

به نام خدا

*Assembly  
Programming*

## برنامه نویسی به زبان اسمبلی برای کامپیوترهای شخصی

### حافظه داخلی

دو نوع حافظه داخلی روی کامپیوتر عبارتند از: حافظه با دستیابی تصادفی (RAM) و حافظه فقط خوانی (Ram) بایت های حافظه به ترتیب با شروع از 00 شماره گذاری می شوند به طوری که هر مکانی دارای یک آدرس منحصر به فردی می باشد

ROM: شامل یک تراشه حافظه بخصوصی است که فقط می تواند خوانده شود. چون دستورالعمل ها و داده ها به طور دائمی در داخل تراشه ها جایگزاری شده اند نمی توانند تغییر داده شوند. سیستم ورودی - خروجی پایه (BIOS) از آدرس 68K شروع شده و دستگاه های ورودی / خروجی نظیر کنترل کننده دیسک سخت را مدیریت می کند. حافظه فقط خوانی که از 960K شروع می شود عملیات اساسی کامپیوتر نظیر امتحان کردن خود کامپیوتر در موقع روشن شدن، الگوهای نقطه ای گرافیکی و بار کننده خود دیسک را کنترل می کند. وقتی که کامپیوتر را روشن می کنید، حافظه ROM کنترل های گوناگونی را انجام داده و اطلاعات مخصوص سیستم را از دیسک به حافظه RAM بار گذاری می کند.

RAM: یک برنامه نویس اصولاً با حافظه RAM ارتباط دارد که بهتر بود در حافظه خواندن - نوشتن نام گذاری می شد. حافظه RAM به عنوان یک برگه کاری برای ذخیره سازی موقت و اجرای برنامه ها بکار می رود چون در موقع خاموش شدن کامپیوتر حافظه RAM (محتوی) از بین می رود باید از حافظه خارجی برای نگهداری برنامه ها و داده ها استفاده کنیم. وقتی که کامپیوتر را روشن می کنید، روال راه اندازی ROM بخشی از سیستم عامل را به داخل RAM بار گذاری می کند. سپس می توانید انجام عملیاتی نظیر بار گذاری یک برنامه از دیسک به داخل RAM را از آن درخواست کنید.

آدرس دهی داده ها در حافظه

بسته به مدل، پردازنده می تواند در هر زمان به یک یا چند بایت از حافظه دسترسی داشته باشد عدد دهدهی 1315 را در نظر بگیرید نمایش هگزا دسیمال آن 0526H است نمایش آن به دو بایت یا یک کلمه از حافظه نیاز دارد. آن شامل با ارزش ترین بایت یعنی 05 و کم ارزش ترین بایت یعنی 29 می باشد.

سیستم داده را در حافظه به ترتیب معکوس ذخیره می کند: بایت دارای کم ارزشتر در آدرس پایین تر و بایت دارای ارزش بیشتر در آدرس بالاتر حافظه ذخیره می شود.

پردازنده انتظار دارد که داده های عددی در حافظه به ترتیب معکوس قرار گیرند و داده ها را مطابق با آن پردازش می کند .  
سگمنت ها و آدرس دهی.

یک سگمنت ناحیه خاصی است که در یک برنامه تعریف شده واز مرز پاراگراف شروع می شود یعنی از مکانی که همواره بر ۱۶ یا ۱۰ شانزده شانزدهی بخش پذیر است. هر چند که یک سگمنت می تواند در هر جایی از حافظه قرار بگیرد و در حالت حقیقی می تواند تا 64k باشد ولی نقطه به اندازه ای فضا لازم دارد که برنامه برای اجرا شدن به آن نیاز دارد.

ممکن است هر تعداد سگمنت وجود داشته باشد ، برای آدرس دهی سگمنت مورد نظر فقط لازم است که آدرس موجود در یک ثبات سگمنت مناسبی را تغییر داد. ۳ سگمنت اصلی عبارتند از سگمنت های داده ، پشته و کد

## سگمنت کد

سگمنت کد شامل دستورالعمل های ماشین است اجرا می شوند. معمولاً اولین دستورالعمل اجرایی در شروعاتین سگمنت قرار دارد. سیستم عامل به آن مکان پیوند داده می شود تا اجرای برنامه را شروع کند همانطور که از نام آن پیداست ، ثبات سگمنت کد (CS) ، سگمنت کد را آدرس دهی می کند. اگر ناحیه کد شما به بیش از 64k نیاز داشته باشد ، برنامه ممکن است نیاز داشته باشد که بیش از یک سگمنت کد را تعریف کند.

## سگمنت داده ها

سگمنت داده ها شامل داده ها ، ثابت ها و نواحی کاری یک برنامه می باشد ثبات سگمنت داده ها (DS) سگمنت داده ها را آدرس دهی می کند . "اگر ناحیه داده به بیش از 64k نیاز داشته باشد در برنامه می توانید بیش از یک سگمنت داده را تعریف کنید ."

## سگمنت پشتتر

به زبان ساده ، پشتتر حاوی هر گونه داده و آدرسی است که برای ذخیره موقت و یا جهت استفاده توسط زیر روال های فراخوانی شده لازم است . ثبات سگمنت پشتتر (SS) سگمنت پشتتر را آدرس دهی می کند.

## حدود یک سگمنت

یک ثبات سگمنت حاوی آدرس شروع یک سگمنت می باشد همانطور که قبلاً ذکر شد یک سگمنت بر روی مرز پارا گراف شروع می شود که آدرسی است که همواره بر ۱۶ تقسیم پذیر است یک سگمنت داده را از مکان حافظه 088EOH شروع می شود در نظر بگیرید چون در این مورد و تمام موارد دیگر راستی ترین رقم شانزده شانزدهی برابر صفر است طراحان کامپیوتر تشخیص دادند که ذخیره کردن رقم صفر در ثبات سگمنت لازم نیست . لذا 038EOH به صورت 038E ذخیره می شود که صفر سمت راست آن بدیهی است. در این کتاب هر جا لازم باشد از گروه ها برای اشاره به سمت راست ترین صفر ما نند [0]38 استفاده خواهد شد.

## آفست های سگمنت

در داخل یک برنامه مکان های حافظه نسبت به آدرس شروع سگمنت بیان می شوند فاصله بایستی از آدرس سگمنت تا یک مکان دیگر در داخل همان سگمنت به صورت یک آفست (جا بجایی) بیان می شود. محدوده یک آفست ۲ بایتی می تواند از 0000H تا FFFFH یا از صفر تا 65,535 باشد. بنابراین اولین بایت سگمنت که در آفست 00، دومین بایت در آفست 01 الی آخر تا آفست 65535 قرار می گیرند برای مراجعه به هر مکان حافظه در یک سگمنت، پردازنده آدرس سگمنت موجود در یک ثبات سگمنت را با یک مقدار آفست ترکیب می کند.

یک سگمنت داده را که از مکان 038EOH شروع می شود در نظر بگیرید. ثبات DS حاوی آدرس سگمنت داده ها یعنی 038E[0]H می باشد و یک دستور العمل به مکانی با آفست 0032H بایت در داخل سگمنت داده ها مراجعه می کند.

بنا بر این مکان حافظه واقعی بایت مراجعه شده به وسیله این دستورالعمل برابر 03912H می باشد. توجه داشته باشید که یک برنامه ممکن است دارای یک یا چند سگمنت باشد که می توانند از هر مکانی از حافظه شروع شده، اندازه آنها متفاوت بوده و به هر ترتیبی قرار بگیرند.

## ظرفیت آدرس دهی

پردازنده های گوناگون انیتس که توسط کامپیوترهای PC مورد استفاده قرار می گیرد دارای قابلیت های آدرس دهی مختلفی می باشند. آدرس دهی 8086/8088: ثبات آنها به احوال ۱۶ بیت می باشند چون یک آدرس سگمنت بر روی مرز پاراگراف (همواره بر ۱۶ تقسیم پذیر است) قرار دارد ۴ بیت سمت راست آدرس آن صفر می باشند همانطور که قبلاً بحث شد یک آدرس سگمنت در ثبات سگمنت ذخیره شده و پردازنده ۴ بیت سمت راست را به صورت شانزده شانزدهی nnnn[0] برابر 0 در نظر می گیرید. حال، FFFF[0]H امکان آدرس دهی تا ۱۰۴۸۵۶۰ بایت را فراهم می کند، اگر

مطمئن نیستید، هر F شانزده شانزدهی را به صورت 1111 دو رویی نوشته و بیت سمت راست را برابر 0 قرار داده و سپس مقادیر بیت ها را با هم جمع کنیم.

آدرس دهی 80286: در حالت حقیقی، پردازنده ۸۰۲۸۶ عمل آدرس دهی را مثل ۸۰۸۶ انجام می دهد در حالت محافظت شده، پردازنده از ۲۴ بیت برای آدرس دهی استفاده می کند، لذا FFFFF[0] امکان آدرس دهی تا ۱۶ میلیون بایت را فراهم می کند. ثبات های سگمنت به عنوان انتخابگرهای دستیابی به آدرس سگمنت ۲۴ بیتی از حافظه عمل کرده و این مقدار را با یک آفست ۱۶ بیتی جمع می کنند.

آدرس پنتیوم / 80386/486: در حالت حقیقی این پردازنده ها عمل آدرس دهی را خیلی شبیه ۸۰۸۶ انجام میدهند. در حالت محافظت شده این پردازنده ها از ۴۸ بیت برای آدرس دهی استفاده می کنند. که امکان آدرس دهی سگمنت ها را تا ۴ میلیون بایت فراهم می کنند. ثبات های سگمنت ۱۶ بیتی به عنوان انتخاب کننده دسترسی به آدرس سگمنت ۳۲ بیتی از حافظه عمل کرده و این مقدار را به یک آفست ۳۲ بیتی اضافه می کنند.

ثبات ها

ثبات های پردازنده برای کنترل دستور العمل های اجرایی، مدیریت آدرس دهی حافظه و انجام محاسبات بکار می روند. ثباتها با استفاده از نام آنها آدرس دهی می شوند مانند SS,DS,CS بیت های یک ثبات به طور قراردادی از راست به چپ و با شروع از صفر شماره گذاری می شوند.

ثبات های سگمنت

یک ثبات سگمنت به طول ۱۶ بیت می باشد و برای آدرس دهی ناحیه ای از حافظه که به عنوان سگمنت جاری شناخته می شود بکار می رود چون یک سگمنت روی مرز پاراگراف تنظیم می شوند، بیت سمت راست آدرس آن در ثبات سگمنت برابر 0 در نظر گرفته می شوند.

ثبات های CS: حاوی آدرس شروع سگمنت که در یک برنامه است. این آدرس سگمنت بعلاوه یک مقدار آفست در ثبات اشاره گر دستورالعمل (Ip) آدرس دستورالعملی را که برای اجرا واکنشی می شود نشان می دهد. برنامه نویسی معمولی مراجعه به ثبات CS لازم نیست.

ثبات DS: حاوی آدرس شروع سگمنت یک برنامه است. دستورالعمل ها برای مراجعه به داده ها از این آدرس استفاده می کنند. این آدرس بعلاوه یک مقدار آفست در داخل یک دستورالعمل واقعی مکانی در سگمنت داده ها را مشخص می کند.

ثبات SS: امکان پیاده سازی پشته ای را که یک برنامه برای ذخیره کردن موقت آدرس ها و داده ها استفاده می کند. سیستم، آدرس شروع سگمنت بعلاوه یک مقدار آفست در ثبات اشاره گر پشته (sp) کلمه جاری در پشته را که آدرس دهی می شود، نشان می دهد. در برنامه نویسی معمولی لازم نیست مستقیماً به ثبات SS مراجعه کنید.

ثبات ES: توسط برخی عملیات رشته ای برای مدیریت آدرس دهی حافظه بکار می رود در اینجا ثبات ES (سگمنت اضافی) با ثبات DI (ثبات اندیس) مرتبط است در برنامه ای که لازم است از ES استفاده شود با یک آدرس سگمنت مناسبی مقدار دهی اولیه می شود.

