LEA Register , Memory

مثال زیر نمونه‌ای از کاربرد این دستور را نشان می‌دهد:

```
Datatlb  db  20 Dup(0)
   ...
   LEA  BX , Datatlb
   MOV  [BX] , 1FH
```

۶-۵- دستور DEC و INC

این دستور فقط یک عملوند خواهند داشت که این عملوند می‌توانند از نوع ثبات و یا حافظه باشد دستور INC مقدار عملوند خود را یک واحد افزایش خواهد داد و دستور DEC مقدار عملوند خود را یک واحد کاهش خواهد داد.در فصل بعد این دستورات مفصلا توضیح داده خواهند شد.

[label:] INC/DEC Register/Memory

شکل بعد برنامه انتقال توسعه یافته را نشان می‌دهد. این برنامه برای انتقال تک تک کاراکترهای رشته‌ی HEADG1 بر روی رشته‌ی HEADG2 نوشته شده است. داخل این برنامه در ابتدای کار عدد 09 بر روی

ثبات CX قرار داده می‌شود (09 مشخص کننده‌ی تعداد کاراکترهای رشته‌ی اولی می‌باشد). در بخش‌های بعدی آدرس آفست رشته‌ی اولی با استفاده از دستور LEA در روی ثبات SI قرا می‌گیرد و آفست رشته‌ی

دومی در روی ثبات DI قرا می‌گیرد. این برنامه ابتدا کاراکتری که آفست آن برابر SI می‌باشد بر روی ثبات AL انتقال داده و سپس از روی ثبات AL بر روی حافظه‌ای که آفست آن توسط DI مشخص شده انتقال

خواهد می‌دهد. سپس مقدار SI و DI یک واحد افزوده می‌شود تا عمل انتقال کاراکتر بعدی انجام شود شمارنده‌ی حلقه (ثبات CX) یک واحد کاسته می‌شود دستور JNZ تا زمانی که محتوی CX مخالف صفر باشد

به برچسب A20 پرش خواهد کرد. این حلقه 9 بار تکرار خواهد شد و هر بار یک کاراکتر از رشته‌ی اولی روی کاراکتر متناظر از رشته‌ی دومی کپی می‌شود. نهایتاً زمانی که CX برابر صفر شد این حلقه تمام

می‌شود.

```
        Page  60,132
TITLE A06MOVE (EXE) Extended move operationa
;-------------------------------------------------------------
        .MODEL  SMALL
        .STACK  64
;-------------------------------------------------------------
        .DATA
HEADG1    DB   'InterTech'
HEADG2    DB   'LaserCorp','$'
;-------------------------------------------------------------
        .CODE
A1OMAIN    PROC  FAR
           MOV  AX , @data           ; InitiaIize segment
           MOV  DS , AX              ; registers
           MOV  ES , AX

           MOV  CX , 09              ; InitiaIize to mov 9 chars
           LEA  SI , HEADG1          ; InitiaIize addrss of headG
           LEA  DI , HEADG2          ; and HEADG2
A20:
           MOV  AL , [SI]            ; Get character from HEADG1,
           MOV  [DI],AL              ; move it to HEADG2
           INC  SI                   ; Incr next cher in HEADG1
           INC  DI                   ; Incr nexs pos'n in HEADC2
           DEC  CX                   ; Decrement cont for loop
           JNZ  A20                  ; Count not zero? Yes loop
                                     ; Finished
           MOV  AH,09H               ; Request dispIay
           LEA  DX,HEADG2            ; of HEADG2
           INT  21H

           MOV  AX,4C00H             ; End processing
           INT  21H
A1OMAIN    ENDP
           END A1OMAIN
```

برنامه انتقال توسعه یافته

بعد از اتمام حلقه با استفاده از تابع 09H از وقفه‌ی 21H محتوی رشته‌ی دوم را پس از انتقال در روی خروجی چاپ خواهد کرد. در این برنامه نحوه چاپ رشته بصورت زیر می‌باشد.

```
MOV  AH, 09H
LEA  DX, HEAOG2
INT  21H
```

فصل ششم

۶- دستورات محاسباتی

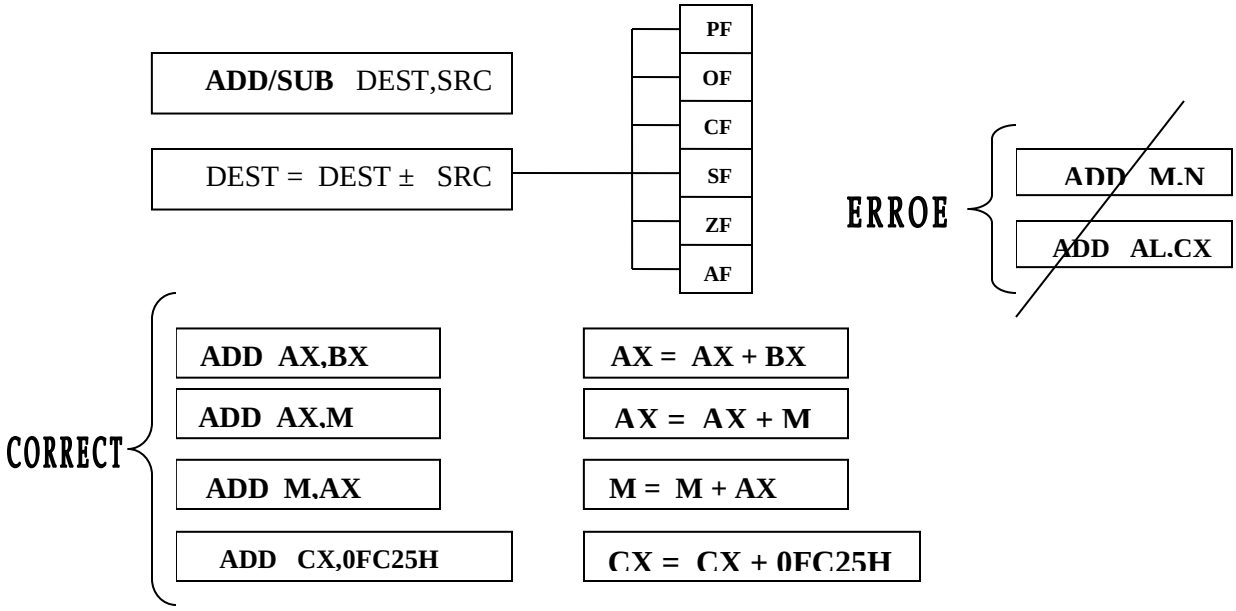
۶-۱- جمع و تفریق دودویی

برای جمع و تفریق دودویی در زبان اسمبلس دو دستور **ADD** و **SUB** وجود دارد. این عملیات ها می‌تواند بر اعداد علامت دار یا بدون علامت انجام شوند .ADD و SUB مقدار دو عملوند را جمع کرده و روی عملوند اول قرار می دهد.نتیچه این عملیات ها محتوای برخی برچم ها را تغییر خواهد داد. ADD و SUB هیچ تفاوتی را مابین اعداد علامت دار و بدون علامت قائل نیست و عمل ثابتی را انجام می‌دهند.

پس از انجام محاسبه ممکن است نتیجه محاسبه صحیح باشد یا صحیح نباشد. از طریق بررسی وجود یا عدم وجود رقم نقلی یا سرریز محاسباتی می توان تشخیص دهیم که محاسبه دودویی برای حالت علامت دار یا بدون علامت صحیح می باشد یا نه .

شکل کلی این دستورات

```
[label:]    ADD/SUB     register,register
[label:]    ADD/SUB     memory,register
[label:]    ADD/SUB     register,memory
[label:]    ADD/SUB     register,immediate
[label:]    ADD/SUB     memory,immediate
```



۱-۱-۶- رقم نقلی محاسباتی

رقم نقلی، بس از انجام محاسبه ممکن است نتیجه درست یا نادرست باشد.از طریق تست OFوCF میتوان اعتبار محاسبات را برای اعداد بدون علامت و با علامت سنجید. خروجی از با ارزشترین بیت در محاسبات دودویی می‌باشد که در روی پرچم CF قرار می گیرد. چون در روی پردازشگر هیچ تفاوتی مابین محاسبه داده های علامتدار و یا بدون علامت وجود ندارد پس این بیت خروجی رقم نقلی حاصل شده از سمت چپ ترین بیت در پردازشهای دو دویی خواهد بود. اگر برابر صفر باشد نتیجه محاسبه برای داده های دودویی بدون علامت معتبر خواهد بود و اگر یک باشد نامعتبر.

۲-۱-۶- سرریز محاسباتی

پرچم OF وجود یا عدم وجود سر ریز محاسباتی را نشان می دهد. سر ریز محاسباتی از طریق رقم نقلی ورودی و رقم نقلی خروجی با ارزشترین بیت (سمت چپ ترین بیت) مشخص می شود. اگر رقم نقلی خروجی از آخرین بیت مخالف رقم نقلی ورودی به آن باشد سرریز محاسباتی اتفاق افتاده و OF = 1 خواهد شد. در غیر این صورت، یعنی زمانی که رقم نقلی خروجی برای رقم نقلی ورودی باشد OF=0 خواهد شد. اگرOF برابر صفر باشد محاسبه برای داده های علامتدار صحیح بوده در غیر این صورت (یعنی OF برابر یک باشد) صحیح نخواهد بود .
مثالهای زیر نمونه هایی را نشان می دهند:

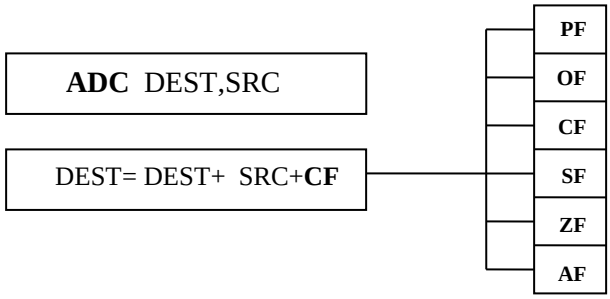
دودویی	دهدهی: بدون علامت	دهدهی: علامتدار	CF	OF
10110010	178	−78		
+ 00011011	+ 27	+ 27	0	0
11001101	205	−51		
11110100	244	−12		
+ 01110011	+ 115	+ 115	1	0
101100111	103	103		
01101000	104	104		
+ 01010111	+ 87	+ 87	0	1
10111111	191	−65		
10010100	148	−108		
+ 11010110	+ 214	+ −42	1	1
101101010	106	+106		

مثال زیر حالتهای مختلف عمل جمع و تفریق را نشان می‌دهد.

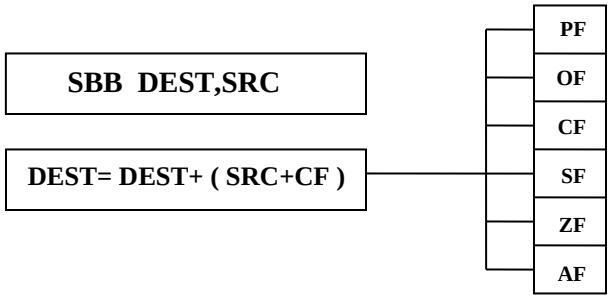
```
TITLE          A13ADD  (COM)ADDandSUBoperation
               .MODEL     SMALL
               .CODE
                 ORG        100H
BEGIN:         JMP        SHORT A10MAIN

    ;.............................................
BYTE1          DB         64H           ;DATA ITEMS
BYTE2          DB         40H
BYTE3          DB         16H
WORD1          DW         4000H
WORD2          DW         2000H
WORD3          DW         1000H
;.............................................
A10MAIN        PROC       NEAR          ;MAIN PROCEDURE:
          CALL        B10ADD        ;CALL ADD ROUTINE
               CALL       C10SUB        ;CALL SUB ROUTINE
               MOV        AX,4C00H      ;END PROCESSING
               INT        21H
A10MAIN        ENDP
;                           EXAMPIES OF ADD BYTES
;                           ....................
B10ADD         PROC
               MOV        AL,BYTE1
               MOV        BL,BYTE2
               ADD        AL,BL         ;REGISTER-TO-REGISTER
               ADD        AL,BYTE3      ;MEMORY-TO-REGISTER
               ADD        BYTE1,BL      ;REGISTER-TO-MEMORY
               ADD        BL,10H        ;IMMEDIATE-TO-REGISTER
               ADD        BYTE1,25H     ;IMMEDIATE-TO-MEMORY
               RET
B10ADD         ENDP
;                           EXAMPLES OF SUB WORDS;
;                           ....................
C10SUB         PROC
               MOV        AX,WORD1
               MOV        BX,WORD2
               SUB        AX,BX         ;REGISTER-FROM-REGISTER
               SUB        AX,WORD3      ;MEMORY-FROM-REGISTER
               SUB        WORD1,BX      ;REGISTER-FROM-MEMORY
               SUB        BX,1000H      ;IMMEDIATE-FROM-REGISTER
               SUB        WORD1,256H    ;IMMEDIATE-FROM-MEMORY
               RET
C10SUB         ENDP
               END        BEGIN
```

۳-۱-۶- جمع با بیت نقلی(ADC)



۴-۱-۶- تفریق به کمک بیت قرضی(SBB)



۲-۶- دستور العمل های DEC,INC

دستور العمل INC,DEG به تزتیب عملوند مقصد را به اندازه یک واحد کاهش و افزایش می دهد این دستورات فقط یک عملوند داشته و این عملوند میتواند از نوع ثبات یا حافظه باشد این دو دستور هیچ تصوری راجع به علامت دار بودن یا بدون علامت بودن عملوند های خود ندارد.÷س ا اجرای این دستور مقدار برچم های PF-AF-ZF-SF-OF تغییر خواهد کرد.اما مقدار برچم CF تغییر بیدا نمی کند.درست است که این دو دستور همانند ADD و SUB عمل خواهد کرد اما کارایی انها نسبت به دستورات جمع و تفریق متناظر بالاتر میباشد.عملوند این دو دستور میتواند ۸-۱۶-۳۲ بیتی باشد.

شکل کلی این دستورات به صورت زیر است

[label:] INC/DEG Register/Memory

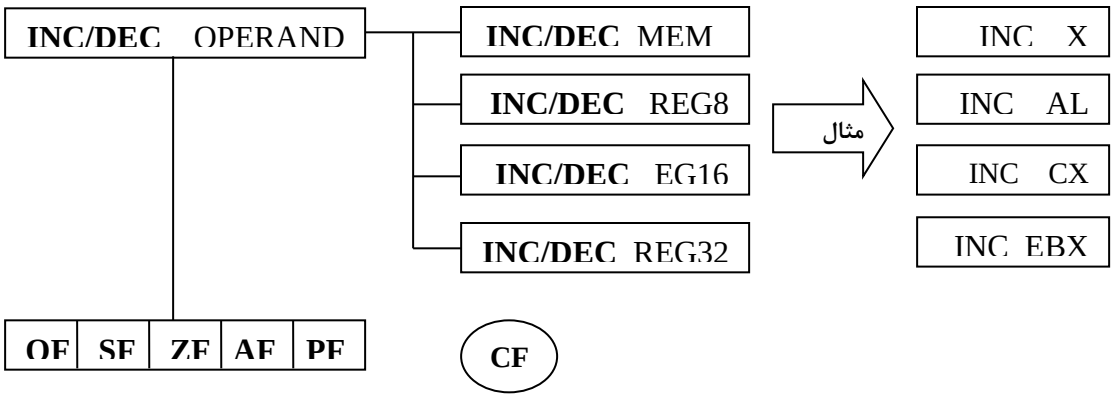
نکات :

این دستورات مقدار عملوند مقصد را به صورت یک عدد بدون علامت در نظر میگیرند

پرچمهای OF,SF,ZF را تغییر می دهند ولی نشانه CF را تغییر نمیدهند

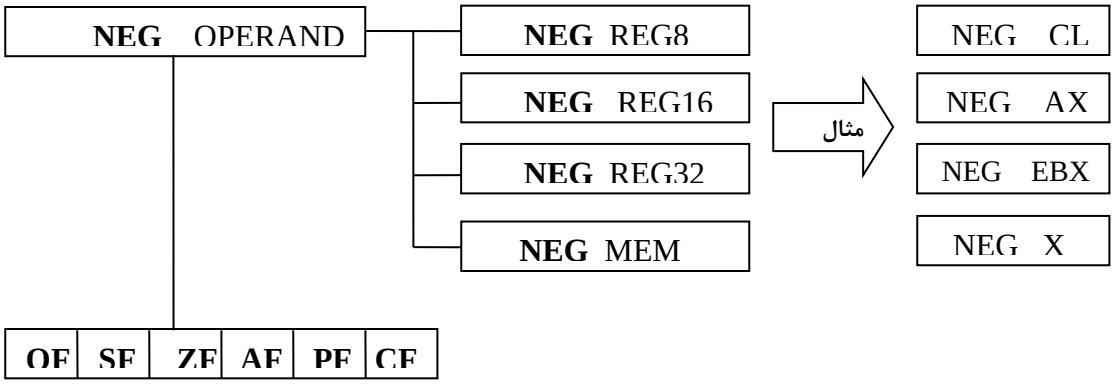
برای افزایش و کاهش شمارنده ها مفیدند و از دستورات جمع و تفریق متناظر کار امد ترند

بهترین مکان برای نگه داشتن شمارنده در صورت امکان ثبات ها می باشد



۳-۶- دستور العمل NEG

این دستورالعمل عملوند خود را منفی میکند یعنی مکمل ۲ ان را محاسبه مینماید.این دستور فقط یک عملوند داشته و این عملوند میتواند ثبات یا حافظه باشد این دستور بس از اجرا مقدار عملوند خود را منفی خواهد کرد به عبارتی مکمل ۲ ان را مخاسبه خواهد کرد.در حقیقت مکمل ۲ ان را محاسبه میکند.این دستور بس از اجرای مقدار ثباتهای OF-SF-ZF-AF-PF-CF را تغییر می دهد.



۴-۶- دستور العملهای ضرب

اسمبلی دارای دو دستور العمل ضرب میباشد:

دستور IMULL:

این دستورعملوند را به صورت علامتدار در نظر میگیردوبرای ضرب اعداد علامتدار استفاده میشود.

دستور MULL :

این دستورعملوند را بصورت بدون علامت در نظر میگیردوبرای ضرب اعداد علامتدار استفاده میشود.

این دستورات فقط یک عملوند دارند.که اندازه عملوند نوع ضرب را مشخص میکند.شکل کلی این دستورات به شکل زیر میباشد:

[label:] MUL/IMUL Register/memory

عملوند ثابت نمی تواند باشدچنانچه عملوند از نوع یک بایتی باشدضرب یک بایتی اتفاق خواهد افتاد.در ضرب یک بایتی محتوی عملونددر محتوی AL ضرب شده نتیجه در AX قرار میگردد.

چنانچه عملوناز نوع دو بایتی(WORD)باشد محتوی عملوند در محتوی AX ضرب شده نتیجه در DX:AX قرار میگیرد یعنی (word)کم ارزش بر روی AX و (word)پر ارزش در DX قرار خواهد گرفت. و محتوی ثبات های DX,AX از بین میرود.بس از عمل ضرب ممکن است مقادیر نشانه های ZF,SF,PF تغییر کند.

مثال:

```
MOV   AL, 10
MOV   X, -8
MUL   X
```

باید دقت داشت که X از نوع یک بایتی باشد تا ضرب یک بایتی اتفاق بیافتد.در این مثال محتوی ثباتها AX برابر 80- میشود.

۵-۶- دستور العمل های تقسیم

اسمبلی دارای دو دستورالعمل تقسیم میباشد:

دستورIDIV:

این دستورعملوند را به صورت علامتدار در نظر میگیرد.

دستورDIV:

عملوند را بدون علامت در نظر میگیرد.

شکل کلی این دستورات به صورت زیر است:

```
[label:]    DIV/IDIV        Register/Memory
```

عملوند ثابت نمیتواند

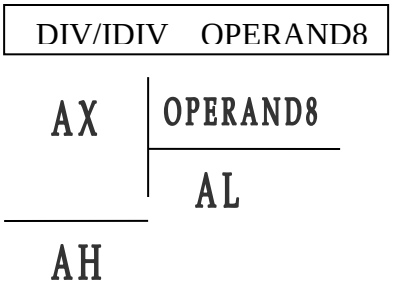
چنانچه عملوند از نوع یک بایتی باشد محتوی AXبر عملوند تقسیم شده نتیجه در AL قرار میگیرد و باقی مانده تقسیم در AH قرار میگیرد.

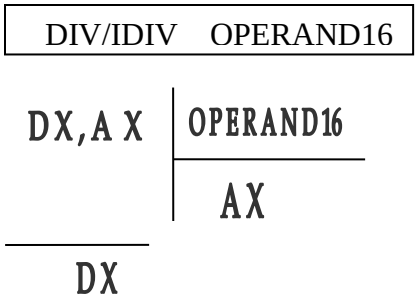
چنانچه عملوند از نوع دو بایتی(WORD)باشد محتوی DX:AX بر محتوی عملوند تقسیم شده نتیجه تقسیم در AX قرار میگیرد.و باقی مانده در DX قرار میگیرد.

- مثال:

```
X       DB          13
MOV     AX, 134
DIV     X
```

بس از اجرای دستورالعمل های فوق محتوی AL برابر با 10 و محتوی AH برابر با ۴ میباشد





۶-۶- دستورات گسترش داده:

- CBW Convert byte to word
- CWD Convert word to doubleword

فصل هفتم

۷- موارد ضروری برنامه نویسی برای منطق و کنترل

۷-۱- انواع آدرس‌ها

اسمبلر از سه نوع آدرس پشتیبانی می‌کند که عبارتند از

۱. آدرس کوتاه (short) : محدوده‌ای در فاصله‌ی 128- تا 127+ بایت می‌باشد.
۲. آدرس نزدیک (Near) : محدوده‌ای در فاصله‌ی 32768- تا 32767+ بایت داخل همان سگمنت را مشخص می‌کند.
۳. آدرس دور (Far) : فاصله‌ی بیشتر از 32k را مشخص می‌کند.