ثبات های FS,GS: ثبات های سگمنت اضافی کمکی روی پردازنده های ۸۰۳۸۶ و بعد از آن. ثباتهای اشاره گر

سه ثبات اشاره گر عبارتند از: BP,SP,IP

ثبات اشاره گر دستورالعمل: (IP) ثبات ۱۶ بیتی IP حاوی آدرس آفست دستورالعمل ۵ بعدی است که اجرا خواهد شد IP با ثبات CS مرتبط می باشد. به این معنی که ثبات IP دستورالعمل جاری را در سگمنت که اجرایی جاری نشان می دهد. معمولاً در یک برنامه به ثبات IP مراجعه نمی کنید ولی می توانید در موقع استفاده از برنامه DEBV6 برای امتحان کردن یک برنامه مقدار آن را تغییر دهید. ۸۰۳۸۶ و پردازنده های بعد از آن دارای یک ثبات IP توسعه یافته ۳۲ بیتی به نام EIP می

باشند. در مثال ، ثبات CS حاوی H [0]39B4 و IP حاوی H 514 می باشد برای پیدا کردن دستورالعمل اجرای بعدی ، پردازنده آدرس موجود در CS را با افسست موجود در IP جمع می کند.

$39B40H+514H=3A054H$  آدرس سگمنت در CS بعلاوه آدرس افسست در IP آدرس دستورالعمل بعدی.

ثباتهای SP (اشاره گربه پشته) و BP (اشاره گر ...): با ثبات SS مرتبط بوده و به سیستم امکان دستیابی به داده های موجود در سگمنت پشته را می دهند.

ثبات اشاره گر پشته (SP): ثبات SP به طول ۱۶ بیت بوده که شامل یک مقدار افسست می باشد و موقعی که این مقدار با محتوی ثبات SS مرتبط می شود به کلمه بالای پشته اشاره می کند پردازنده ۸۰۳۸۹ و پس از آن دارای یک اشاره گر پشته ۸۲ بیتی توسعه به نام ثبات ESP می باشند.

سیستم به طور اتوماتیک این ثبات ها را مدیریت می کند.

ثبات اشاره گر مبنا (BP): ثبات ۱۶ بیتی BP ارجاع به پارامترهای که داده ها و آدرسهای هستند که یک برنامه از طریق پشته ارسال می کند، تسهیل می کند. ۸۰۳۸۶ و پردازنده های بعد از آن دارای یک ثبات ۳۲ بیتی توسعه به نام EBP می باشند.

ثبات های همه منظوره

ثبات های همه منظوره AX, BX, CX, DX مبنای کار سیستم هستند . در اینکه می توانید آنها را به عنوان کلمه یا به صورت بایت آدرس دهی کنید منحصر به فرد هستند. سمت چپ ترین بایت قسمت بالایی و سمت راست ترین قسمت پایینی می باشند برای مثال AX شامل یک بخش بالایی AH و یک بخش پایینی AL می باشد و می توانید به هر بخش به نام آن مراجعه کنید. ۸۰۳۸۶ و پردازنده های بعد از آن از تمام ثباتهای همه منظوره بعلاوه نسخه توسعه یافته ۳۲ بیتی آنها یعنی EDX, ECX, EBX, EAX پشتیبانی می کنند.

دستورالعملهای زیر عدد 0 را به ثباتهای ECX, BX, AX انتقال می دهند:

MOV AX,00 2/MOV BH,00 3/MOV ECX,00

ثبات AX: انباره اصلی، در عملیاتی که مستلزم ورودی / خروجی و محاسبات زیاد هستند به کار می روند. برای مثال دستورالعملهای ضرب، تقسیم و ترجمه از AX استفاده کنند که کار آمدتری تولید می کنند.

ثبات BX: از آنجا که این ثبات تنها ثبات همه منظوره ایست که می تواند بعنوان یک اندیس برای توسعه آدرس دهی به کار رود به عنوان ثبات... معروف است یکی دیگر از کار بردهای معمول BX بکار گیری آن در انجام محاسبات است.

ثبات CX: CX به عنوان ثبات شمارش معروف است. ممکن است حاوی مقداری برای کنترل تعداد دفعات تکراری یک حلقه یا مقداری برای انتقال بیت ها به سمت چپ یا راست باشد، از CX می توان برای انجام محاسبات نیز استفاده کرد.

ثبات DX: DX به عنوان ثبات داده معروف است. برخی از عملیات ورودی / خروجی نیاز به این ثبات دارند و عملیات ضرب و تقسیم که با اعداد بزرگ سروکار دارند از زوج ثباتهای AX, DX با هم استفاده می کنند.

می توانید از هر یک از این ثباتهای همه منظوره برای جمع و تفریق مقادیر ۸ بیتی، ۱۶ بیتی و ۳۲ بیتی استفاده کنید.

ثبات های اندیس:

ثباتهای DI, SI برای آدرس دهی شاخص دار و استفاده در جمع و تفریق بکار می روند.

ثبات SI: ثبات اندیس منبع ۶ بیتی در برخی از عملیات رشته ای مورد نیاز می باشد در این مورد ثبات SI با ثبات DS مرتبط می باشد ۸۰۳۸۶ و پردازنده های پس از آن از یک ثبات توسعه یافته ۳۲ بیتی به نام ESI پشتیبانی می کنند.

ثبات DI: ثبات اندیس مقصد ۱۶ بیتی نیز برای انجام برخی عملیات رشته ای مورد نیاز می باشد. در این مورد، DI با ثبات ES مرتبط می باشد.

## ثبات نشانه ها (Flags)

نه بیت از ۱۶ بیت ثبات نشانه ها در تمام پردازنده های خانواده ۸۰۸۶ مشترک می باشد وضعیت جاری کامپیوتر و نتایج حاصل از پردازش را نشان می دهند بسیاری از دستورالعمل های مقایسه ای و حسابی وضعیت **flag** ها را تغییر داده و برخی از دستورالعمل ها نیز برای تعیین عمل بعدی می توانند این نشانه ها را امتحان کنند.

**OF** (نشانه سرریزی) سرریزی بیت مرتبه بالایی (سمت چپ ترین) را پس از عمل حسابی نشان می دهد.

**DF** (نشانه جهت) سمت چپ یا راست را برای انجام عملیات انتقال یا مقایسه داده های رشته ای معین می کند.

**IF** (نشانه وقفه) نشان می دهد که تمام وقفه های خارجی نظیر ورودی از صفحه کلید باید مورد پردازش قرار گرفته و یا صرف نظر شود.

**TF** (نشانه تله) به پردازنده اجازه عملیات را در حالت تک مرحله ای می دهد. برنامه های اشکال دار نظیر **DEBVG** این **Flag** را برابر یک می کشد تا بتوانند برای امتحان کردن تاثیرات یک دستورالعمل بر روی ثبات ها و حافظه، آن دستورالعمل را به تنهای اجرا کنند.

**SF** (نشانه علامت) حاوی علامت نتیجه یک عمل حسابی است (0 = مثبت و 1 = منفی)

**ZF** (نشانه صفر) نتیجه یک عمل حسابی یا مقایسه ای را نشان می دهد (ZF=0 یعنی نتیجه غیر صفر بوده و ZF=1 یعنی نتیجه برابر صفر می باشد)

(نشانه رقم نقلی کمکی) حاوی یک رقم نقلی خروجی از بیت شماره ۳ از یک داده ۸ بیتی در

**AF** یک عمل حسابی ویژه می باشد.

**PF** (نشانه توازن) توازن زوج یا فرد هشت بیت پایین نتیجه یک عمل ثابت را نشان می دهد.

**CF** (نشانه رقم نقلی) حاوی رقم نقلی از یک بیت بالای (سمت چپ ترین بیت) بعد از انجام یک عمل حسابی بوده و نیز حاوی آخرین بیت عملیات انتقال یا دوران می باشد.

Flag ها در ثبات نشانه ها در مکانهای زیر قرار دارند.

Flag های که در برنامه نویسی زیاد به کار می روند عبارتند از: CF,ZF,SF,OF برای انجام

عملیات مقایسه و حساب و DF برای تطبیق جهت عملیات رشته ای .

۸۰۲۸۶ وپردازنده های بعد از آن دارای flag های می باشند که در عملیات داخلی پردازنده بکار

می روند و با حالت محافظت شده کامپیوتر مرتبط می باشند .



## فصل دوم

### ویژگیهای سیستم عامل

سیستم عامل دسترسی کلی و مستقل از دستگاه به منابع یک کامپیوتر برای دستگاههای نظیر صفحه کلید صفحه نمایش و دیسک گردان را در اختیار شما قرار می دهد . استقلال از دستگاه بدین معنی است که مجبور نیستید که دستگاهها را به طور مشخص آدرس دهی کنید زیرا سیستم می تواند عملیات I/O را در سطح دستگاه ، بدون نیاز به برنامه ای که این عمل را درخواست کرده است ، مدیریت کند. از بین توابع DOS که در این کتاب مورد نظر می باشند می توان موارد زیر را نام برد.

مدیریت فایل: Dos: فهرستها و فایل ها را روی دیسک های سیستم نگاه داری می کند .برنامه های فایلها را ایجاد کرده و تغییر می دهند ولی سیستم مسئولیت مدیریت مکان آنها را در روی دیسک به عهد دارد .

ورودی /خروجی :برنامه ها داده های ورودی را از سیستم درخواست نموده ویا چنین داده های را با استفاده از وقفه ها وارد سیستم می کند .برنامه نویس در سطح پایین آسوده می باشد. بار کردن برنامه :وقتی که یک کار بر یا یک برنامه، درخواست اجرای یک برنامه را می کند بار کننده برنامه مراحل را که شامل دسترسی به برنامه از دیسک ، قرار دادن آن در حافظه و مقدار دهی اولیه آن برای اجرا می باشند ، مدیریت می کند .

مدیریت حافظه: وقتی که بار کننده برنامه را از دیسک جهت اجرا به داخل حافظه بار می کند ، یک فضای بسیار بزرگ در داخل حافظه به برنامه مورد نظر و داده های آن اختصاص می دهد.برنامه ها

می توانند داده های موجود در ناحیه حافظه خود را پردازش کرده حافظه ای را که نیاز ندارند آزاد کرده در صورت نیاز حافظه اضافی در خواست کنند .

مدیریت وقفه: سیستم اجازه نصب و راه اندازی برنامه های مقیم ی را به کار بران می دهد که برای انجام عملیات ویژه ای خود را به سیستم وقفه ضمیمه می کنند .  
ساختار سیستم عامل:

سه جزء اصلی سیستم عامل عبارتند از: **CO.MMAND.Com,MSDOS.SysIO.sys**

**IO.sys**: عملیات مقدار دهی اولیه را در زمان راه اندازی سیستم انجام می دهد و همچنین شامل توابع ورودی / خروجی مهم و دستگاه کردانهای است که پشتیبانی ورودی / خروجی را در بایوس سیستم را تکمیل می کنند. این قسمت در روی دیسک به صورت یک فایل سیستم مخفی ذخیره شده و در **pc-Dos** به نام **IBMBIO.COM** معروف است.

**MSDOS.sys**: به عنوان هسته مرکزی سیستم عامل کرده و با مدیریت فایل ، مدیریت حافظه و ورودی / خروجی در ارتباط است . این قسمت در روی دیسک به صورت فایل سیستم مخفی ذخیره شده و در **pc-Dos** به نام **IBMDOS.COM** معروف است .

**COMMAND.COM**: پردازشگر فرمان یا پوسته ای است که به عنوان رابط بین کار بر و سیستم عامل عمل می کند . این قسمت اعلان کار بر را نمایش داده ، صفحه کلید را کنترل کرده و فرمانهای کار بر نظیر حذف یک فایل یا باز کردن یک برنامه برای اجرا را پردازش می کند.

فرایند راه اندازی کامپیوتر **Boot Process**

روشن کردن کامپیوتر موجب یک راه اندازی سرد (**Cold boot**) می شود. پردازنده یک وضعیت آغازین شده ، تمام مکانهای حافظه را پاک کرده ، کنترل توازن حافظه را انجام داده و ثبات **CS** را برابر آدرس سگمنت **FFFF[0] H** و ثبات **IP** را برابر با افسست صفر قرار می دهد. بنابراین اولین

دستورالعمل اجرایی در آدرسی که متشکل از زوج CS:IP بوده برابر FFFF0H که همان نقطه ورود به بایوس در حافظه ROM می باشد قرار دارد.