برخی از دستورات مانند LOOP، CALL، JMP و Jnnn می‌توانند همه یا برخی از این آدرس‌ها را پشتیبانی کنند. به عنوان مثال دستور JMP همه‌ی این آدرس‌ها را پشتیبانی می‌کند.

۷-۲- دستور JMP

با استفاده از دستور JMP می‌توان به بخش مشخص شده‌ای از برنامه بدون هیچ شرطی پرش نمود. ساختار کلی این دستور به صورت شکل زیر می‌باشد:

```
[label:]        JMP        short, near, or far address
```

با استفاده از دستور JMP می‌توان به یک آدرس Short و Near و یا Far پرش نمود. شکل زیر نمونه مثالی را از به کارگیری دستور JMP نشان می‌دهد.

```
                                Page   60 , 132
          TITLE  A08JUMP  (COM) using JMP for looping
                 .MODEL  SMALL
                 .CODE
0100                    ORG  100H
0100             A10MAIN  PROC  NEAR
0100  B8 0001            MOV  AX , 01
0103  BB 0001            MOV  BX , 01
0106  B9 0001            MOV  CX , 01
0109             A20:
0109  05 0001            ADD  AX , 01        ;Add 01 to AX
010C  03 D8              ADD  BX , AX        ;Add AX to BX
010E  D1 E1              SHL  CX , 1         ;Double CX
0110  EB F7              JMP  A20            ;Jump to A20 label
0112             A10MAIN  ENDP
                    END  A10MAIN
```

استفاده از دستور JMP

۳-۷- دستور LOOP

از این دستور برای ایجاد حلقه‌ها می‌توان استفاده کرد در این حالت باید تعداد تکرارهای حلقه را در روی ثبات CX قرار دهیم. هر بار که دستور LOOP اجرا می‌شود مقدار CX را یک واحد کاهش می‌دهد و در صورتی که CX مخالف صفر باشد به بر جسب مشخص شده پرش خواهد کرد. این دستور فقط از آدرس‌های کوتاه پشتیبانی می‌کند یعنی پرش می‌تواند از در فاصله‌ی 128- تا 127+ بایت باشد.

[label:] LOOP short address

```
                Page  60 , 132
          TITLE  A08LOOP (COM) Illustration of LOOP
                 .MODEL  SMALL
                 .CODE
0100                     ORG  100H
0100             A10MAIN  PROC  NEAR
0100  B8 0001            MOV  AX , 01       ; Initialize AX ,
0103  BB 0001            MOV  BX , 01       ; BX , and
0106  BA 0001            MOV  DX , 01       ; DX to 01
0109  B9 000A            MOV  CX , 10       ; Initialize for
010C             A20:                       ; ten loops
010C  40                 INC  AX            ; Add 01 to AX
010D  03 D8              ADD  BX , AX       ; Add AX to BX
010F  D1 E2              SHL  DX , 1        ; DOUBLE DX
0111  E2 F9              LOOP A20           ; Decrement CX ,
                                            ; LOOP  if  nonzero
0113  B8 4C00            MOV  AX , 4C00H    ; End processing
0116  CD 21              INT  21H
0118             A10MAIN  ENDP
                    END  A10MAIN
```

استفاده از دستور LOOP

در این برنامه اولین دستور، دستور ORG 100H می‌باشد. این دستور به اسمبلر می‌فهماند که چینش برنامه باید از آفست 100H شروع شود به همین دلیل در این مثال اول آفست برابر 0100H می‌باشد. برای دستور ORG هیچ کد زبان ماشین تولید نمی‌شود به همین دلیل آفست بدی همان آفست قبلی یعنی 0100H می‌باشد. سطر بعدی دستور PROC می‌باشد که اول تابع A10MAIN را مشخص می‌کند. برای این دستور هیچ کد زبان ماشین تولید نمی‌شود. پس آفست بعدی همان آفست قبلی خواهد بود. در سطر بعدی دستور MOV AX,01 قرار دارد.کدمایشن دستور MOV AX برابر B8 می‌باشد و عددی هم که باید روی AX قرار بگیرد 0001 خواهد بود برای ذخیره سازی این کد B8 0001 سه بایت مورد نیاز می‌باشد پس ofsset برابر خواهد بود با ofsset قبلی به اضافه سه (0103=3+0100).

نکته: دستور MOV به صورت MOV AX, 01 می‌باشد و کد زبان ماشین نیز به صورت B8 0001 نوشته شده، در این مثال اگر دقت کنید عددی که قرار است روی AX قرار گیرد 01 می‌باشد اما که زبان ماشین آن بصورت 0001 نوشته شده است. این کار به این دلیل می‌باشد که AX یک ثبات 2 byte می‌باشد و عددی که روی آن قرار خواهد گرفت باید به صورت۲ بایتی تعریف شود به این دلیل 0001 نوشته شده است.

برچسب A20 هیچ کد زبان ماشینی نیاز ندارد به همین دلیل آفست بعدی همان آفست جاری خواهد بود (010C) دستور LOOP مقدار CX را یک واحد کاهش خواهد داد و تا زمانیکه CX مخالف صفر باشد به برچسب A20 پرش

می‌کند. کد زبان ماشین LOOP برابر E2 می‌باشد و آدرسی هم که پشتیبانی می‌کند یک آدرس کوتاه خواهد بود (+127 تا 128-) پس یک بایت برای مشخص کردن این آدرس کافی خواهد یود (اختلاف مکان). زمانیکه دستور LOOP در حال اجرا می‌باشد ثبات IP آفست دستور بعدی (0113) را مشخص خواهد کرد، اختلاف مکان IP با برچسب A20 (آفست A20 برابر 010C می‌باشد) منفی هفت بایت (010C-0113=-0007) می‌باشد. عدد 7- در مبنای شانزده برابر F9 می‌باشد. نوشتن F9 در کد ماشین دستور LOOP A20 به این دلیل می‌باشد.

$$(-7)_{10} = (-0000111)_2 = (\underbrace{1111}_{F}\underbrace{1001}_{9})_2 = (F9)_{16}$$

۴-۷- دستور CMP

این دستور (Compare) جهت مقایسه‌ی دو فیلد داده‌ای استفاده می‌شود. این دستور دو عملوند خواهد داشت عملوند اول می‌تواند حافظه یا ثبات و عملوند دوم می‌تواند حافظه، ثبات یا Immediate باشد. پس از اجرای این دستور پرچم‌های CF ، PF ، AF ، ZF ، SF و OF تغییر پیدا خواهند کرد. از طریق مقدار این پرچم‌ها دستورات بعدی می‌توانند تصمیم خاصی را اتخاذ کنند. شکل کلی این دستور به صورت زیر می‌باشد:

[label:] CMP register/memory , register/memory/immediate

در مثال زیر دستور CMP مقدار ثبات AX را با BX مقایسه می‌کند و پس از مقایسه‌ی این مقادیر دستور بعدی JE (پرش کن اگر برابر باشند) می‌تواند تصمیم بگیرد که به برچسب A20 پرش کند یا نه این کار با بررسی وضعیت پرچم‌ها انجام خواهد داد.

```
    CMP AX , BX
    JE  A20
    ...
A20:
```

۵-۷- پرش شرطی

دستورات پرش شرطی بر اساس وضعیت برخی از پرچم ها تصمیم می‌گیرند که به یک بر چسب پرش کنند یا نه. شکل کلی این دستورات به صورت زیر می‌باشد:

[label:] Jnnn short address

جداول زیر این دستورات را نشان می‌دهند.

سمبول	توضیح	پرچم های تست شده
JE/JZ	پرش مساوی یا پرش صفر	ZF
JNE/JNZ	پرش غیرمساوی یا پرش غیر صفر	ZF
JG/JNBE	پرش بیشتریا پرش کمترنبودن یا مساوی	CF,ZF
JAE/JNB	پرش بیشتر نبودن یا مساوی یا پرش کمتر نبودن	CF
JB/JNAE	پرش کمتر یا پرش بیشتر نبودن یا مساوی	CF
JBE/JNA	پرش کمتر یا مساوی یا بیشتر نبودن	ZF,CF

پرشهای مربوط به داده های بدون علامت (منطقی)

سمبول	توضیح	پرچم های تست شده
JE/NZ	پرش مساوی یا پرش صفر	ZF
JNE/JNZ	پرش غیرمساوی یا پرش غیر صفر	ZF
JA/JNLE	پرش بزرگتر یا پرش غیر کو چکتر یا مساوی	OF,SF ZF
JCE/JNL	پرش غیر بزرگتر یا پرش غیر کوچکتر	OF, SF
JL/NGE	پرش کوچکتر یا پرش غیر بزرگتر یا مساوی	OF,SF
JLE/JNG	پرش کوچکتر یا مساوی یا پرش غیر بزرگتر	OF,SF,ZF

پرشهای مربوط به داده های علامت دار (محاسباتی)

سمبول	توضیح	پرچم های تست شده
JCXZ	پرش اگر CX صفر است	هیچ یک
JC	پرش اگر رقم نقلی وجود دارد	CF
JNC	پرش اگر رقم نقلی وجود ندارد	CF
JO	پرش اگر سرریز وجود دارد	OF
JNO	پرش اگر سرریز وجود ندارد	OF
JP/JPE	پرش اگر توازن وجود دارد یا توازن زوج است	PF
JNP/JPO	پرش اگر توازن وجود ندارد یا توازن فرد است	PF
JS	پرش اگر علامت دارد (منفی)	SF
JNS	پرش اگر علامت ندارد (مثبت)	SF

پرشهای محاسباتی خاص

این دستورات فقط می‌تواند به آدرس‌های کوتاه پرش کننده.

۶-۷- دستور CALL و دستور RET

عبارت PROC تعریف کننده Procedure (رویه) می باشد. هر رویه با عبارت PROC آغاز شده و با عبارت ENDP خاتمه پیدا می کند برای فراخوانی رویه می توان از دستور CALL استفاده کرد و برای برگشتن از رویه به رویه فراخوانی کننده می توان از دستور RET استفاده کرد. زمانی که با استفاده از دستور CALL یک رویه فراخوانی می شود اجرای برنامه به اولین سطر رویه مشخص شده منتقل می شود، دستورات رویه سطر به سطر اجرا شده و نهایتا با اجرای دستور RET کار رویه تمام شده و اجرای برنامه از دستور بعد از CALL ادامه پیدا می کند.

باید توجه داشته باشید که برای برگشتن از تابع حتما باید از دستور RET استفاده شود. داخل یک رویه RET آخرین دستوری است که اجرا خواهد شد و کل دستورات بعد از آن داخل همان رویه اجرا نخواهد شد. همچنین معمولا در یک برنامه exe رویه اصلی از نوع FAR تعریف می شود و بقیه رویه ها از نوع NEAR اما نباید در برنامه های COM از رویه FAR استفاده کرد.

شکل زیر نمونه ای از تعریف روال و فراخوانی آنها را نشان می‌دهد.

```
                                page 60,132
                        TITLE A08CALLP (EXE) Calling procedure
                              .MODEL SMALL
                              .STACK 64
                              .DATA
                        ;-----------------------------------------
                              .CODE
0000                    A10MAIN  PROC  FAR
0000  E8 0008 R               CALL   B10              ; Call B10
                        ;       ...
0003  B8 4C00                 MOV    AX, 4C00H        ; End processing
0006  CD 21                   INT       21H
0008                    A10MAIN  ENDP
                        ;-----------------------------------------
0008                    B10   PROC  NEAR
0008  E8 000C R               CALL C10                ; Call C10
                        ;     ...
000B  C3                      RET                     ; Return to
000C                    B10   ENDP                    ; caller
                        ;-----------------------------------------
000C                    C10   PROC   NEAR
                        ;     ...
000C  C3                      RET                     ; Return to
000D                    C10   ENDP                    ; caller
                        ;-----------------------------------------
                              END A10MAIN
```

روالهای فراخوان

فصل هشتم

۸- عملیات دودویی

دستورات بول که در زبان اسمبلی وجود دارند عبارتند از AND ، OR ، XOR ، TEST و NOT شکل کلی این دستورات به صورت زیر می باشد

[label:] operator register/memory , register/memory/immediate

اجرای این دستورات پرچم های ZF و SF و DF و OF و CF را تغییر خواهد داد.

AND :

دستور AND حاصل بیت های متناظر را یک قرار خواهد داد اگر هر دو بیت متناظر برابر یک باشد.

OR :

دستور OR بیت متناظر را یک قرار خواهد داد اگر حداقل یکی از بیت ها برابر یک باشد.

XOR :

دستور XOR نتیجه ی صفر خواهد داشت، اگر بیت های متناظر برابر باشند، و نتیجه ی یک خواهد داشت اگر بیت های متناظر مخالف هم باشند.

TEST :

دستور TEST پرچم ها را مانند عملیات AND تنظیم خواهد کرد با این تفاوت که مقدار عملونداول تغییر نخواهد کرد.

NOT :

دستور NOT فقط یک عملوند خواهد داشت از نوع ثباتی یا حافظه این دستور بر روی عملوند خود کلیه بیت های صفر را به یک و یک را به صفر تغییر خواهد داد.

مثال تبدیل حروف بزرگ به حروف کوچک : تفاوت ما بین حروف بزرگ و کوچک در بیت ششم آنها می باشد. در حروف بزرگ این بیت برابر صفر و در حروف کوچک این بیت برابر یک می باشد. اگر بخواهیم حروف بزرگ را به حروف کوچک تبدیل کنیم می توان این بیت را از صفر به یک تغییر داد. همچنین برای تبدیل حروف کوچک به حروف بزرگ می توان این بیت را از یک به صفر تغییر داد. برنامه زیر این عمل را انجام می‌دهد.

```
        Page   60 , 132
TITLE   A08CASE  (COM) Change uppercase to lowercase
        .MODEL  SMALL
        .CODE
        ORG  100H
BEGIN: JMP  A10MAIN
;----------------------------------
CONAME   DB   'INTERTECH SYSTEM','$'
;----------------------------------
A10MAIN  PROC  NEAR
     LEA  BX , CONAME+1
          MOV  CX , 15
A20:
     MOV  AH , [BX]
     CMP  AH , 41H
     JB   A30
     CMP  AH , 5AH
     JA   A30
     XOR  AH , 00100000B
     MOV  [BX] , AH
A30:
     INC  BX
     LOOP A20

     MOV  AH , 09H
     LEA  DX , CONAME
     INT  21H

     MOV  AX , 4C00H
     INT  21H
A10MAIN  ENDP
          END  BEGIN
```

تغییر حروف بزرگ به کوچک

۱-۸- شیفت بیتها

این دستورات باعث حرکت بیتها ی مربوط به یک عدد خواهد شد. شکل کلی این دستور بصورت زیر خواهد بود.

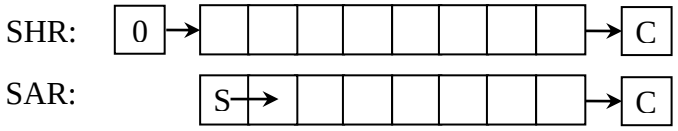
[label:] shift register/memory , CL/immediate

در پردازنده های 8086\8088 فقط شیفت یک واحدی پذیرفته شده می باشد. یعنی عملوند دوم فقط باید عدد یک باشد اما اگر تعداد شیفت ها بیش از یک باشد باید ابتدا تعداد شیفتها را داخل ثبات CL قرار دهیم و این ثبات را در عملوند دوم دستور شیفت استفاده کنیم در پردازنده های رده بالاتر این کار لزومی ندارد.

۱-۱-۸- شیفت به راست بیتها

برای این منظور دو دستور SHR و SAR وجود دارد. SHR برای شیفت منطقی به راست (برای داده های بدون علامت) می‌باشد. این دستور کل بیت ها را به سمت راست shift خواهد داد از سمت چپ صفر وارد می شود و سمت راست ترین بیتی که پایین می افتد داخل CF قرار می‌گیرد.SAR برای شیفت محاسباتی به راست (برای داده های علامت دار)

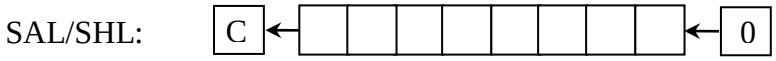
می‌باشد. SAR بیت علامت را از سمت چپ کپی خواهد کرد و سمت راست ترین بیت که پایین ترین بیت که پایین خواهد افتاد روی CF قرار می گیرد. شکل زیر نحوه عملکرد این دو دستور را نشان می‌دهد.



نکته : یک واحد شیفت به راست باعث می شود عدد تقسیم بر ۲ شود.

۲-۱-۸- شیفت به سمت چپ

برای شیفت به چپ می توان از SHL و SAL استفاده نمود. برای هر دوی این حالات از سمت راست بیت صفر وارد می شود و سمت چپ ترین بیتی که خارج می شود روی CF قرار می گیرد. هر واحد شیفت به سمت چپ باعث می شود تا عدد در ۲ ضرب شود.

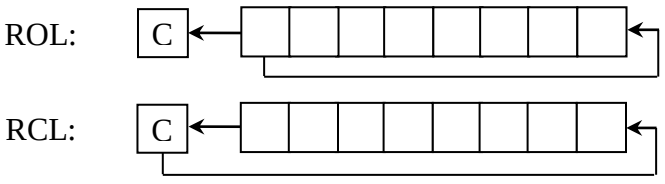


۲-۸- چرخش بیت ها به راست

برای چرخش به راست دو دستور ROR و RCR وجود دارد دستور ROR برای چرخش منطقی به راست برای داده های بدون علامت استفاده می شود و RCR برای چرخش با رقم نقلی به راست برای داده های علامت دار استفاده می شود. نحوه عملکرد این دستورات بصورت شکل زیر است.

۳-۸- چرخش بیت ها به چپ

برای چرخش به چپ دو دستور ROL و RCL وجود دارد دستور ROL برای چرخش منطقی به چپ برای داده های بدون علامت و RCR برای چرخش با رقم نقلی به چپ برای داده های علامت دار می‌باشد. نحوه عملکرد این دستورات بصورت شکل زیر است.



```
                        TITLE  A08JMPTB (EXE)  Using a jump table
                              .MODEL SMALL
                              .STACK 64
                              .DATA
0000  001E R                  CUSTTAB  DW   B10CDE0     ; Table of addresses
0002  0025 R                         DW   B11CDE1
0004  002C R                         DW   B12CDE2
0006  0033 R                         DW   B13CDE3
0008  003A R                         DW   B14CDE4
000A  43 6F 64 65 20 30       MESSG0   DB  'Code 0 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
001C  43 6F 64 65 20 31       MESSG1   DB  'Code 1 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
002E  43 6F 64 65 20 32       MESSG2   DB  'Code 2 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
0040  43 6F 64 65 20 33       MESSG3   DB  'Code 3 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
0052  43 6F 64 65 20 34       MESSG4   DB  'Code 4 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
                        ;-------------------------------------
                              .CODE
                              .386
0000                          A10MAIN  PROC  FAR
0000  B8 ---- R                       MOV   AX,@data      ; Initialize
0003  8E D8                           MOV   DS, AX        ; segment
0005  8E C0                           MOV   ES, AX        ; registers
0007  E8 000F R                       CALL  B10JUMP
000A  B8 4C00                         MOV   AX, 4C00H     ; End processing
000D  CD 21                           INT   21H
000F                          A10MAIN  ENDP
                        ;
000F                          B10JUMP  PROC  NEAR
000F  B4 10                           MOV   AH, 10H       ; Get KB char
0011  CD 16                           INT   16H           ; into AL
0013  24 07                           AND   AL, 00000111B ;clear left 5 bits
0015  0F B6 D8                        MOVZX BX, AL        ; move AL to BX
0018  D1 E3                           SHL   BX, 01        ; Double  value
001A  FF A7 0000 R                    JMP   [CUSTTAB+BX]  ;Jump to cust  rtne
001E  8D 16 000A R            B10CDE0: LEA   DX , MESSG0   ;Code 0 routine
0022  EB 1D 90                        JMP   B90
0025  8D 16 001C R            B11CDE1: LEA   DX , MESSG1   ;Code 1 routine
0029  EB 16 90                        JMP   B90
002C  8D 16 002E R            B12CDE2: LEA   DX , MESSG2   ;Code 2 routine
0030  EB 0F 90                        JMP   B90
0033  8D 16 0040 R            B13CDE3: LEA   DX , MESSG3   ;Code 3 routine
0037  EB 08 90                        JMP   B90
003A  8D 16 0052 R            B14CDE4: LEA   DX , MESSG4   ;Code 4 routine
003E  EB 01 90                        JMP   B90
0041                          B90 :
0041  B4 09                           MOV     AH,09H      ;Display
0043  CD 21                           INT       21H
0045  C3                              RET
0046                          B10JUMP  ENDP
                               END   A10MAIN
```

استفاده از جدول پرش

این برنامه تابع PROC B10JUMP را فراخوانی می کنید. این رویه در ابتدای کار با استفاده از تابع 10H از وقفه ی شماره 16H کیلدی را از صفحه کلید خوانده و کد آن را داخل ثبات AL برگشت می دهد در سطر بعد این کد با عدد AND 00000111B می شود تا اینکه نهایتا ۵ بیت پردازش AL به صفر تبدیل شود با این کار ۳ بیت کم ارزش مقدار قبلی خود را حفظ خواهد کرد. با استفاده از MOVZX مقدار AL بر روی BX منتقل می شود، تا این عدد که ما بین صفر تا چهار می‌باشد روی BX قرار بگیرد. می خواهیم از طریق این کد به برچسبی پرش کنیم که بر روی CUSTTAB مشخص شده است. هر یک از عناصر این متغیر دو بایتی می باشد (DW) برای این منظور BX یک واحد به سمت چپ شیفت داده می شود (دو برابر می‌شود) تا با این کار به هر یک از عناصر این جدول دسترسی پیدا کنیم. دستور JMP بعدی به بر حسب آندیسی که BX مشخص می‌کند به برچسب مورد نظر برش خواهد کرد.