روتین بایوس که از مکان FFFFOH شروع می شود ، پورت های گوناگون را برای شناسای و ارزش دهی دستگاههایی که به کامپیوتر متصل می باشند، کنترل می کند:

۱. یک جدول بردار وقفه که در ناحیه پایین حافظه از مکان صفر شروع شد و حاوی آدرس های وقفه هایی است که روی می دهند.

۲. یک ناحیه داده های بایوس که ابتدای آن در مکان [0]40 بوده و بیشتر با دستگاههای متصل به کامپیوتر در ارتباط است .

بایوس پس از آن کنترل می کند که آیا دیسک حاوی فایل های سیستم قابل دستیابی است و یا نه که در صورت آماده بودن آن ، به بار کننده برنامه راه اندازی سیستم (boot start loader) از دیسک دستیابی می کند این برنامه فایل های سیستم یعنی MSDOS,To.sys را از دیسک به داخل حافظه بار کرده و کنترل را به نقطه ورودی To.sys که حاوی کد دستگاه گردان ها و سخت افزارهای مخصوص دیگر می باشد منتقل می کند.To.sys در حافظه جای گرفته و سپس کنترل را به MSDOS.sys انتقال می دهد. این ماژول جداول داخلی DOS و بخش DOS در داخل جدول بردار وقفه ها را مقدار دهی اولیه می کند.همچنین فایل CONFIG.sys را خوانده و فرامین داخل آن را اجرا می کند . در نهایت ، MSDOS.sys کنترل را به COMMAND.COM منتقل می کند که فایل AUTOEXE.BAT را پردازش کرده، اعلان آن را نشان داده و صفحه کلید را برای ورود اطلاعات کنترل می کند.

در اینجا، حافظه متعارف تا 640k بایت به صورت نشان داده شده در شکل دیده می شود تحت مدیریت حافظه، بخش سیستم ممکن است در داخل حافظه فوقانی جای گیرد.

## رابطه ورودی- خروجی

بایوس دارای مجموعه ای از روتین ها در حافظه ROM برای ارائه پشتیبانی از دستگاههای مختلف می باشد ، بایوس دستگاههای متصل را امتحان کرده و آنها را مقدار دهی اولیه کرده سرویس های را ارائه می کند که برای خواندن از دستگاهها و نوشتن به آنها به کار می روند . یکی از وظایف Dos برقراری ارتباط با پایاس در زمانی است که می خواهد از امکانات آن استفاده کند .

وقتی که برنامه ای یک سرویس Llo از سیستم عامل در خواست می کند این درخواست را به بایوس منتقل می کند که در نتیجه به دستگاه مورد نظر دستیابی می کند . اما برخی مواقع ، یک برنامه از بایوس به طور مستقیم درخواست می کند به ویژه برای سرویس ها مربوط به صفحه نمایش و صفحه کلید. درمواقعی دیگر- هر چند بندرت- یک برنامه می تواند هم از Dos و هم از بایوس برای دستیابی مستقیم به یک دستگاه استفاده کند .

## بار کننده برنامه سیستم

دو نوع از برنامه های قابل اجرا، برنامه های EXE,COM می باشد . یک برنامه COM تنها از یک سگمنت تشکیل می شود که شامل کد ، داده ها وپشته . یک برنامه COM به عنوان یک برنامه سود

مند کوچک و یا یک برنامه مقیم در حافظه (برنامه ای که در حافظه نصف شده است و در موقع اجرای سائز برنامه ها قابل دسترس است)، می باشد. یک برنامه EXE از شگفتیهای کد، داده ها و پشته جداگانه ای تشکیل می شود و از آن برای برنامه های بسیار مهم استفاده می شود وقتی که از سیستم می خواهید که یک برنامه EXE را جهت اجرا از دیسک به داخل حافظه بار کند، بار کننده مراحل زیر را انجام می دهد:

۱. به برنامه EXE روی دیسک دستیابی می کند .
  ۲. یک پیشوند سگمنت برنامه (PSP) 266 بایتی را روی مرز پاراگراف یعنی مضروب صحیحی از ۱۶ در حافظه داخلی ایجاد می کند .
  ۳. برنامه مزبور را بلا فاصله پس از PSP در حافظه ذخیره می کند .
  ۴. آدرس PSP را در ثبات های ES,DS بار می کند .
  ۵. آدرس سگمنت کد را در ثبات CS بار کرده و ثبات LP را برابر افست اولین دستورالعمل (معمولا صفر) موجود در هر سگمنت کد قرار می دهد.
  ۶. آدرس پشته را در ثبات SS بار کرده و اندازه پشته را در ثبات SP قرار می دهد .
  ۷. کنترل را جهت اجرا به برنامه منتقل می کند که معمولا از اولین دستورالعمل موجود در سگمنت کد شروع می شود.
- در نتیجه بار کننده برنامه ثباتهای SS:SP,CS:LP را به درستی مقدار دهی می کند، اما توجه کنید که بار کننده برنامه آدرس PSP را هم در ثبات DS و هم در ثبات ES ذخیره می کند، هر چند که برنامه شما به طور طبیعی به آدرس سگمنت داده ها، مقدار دهی می کند .
- پشته تمام برنامه های EXE,COM به یک ناحیه به نام پشته در برنامه ها نیاز دارد. مقصود از پشته فراهم کردن فضای برای ذخیره موقت آدرس ها و اقلام اطلاعاتی می باشد .
- بار کننده برنامه بطور اتوماتیک، پشته را برای یک برنامه COM تعریف می کند، در حالیکه شما باید به طور صریح برای یک برنامه EXE پشته ای را تعریف کنید. طول هر قلم داده در پشته یک

کلمه (دو بایت) می باشد. ثبات SS همانطور که توسط بار کننده مقدار دهی شده است ، حاوی آدرس شروع پشته می باشد. در ابتدا، ثبات SP حاوی اندازه پشته که مقداری است که به بایت بعد از انتهای پشته اشاره می کند ، می باشد. پشته در روش ذخیره سازی داده ها با سگمنت های دیگر تفاوت دارد: ذخیره داده ها را از بالاترین مکان سگمنت شروع کرده و داده ها را در حافظه به سمت پایین ذخیره می کند.

### دستورالعمل های POP,PUSH

دو تا از دستورالعمل های هستند که محتوی ثبات 6P را تغییر داده و برای ذخیره داده ها در روی پشته و بازیابی آنها بکار می روند. دستورالعمل PUSH اندازه SP را دو واحد کاهش داده و آنرا به کلمه پایین تر بعدی حافظه اشاره داده و یک مقداری را در آن ذخیره می کند POP مقداری را از پشته بازیابی کرده و مقدار SP را دو واحد افزایش می دهد تا به کلمه بالایی بعدی حافظه اشاره کند .

همیشه باید اطمینان حاصل کنید که برنامه شما اضافه کردن اطلاعات بر روی پشته و برداشتن آنها از پشته را بطور هماهنگ انجام می دهد. اگر چه این یک نیاز نسبتا ساده و واضحی است ولی یک خطا می تواند موجب اختلال در عملکرد یک برنامه شود همچنین پشته ای که برای یک برنامه EXE تعریف می کنید باید به قدر کافی بزرگ باشد تا بتواند تمام مقادیری را که روی آن قرار می دهید در خود جای دهد .

دستورالعملهای مرتبط دیگر که مقدارهایی را به روی پشته اضافه کرده اطلاعات را از روی آن بر می دارد عبارتند از PushF و popF: وضعیت فلگ ها را ذخیره و بازیابی می کنند. popA, pushA (برای پردازنده ۸۰۲۸۶ و سپس از آن) محتوی تمام ثبات های همه منظوره را ذخیره و بازیابی می کنند.

## آدرس دهی دستورالعملها و داده ها

برنامه نویسی اسمبلی یک برنامه را در کد نمادی نوشته و از اسمبلر برای ترجمه آن به کد ماشین استفاده می کند. برای اجرای برنامه سیستم فقط کد ماشین را به داخل حافظه بار می کند. هر دستورالعمل شامل حداقل یک کد عمل نظیر انتقال، جمع و برگشت می باشد. بسته به عمل مورد نظر، یک دستورالعمل ممکن است دارای یک یا چند عملوند نیز باشد که به داده هایی که دستورالعمل مزبور پردازش می کند اطلاق می شود.

برای یک برنامه EXE، ثبات CS حاوی آدرس شروع سگمنت کد برنامه بوده و ثبات DS حاوی آدرس شروع سگمنت داده ها می باشد، سگمنت کد حاوی دستورالعملهای است که اجرا می شوند، درحالیکه سگمنت داده ها، حاوی داده های است که دستورالعملها به آنها مراجعه می کنند. ثبات IP آدرس افسست دستورالعمل جاری در سگمنت کد را که اجرا می شود، نشان می دهد. عملوند یک دستورالعمل آدرس افسستی را که در سگمنت داده ها به آن مراجعه می شود، نشان می دهد.

### عملوندهای دستورالعمل

یکی از ویژگیهای عملوندهای دستورالعملهای برای روشنی و گویایی برنامه استفاده از نام های عادی، نام های داخل کروشه ها و اعداد می باشد در مثال زیر دستور Wordx, DW را به صورت

Wordx DW 0 یک کلمه (۲بایت) تعریف می کند.

MOV CX, Wordx محتوی Wordx را به CX منتقل می کند

MOV CX, 25 مقدار ۲۵ را به CX انتقال میدهد

MOV CX, DX محتوی DX را به CX منتقل می کند

MOV CX, [DX] محتوی مکان آدرس دهی شدن توسط ثبات DX را به CX منتقل می کند

- دستورالعمل MOV اول، داده را بین حافظه و یک ثبات انتقال می دهد.

- دستورالعمل MOV دوم، داده بلاواسطه را به یک ثبات منتقل می کند.

- دستورالعمل MOV سوم، داده را بین ثبات منتقل می کند.

- کروش های دستورالعمل MOV چهارم یک عملگر نویس تعریف می کند که به این صورت می باشد: از آدرس افست داخل DX (که با آدرس سگمنت واقع در DS بصورت DS:DX ترکیب می شود) برای پیدا کردن کلمه ای در حافظه استفاده می کرده و محتوی آنرا به CX منتقل می کند.

## فصل ۴

برنامه نویسی به زبان اسمبلی

یک توضیح با یک کارکتر سمیکالن شروع شده و هر جا که نوشته شود اسمبلر فرض می کند که تمام کارکترهای سمت راست آن تا انتهای خط، توضیحات محسوب می شوند. چون یک توضیح فقط در لیست یک برنامه اسمبل ظاهر شده و هیچ کد ماشینی را تولید نمی کند می توانید بدون اینکه تاثیری در اندازه برنامه و زمان اجرای آن داشته باشد هر تعداد از توضیحات را در داخل یک برنامه بگنجانید.

روش دیگر برای نوشتن توضیحات، استفاده از دستورالعمل Comment می باشد.

کلمات رزو شده

دستورالعملها مانند ADD , MOV که اعمالی می باشد که کامپیوتر می تواند اجرا کند، دستورات اسمبلر مانند END یا SEGMENT که برای ارائه اطلاعاتی به اسمبلر مورد استفاده قرار می گیرند عملکردها نظیر FAR, SIZE که در عبارات مورد استفاده قرار می گیرند و نمادهای از پیش تعریف

شده مانند **Data** , **Model** که اطلاعات را به برنامه شما برمی گرداند. در صورتی که از کلمات رزرو شده برای منظوره‌های دیگر استفاده شود اسمبلر یک پیغام خطایی را تولید می کند. شناسه ها

دو نوع شناسه ها نام و برچسب (**label**) می باشند:

۱. نام به آدرس یک داده اشاره می کند.

۲. برچسب ، به آدرس یک دستورالعمل روال یا سگمنت اشاره می کند، مانند **MAIN** در

دستورالعمل زیر: **MAIN PROC FAIR**

قوانین یکسانی برای نام ها و برچسب ها ، مورد استفاده قرار می گیرد. یک شناسه می تواند از کاراکترهای زیر استفاده کند.