فصل نهم

۹- مقدمه‌ای بر پردازش صفحه کلید و صفحه نمایش

۹-۱- صفحه نمایش

صفحه ی نمایش در حالت متنی از ۲۵ سطر و ۸۰ ستون تشکیل شده است. مختصات گوشه ی سمت چپ بالای صفحه برابر (0,0) و گوشه ی سمت راست پایین صفحه برابر (24,79) (در مبنای ۱۶، 18,4F) می باشد. پس با این مفروضات مرکز صفحه در سطر ۱۲ و ستون ۳۹ یا ۴۰ قرار خواهد داشت.

۹-۱-۱- انتقال مکان نما

برای انتقال مکان نما بر روی صفحه ی نمایش می توان از تابع 02H وقفه ی شماره 10H استفاده کرد. در این حالت باید شماره صفحه را در ثبات BH (معمولا برابر صفر است)، شماره سطر را در ثبات DH و شماره ستون را در ثبات DL قرار دهیم. در مثال زیر با استفاده از این وقفه مکان نما به سطر هشتم (08H) و ستون پانزدهم (0FH) از صفحه ی شماره صفر منتقل خواهد شد.

```
MOV   AH,02H      ; درخواست تنظیم مکان نما
MOV   BH,00       ; شماره صفحه صفر
MOV   DH,08H      ; سطر ۸
MOV   DL,0FH      ;سطون ۱۵
INT   10H         ; فراخوانی سرویس وقفه
```

می توان دو دستور MOV مربوط به DH و DL (سطر ۳ و ۴)را با هم دیگر ترکیب نمود و آنها را در قالب یک دستور نوشت.

```
MOV   DX,080FH    ;  سطر ۸ و ستون ۱۵
```

۹-۱-۲- پاک کردن صفحه نمایش

می توان از تابع 06H از وقفه ی 10H برای پاک کردن صفحه ی نمایش استفاده کرد در این حالت باید ثباتهایی که در زیر آمده اند به ترتیب مقداردهی شوند.

AH = تابع 06H.

AL = شماره خطهایی که حرکت طوماری دارند، صفر برای کل صفحه.

BH = مقدار صفت (رنگ قلم، رنگ پس‌زمینه و حالت چشمک زن)

CX = ستون : سطر شروع.

DX = ستون : سطر پایان.

نکته:خصوصیات صفحه مشخص کننده رنگ قلم و رنگ پس زمینه می باشد که یک عدد یک بایتی می باشد بیت های صفر تا سه از این عدد مشخص کننده ی رنگ قلم بیت های ۴ تا ۶ مشخص کننده رنگ پس زمینه و بیت هفتم مشخص کننده حالت چشمک زن می باشد. پس برای یک صفحه می توان ۱۶ رنگ قلم و هشت رنگ پس زمینه مشخص نمود. در مثال زیر رنگ قلم شماره یک و رنگ پس زمینه ی هفت مشخص شده است و کل صفحه با شروع از مختصات 00:00تا مختصات 18H:4FH پاک خواهد شد.

```
MOV   AX,0600H      ; AH=0 (حرکت طوماری) AL=0 (صفحه کامل)
MOV   BH,71H        ; پس زمینه سفید (7) ، پیش زمینه آبی (1)
MOV   CX,000H       ; ستون : سطر بالای چپ
MOV   DX,184FH      ; ستون : سطر پایین راست
INT   10H           ; فراخوانی سرویس وقفه
```

۳-۱-۹- چاپ رشته در خروجی

از تابع شماره 09H وقفه ی 21H می توان برای چاپ کردن رشته در خروجی استفاده کرد. این تابع کل کاراکترهای رشته از اولین کاراکتر تا رسیدن به کاراکتر $ را در خروجی چاپ می‌کند. برای این منظور باید 09H را روی ثبات AH و آدرس آفست اولین خانه ی رشته را روی DX قرار دهیم. مثال زیر نمونه ای از کاربرد این تابع را نشان می دهد.

```
Custmsg    DB    'Hello', '$'
...
Mov   AH,09H
LEA   DX,Custmsg
INT   21H
```

با فراخوانی وقفه 21H رشته custmsg که برابر Hello می‌باشد در خروجی چاپ خواهد شد.

نمایش مجموعه کاراکترهای ASCII

مثال زیر کاربردی از وقفه های گفته شده را برای نمایش مجموعه کاراکترهای ASCII را نشان می‌دهد.

```
                                page 60,132
                        TITLE   A09DISAS    (COM)Display ASCII character set
                                .MODEL       SMALL
                                .CODE
0100                            ORG         100H
0100  EB 02             BEGIN: JMP         SHORT A10MAIN
0102  00 24             ASCHAR DB          00,'$'                ;Display character
                        ;
                        ;                     Main procedure
                        ;                    -----------------


0104                    A10MAIN     PROC          NEAR
0104  E8 0112 R                    CALL          B10SCRN       ;Clear screen
0107  E8 0120 R                    CaLL          C10CURS       ;Set cursor
010A  E8 012A R                    CALL          D10DISP       ;Display characters
010D  B8 4C00                      mov           AX,4C00H      ;END
0110  CD 21                        INT           21H           ;processing
0112                    A10MAIN     ENDP
                        ;
                        ;                     Clear screen:
                        ;                     -------------
0112                    B10SCRN     PROc          NEAR
0112  B8 0600                      MOV           AX,0600H      ;Scroll full screen
0115  B7 07                        MOV           BH,07         ;Attribute:white on black
0117  B9 0000                      MOV           CX,0000       ;Upper left  location
011A  BA 184F                      MOV           DX,184FH      ;Lower right location
011D  CD 10                        INT           10H           ;Call BIOS
011F  C3                           RET                         ;Return to caller
0120                    B10SCRN     ENDP
                        ;                     Set cursor to 00,00
                        ;                     --------------------
0120                    C10CURS     PROC          NEAR
0120  B4 02                        MOV           AH,02H        ;Reguest set curser
0122  B7 00                        MOV           BH,00         ;Pagevnumber 0
0124  BA 0000                      MOV           DX,0000       ;Row0,column 0
0127  CD 10                        INT           10H           ;Call BIOS
0129  C3                           RET                         ;Return to caller
012A                    C10CURS     ENDP
                        ;                     Display ASCII characters:
                        ;                     -------------------------
012A                    D10DISP     PROC
012A  B9 0100                      MOV           CX,256        ;Initialize 256 atentions
012D  8D 16 0102 R                 LEA           DX,ASCHAR     ;Initialize address of ASCHAR
0131                    D20:
0131  B4 09                        MOV           AH,09H        ;Display ASCII ASCHAR
0133  CD 21                        INT           21H
0135  2E: FE 06 0102 R             INC           ASCHAR        ;Increment for next character
013A  E2 F5                        LOOP          D20           ;Decrement CX,loop nonzero
013C  C3                           RET                         ;Return to caller
013D                    D10DISP     ENDP
                                    END           BEGIN
```

نمایش مجموعه کاراکتر های ASCII

۲-۹- تابع OAH از وقفه 21H برای ورودی صفحه کلید

برای اینکه بتوان از این تابع استفاده کرد باید ابتدا در بخش داده‌ای لیست پارامترهای مورد نیاز آن را مشخص کنیم. در زیر نمونه‌ای از تعریف این لیست پارامترها نشان داده شده است :

```
Paralist  Label  Byte
Maxlen    DB     20
Actlen    DB     ?
KBData    DB     20 Dup(' ')
```

در این لیست Maxlen مشخص کننده حداکثر تعداد کاراکترهایی است که باید از صفحه کلید خوانده شود. Actlen پس از اجرای وقفه مشخص کننده تعداد کاراکترهایی می‌باشد که از کار برگرفته شده. KBData کد کاراکترهای گرفته شده را نشان می‌دهد. حداقل اندازه این آرایه باید برابر Maxlen باشد.

هنگام فراخوانی این وقفه در ثبات AH مقدار 0AH و آدرس آفست لیست پارامترها را در روی DX قرار خواهیم داد. نحوه فراخوانی بصورت زیر خواهد بود:

```
MOV  AH , 0AH
LEA  Dx , Paralist
INT  21H
```

برنامه پذیرش و نمایش نام‌ها

برنامه شکل زیر از کاربر میخواهد تا یک نام را وارد کند و سپس نام را در مرکز صفحه نمایش چاپ خواهد کرد و بلندگو به صدا درمی‌آید.

```
        PAGE 60,132
TITLE AO9CTRNM(EXE) ACCeptname , centeron screen
;-----------------------------------------
        .MODEL SMALL
        .Stack 64
;-----------------------------------------
        .DATA
PARALIST LABEL BYTE              ; name parameter list.
MAXNLEN  DB    20                ; maximam lengtho fnam
ACTULEN  DB    ?                 ; noofcharactes enteres
KBNAME   DB    21 Dup(' ')       ; entere name
PROMPT   DB    'Name?', '$'
;-----------------------------------------
      .code
      .386
A10MAIN   PROC  FAR
      MOV   AX , @data          ; initialized segment
      MOV   DS , AX             ; registers
      MOV   ES , AX
      CALL  Q10CLR              ; clear screen
A20LOOP:
      MOV   DX , 0000           ; set cursor to 00,00
      CALL  Q20curs
      CALL  B10PRMPT            ; Display promt
      CALL  C10INPT             ; provide for inpat of name
      CALL  Q10CLR              ; clear screen
      CMP   ACTULEN,00          ; name entered?
      JE    A30                 ;   no , exit
      CALL  D10CODE             ; set bell and $
      CALL  E10CENT             ; senter , display name
      JMP   A20LOOP
A30:
      MOV   AX,4C00H            ; End processing
      INT   21H
A10MAIN     ENDP
;               Display prompt :
;               ----------------
B10PRMPT  PROC  NEAR
      MOV   AH,09H              ; Reguest display
      LEA   DX,PROMPT
      INT   21H
      RET
B10PRMPT  ENDP
;               Accept input of name
;               --------------------
C10INPT   PROC  NEAR
      MOV   AH,0AH              ; Request keyboard
      LEA   DX,PARALIST         ;    input
      INT   21H
      RET
C10INPT   ENDP
;                 Set bell and '$' delimiter:
;                 ---------------------------
D10CODE   PROC  NEAR
      MOVzx Bx,ACTULEN          ; Replace 0DH with 07H
      MOV   KBNAME[Bx],07
      MOV   KBNAME[Bx+1],'$'    ; setdisplay delimeter
      RET
D10CODE     ENDP
```

```
;                   Center and display name:
;                   ------------------------
E10CENT   PROC  NEAR
      MOV   DL,ACTULEN        ; locate center column:
      SHR   DL,1              ; dividelengthby2
      NEG   DL                ; revers sign
      ADD   DL,40             ; add 40
      MOV   DH,12             ; center row
      CALL  Q20CURS           ; set cursor
      MOV   AH,09H
      LEA   DX,KBNAME         ; display name
      INT   21H
      RET
E10CENT   ENDP
;                   Clear screen:
;                   -------------
Q10CLR    PROC  NEAR
      MOV   AX,0600H          ; Request scroll screen
      MOV   BH,30             ; color atribute
      MOV   CX,0000           ; From 00,00
      MOV   DX,184FH          ; to 24,79
      INT   10H
      RET
Q10CLR    ENDP
;                   Set carsor row/column :
;                   -----------------------
Q20CURS   PROC  NEAR              ; Dx set on entry
      MOV   AH,02H            ; Request set sursor
      MOV   BH,00             ; page #0
      INT   10H
      RET
Q20CURS   ENDP
      END   A10MAIN
```

برنامه پذیرش و نمایش نامها

در روال E10CENT ، دستور SHR ثبات DL را یک بیت به راست شیفت میدهد که تاثیر آن این است که طول کاراکترهای گرفته شده تقسیم بر ۲ می‌شود. دستور NEG علامت DL را معکوس می‌کند. یعنی یک عدد مثبت به عدد منفی تبدیل خواهد شد. ADD مقدار ۴۰ را با این عدد جمع می‌کند، تا موقعیت ستون شروع چاپ در ثبات DL قرار گیرد. یعنی در حقیقت از ۴۰ نصف طول نام گرفته شده کم می‌شود تا شماره ستون چاپ حاصل شود (نام در وسط صفحه چاپ شود). با تنظیم مکان‌نما در سطر ۱۲، و ستون مشخص شده نام در مرکز صفحه نمایش چاپ خواهد شد. برای مثال ، اگر کاربر نام "Hamid Reza"را وارد کند، برنامه موارد زیر را انجام میدهد :

۱. تقسیم طول ۱۰ بر ۲ (۵=۲÷۱۰)
۲. این مقدار از ۴۰ کم می‌شود (۳۵ = ۵ – ۴۰)
۳. نام در سطر ۱۲ وستون حاصل (۳۵) چاپ خواهد شد.

تازمانی که کاربر نامی را وارد کند این برنامه آنرا گرفته و در وسط صفحه نمایش چاپ خواهد کرد. به محض اینکه کاربر یک رشته خالی را وارد کند این برنامه خاتمه پیدا خواهد کرد

فصل دهم

۱۰- نکات پیشرفته پردازش صفحه کلید

۱-۱۰- صفحه کلید

صفحه کلید سه نوع اصلی کلید دارد

۱. **کلیدهای استاندارد**، شامل حوف A تا Z، شماره های ۰ تا ۹ و برخی کاراکترها مانند %، $، # و ...
۲. **کلیدهای تابعی توسعه یافته** که شامل :
   - کلیدهای تابعی برنامه مانند <F1> و <Shift>+<F1> .
   - کلیدهای عددی با حالت Num Lock خاموش: <Home>، <End>، <Arrows>، <Del>، <Ins>، <Page Up> و <Page Down> و کلیدهایی که در صفحه کلید مانند همین کلیدها عمل می‌کنند.
   - <Alt> + حروف الفبا و کلیدهای تابعی برنامه + <Alt>
۳. **کلیدهای کنترلی** <Alt>، <Ctrl> و <Shift> که در ارتباط با سایر کلیدها عمل می‌کنند.

۲-۱۰- وضعیت شیفت صفحه کلید

در موقعیت 40:07H یک بایت قرار گرفته که این بایت اولین بایت وضعیت شیفت صفحه کلید می‌باشد. هریک از بیت‌های این بایت وضعیت کلیدهایی را که در جدول زیر نشان داده شده مشخص خواهند کرد.

Bit	Action	Bit	Action
7	Insert active	3	<Alt> pressed
6	CapsLock sate active	2	<Ctrl> pressed
5	NumLock sate active	1	<Left Shift> pressed
4	Scroll Lock sate active	0	<Right Shift> pressed

بایت دیگری 40:96H قرار دارد که وضعیت برخی دیگر از کلیدهای صفحه را نشان میدهد. عناصری که بر ما اهمیت دارد بیت ۴ است که وقتی ۱ باشد یعنی یک صفحه کلید توسعه یافته نسب شده است.

Bit	Action	Bit	Action
7	Insert pressed	3	Ctrl/Numlock (pause) active
6	Caps Look pressed	2	Sys Reg pressed
5	Num Lock pressed	1	Left Alt pressed
4	Scroll Look pressed	0	Left ctrl pressed

۳-۱۰- بافر صفحه کلید

یک عنصر سودمند در ناحیه داده BIOS در 40:1EH به نام **بافر صفحه کلید** است. وقتی کاربر کلیدی را از صفحه کلید فشار می‌دهد بصورت خودکار 09H از وقفه‌های BIOS اجرا شده کد پیمایشی را که توسط صفحه کلید تولید شده دریافت کرده به کد معادل ASCII تبدیل نموده و به بافر صفحه کلید تحویل می‌دهد. بافر صفحه کلید این کدها را به ترتیب نگهداری می‌کند تا اینکه وقفه‌ای (مانند وقفه 16h) از آن کلیدی را تقاضا کند. سپس بافر به ترتیب این کدها را برگشت می‌دهد.

۴-۱۰- تابع 01H از وقفه 21H : ورودی صفحه کلید با انعکاس

از این تابع برای خواندن کلید از صفحه استفاده می‌شود. اگر در بافر صفحه کلید، کدی وجود داشته باشد آنرا برگشت میدهد وگرنه منتظر فشار دادن کلید از کاربر می‌ماند. پس از فشار دادن کلید کد کلید فشار داده شده روی AL برگشت داده می‌شود.

این تابع توانایی خواندن کلیدهای تابعی توسعه یافته (کلیدهای دو کده) را دارا می‌باشد. اگر کلید معمولی فشار داده شده باشد که اسکی آن بر روی AL برگشت داده می‌شود. اما اگر کلید تابعی توسعه یافته فشار داده شده باشد بر روی AL مقدار صفر برگشت داده خواهد شد. برای خواندن کد دوم باید این وقفه دوباره فراخوانی شود. مثال زیر نمونه‌ای از کاربرد این وقفه را نشان می‌دهد:

```
MOV   AH,01H      ; درخواست ورودی صفحه کلید
INT   21H         ; فراخوانی سرویس وقفه
CMP   AL,00       ; آیا کلید تابعی توسعه یافته فشار داده شده؟
JNZ   …           ; نه کاراکتر ASCII
INT   21H         ; بله تکرار عملیات برای خواندن کد دوم
…                 ; برای کد پیمایش
```

۵-۱۰- تابع 06H از وقفه 21H : I/O کنسول مستقیم

از این تابع برای خواندن کلید از صفحه کلید استفاده می‌شود. این تابع تقریباً شبیه تابع 01H می‌باشد با این تفاوت که پس از تست کردن بافر صفحه کلید اگر کلیدی وجود نداشته باشد دیگر منتظر فشار دادن کلید از طرف کاربر نمی‌ماند. در این حالت این تابع پرچم صفر (ZF) را را برابر یک قرار داده و برای ورودی منتظر نمی‌ماند. اما اگر در بافر صفحه کلید کد وجود داشته باشد، این عملیات کد را در ثبات AL قرار داده و پرچم صفر را برابر صفر قرار میدهد. این عملیات کاراکتر را بر روی صفحه نمایش منعکس نمی‌کند و همچنین کلیدهای <prtsc> + <Ctrl> یا <Ctrl>+<Break> را بررسی نمی‌کند.

۶-۱۰- تابع 07H از وقفه 21H : ورودی مستقیم صفحه کلید بدون انعکاس

این تابع شبیه تابع 01H عمل می‌کند با این تفاوت که کاراکتر را بر روی صفحه منعکس نکرده و کلید + < Ctrl> <Break> واکنش نشان نمی‌دهد.

۷-۱۰- تابع 08H از وقفه 21H : ورودی صفحه کلید بدون انعکاس

این تابع شبیه ۰۱ عمل می‌کند با این تفاوت که کلید فشار داده شده را بر روی صفحه نمایش منعکس نمی‌کند.

۸-۱۰- تابع 0AH از وقفه 21H : ورودی صفحه کلید بافر شده

این تابع در مباحث قبلی توضیح داده شده است.

۹-۱۰- تابع 0BH از وقفه 21H : بررسی وضعیت صفحه کلید

این تابع بافر صفحه کلید را بررسی کرده و اگر در بافر کلیدی در دسترس باشد در روی AL عدد FFH را قرار می‌دهد و در غیر اینصورت (هیچ کلیدی در دسترس نباشد) 00H را قرار خواهد داد.

۱۰-۱۰- تابع 0CH از وقفه 21H : پاک کردن بافر صفحه کلید و برداشتن تابع

این تابع در ارتباط با توابع 01H ، 06H ، 08H یا 0AH استفاده می‌شود .

تابع مورد نیاز در AL قرار می گیرد:

```
MOV   AH,0CH          ; درخواست ورودی صفحه کلید
MOV   AL,fUNCTION     ; تابع مورد نیاز
MOV   DX,ABAREA       ; ناحیه ورودی صفحه کلید
INT   21H             ; فراخوانی سرویس وقفه
```

۱۱-۱۰- تابع 00H از وقفه 16H : خواندن یک کاراکتر

این تابع فقط کلیدهای صفحه کلید ۸۳ تایی را دستکاری می‌کند و ورودی از کلیدهای اضافی بر روی صفحه کلیدهای گسترش یافته مانند <F11> و <F12> نمی پذیرد.

۱۲-۱۰- تابع 02H از وقفه 16H : بازگرداندن وضعیت شیفت جاری

این تابع وضعیت شیفت صفحه کلید را که در ناحیه داده‌ای BIOS در موقعیت 40:17H قرار دارد در روی AL برگشت می‌دهد.

۱۳-۱۰- تابع 05H از وقفه 16H : نوشتن بر روی صفحه کلید

این تابع به ما این اجازه را می‌دهد که بتوانیم کلیدی را در بافر صفحه کلید بنویسیم. همانند اینکه کلیدی فشار داده شده است. کاراکتر ASCII را در CH و کد پیمایش آنرا در CL وارد می‌کنیم. تا زمانی که بافر صفحه کلید پر شود می‌توان این عملیات را تکرار کرد.

فصل یازدهم

۱۱- ماکروها

۱-۱۱- ماکرو

برای هر یک از دستورات اسمبلر یک دستور زبان ماشین تولید میکند.به عبارت دیگر برای هر دستور نوشته شده به یک زبان سطح بالا مانند C یا BASIC کمپایلر ممکن است تعداد زیادی دستورات زبان ماشین تولید کند بدین ترتیب میتوان یک زبان سطح بالا را شامل یک مجموعه از جملات ماکرو در نظر گرفت.

اسمبلر نیز وسایلی برای تعریف ماکرو ها دارد.یک نام منحصر به فرد برای ماکرو با مجموعه ی از دستورات زبان اسمبلی که تولید ماکرو تولید می نماید تعریف میشود

ماکرو ها جهت اهداف زیر مفید می باشند:

۱. ساده سازی و کاهش میزان دستورات تکراری
۲. کاستن خطاهایی که ممکن است در دستورات تکراری پدید ایند
۳. افزایش خوانایی برنامه اسمبلی

مثال هایی از توابعی که ممکن است با ماکرو ها انجام داد عملیات ورودی خروجی است که ثبات ها را بار گذاری نمود و وقفه ها را انجام می دهد تبدیل داده های ASCII به دودویی عملیات محاسباتی چند کلمه ای و روتین های دستکاری رشته ها.