- حروف الفبا **A تا Z و Q تا Z**

- ارقام **0 تا 9** (نباید اولین کاراکتر یک رقم باشد.)

- کاراکترهای ویژه علامت (؟) خط زیرین (-) علامت (\$) کاراکتر (@)

نقطه (.) (نباید اولین کاراکتر باشد)

### احکام (state ments)

یک برنامه اسمبلی دارای مجموعه ای از احکام می باشد. دو نوع از احکام عبارتند از:

۱. دستورالعملهای مانند **ADD,MOV** که اسمبلر آنها را به کد هدف ترجمه می کند.

۲. دستورات (**directives**) که به اسمبلر می گویند که عمل معینی مانند تعریف یک قلم داده را

انجام دهد.

فالب یک حکم در اینجا آمده است که گروه ها یک عامل اختیاری را نشان می دهند.

## دستورات Title,Page

دستورات Title,Page اسمبلر برای کنترل فرم لیست یک برنامه اسمبل شده به کار می روند هدف این دو دستور همین بوده و هیچ تاثیری روی اجرای بعدی یک برنامه ندارند.

Page : در ابتدای یک برنامه ، دستور Page اسمبلر ، تعداد ماکسیمم خطوط برای لیست کردن در یک صفحه و ماکسیمم تعداد کاراکترهای روی یک خط را نشان می دهد.

شکل کلی: Page {پهنا} {طول}

ممکن است بخواهید که در فایل لیست برنامه یک صفحه در خط مشخصی مانند انتهای یک سگمنت به اول صفحه بعدی پرش کند برای این کار در خط مورد نیاز دستور Page را بدون عملوندی را بنویسید.

Title : می توانید از دستور Title اسمبلر استفاده کرده و عنوان یک برنامه را در خط دوم در هر صفحه از لیست برنامه چاپ کنید.

## دستور SAGMENT اسمبلر

یک برنامه اسمبلی در قالب EXE دارای یک یا چند سگمنت می باشد. یک سگمنت پشته که حافظه پشته را تعریف کرده، یک سگمنت داده ها که اقلام داده ها را تعریف کرده و.....

دستورات اسمبلری که برای تعریف یک سگمنت بکار می روند عبارتند از : segments , ENDS و دارای قالب زیر می باشد.

توضیحات	عملوند	عمل	نام
Begin segment و	پارامترها	segment	نام
End segment و		ENDS	نام

عملوند یک دستور سگمنت می تواند دارای سه نوع پارامتر باشد: تنظیم (align)

ترکیب (combine) ورده (class) که در قالب زیر نوشته می شوند :

نام	segment	تنظیم	ترکیب	"رده"
-----	---------	-------	-------	-------

نوع تنظیم: پرامتر تنظیم مرزی را نشان می دهد که سگمنت از آنجا شروع خواهد شد. اگر نوع تنظیم برابر **PARA** باشد، سگمنت مزبور روی مرز پراگراف تنظیم می شود، به این معنی که آدرس شروع آن بر ۱۶ یا 10H بخش پذیر بوده.

اگر از پرامتر تنظیم چشم پوشی کنید اسمبلر بطور پیش فرض ، **para** در نظر می گیرد.

نوع ترکیب: پرامتر ترکیب نشان می دهد که این سگمنت را با سگمنت های دیگر پس از اسمبل شدن در مرحله لینک ترکیب کنند. انواع ترکیب عبارتند از: **public , common , stack** و عبارت **AT** برای مثال، سگمنت پشته معمولا به صورت زیر تعریف می شود.

نام	Sgment	PARA	stack
-----	--------	------	-------

اگر بخواهید برنامه های اسمبل شده جداگانه رادر موقع لینک کردن، ترکیب کنید می توانید از **public , common** استفاده کنید، در غیر این صورت، اگر یک برنامه قرار نیست با برنامه های دیگر ترکیب شود، باید این پرامتر را حذف کرده و یا کلمه **none** را بنویسید.

نوع رده: پرامتر رده (**class**) که در داخل دو آپوستروف ( ' ) قرار می گیرد ، برای گروه بندی سگمنتهای مرتبط در هنگام لینک کردن مورد استفاده قرار می گیرد. این کتاب از رده های 'code' برای سگمنت کد (توسط میکروسافت معرفی شده) ، 'data' برای سگمنت داده ها و 'stack' برای سگمنت پشته استفاده می کند.

### دستور PROC

سگمنت که حاوی کد قابل اجرای یک برنامه می باشد که از یک یا چند روال تشکیل می شود که با دستور PROC تعریف می شوند.

### دستور ASSUME

یک برنامه **EXE** از ثبات **SS** برای آدرس دهی پشته ، از ثبات **DS** برای آدرس دهی سگمنت داده ها و از ثبات **CS** برای آدرس دهی سگمنت کد استفاده می کند.

در اینجا ، باید هدف از هر سگمنت برنامه را به اسمبلر اطلاع دهید. دستور مورد نیاز برای این کار دستور ASSVME می باشد که در سگمنت کد به صورت زیر نوشته می شود.

عمل

عملوند

ASSVME

نام سگمنت کد: CS و نام سگمنت داده: DS و نام پشته: SS

عملوند نام پشته: SS به این معنی است که اسمبلر باید نام سگمنت پشته را با ثبات SS مرتبط کند و به روش مشابه برای عملوندهای نشان داده شده دیگر عمل می کند. عملوندها می توانند به هر ترتیبی ظاهر شوند ASSVME می تواند دارای یک عامل نیز برای ES به صورت نام سگمنت داده ها: ES باشد.

اگر برنامه شما از ثبات ES استفاده نمی کند، می توانید از رجوع به آن صرفنظر کرده و یا عملوند ES:nothing را وارد کنید.

همانند سایر دستورات اسمبلر ، ASSVME فقط پیغامی است تا اسمبلر در تبدیل کد نمادی به کد ماشین کمک کند و هنوز ممکن است مجبور باشید که دستوراتی را بنویسید که بطور فیزیکی آدرسهایی را در ثباتهای سگمنت در زمان اجرا قرار دهید.

**دستور END**

همانطور که قبلا ذکر شد، دستور ENDS به یک سگمنت و دستور ENDP به یک روال خاتمه می دهند. یک دستور END به کل برنامه خاتمه داده و به عنوان آخرین دستور در یک برنامه ظاهر می شود قالب کلی آن به صورت زیر است.

عمل

عملوند

END

[procname]

اگر قرار نیست برنامه اجرا شود، عملوند می تواند خالی باشد، برای مثال، ممکن است بخواهید که فقط تعارف داده ها را اسمبل کرده و یا ممکن است بخواهید که برنامه را با یک ماژول دیگری لینک کنید. در بسیاری از برنامه ها این عملوند شامل نام اولین و یا تنها PROC معرفی شده به صورت FAR می باشد که اجرای برنامه قرار است از آنجا شروع شود.

دستورالعملهای مقدار دهی اولیه یک برنامه

وقتی بار کننده برنامه یک برنامه EXE را برای اجرا از دیسک به درون حافظه می خواند یک psp بطول ۲۵۶ بایت روی مرز پاراگراف در داخل حافظه ساخته و بلافاصله برنامه را به دنبال آن ذخیره می کند. سپس بار کننده کارهای زیر را انجام می دهد:

- آدرس سگمنت کد را در CS بار می کند.

- آدرس پشته را در SS بار کرده و آدرس psp را در ثباتها.

### دستورات ساده شده سگمنت

اسمبلر برخی راههای میان بر در تعریف سگمنت ها را ارائه می دهد. برای استفاده کردن از آنها باید قبل تعریف هر سگمنت، مدل حافظه را مقدار دهی کنید. قالب کلی به صورت زیر می باشد.

MODEL حافظه

مدل حافظه می تواند tiny, small, medium, compact, large باشد نیازهای هر مدل عبارتند از:

مدل	تعداد سگمنتهای کد	تعداد سگمنتهای داده
Tiny	*	*
small	۱	۱
medium	بیش از ۱	۱
compact	۱	بیش از ۱
Large	بیش از ۱	بیش از ۱

می توانید از هر یک از این مدل ها برای برنامه مستقل ( برنامه ای که به برنامه دیگر liny نشده

باشد) استفاده کنید. در اسمبلرهای TASM4, MASM6 مدل Tiny برای استفاده کردن از برنامه com

طراحی شده است که داده ها، کد و پشته در یک سگمنت 64k بایت قرار دارند. در مدل small کدها

در داخل یک سگمنت 64k و داده ها در یک سگمنت 64k دیگری جا می گیرند با دستور model

اسمبلر بطور اتوماتیک دستور Assume مورد نیاز را تولید می کند.

قالب های کلی دستوراتی که سگمنت های پشته، داده ها و کد را تعریف می کنند به صورت زیر می

باشند: `stack` {اندازه}

`Data`

`code` {نام}

هر یک از این دستورات موجب می شوند که اسمبلر دستور `segment` مورد نیاز و دستور `Ends`

متناظر را تولید کند. نام سگمنت های پیش فرض (که نباید تعریف کند) عبارتند از:

`text, data, stack` (برای سگمنت کد). کاراکتر خط زیرین در ابتدای `text, data` ضروری می باشند

همانطور که قالب کد گذاری نشان می دهد، می توانید نام پیش فرض سگمنت کد را تغییر دهید

اندازه پیش فرض پشته برابر ۱۰۲۴ بایت بوده و می توانید آنرا صریحا انتخاب کنید.

دستورالعملهایی که اکنون برای مقدار دهی اولیه آدرس سگمنت داده ها در داخل ثبات `ds` استفاده

می کنید عبارتند از `MOV DS, AX`, `MOVS AX, @data`

دستورات `Startup`, `EXIT` اسمبلر

اسمبلر `MASM` برای آسان نمودن مقدار دهی اولیه برنامه و خاتمه دادن آن دستورات `Exit`.

`startup` را معرفی کرده است. دستورالعملهایی را برای مقدار دهی اولیه ثباتهای سگمنت

تولید می کند در حالیکه دستور `Exit` دستورالعملهای تابع `4ch` از `INI 21h` را برای خارج شدن از

برنامه تولید می کند.

اسمبلر `TASM` از اصطلاحات `ExitCode, startup code` استفاده می کند.

تعریف داده ها:

همانطور که بیان شد. هدف سگمنت داده ها در یک برنامه `EXE` تعریف ثابتها، نواحی کاری و

نواحی ورودی/خروجی می باشد. اسمبلر اجازه تعریف اقلام داده ها با طولهای گوناگون را بر حسب

دستورات اسمبلری که داده ها را تعریف می کنند می دهد: برای مثال، `DB` یک بایت و `DW` یک کلمه را

تعریف میکند. یک قلم داده ممکن است دارای یک مقدار تعریف نشده (مقدار دهی نشده) بوده و یا

حاوی یک مقدار ثابت باشد. که بصورت یک رشته کاراکتری یا یک مقدار عددی تعریف شده

است. قالب کلی داده هابه صورت زیر می باشد. `[name] Dn expressino`

**name**: برنامه ای که به یک قلم داده مراجعه می کند، این کار را به وسیله یک نام انجام می دهد.

دستور اسمبلر "Dn": دستوراتی که اقلام داده ها را تعریف می کنند عبارتند از: **DB** (بایت)، **DW**

(کلمه)، **DD** (کلمه مضاعف)، **DF** (کلمه دور)، **DQ** (چهار کلمه یا ۸ بایت) و **DT** (۱۰ بایت) که هر کدام به

صورت صریح طول داده تعریف شده را نشان می دهد. اسمبلر **MASM6** به ترتیب کلمات

**Tword, Qword, Fword, Dword-22/word-Byte** برای دستورالعملها معرفی کرده است.

عبارت (**expression**) عبارت در یک عملوند ممکن است یک مقدار بدون ارزش اولیه یا یک ثابت

اولیه باشد برای نشان دادن یک داده بدون مقدار اولیه عملوند را با یک علامت سوال تعریف کنید.

مثال : `FLDA DB ?`

یک عبارت ممکن است دارای چند ثابت باشد که بوسیله کاراکتر کاما(,) از یکدیگر جدا شده است

و فقط به اندازه طول یک خط محدود می باشد، مثل `FLDA DB/Byte 21,22,23,.....`

اسمبلر این ثابت ها را در بایتهای مجاور تعریف می کند. مراجعه به **FLDA** معادل مراجعه به اولین

ثابت یک بایتی یعنی ۲۱ می باشد (می توانستید بایت اول را به صورت **FLDA+0** در نظر بگیرید) و

مراجعه به **FLDA+1** معادل مراجعه به ثابت دوم یعنی ۲۲ می باشد برای مثال دستورالعمل

`MOV AL,FLDA+3`

مقدار **24(18h)** را به ثابت **AL** انتقال می دهد همچنین تکرار ثباتها در یک عبارت در یک قلم به

صورت قلمی (عبارت) **Dup** تعداد **DN name** مجاز می باشد.

1/ `DW/word 10 Dup (?)`

2/ `DB/Byte 5 Dup (12)`

3/ `DB/Byte 3 Dup (5 Dup 4)`

مثال سوم پنج کپی از رقم **4(44444)** را تولید کرده و این مقدار را سه بار تکرار کرده و مقدار دهی

اولیه کند.

رشته های کاراکتری:

رشته های کاراکتری برای داده های توصیفی نظیر نام افراد و توصیف محصولات مختلف به کار می رود پشته در داخل کوتیشن های تکی نظیر "PC" تعریف شده و یا داخل کوتیشن مضاعف نظیر "PC" تعریف می شود اسمبلر رشته های کاراکتری را به صورت کد هدف در قالب عادی اسکی و بدون کوتیشن های ابتدا و انتها ذخیره می کند. در اسمبلر DB, MASM (یا Byte) تنها قالبی است که یک رشته کاراکتری با بیش از دو کاراکتر را تعریف کرده و آنها را به ترتیب از چپ به راست ذخیره می کند (شبهه اسامی و آدرسها) در نتیجه DB قالب متعارف برای تعریف داده های کاراکتری با هر طولی می باشد یک مثال عبارت است از:

23/ DB , strawberry jam

اگر رشته شامل یک کوتیشن تکی یا کوتیشن مضاعف باشد می توانید آنرا به یکی از دو روش زیر

تعریف کنید. DB "crazy sams CD Emporium"

DB Crazy sam "s CD Emporium"

ثابت های عددی:

ثابت های عددی برای تعریف مقادیر حسابی و آدرسهای حافظه بکار می روند. یک ثابت در داخل کوتیشن قرار نمی گیرد. اما پس از آن می توان یک مشخص کننده مبنا ذکر نمود مانند H در مقدار شانزده شانزدهی برای اکثر دستورات تعریف داده ها ، اسمبلر ثابت های عددی تعریف شده را به مبنای شانزده شانزدهی تبدیل کرده و بایت های تولید شده را در کد هدف به ترتیب معکوس، از راست به چپ ذخیره می کند.

قالب های عددی مختلف به شرح زیر می باشند:

دودویی: در قالب دودویی می توان از ارقام دودویی 1,0 به دنبال آن از حرف B استفاده کرد استفاده معمولی از قالب دودویی ذکر کردن مقادیر برای دستورالعملهای پردازش کننده بیت ها یعنی Test,xor,OR,AND می باشند.

دهدهی: در قالب دهدهی می توان از ارقام ۰ تا ۹ و به دلخواه به دنبال آن از حرف D استفاده کرد  
 اسمبلر مقادیر دهدهی را به کد هدف دودویی تبدیل کرده و آنها را در قالب شانزده شانزدهی نمایش  
 می دهد.

شانزده شانزدهی: در این قالب از ارقام 0 تا F و به دنبال آن از حرف H استفاده می شود.

چون اسمبلر عددی را که با یک حرف شروع شود برابر نام یک نماد فرض می کند، لذا رقم اول  
 یک ثابت شانزده شانزدهی بایستی از ۰ تا ۹ باشد.

اعداد حقیقی: اسمبلر یک مقدار حقیقی داده شده را برای استفاده کردن با یک کمک پردازنده عددی  
 به قالب ممیز شناور تبدیل می کند از تفاوت بین استفاده از کاراکترها و ثابت های عددی اطمینان  
 حاصل کنید.

### عملوندهای دستورالعملها

یک عملوند منبع داده ها برای یک دستورالعمل جهت پردازش می باشد. برخی از دستورالعملها  
 مانند CLC و RET به عملوند نیازی ندارند. در حالیکه دستورالعملهای دیگر دارای یک یا چند  
 عملوند می باشد اگر در جایی دو عملوند وجود داشته باشد، عملوند دوم، عملوند منبع می باشد که  
 حاوی داده ای است که ارسال شده (بلا واسطه) و یا حاوی آدرس (ثبات یا حافظه) داده مورد نظر می  
 باشد. عملوند منبع توسط دستورالعمل تغییر داده نمی شود عملوند اول، عملوند مقصد می باشد که  
 حاوی داده مورد نظر در یک ثبات یا حافظه می باشد که قرار است پردازش شود قالب  
 دستورالعملها به صورت زیر می باشد.

[label:] operation operand 1, operand 2

عملوندهای ثابت

در این نوع، ثبات نشان دهنده نام یکی از ثبات های ۸ و ۱۶ یا ۳۲ بیتی می باشد. بسته به دستورالعمل  
 ثبات ممکن است در عملوند اول، عملوند دوم و یا در هر دو ظاهر شود.

Worda Dw ? define a word  
 Mov DX, Worda , register in first operand

Mov worda .cx ,register in second ... ..

Mov EDX , EBX ,register in both operand

چون پردازش داده ها بین ثباتها هیچ مراجعه ای به حافظه ندارد لذا این سریعترین نوع عمل است .

عملوندهای بلا واسطه

در قالب بلاواسطه عملوند دوم حاوی یک مقدار ثابت یا یک عبارت می باشد(عملوند اول هرگز

نبایستی یک مقدار بلاواسطه باشد.) مقصد در عملوند اول طول داده را تعریف می کند و ممکن

است یک ثبات یا یک مکان حافظه باشد.

Count DB ?

ADD BX,25 ,add 25 to BX

Mov count,50 ,move 50 to count

عملوندهای مستقیم حافظه

در این قالب ، یکی از عملوندها به یک مکان حافظه مراجعه کرده و عملوند دیگر به ثبات مراجعه

می کند(تنها دستورالعملهایی که هر دو عملوند می توانند به طور مستقیم حافظه را آدرس دهی کنند

عبارتند از Cmps,movs). ثبات DC ، ثبات سگمنت پیش فرض برای آدرس دهی داده ها در

حافظه می باشد.

Worda DW o ,Define a word

Bytea DB o ,define a Byte

Mov BX, worda

ADD ByteA , DL

Mov CX,DS:[38BOH]

INC ByteA PTR [1BOH]

دو مثال آخر از گروه برای نشان دادن اندیس جهت مراجعه به حافظه استفاده می کند(یک مقدار

افست مانند 38BOH با آدرس واقع در DS ترکیب می شود) اگر از گروه استفاده نشود مثلا در

38BOH و move cx یک مقدار بلاواسطه را نشان می دهد.

مثال آخر بایتی از حافظه را که در آفت 1BOH قرار دارد یک واحد افزایش می دهد.

از آنجا که عملوند [1BOH] فقط یک مکان حافظه را نشان می دهد لازم است که از تغییر دهنده

Byte PTR برای تعریف طول استفاده کنید.

Codetbl DB 20 DUP (?)

Mov CL, Codetbl [3]

Mov CL, Codetbl+3

اولین دستورالعمل mov از یک مشخص کننده اندیس برای دستیابی به بایت چهارم استفاده می کند دستورالعمل mov دوم از عملکرد + استفاده کرده و دقیقاً همان کار قبلی را انجام می دهد.

عملوندهای غیر مستقیم

آدرس دهی غیر مستقیم تکنیک پیشرفته ای است که از قابلیت کامپیوتر برای آدرس دهی افسست سگمنت استفاده می کند. ثباتهای که برای این منظور به کار می رود، عبارتند از BP,SI,DI,BX می باشند که در داخل گروه به عنوان عملگیر اندیس نوشته می شوند. ثباتهای SI,DI,BX برای پردازش داده های داخل سگمنت داده ها با ثبات DS بصورت DS:SI,DS:DI,DS:BX مرتبط می باشند ثبات BP برای مدیریت داده ها در پشته با ثبات SS به صورت SS:BP مرتبط می باشد که برای فراخوانی زیر برنامه ها و انتقال دادن پارامترها مورد بحث قرار خواهد گرفت.

موقعی که عملوند اول شامل یک آدرس غیر مستقیم می باشند. عملوند دوم به یک ثبات یا یک مقدار بلاواسطه مراجعه می کند. وقتی که عملوند دوم شامل یک آدرس غیر مستقیم می باشد، عملوند به یک ثبات مراجعه می کند. یک آدرس غیر مستقیم مانند [DI] به اسمبلر می گوید وقتی که برنامه بعداً اجرا می شود آدرس حافظه مورد استفاده در ثبات DI خواهد بود.

در مثال زیر Mov اول BX را با آدرس افسست Dataval مقدار دهی می کند. Mov دوم از آدرس واقع

در BX برای ذخیره کردن عدد ۲۵ در مکان حافظه ای که به آن اشاره می کند یعنی Dataval

Dataval DB 50 استفاده می کند.

MOV BX,offset Dataval

MOV [BX],25

تأثیر دو دستورالعمل MOV معادل با نوشتن دستورالعمل 25, mov Dataval می باشد. هر چند که کاربردهای آدرس دهی دارای اندیس زیاد بدیهی نیست. دستورالعمل زیر عدد صفر را به مکانی که بلافاصله دو بایت پس از Dadval قرار دارد ذخیره می کند:

```
mov [BX+2], 0
```

می توانید ثباتها را نیز در یک آدرس غیر مستقیم ترکیب کنید: برای مثال [BX+DI] به معنی آدرس واقع در BX به علاوه آدرس واقع در DI می باشد.

توجه کنید که یک مراجعه در داخل گروه به ثبات DI, BX, BP یا SI نشان دهنده یک عملوند غیر مستقیم بوده و پردازنده وقتی که برنامه در حال اجراست با محتوی ثبات مورد نظر به عنوان یک آدرس افست رفتار می کند.

```
ADD CL,[BX] ,2nd operond =DS:BX
MOV Byte ptr[DI],25 ,1st operond =DS:DI
ADD [BP] ,CL ,1st operond =SS:BP
```

جابجایی آدرس: این روش از جابجایی آدرس برای یک عملوند استفاده می کند. دستورالعمل MOV زیر محتوی BL را به Data TAB (یک جدول ۴۰ بایتی) انتقال می دهد، دقیقاً جای دو Data TAB که با محتوی DI در موقع اجرای برنامه تعیین خواهد شد.

```
Datatab DB 40 DUP(?)
MOV datatab[DI] ,BL
```

اندیس گذاری روی ۸۰۳۸۶ و پردازهای پس از آن در این پردازها، یک آدرس از ترکیب یک یا چند ثبات عمومی، یک افست و یک عامل مقیاس گذاری (۱ یا ۲ یا ۸) کد مربوط با محتوی یکی از ثباتها می باشد بدست می آید.

### دستورالعمل MOV

دستورالعمل MOV، داده ای را که با آدرس عملوند دوم مشخص می شود، به آدرس عملوند اول منتقل می نماید (کپی می کند) فیلد ارسالی تغییر داده نمی شود. عملوندهای که به حافظه یا ثباتها مراجعه می کنند باید از نظر اندازه سازگار باشند (یعنی هر دو باید، طول بایت، کلمه یا کلمه مضاعف

باشند قالب کلی دستورالعمل MOV به صورت زیر است. مقدار بلاواسطه/حافظه/ثبات

[alde:] MOV و حافظه/ثبات

در اینجا ۴ مثال معتبر MOV با در نظر گرفتن ارقام زیر آمده است:

Bytefld	BW	?
Wordfld	DW	?