در اینجا تعریف کلی ماکرو امده است:با عبارت ماکرو شروع شده و به عبارت END M تمام می شود.برنامه (۱-۲۲) نمونه ای ازتعریف و استفاده از ۲ ماکرو به نام INTIZو FINISH را نشان میدهد.

۲-۱۱- دو تعریف ماکروی ساده

ابتدا یک تعریف ماکروی ساده که ثبات های سگمنت را برای یک برنامه EXE مقدار دهی می کند بررسی می کند.نام این ماکرو INTZ است اما هر نام منحصر به فرد دیگری نیز قابل پذیرش خواهد بود نامی که در تعریف ماکرو ارجاع میشود –ES,DS,AX,@data باید در جایی از برنامه تعریف شوند یا برای اسمبلر شناخته شوند.

```
INITZ    MACRO              تعریف ماکرو:
          MOV   AX,@data    بدنه:
          MOV   DS,AX       تعریف:
          MOV   ES,AX       ماکرو:
       ENDM                 انتهای ماکرو:
```

ممکن است بعد از دستور ماکرو INITZ در سگمنت کد هر جا که ثبات ها را مقدار دهی می کنید استفاده نمایید.وقتی اسمبلر به دستور ماکرو INITZ می رسد جدولدستورات سمبولیک را پیمایش میکند و در صورت پیدا نکردن یک ورودی دستورات ماکرو را بررسی میکند.چون برنامه شامل یک تعریف از ماکرو INITZ است اسمبلر بدنه تعریف را جایگزین کرده و دستورات مورد نظر جایگزین میشود-بسط ماکرو یک برنامه ممکن است از دستور ماکرو INITZ فقط یک بار استفاده کند.اگرچه ماکرو ها طوری طراحی شده اند که هر تعداد دفعه قابل استفاده اند و هر بار اسمبلر بسط ماکرو را تولید می کند

ماکرو دوم را نیز با نام FINISH تعریف میکنیم.که خروج طبیعی از یک برنامه راتعریف میکند

شکل ۱-۲۲ یک لیست از برنامه اسمبل شده ای است مه ماکرو های INITZو FINISH را تعریف و استفاده میکند,فراهم می سازد این نسخه اسمبلر خاص در کنار بسط ماکرو عدد یک را در سمت چپ هر دستور قرار می دهد تا مبنی بر تولید دستور ماکرو باشد.

```
FINISH  MACRO           تعریف ماکرو:
        MOV   AX,4C00H  درخواست:
        INT   21H       انتهای پردازش:
        ENDM            انتهای ماکرو:
```

```
                                Page   60,132
                            TITLE    A22MACR1  (EXE)  simple    macros
                            ;-------------------------------------------------------
                                   INITZ   MACRO          ;Define   macro
                                      MOV   AX,@data      ;initialize   segment
                                      MOV   DS,AX         ;registers
                                      MOV   ES,AX
                                   ENDM                   ;END MACRO
                            ;-------------------------------------------------------
                                   FINISH        MACRO    ;Define macro
                                      MOV   AX,4C00H      ;End processing
                                      INT   21H
                                   ENDM                   ;End macro
                            ;-------------------------------------------------------
                                  .MODEL    SMALL
                                  .STACK       64
                                  .DATA
0000  54 45 53 54 20 4D       MESSGE  DB  'TEST MACRO INISTRUCTION',13,10,'$'
      41 43 52 4F 20 49
      4E 49 53 54 52 55
      43 54 49 4F 4E 0D
      0A 24

                            ;-------------------------------------------------------
                                  .CODE
0000                          BEGIN  PROC     FAR
                                      INITZ               ;macro inistruction
0000  B8 ---- R             1         MOV   AX,@data      ;initialize   segment
0003  8E D8                 1         MOV   DS,AX         ;registers
0005  8E C0                 1         MOV   ES,AX
0007  B4 09                           MOV  AH,09          ;request display Message
0009  8D 16 0000 R                    LEA  DX,MESSGE
000D  CD 21                           INT  21H
                                      FINISH
000F  B8 4C00               1         MOV   AX,4C00H      ;End processing
0012  CD 21                 1         INT   21H
0014                          BEGIN  ENDP
                                      END      BEGIN
```

شکل ۱-۲۲ دستورات اسمبل شده ساده

۳-۱۱- استفاده از پارامتر ها در ماکروها

برای انعطاف بیشتر یک ماکرو ها میتوان پارامتر هایی در عملوند ها بصورت ارگومان های فرضی تعریف کنید.تعریف ماکروی زیر با نام PROMPT برای استفاده از INT 21H تابع 09H جهت نمایش پیغام می باشد:

```
PROMPT  MACRO    MESSGE          ارگومان فرضی:
        MOV      AH,09H
        LEA      DX,MESSGE
        INT      21H
        ENDM                     انتهای ماکرو:
```

وقتی از این دستور ماکرو استفاده کنید می توانید نام پیغام را طوری قرار دهید که به یک ناحیه داده که با یک علامت دلار خاتمه می یابد ارجاع کنید.

یک ارگومان فرضی در تعریف ماکرو به اسمبلر می گویند میتوانید نام پیغام را طوری قرار دهید که به یک ناحیه داده که با یک علامت دلار خاتمه می یابد ارجاع کنید.

یک ارگومان فرضی در تعریف ماکرو هابه اسمبلر می گویند که این نام را هر نام مشابه در بدنه ماکرو مرتبط کند برای مثال ارگومان فرضی MESSAGE در دستور LEA نیز وجود دارد برنامه یک اعلان با نام MESSAGE 2 تعریف می کند.

```
MESSAGE2    DB    'Enter the data as mm\dd\yy'
```

حال اگر بخواهیم برای نمایش meesage2 از ماکروی prompt استفاده کنیم به شکل زیر عمل خواهد کرد.

```
Prompt    MESSAGE2
```

پارامتر (MESSAGE2) در دستور ماکرو با ارگومان فرضی (MESSAGE) در تعریف ماکرو اصلی مرتبط می شود.

Macro definition:	ROMPT MACRO MESSAGE	(ارگومان)
Macro instruction:	PROMPT MESSAGE 2	(پارامتر)

حال اسمبلر ارگومان تعریف ماکروی اصلی را با عملوند جمله LEA جور می کند حال پترامتر های دستور ماکرو (MESSAGE2) را به جای ارگومان فرضی MESSAGE درتعریف ماکرو جانشین میکند.اسمبلر MESSAGE2 را جانشین MESSAGE در دستور LEA جانشین هر یک از رویداد های دیگر MESSAGE مینماید.

یک ارگومان فرضی ممکن است شامل هر نام معتبر و یک نام ثبات مانندCX باشد.ممکن است یک ماکرو را با هر تعداد از ارگومان های فرضی که با کاما جدا میشود و حداکثر تا سلول ۱۲۰ یک خط هستند تعریف کنید (نسخه اسمبلر بستگی دارد)اسمبلر پارامتر های دستور ماکرو را از چپ به راست و یکی یکی جانشین ارگومان های فرضی در تعریف ماکرو می کند.

۴-۱۱- توضیحات ماکرو

برای توصیفی تر کردن ماکرو ها می‌توان توضیحاتی را در آن به کار برد یک ; یا پیش پردازنده COMMENT منبی بر یک خط توضیح است. مثال زیر برای توضیح از ; استفاده می‌کند :

```
PROMPT    MACRO  MESSAGE
          ;This macro permits a display of messages
          MOV        AH, 09H          درخواست نمایش:
           LEA        DX,MESSAGE
          INT        21H
       ENDM
```

```
                                page 60,132
                        TITLE A22MACR2  (EXE)  Use of parameters
                        ;-----------------------------------------------
                        INITZ   MACRO                ;Define macro
                                MOV     AX,@data     ;Initialize segment
                                MOV     DS,AX        ;Registers
                                MOV     ES,AX
                                ENDM                 ;End macro
                        ;-----------------------------------------------
                        PROMPT MACRO  MESSGE         ;Define macro
                                MOV     AH,09H       ;Request display
                                LEA     DX,MESSGE    ;Prompt
                                INT     21H
                        ENDM                         ;End macro
                        ;-----------------------------------------------
                        FINISH MACRO                 ;Define macro
                                MOV     AX,4C00H
                                INT     21H
                        ENDM
                        ;-----------------------------------------------
                                .MODEL  SMALL
                                .STACK  64
                                .DATA
0000  43 75 73 74 6F 6D           MESSG1  DB  'Customer name?','$'
      65 72 20 6E 61 6D
      65 3F 24
000F  43 75 73 74 6F 6D           MESSG2  DB  'Customer address?','$'
      65 72 20 61 64 64
      72 65 73 73 3F 24
                        ;-----------------------------------------------
                                .CODE
0000                       BEGIN   PROC  FAR
                                INITZ
0000  B8 ---- R            1        MOV     AX,@data     ;Initialize segment
0003  8E D8                1        MOV     DS,AX        ;Registers
0005  8E C0                1        MOV     ES,AX
                                PROMPT  MESSG2
0007  B4 09                1        MOV     AH,09H       ;Request display
0009  8D 16 000F R         1        LEA     DX,MESSG2    ;Prompt
000D  CD 21                1        INT     21H
                                FINISH
000F  B8 4C00              1                MOV     AX,4C00H
0012  CD 21                1                INT     21H
0014                           BEGIN   ENDP
                                END     BEGIN
```

شکل۲-۲۲ استفاده از پارامتر های ماکرو

چون پیش فرض فقط لیست کردن دستوراتی است که کد مقصد تولید میکند اسمبلر بطور خودکار وقتی تعریف ماکرو را بسط میدهد توضیحات را نمایش نمی دهد اگر میخواهید توضیحات ماکرو نیز شوند شبه عمل لیست LALL.(لیست کردن همه شامل نقطه نام)را قبل از تقاضای دستور العمل ماکرو بکار ببرید:

```
.LALL
PROMPT  MESSAGE 1
```

یک تعریف ماکرو میتوانید دارای تعدادی توضیح باشد که ممکن است بخواهید برخی لز انها در بسط ماکرو دیده شوند و برخی ظاهر نشوند.این کار نیز با دستور LALL.عملی است ولی قبل از توضیحاتی که نباید ظاهر شوند یک جفت علامت (;;) قرار دهید پیش فرض اسمبلر XALL است.که سبب میشوند فقط دستوراتی که کد مقصد تولید

میکنند لیست شوند و در نهایت ممکن است نخواهید هیچ کد اسمبلی لیست شوند بخصوص اگر دستور ماکرو چندین بار در برنامه استفاده شده باشد.بدین منظوراز پیش پردازنده SALL.(موقوف گذراندن همه)استفاده کنید که اندازه برنامه چاپ شده را کاهش میدهد لیکن هیچ تاثیری بر اندازه مقصد ندارد.

یک پیش پردازنده لیست گیری تا زمانی که پیش پردازنده لیست دیگری ظاهر نشده است تاثیرش رادر برنامه حفظ میکنند میتوانید انها را در برنامه قرار دهید تا برخی ماکرو ها توضیحات را نمایش دهند.برخی بسط ماکرو را نمایش دهند و برخی دیگر نیز نمایش ندهند.