۱. انتقال ثبات

MOV	EDX,ECX	,Register to Register
MOV	DS , BX	,Register to segment register
MOV	Byteefld,DH	,Register to memory,direct
MOV	[DI] ,BX	,Register to memory , Indirect

۲. انتقال مقادیر بلاواسطه

MOV	CX,40	,Immediate to register
MOV	Bytefld,40	,Immediate to memory-direct
MOV	Wordfld [BX],40	,Immediate to memory-indirect

۳. انتقال حافظه مستقیم

MOV	CH,Bytefld	,memory to Register,direct
MOV	CX,Wordfld [BX]	,memory to Register,indirect

۳. انتقال ثبات سگمنت

MOV	CX,DS	,segment register to Register
MOV	Wordfld,DS	,segment Register to memory

می توانید به یک ثبات یک بایت، یک کلمه یا یک کلمه مضاعف را منتقل کنید. عملوند دوم فقط

روی بخش مورد نظر ثبات عملوند اول تاثیر می گذارد، برای مثال انتقال دادن یک بایت به CH

روی CL تاثیری ندارد، عملیات غیر معتبر MOV عبارتند از:

حافظه-به-حافظه (این را همیشه به خاطر داشته باشید) مقدار بلاواسطه-به ثبات سگمنت و ثبات

و ثبات سگمنت-به-ثبات سگمنت. انجام این عملیات به بیش از یک دستورالعمل نیاز دارد.

فصل ۷، مجموعه دستورالعملهای کامپیوتر:

هدف: تشریح کد ماشین و ارائه توصیفی از مجموعه دستورات عملهای کامپیوتر.

مقدمه: این فصل کد ماشین را شرح می دهد و لیستی از دستورات عملهای نمادی به همراه شرحی از اهداف آنها ارائه می دهد.

بسیاری از دستورات عملها دارای هدف مشخصی می باشند. بطوریکه یک کد دستورالعمل زبان ماشین ۱ بایتی کافی باشد.

هیچ یک از این دستورات عملها، موجب مراجعه مستقیم به حافظه نمی شوند. دستورات عملهای که یک عملوند بلاواسطه، دو ثبات، یا یک رجوع به حافظه را مشخص می کنند خیلی پیچیده هستند و به دو یا چند بایت کد ماشین نیاز دارند.

کد ماشین دارای یک قید ویژه برای نشان دادن یک ثبات مخصوص و قید دیگری برای مراجعه به حافظه به وسیله یک بایت حالت آدرس دهی می باشد.

نشان گذاری ثبات

دستورات عملهایی که به یک ثبات مراجعه می کنند ممکن است حاوی ۳ بیت باشند که ثبات مخصوص را نشان می دهند و یک بیت که نمایانگر این است که پهنای یک بایت (0) یا یک کلمه (1) است. همچنین، فقط دستورات عملهای معینی می توانند به ثبات های سگمنت دسترسی پیدا کنند. شکل 1-27 نشان گذاری ثبات کامل را نشان می دهد. برای مثال، مقدار بیتی 000 به معنی AH است در صورتیکه بیت W برابر 0 باشد و AX است اگر این بیت برابر 1 است.

کد نمادی و ماشین یک دستورالعمل MOV با یک عملوند بلاواسطه ۱ بایتی به صورت زیر می باشد

MOV AH,00 10110 100 00000000

در این مورد بایت اول کد ماشین یک پهنای ۱ بایتی را نشان می دهد و به ثبات AH(100) اشاره می کند. دستورالعمل MOV زیر دارای یک عملوند بلاواسطه ۱ کلمه ای است که به همراه آن کد ماشین ایجاد شده آن نیز آمده است:

```
MOV    AX,00 10111 000 00000000 00000000
```

بایت اول کد ماشین یک پهنای ۱ کلمه ای را نشان می دهد و به ثبات AX(000) اشاره می کند. برای دستورالعملهای دیگر reg,w ممکن است مکانهای متفاوتی را اشغال نمایند.

بایت حالت آدرس دهی

بایت حالت (mode) در زمانی که وجود داشته باشند، بایت دوم کد ماشین را اشغال می کند و شامل سه عنصر زیر می باشد:

mod یک حالت ۲ بیتی که مقادیر 10,01,00 به مکانهای حافظه اشاره می کند و 11 به یک ثبات اشاره می کند.

Rey یک مراجعه ۳ بیتی به یک ثبات

R/m یک مراجعه ۸ بیتی به یک ثبات یا حافظه، که ۲ ثبات را مشخص می کند و m یک آدرس حافظه را نشان می دهد.

همچنین بایت اول کد ماشین ممکن است دارای یک بیت d باشد که جهت (چپ/ راست) جریان را نشان می دهد. در مثال زیر اضافه کردن AX به BX

```
MOV    BX,AX    00 00 00 11    11    011    000
```

D=q به این معنی است که mod(11) و reg(011) عملوند اول را توصیف می کنند و (r/m)(000)

عملوند دوم را توصیف می کند. چون  $w=1$  می باشد، پهنای یک کلمه است. بنابراین، این دستورالعمل AX(000) را به BX(011) اضافه می کند بایت دوم کد هدف نمایانگر بسیاری از حالات آدرس دهی حافظه می باشد. می توانید از DEBU6 برای کنترل مثال مزبور استفاده کنید.

MOV : انتقال داده ها

عمل داده‌ها را بین دو ثبات پایین یک ثبات و حافظه منتقل می‌کند و داده‌های بلاواسطه را به یک ثبات یا حافظه انتقال می‌دهد. داده‌ها مراجعه شده تعداد بایت‌های انتقال داده شده (۱ یا ۲ یا ۴) را تعریف می‌کند عملوندها باید از لحاظ اندازه سازگار باشند MOV نمی‌تواند داده‌ها را بین مکان‌های حافظه (از MOV استفاده کنید) از داده بلاواسطه به یک ثبات سگمنت یا از یک ثبات سگمنت به یک ثبات سگمنت، انتقال دهد.

فگل‌ها: هیچ یک از فگل‌ها تاثیر نمی‌پذیرند.

کد منبع: مقدار بلاواسطه/حافظه/ثبات و حافظه/ثبات MOV

کد هدف: هفت قالب:

100010	dw	mod	reg	r/m	:Reg / reg/mem
1100011	w	mod	oor/m	.....data.....data	ifw=1
1011w	reg	.....data.....data	if	w=1	Immed به ثبات:
1010000	w	addr-low	addr-high		mem به آکومولاتور:
1010001	w	addr-low	addr-high		آکومولاتور به mem
10001110	mod	O	sgr/m	(sg=seg reg)	seg reg به rem/mem
10001100	mod	o	sgr/m	(sg=seg reg)	reg/mem به seg reg

## فصل 9

دستورالعمل INT (وقفه=Interrupt) ورودی و خروجی را برای اکثریت اهداف انجام می‌دهد. توابع INT10H برای مدیریت صفحه نمایش و توابع INT21 H برای نمایش اطلاعات خروجی روی صفحه نمایش و دریافت اطلاعات ورودی از صفحه کلید هستند.

این توابع (یا سرویسها) یک عمل ویژه ای را درخواست می کنند، برای این کار مقدار تابع را در ثبات AH برای مشخص کردن نوع سرویسی که وقفه انجام خواهد داد، وارد می کنید.

عملیات سطح پایین بایوس مانند INT10 H کنترل اجرا را به طور مستقیم به بایوس منتقل می کند. ولی برای ساده کردن برخی از عملیات پیچیده تر، INT 21 H سرویس وقفه ای را ارائه می کند که ابتدا کنترل را به DOS انتقال می دهد. برای نمونه به ورود اطلاعات از صفحه کلید ممکن است مستلزم شمارش کاراکترهای وارد شده و کنترل تعداد ماکسیمم آلفا باشد.

وقفه INT21 H بسیاری از این پردازش سطح پایین اضافی را انجام داده و سپس کنترل را به طور اتوماتیک به بایوس منتقل می کند که بخش سطح پایین عمل مورد نظر را انجام بدهد.

به عنوان یک قرارداد این کتاب مقدار ODH را به عنوان کاراکتر enter برای صفحه کلید و کاراکتر برگشت به اول سطر جاری (carriage Return=CR) صفحه نمایش و چاپگر در نظر می گیرد.

### صفحه نمایش

یک صفحه نمایش معمولی دارای ۲۵ سطر (از ۰ تا ۲۴) و ۸۰ ستون (از ۰ تا ۷۹) می باشد. این سطرها و ستون ها شبکه ای از مکان های قابل آدرس دهی را به وجود می آورند که در هر یک از آنها مکان نما می تواند قرار داده شود.

مکان صفحه نمایش	قالب ده دهی		قالب شانزده شانزدهی	
	ستون	سطر	ستون	سطر
گوشه سمت چپ فوقانی	00H	00H	00	00
گوشه سمت راست فوقانی	4FH	00H	79	00
مرکز صفحه نمایش	27H/28H	00H	39/40	12
گوشه سمت چپ تحتانی	00H	18 H	00	24

سیستم دارای فضای در داخل حافظه به نام ناحیه نمایش ویدیو یا بافر می باشد. ناحیه نمایش تک رنگ از مکان B000COJH بایوس شروع شده و از 4K حافظه پشتیبانی می کند که 2K از آن برای کاراکتر بوده و 2K نیز برای بایت های مشخصه کاراکترها مانند نمایش معکوس چشمک زدن، شدت بالا و دارای خط زیرین، قابل استفاده می باشد. ناحیه نمایش ویدیوی گرافیک رنگی پایه از 16k بایت پشتیبانی می کند که از مکان بایوس B800[0]h شروع می شود. می توانید برای نمایش کاراکترها در حالت متنی (text mode) و یا در حالت گرافیکی (graphics mode) کار کنید. در حالت متنی، ناحیه نمایش صفحات نمایش را فراهم می کند که برای یک صفحه نمایش ۸۰ ستونی از شماره ۰ تا ۳ می باشد که یک بایت برای هر کاراکتر و یک بایت نیز برای بایت مشخصه آن (مانند رنگ) در نظر گرفته می شود.

#### تنظیم مکان نما

حالت گرافیکی از مکان نما پشتیبانی نمی کند - وقفه INT 10 H در بایوس برای مدیریت صفحه نمایش بوده و تابع 02 H در ثبات AH به این وقفه می گوید که مکان نما را تنظیم کند. شماره صفحه (یا شماره صفحه نمایش) مورد نظر، معمولا O، را در ثبات BH، شماره سطر را در DH و شماره ستون را در DL بار کنید. محتوی ثباتهای دیگر، مهم نمی باشد.

مثال. مکان نما را در سطر 08 و ستون ۱۵ قرار می دهد:

```
MOV AH,02 H ,Request set sursor
MOV BH,00 ,Page number 0
MOV DH,08 ,Row 8
MOV DL,15 ,Solumn 15
INT 10H ,Call interrupt service.
```

پاک کردن صفحه نمایش

تابع 06H از INT 10 H پاک کردن صفحه نمایش یا حرکت دادن آنرا انجام می دهد می توانید تمام صفحه نمایش و یا قسمتی از آنرا با شروع از هر مکان صفحه نمایش که پایان آن بایستی در یک مکان بالاتر، قرار داشته باشد، پاک کنید، ثبات های زیر را بار کنید.

• AH = تابع 06H

• AL = تعداد خطوطی که حرکت داده می شود، یا 00H برای صفحه نمایش کامل .

• BH = مقدار بایت مشخصه (رنگ، تصویر معکوس، چشمک زدن)

• CX = ستون: سطر شروع

• DX = ستون: سطر پایانی

CX,DX با هم ناحیه مورد نظر صفحه نمایش (پنجره) را تعریف می کند که حرکت داده می شود و ثبات AL تعداد خطوطی را که باید حرکت کنند مشخص می کند. برای پاک کردن کل صفحه نمایش، ستون: سطر شروع را در CX به صورت 00:00H و ستون: سطر پایانی را به صورت 18:4FH در ثبات DX مشخص کنید. صفت مشخصه 71H در مثال زیر کل صفحه نمایش را به رنگ زمینه سفید (مشخصه ۷) با رنگ نوشتاری آبی (مشخصه ۱) در می آورد.

```
MOV    AX,0600H      ,AH =06(scroll),AL=00(full screen)
MOV    BH,71H        ,white background (l), biue foreground (1)
MOV    BH,0000H      ,upper left row: column
MOV    DX,184 FH     ,lower row : column
INT    10 H          ,Call interrupt service
```

برای نمونه جهت حرکت دادن پنجره صفحه نمایش از سطر 05 و ستون 00 تا سطر ۱۲ و ستون ۷۹ عدد 0500H را در CX و 0c4FH را در DX بار کنید:

دقت کنید که از روی اشتباه مکان سمت راست تحتانی صفحه نمایش را بیشتر از 184FH قرار ندهید. تابع 09H از INT 21H برای نشان دادن اطلاعات روی صفحه نمایش

برای این کار لازم است که یک رشته نمایشی را در ناحیه داده ها تعریف کرده و بلافاصله پس از آن یک علامت دلار قرار داد که از آن بعنوان پایان رشته نمایش استفاده می کند.

CUSTMSG DB , customer name ? "\$"

می توانید علامت دلار را بلافاصله پس از رشته نمایشی به صورت \$customer name? یا در خط بعدی به صورت \$ DB بنویسید. اما ضعفی که این تابع دارد این است که نمی توانید از تابع برای نمایش کاراکتر \$ روی صفحه نمایش استفاده کنید.

تابع 09H را در ثبات AH قرار داده ، از دستورالعمل LEA برای باز کردن آدرس رشته نمایشی در DX استفاده کرده و سپس دستورالعمل INT 21 H را بنویسید.

MOV AH,09H

LEA DX,Custmsg ,load address of prompt

INT 21H

دستورالعمل INT کاراکترها را از چپ به راست نمایش داده و در صورت مواجه شدن با علامت دلار پایان داده ها را تشخیص می دهد. این عمل محتوی ثبات ها را تغییر نمی دهد. یک رشته نمایشی داده شده که از سمت راست ترین ستون صفحه نمایش تجاوز می کند. بطور اتوماتیک از سطر بعدی ادامه یافته و در صورت لزوم صفحه نمایش را حرکت می دهد.

استفاده از تابع 09H برای نمایش کاراکترهای اسکی

اکثر ۲۵۶ کاراکتر اسکی به وسیله نمادهایی نشان داده می شوند که می توانند روی صفحه نمایش نمایش داده شوند. برخی از مقادیر مانند FFH,00H هیچ نماد قابل نمایشی ندارند و به صورت کاراکتر خالی ظاهر می شوند. هر چند که کاراکتر خالی اسکی واقعی برابر 20 H می باشد.

تابع 06 H از INT 10 H: حرکت دادن صفحه نمایش به بالا

در صورتی که برنامه ای متنی را روی صفحه نمایش نشان می دهد که از پایین ترین خط آن تجاوز می نماید، خط بعدی از اول صفحه نمایش نشان داده می شود. اما حتی اگر دستورالعمل INT ستون O را مشخص کرده باشد، خطوط جدید بطور نامناسبی تو گذاری شده و خطوط بعدی ممکن است به شکل بدی کج ظاهر شوند. راه حل این مشکل ، حرکت دادن صفحه نمایش می باشد بطوریکه خطوط نشان داده شده از بالا به بیرون حرکت کرده و خطوط خالی در پایین صفحه ظاهر می شوند. قبلا در فصل و از تابع 06 H برای پاک کردن صفحه نمایش استفاده کردید که با قرار

دادن مقدار صفر در AL کل صفحه نمایش به بالا حرکت می کرد و در واقع پاک می شد. با قرار دادن یک مقدار غیر صفر در AL تعدادی از خطوط به بالا حرکت می کنند. ثبات های زیر را بار کنید:

ستون : سطر آغازین = CX ، تعداد سطرها (00 برای صفحه کامل) = AL  
 ستون : سطر پایانی = DX ، صفت مشخصه = BH

مثال زیر یک مشخصه رنگی را تعیین کرده و کل صفحه نمایش را یک خط به بالا حرکت می دهد:

```
MOV    AX,0601H    ,Request Scroll upone Line
MOV    BH,61H     ,Bronn Background , Blue Foregronnd
MOV    CX,0000    ,From 00:00 throogh
MOV    DX,184FH   ,24:79-(Full screen)
INT    10H        ,call interrupt service
```

یک روش استاندارد برای حرکت دادن به اندازه یک خط به صورت زیر می باشد.

۱. برای تعیین موقعیت سطری مکان نما ، یک عنصر جدید مثلا به نام ROW تعریف کرده و مقدار اولیه آنرا برابر 0 قرار دهید.

۲. یک خط را نمایش داده و مکان نما را به خط بعدی حرکت دهید.

۳. کنترل کنید که آیا ROW نزدیک به آخر صفحه نمایش می باشد (CMP ROW,22)

۴. در صورت مثبت بودن پاسخ مقایسه فوق ، یک خط به بالا حرکت کرده و از ROW برای تنظیم مکان نما استفاده کرده و ROW را برابر 00 قرار دهید.

۵. در صورت منفی بودن پاسخ مقایسه فوق ، ROW را یک واحد افزایش دهید (INC ROW) ثبات

های CX و DX حرکت دادن هر قسمتی از صفحه نمایش را اجازه می دهند. دقت کنید که مقدار AL را با فاصله واقع در DX: CX هماهنگ کنید. ویژه زمانی که به بخشی از نمایش مراجعه می کنید. دستورالعمل های زیر پنجره ای را (با مشخصات مخصوص خود) به اندازه ۷ سطر و ۳۶ ستون ایجاد می کنند که مختصات سمت چپ فوقانی آن برابر ۲۵:۱۲، سمت راست فوقانی برابر ۵۴:۱۲، سمت چپ تحتانی برابر ۲۵:۱۸ و سمت راست تحتانی برابر ۵۴:۱۸ می باشد:

```
MOV    AX,0607H    ,request scroll 7 lins
MOV    BH,80H     ,cyan background,black foreground
```

```
MOV    CX,0C19 H    ,from    cow12,column25 tehrough
INT    10 H        ,call    int  errupt- service
```

این مثال حرکت دادن ۷ خط را مشخص می کند که همان مقدار فاصله بین سطرهای ۱۲ و ۱۸ می باشد لذا فقط پنجره مزبور پاک می شود. در هنگام ایجاد یک پنجره و حرکت دادن تمام سطرهای آن، روش معمول این است سطرها را به ترتیب یک سطر در هر زمان حرکت داد. چون مشخصه یک پنجره همچنان باقی می ماند تا زمانی که عمل دیگری آنرا تغییر دهد، می توانید در یک زمان صفات مختلفی را به پنجره های گوناگون نسبت دهید.

تابع 09 H از INT 10 H: نمایش بایت مشخصه یا کاراکتر در موقعیت مکان نما

این عمل، عمل مفیدی است که تعداد مشخصی از کاراکترها را در حالت متنی یا گرافیکی مطابق مشخصه داده شده نمایش می دهد. ثبات های زیر را مقدار دهی کنید:

صفت مشخصه=BL کاراکتر اسکی که نمایش داده می شود=AL

تعداد=CX شماره صفحه=BH

عدد واقع در CX تعداد دفعاتی را که این عمل بطور مکرر کاراکتر واقع در AL را نشان خواهد داد، مشخص می کند. مثال زیر یک مشخصه رنگی را تعیین کرده و C60 کاراکتر "چهره خوشحال" (01H) را نشان می دهد.

```
MOV    AH,09H      ,request    display
MOV    AL,01H      ,happy    face    for    display
MOV    BH,0        ,page    number    o    (normal)
MOV    BL,16H      ,blue    background    ,brown    foreground
MOV    CX,60       ,60repeated    characters
INT    10H        ,call    interrupt    service
```

عمل فوق مکان نما را حرکت نداده و یا به کاراکترهای بوق کامپیوتر (Bell) بازگشت به اول سطر، خط خور و Tab عکسالعمل نشان نمی دهد، ولی در عوض مبادرت به نمایش آنها به صورت کاراکترهای اسکی می کند.

نمایش دادن کاراکترهای مختلف به یک حلقه نیاز دارد، در حالت متنی و گرافیکی، وقتی کد نمایش از سمت راست ترین ستون تجاوز می کند، تابع 09 H بطور اتوماتیک عمل نمایش را از سطر بعدی و از ستون 00 ادامه می دهد. برای نمایش یک اعلان یا پیغام روتینی را بنویسید که CX را برابر 01

قرار داده و برای انتقال یک کاراکتر در هر زمان از حافظه به AL ایجاد حلقه می کند(از آنجا که CX اشغال می باشد نمی توانید به سادگی از دستورالعمل LOOP استفاده کنید)همچنین پس از نمایش هر کاراکتر از تابع 02H در 10H INT جهت حرکت دادن مکان نما به ستون بعدی استفاده کنید.

برنامه : پذیرش و نمایش اسامی

برنامه از کاربر می خواهد که اسمی را وارد کند و سپس این اسم را در مرکز صفحه نمایش نشان داده و بوق کامپیوتر بصدا در می آید. اگر برای مثال کاربر اسم Dana porter را وارد کند این برنامه اعمال زیر را انجام می دهد:

۱. ۱۱ را بر ۲ تقسیم کرده و از باقی مانده صرف نظر می کند  $11/2=5$

۲. این مقدار را از ۴۰ کم می کند  $40-5=35$

page 60,132

```
Title Ao9CTRM (EXE) Accept names ,Center on screen
.model small .stack 64
```

Data

```
Parlist label byte
Maxnlen DB 20
Actulen DB ?
KB name DB 21 DUP
Prompt DB name? , $
```

.Code .886

```
Alomain proc far ,MOV AX,@ data,MOV DS,AX
MOV ES,AX, CALL QLOCLR
```

عدد واقع در CX تعداد دفعاتی را که این عمل بطور مکرر کاراکتر واقع در AL را نشان خواهد داد، مشخص می کند مثال زیر یک مشخصه رنگی را تعیین کرده و ۶۰ کاراکتر "چهره خوشحال"(01H) را نشان می دهد:

```
MOV AH,09H ,Request display
```

MOV AL,01H ,Happy face for display

MOV BH,0 ,page number 0 (normal)

MOV BL,16 H ,blue background , brown foreground

MOV CX,60 ,60 repeated characters

INT 10 H ,call interrupt service

عمل فوق مکان نما را حرکت نداده و یا به کاراکترهای بوق کامپیوتر (Bell) باز گشت به اول سطر خط خور و TAB عکس العمل نشان نمی دهد، ولی در عوض مبادرت به نمایش آنها به صورت کاراکترهای اسکی می کند.

نمایش دادن کاراکترهای مختلف به یک حلقه نیاز دارد، در حالت متنی نه گرافیکی، وقتی که نمایش از سمت راست تمرین ستون تجاوز می کند، تابع 09 H بطور اتوماتیک عمل نمایش را از سطر بعدی و ستون 00 ادامه می دهد. برای نمایش یک اعلان یا پیغام، روتینی را بنویسید که CX را برابر 01 قرار داده و برای انتقال یک کاراکتر در هر زمان از حافظه به AL ایجاد حلقه می کند (از آنجا که CX اشغال می باشد نمی توانید به سادگی از دستورالعمل LOOP استفاده کنید) همچنین پس از نمایش هر کاراکتر، از تابع 02H در INT 10H جهت حرکت دادن مکان نما به ستون بعدی استفاده کنید.