برنامه ۳-۲۲ خصوصیات فوق را تشریح میکند برنامه ماکرو های INITZ,FINISH,PROMPTکه قبلا توصیح داده شده است.سگمنت کد حائی پیش پردازنده لیست گیری SALL.است که از لیست گیری بسط INITZو FINISH و اولین بسط PROMPT جلوگیری میکند.برای دومین PROMPT پیش پردازنده LALLسبب میشود.تا اسمبلر همه توضیحات و بسط ماکرو را نمایش دهد اما توجه کنید که در تعریف ماکروی PROMPT توضیح بسط ماکرو که علامت (;;)دارد لیست نشده است.

۵-۱۱- استفاده از یک ماکرو داخل یک تعریف ماکرو

یک تعریف ماکرو ممکن است حاوی ارجاع به ماکروی تعریف شده دیگری باشد یکک ماکروی ساده با نام INT 21 که یک تابع را در ثبات AH قرار می دهد و INT 21H را صادر می کند.

```
INT21   MACRO   FUNCTN
        MOV     AH,FUNCTN
        INT     21H
    ENDM
```

برای انکه ماکرو INT 21 را استفاده کنید و ورودی را از صفحه کلید بپذیرید.دستورات زیر را به کار برید :

```
LEA     DX,NAMEPAR
INT21   0AH
```

برای تولید کد برای INT 21 تابع 0AH را در AH قرا دهید وint 21h را برای ورودی صفحه کلید صادر کنید حال فرض کنید ماکروی دیگری دارید با نام DISP که INT 21H تابع 02H را در ثبات AH برای نمایش یک کاراکتری قرار دهد:

```
DISP    MACRO   CHAR
        MOV         AH,02H
        MOV         DL,CHAR
        INT         21H
ENDM
```

برای نمایش یک علامت سوال ماکرو را چنین بنویسید ’DISP’?’ اما می توانید برای اخذ مزیت ماکرو INT 21 با اشاره به INT 21 در تعریف DISP تغییرات ایجاد کنید

```
DISP    MACRO   CHAR
        MOV     DL,CHAR
        INT21   02H
ENDM
```

حال اگر ماکرو DISP را به صورت ’?‘ DISP کد نمایید اسمبلرکد زیر را تولید میکند.

```
MOV     D,'?'
```

```
MOV       AH,02H
INT       21H
```

۱-۵-۱۱- پیش پردازنده LOCAL

تعدادی از ماکرو ها به تعریف یک عنصر داده یا برچسب یک دستور نیاز دارد اگر ماکرو را پیش از یک بار در برنامه استفاده کنید اسمبلر عنصر داده یا برچسب را برای هر رخداد را تعریف میکند و یک پیغام خطا یه سبب نسخه های متعدد نام ها تولید میشوند.پیش پردازنده LOCAL بلا فاصله بعد از جمله MACRO حتی قبل از توضیح کد میشود قالب کلی ان چنین است.

یک یا چند ارگومان فرضی: LOCAL dummy1,dummy2,…

```
                                page 60,132
                        TITLE  A22MACR3  (EXE)  Use .LALL  & .SALL
                        ;-------------------------------------------------------
                        INITZ   MACRO                          ;Define macro
                                MOV     AX,@data               ;Initialize segment
                                MOV     DS,AX                  ;Registers
                                MOV     ES,AX
                        ENDM                                   ;End macro
                        ;-------------------------------------------------------
                        PROMPT  MACRO  MESSGE                  ;Define macro
                        ;This macro display any message
                        ;;Generates cod that request display
                               MOV     AH,09H                 ;Request display
                               LEA     DX,MESSGE              ;Prompt
                               INT     21H
                        ENDM                                   ;End macro
                        ;-------------------------------------------------------
                        FINISH  MACRO                          ;Define macro
                               MOV     AX,4C00H               ;End processing
                               INT     21H
                        ENDM
                        ;-------------------------------------------------------
                               .MODEL  SMALL
                               .STACK  64
                               .DATA
0000  43 75 73 74 6F 6D        MESSG1  DB  'Customer name?',10,13,'$'
      65 72 20 6E 61 6D
      65 3F 0A 0D 24
0011  43 75 73 74 6F 6D        MESSG2  DB  'Customer address?',10,13,'$'
      65 72 20 61 64 64
      72 65 73 73 3F 0A
      0D 24
                        ;-------------------------------------------------------
                               .CODE
0000                    BEGIN   PROC  FAR
                               .SALL
                               INITZ
                               PROMPT  MESSG1
                               .LALL
                               PROMPT  MESSG2
                      1 ;This macro display any message
                      1 ;
000F  B4 09                     1          MOV     AH,09H         ;Request display
0011  8D 16 0011 R              1          LEA     DX,MESSG2      ;Prompt
0015  CD 21                     1          INT     21H
                               .SALL
                               FINISH
001C                    BEGIN   ENDP
                               END     BEGIN
```

شکل ۳-۲۲ لیست کردن و جلوگیری از بسط ماکرو

۶-۱۱- مشمول نمودن ماکرو ها از یک کتابخانه

تعریف ماکرو هایی مانند INITZ,FINISH,PROMPT و استفاده از انها فقط یک بار در برنامه ممکن است حاصلی نداشته باشد یک روش بهتر ثبت نام تمام ماکرو ها در یک کتابخانه دیسک تحت یک نام مانند MACRO.LBY است.باید به راحتی همه تعاریف ماکرو را در یک یا چند فایل روی دیسک ذخیره کنید.

```
INITZ   MACRO
        ...
        ENDM
        ...
PROMPT  MESSGAE  MACRO
        ...
        ENDM
```

میتوانید از یک پیرایشگر یا پردازشگر کلمات برای نوشتن در فایل استفاده کنید اما باید یک فایل ASCII قالب بندی نشده باشد.در مثال زیر فرض شده است که فایل با نام MACRO.LBY روی درایو :F ذخیره شده است.برنامه شما میتواند از هر یک از ماکرو های ثبت شده استفاده کنید اما به جای درج دستورات MACRO در شروع برنامه از پیش پردازنده INCLUDE استفاده کنید: INCLUDE F:\MACRO.LBYاسمبلر فایل MACRO.LBY. را روی درایو :F دستیابی میکند و تمامی تعاریف ماکروی ثبت شده در ان شامل برنامه خوهند شد.اگر چه برنامه شما ممکن است به بعضی از انها نیاز داشته باشد لیست اسمبل شده حاوی یک کپی از تعاریف ماکرو است.که با حرف C در ستون ۳۰ فایلLST. در برخی از نسخه های اسمبلی مشاهده میشود.

چون برخی اسمبلر ها مستلزم عملیات عملیات دو مرحله ای هستند می توانید با استفاده از از دستورات زیر کاری کنید که از INCLUDE فقط در مرحله اول استفاده کنید.(بجای هر دو مرحله)

```
IF1
        F:\MACRO.LBY  INCLUDE
ENDIF
```

IF1 و ENDIF پیش پردازنده های شرطی هستند .IF1 به اسمبلر می گوید تا کتابخانه را فقط در گذر اول فرایند اسمبلی دستیابی کند.پیش پردازنده ENDIF منطقIF راخاتمه می دهد

۱-۶-۱۱- پیش پردازنده PURGE

اجرای یک دستور INCLUDEسبب میشود که اسمبلر تمام تعاریف ماکروی موجود در کتابخانه را در برنامه قرار دهد فرض کنید که یک کتابخانه حاوی ماکرو های INITZ,FINISH,PROMPT,DIVIDE است.اما برنامه فقط INITZو FINISH را نیاز دارد.پیش پردازنده PURGE به شما توان حذف ماکرو های نا خواسته PROMPT و DIVIDE را می دهد.

```
IF1
INCLUDE         D:\MACRO.LBY        کتابخانه کامل:
ENDIF
PURGE           PROMPT,DIVIDE       حذف ماکرو های نا خواسته:
...
INITZ   ...                         استفاده از ماکروهای باقیمانده:
```

۲-۶-۱۱- پیش پردازنده تکرار

پیش پردازنده های تکرار سبب میشوند تا اسمبلر بلوکی از احکام را که به وسیله جمله ENDM خاتمه می یابد تکرار کند.MASM6.0 عبارت های REPEAT,FORCوFOR را برای REPT,IRP,IPRC معرفی میکند.این پیش پردازنده ها در یک تعریف MACRO نباید باشد ولی اگر وجود داشته باشد باید یک ENDM برای خاتمه هر پیش پردازنده تکرار قرار دهید.و یک ENDM دیگر برای خاتمه تعریف MACRO بگذارید.

پیش پردازنده تکرار:REPT

پیش پردازنده REPT سبب می شود تا بلوکی از جملات حداکثر تا ENDM بر اساس تعداد دفعات ورودی عبارت تکرار شود :

REPT expression

اولین مثال دستور dec را چهار بار تولید می کند:

```
REPT    4
   DEC      SI
ENDM
```

۳-۶-۱۱- IRP:پیش پردازنده تکرار نا محدود

پیش پردازنده IRPC سبب میشود تا اسمبلر یک بلو ک از دستورات تا ENDM را تکرار کند شکل کلی ان چنین است:

IRP dummy,<argument>

اگر ارگومانهای دستور میتواند هر تعداد نماد های معتبرو رشته ای وعددی یا ثابت حسابی باشد اسمبلر بلوکی از کد را برای هر ارگومان تولید میکند

۴-۶-۱۱- IRPC:پیش پردازنده تکرار نامحدود کاراکتری

پیش پردازنده های IPRC یا (FORC) سبب تکرار بلوکی از احکام تا ENDM میشود شکل کلی ان در زیر نشان داده شده است.

IRPC dummy,string

۵-۶-۱۱- پیش پردازنده های شرطی

پیش پردازنده های اعمال شرطی اکثرا برای استفاده در داخل تعریف یک ماکرو مفیدند لیکن برای این منظور محدود نمی شوند.هر پیش پردازنده IF باید دارای یک END IF متناظر باشد شرط تست شونده را خاتمه دهد.یک ELSE اختیاری یک عمل متفاوتی را مهیا می سازد در اینجا شکل کلی خانواده IF از پیش پردازنده های شرطی اورده شده است":

IFXX (شرط)
..... شرطی
ELSE
.... بلوک
ENDIF (انتهای IF)

اگر شرط بررسی شده درست باشد اسمبلر بلوک شرطی را تا ELSE و اگرELSE نباشد حداکثر تا ENDIF اجرا میکند اگر شرط غلط باشد اسمبلر بلوک شرطی را بعد از ELSE اجرا میکند .اگر ELSE نباشد هیچ بلوک شرطی را تولید نمی‌کند.

-عبارت IF:اگر اسمبلر عبارت را غیر صفر ارزیابی کند دستورات داخل بلوک شرطی را اسمبل میکند
-عبارت IFE:اگر اسمبلر عبارت صفر را ارزیابی کند دستورات داخل بلوک شرطی را اسمبل میکند
-(عبارت نیست)IF1:اگر اسمبلر گذر اول را پردازش میکند روی دستورات داخل بلوک شرطی را عمل میکند.
-(عبارت نیست)IF2:اگر اسمبلر عبارت دوم را پردازش کند روی دستورات داخل بلوک شرطی را عمل میکند.