A20LOOP:

MOV DX,0000 ,set wsr or to 00,00

Call Q20CURS

Call B10 PRMPT ,display-prompt

Call C10 INPT ,provide for input of name

Call Q10 CLR ,clear screen

```

CMP     ACTULEN,OO     ,name entered?

JE      A30            ,no,exit

Call    D10 CODE      , setbell  ande $

Call    E10 CENT      ,cinter,display name

JMP     A20 LOOP

A30:

MOV     AX,4COOH ?    ,end processing

INT     21 H

A10 MAIN          ENDP

                                Display  Prompt
                                PROC      Near

B10 PROMPT

                                MOV      AH , 09H      ,Request  display

                                Lea      DX,PROMPT

                                INT      21H

                                RET

                                B10 Prompt      Endp
                                Accept+input of name

C10 INPT          Proc      near
                                MOV      AH,OAH      ,Request  Key board
                                Lea      DX,PARlist    ,input
INT     21 H      ...      RET      .... C10INPT      Endp

```

```

                Set bell and $ delimiter
D10 Code      PROC      near
                Movsx    BX,Actulen  Replace  ODH  with  0H
                Mov      KB name [BX],0
                Mov      KB name[BX+1] $
                Ret
D10CCode      Endp

```

```

                Center and display name:
E10cent      PROC      near

                MOV      DL,Actulen  ,locate center column

                Shir     DL,1        ,divide lenth by 2

                NEG      DL          ,reverse sign

                ADD      DL,40       ,add 40

                MOV      DH,12      ,center row

                Call     Q20coRs    ,set cursor

                MOV      AH,09H

                LEA      DX,kbname  ,display name

                INT      21 H
                RET
E10Cent      Endp

```

```

                Clear Screen
Q10CLR      PROC      Near

                MOV      AX,0600H    ,Request scroll screen

                MOV      BH,30

```

```
MOV CX,0000
```

```
MOV DX,184FH
```

```
INT 10H
```

```
RET
```

```
Q10 CLR Endp
```

```
Set eursor row co umn
```

```
Q20CUPRS PROC near ,DX set on entry
```

```
MOV AH,02H ,Request set cursor
```

```
MOV BH,00 ,page #00
```

```
INT 10H
```

```
RET
```

```
Q20CURS ENDP
```

```
End A10 main
```

در روال E10 CENT دستورالعمل SHR طول ۱۱ را یک بیت به راست شیفت می دهد و در حقیقت بر ۲ تقسیم می کند.

دستورالعمل NEG علامت را برعکس کرده و آنرا از +۵ به -۵ تغییر می دهد. دستورالعمل ADD مقدار ۴۰ را به DL اضافه می کند تا مکان شروع ستون یعنی ۳۵ در DL بدست آید با قرار دادن مکان نما در سطر ۱۲ و ستون ۳۵ اسم مورد نظر به صورت زیر روی صفحه نمایش ظاهر می شود.

۱۲ سطر: Dana porter

به دستورالعملهای داخل روال D10Ccode که کاراکتر بوق کامپیوتر (01H) را در ناحیه ورودی بلافاصله بدنبال اسم مزبور درج می کنند، توجه کنید:

```
MOV ZX BX,Actulen
```

```
MOV Kbname[BX],01H
```

دستورالعمل MOVzx در BX تعداد کاراکترها را قرار می دهد. دو دستورالعمل MOV مشخص کننده اندیس در داخل گروه ها بدین معنی است که BX به عنوان یک ثابت اندیس برای آدرس دهی توسعه یافته بکار می رود. دستورالعمل mov طول واقع در bx را با آدرس KB name ترکیب کرده

و 07H را به آدرس محاسبه شده انتقال می دهد. برای طول ۱۱، این دستورالعمل 07H را در بایت KB name+11 بدنبال اسم مزبور وارد می کند (کد جایگزین کاراکتر Enter می شود). آخرین دستورالعمل در روال D10 code یک علامت \$ را بدنبال 07H درج می کند تا تابع 09H از INT 21 H بتواند اسم مزبور را نمایش داده و بوق کامپیوتر را به صدا در آورد.

این برنامه تا زمانی ادامه می یابد که کاربر فقط کلید <Enter> را به عنوان پاسخی به اعلان برنامه فشار دهد. تابع 09 H از INT 21 H آنرا پذیرفته و طول 00H را به صورت زیر در لیست پارامتر درج می کند:

اگر طول اطلاعات ورودی برابر صفر باشد، برنامه تشخیص می دهد که اطلاعات ورودی پایان یافته است، که این کار توسط دستورالعمل 00, cmp Actulen در بخش A20 loop صورت می گیرد.

### تابع OAH از INT 21 H برای ورود اطلاعات از صفحه کلید

تابع OAH از INT 21 H برای خواندن اطلاعات از صفحه کلید تابع قدرتمندی می باشد. ناحیه ورود برای کاراکترهای ورودی به یک لیست پارامترها نیاز دارد که شامل فیلدهای مشخصی است که دستورالعمل INT پردازش خواهد کرد. در ابتدا این دستورالعمل نیاز دارد که طول حداکثر کاراکترهای ورودی را بداند. هدف از آن، جلوگیری کاربران از وارد کردن کاراکترهای بیش از حد می باشد. این عمل موجب می شود که بوق کامپیوتر بصدا در آمده و کاراکترهای اضافه را قبول نکند. سپس این دستورالعمل تعداد بایتهایی را که بطور واقعی وارد شده اند، به لیست پارامترها ارسال می کند. لیست پارامترها از عناصر زیر تشکیل می شود:

۱. عنصر اول نشان دهند نام لیست پارامترها بصورت ABEL BYTE می باشد.

**LABEL** دستوری است با مشخصه بایت که به سادگی موجب تنظیم روی مرز بایت می شود. چون این تنظیم به صورت عادی صورت می گیرد، اسمبلر شمارنده مکان خود را تغییر نمی دهد. با استفاده از دستور **LABEL** می توانید اسمی را به لیست پارامترها نسبت دهید.

۲. بایت اول لیست پارامترها شامل عددی برای حداکثر تعداد کاراکترهای ورودی می باشد. چون این یک فیلد یک بایتی می باشد، حداقل برابر **O** و حداکثر برابر **FFh** یا **۲۵۵** می باشد.

بر اساس نوع داده هایی که انتظار دارید کاربران وارد کنند، روی فیلد حداکثر تصمیم گیری کنید.

۳. بایت دوم برای این است که این دستورالعمل تعداد واقعی کاراکترهای تایپ شده را به صورت یک مقدار دو رویی در آن ذخیره کند.

۴. بایت سوم فیلدی را شروع می کند که شامل کاراکترهای تایپ شده از چپ به راست خواهد بود. این دستورالعمل منتظر می ماند تا کاربر کاراکترهایی را تایپ کند و کنترل می کند که تعداد اینها از تعداد حداکثر ۲۰ تجاوز نکند. این عمل همچنین کاراکتر **Enter** را به فیلد ورودی انتقال می دهد ولی آن را در شمارش طول واقعی اطلاعات ورودی در نظر نمی گیرد.

این عمل از کلید های تابع توسعه یافته نظیر **F1** , **home** , **pgup** و کلیدهای جهت صرف نظر می کند.

هندل فایل ها

این بخش استفاده از هندل فایل ها را برای عملیات صفحه نمایش و صفحه کلید که بیشتر در **OS** **12,unix** به کار می روند مورد بررسی قرار می دهد هندل یک فایل عددی است که به دستگاه ویژه ای اشاره می کند. با توجه به اینکه هندل فایل های استاندارد زیر از قبل تعریف شده اند نیازی به تعریف آنها ندارید.

هندل

دستگاه

ورودی، معمولاً صفحه کلید **(con)**، ولی ممکن است تغییر جهت داده شود

00

- 01 خروجی، معمولا صفحه نمایش (con)، ولی ممکن است تغییر جهت داده شود.
- 02 خروجی خطا، صفحه نمایش (con) نمی توان آنرا تغییر جهت داد.
- 03 دستگاه کمکی (AVX)
- 04 چاپگر (LPT1—PRN)

همانطور که ملاحظه می کنیم، هندل های عادی فایل عبارتند از: 01,00 هندل فایل ها برای سرویس های دیسک باید توسط برنامه شما تعیین شوند. همچنین می توانید از این سرویس ها برای تغییر جهت ورودی و خروجی به دستگاههای دیگر استفاده کنید.

### تابع 40h از INT 21h برای نشان دادن اطلاعات در روی صفحه نمایش

این تابع از هندل فایل ها برای درخواست نمایش اطلاعات استفاده می کند برای درخواست این سرویس ثباتهای زیر را بار کنید.

AH=40h تابع

CX=تعداد کاراکترهایی که نمایش داده می شوند

BX=01 هندل فایل

DX=آدرس ناحیه شمارش

یک عملیات موفقیت آمیز INT فلگ رقم نقلی را برابر یک کرده و کد خطای مربوطه را در AX برمی گرداند:

05H = دستیابی غیر مجاز (برای یک دستگاه غیر معتبر یا قطع ارتباط) یا

06H = هندل غیر معتبر چون AX می تواند شامل طول یا یک کد خطا باشد، تنها راه تشخیص

وضعیت خطا کنترل فلگ رقم نقلی می باشد هر چند که این خطاها بندرت اتفاق می افتد:

JC error-routine , test for display error

این تابع در مورد کاراکترهای کنترل 07H (بوق) (back spacs) 08H, (oalt, (خط خور) و 0DH (بازگشت به اول سطر) دقیقاً مثل تابع 09H از INT 21H عمل می کند.

Cotitle	DB	Intertech corp ,	ODH,OAH	,Dispiay area
	MOV	AH,40H		,kequest display
	MOV	BX,01		,File handle for screen
	MOV	ex,17		,Display 17 characters
	Lea	BX,Cotitle		,Display area
	INT	21 H		,Call interrupt service

توجه کنید که طول Cotitle شامل OAH,ODH می باشد

### تابع 3FH از INT 21 H برای ورود اطلاعات از صفحه کلید

این تابع از هندل یک فایل برای درخواست ورودی از صفحه کلید استفاده می کند هر چند که این کار تا اندازه ای ناهنجار می باشد. ثبات های زیر را بار کنید:

تابع AH=3FH

هندل فایل BX=00

حداکثر تعداد کاراکترهای قابل قبول CX=

آدرس ناحیه ای جهت وارد کردن کاراکترها=DX

یک عمل INT موفقیت آمیز فلگ رقم نقلی را صفر کرده و تعداد کاراکترهای وارد شده را در AX قرار می دهد.

یک عمل INT غیر موفقیت آمیز می تواند به دلیل هندل غیر معتبر اتفاق بیافتد در نتیجه فلگ رقم نقلی را یک کرده و کد خطای مربوطه را در AX درج می کند: 05H = دسترسی غیر مجاز یا 06H = هندل غیر معتبر. از آنجا که AX می تواند شامل طول یا یک کد خطا باشد تنها راه تشخیص وضعیت خطا، کنترل فلگ رقم نقلی می باشد، هر چند که خطاهای صفحه کلید بندرت اتفاق می افتند.

مانند تابع Oah از INT 21h تابع 3Fh نیز کاراکتر (back spaces) را قبول می کند ولی از کلیدهای تابع توسعه یافته صرفنظر می کند که وسعت عملکرد آنرا به شدت محدود می کند.

KB nput	DB	20DUP
MOV	AH,3FH	,Request keyboard input
MOV	BX,00	,File handle for key board
MOV	CX,20	,maximum 20 characters
Lea	DX,KBInput	,Input area
INT	21 H	,call interrupt service

این تابع منتظر می ماند تا کاراکترهایی را وارد کنید، ولی متاسفانه کنترل نمی کند که آیا تعداد کاراکترها از تعداد ماکزیمم واقع در ثبات CX تجاوز می کند. فشار دادن کلید Enter پایان ورودی را مشخص می کند.

بعد از کاراکترهای تایپ شده بلافاصله یک کاراکتر **Enter (ODH)** قرار دارد که شما تایپ کرده اید، و نیز یک کاراکتر خط خور (**Oah**) که شما تایپ نکرده اید به همین دلیل برای حداکثر تعداد طول ناحیه ورودی که تعریف می کنید، بایستی این دو کاراکتر اضافی را در نظر بگیرید. اگر کاراکترهای کمتری از حداکثر تعداد وارد کنید، مکانهای حافظه که بدنبال کاراکتر تایپ شده قرار دارند، هنوز محتوی قبلی خود را حفظ خواهند کرد. یک عمل **INT** موفقیت آمیز فلگ رقم نقلی را پاک کرده و در **AX** تعداد کاراکترهای وارد شده را قرار می دهد.