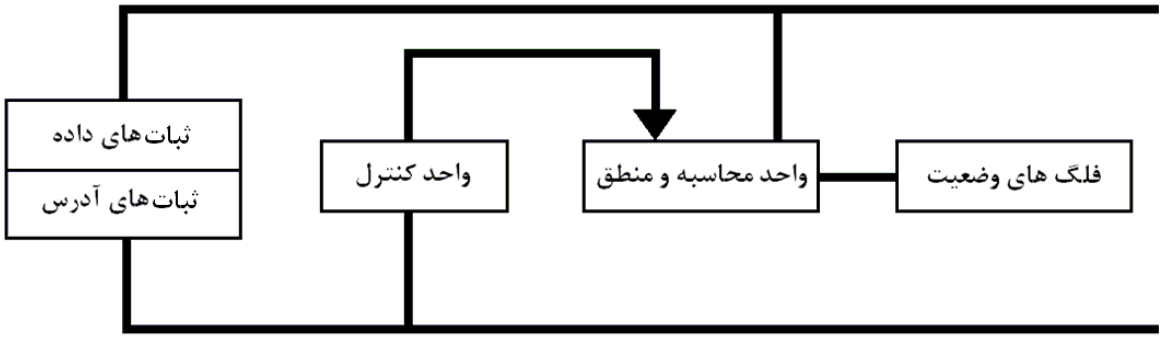
جلسه اول : مقدمات

برنامه‌نویسی به زبان اسمبلی مستلزم آشنایی مختصر با ساختمان کامپیوتری است که می‌خواهیم برای آن کد بنویسیم. هر کامپیوتر معماری خاص خود را داشته و زبان اسمبلی مختص خود را نیز دارد. در اینجا هدف یادگیری زبان اسمبلی پردازنده‌ی ۸۰۸۶ و نسل‌های بعدی آن می‌باشد لذا به بررسی ساختمان این پردازنده می‌پردازیم.

شرکت اینتل یکی از معروف‌ترین شرکت‌ها در زمینه ی ساخت پردازنده می‌باشد. پردازنده‌های ۸۰۸۶ ، ۸۰۱۸۶ ، ۸۰۲۸۶ ، ۸۰۳۸۶ و ۸۰۴۸۶ که در سال‌های گذشته به وفور مورد استفاده قرار می‌گرفتند ، ساخته‌ی این شرکت می‌باشند. اینتل پس از پردازنده‌های فوق، نسل جدیدی از پردازنده‌ها به نام پنتیوم و پنتیوم‌پرو را روانه‌ی بازار کرد که امروزه این کامپیوترها مورد استفاده قرار گرفته و نسل‌های قبلی خویش را از دور خارج کرده‌اند. لازم به ذکر است که هرکدام از این پردازنده‌ها با نسل قبلی خویش سازگاری کامل دارند، به این معنا که تمام دستورات نسل قبلی خویش را می‌شناسند و قادر به اجرای آن‌ها هستند.

پردازنده که قلب سیستم می‌باشد، وظیفه ی اجرای دستورات را به عهده دارد. اجرای هر دستورالعمل شامل سه بخش زیر است :

- واکشی[1] دستورالعمل : خواندن کد دستورالعمل از حافظه ی اصلی
- رمزگشایی[2] دستورالعمل : تعیین اینکه کد خوانده‌شده از حافظه مربوط به چه دستورالعملی است (جمع ، ضرب ، تقسیم ، انشعاب و ....) و واکشی عملوندهای[3] آن از حافظه
- اجرای[4] دستورالعمل : انجام محاسبات مورد نیاز ، ذخیره ی نتیجه در حافظه و ....



نمای ساده‌ای از پردازنده

[1] Fetch
[2] Decode
[3] Operand fetch
[4] Execute

شکل صفحه ی قبل ، بخش‌های اصلی یک پردازنده را نشان می‌دهد.

- واحد محاسبه و منطق[1] : جایی است که تمام محاسبات ریاضی و منطقی نظیر جمع، تفریق، ضرب، تقسیم، شیفت ، چرخش[2] ، And ، Or ، Not و ... را انجام می‌دهد.
- واحد کنترل[3] : کنترل داخلی پردازنده ، رمزگشایی دستورات و فراهم کردن داده‌های مورد نیاز واحد محاسبه و منطق و ... را به عهده دارد. دستورات را از حافظه خوانده ، داده‌ها را فراهم کرده و در اختیار واحد محاسبه و منطق قرار می‌دهد آنگاه به ALU فرمان می‌دهد که چه کاری را انجام دهد.
- فلگ‌های وضعیت : وضعیت پردازنده یا نتیجه عملیات محاسباتی را نشان می‌دهد. مانند نقلی ، سرریز ، علامت ، توازن[4] و ...
- ثبات‌ها[5] : حافظه‌هایی بسیار سریع در داخل پردازنده هستند که مستقیما به واحد کنترل و واحد محاسبه و منطق متصل هستند. از آنجا که دستیابی به ثبات‌ها بسیار سریع‌تر از دستیابی به حافظه‌ی اصلی است ، دستوراتی که فقط از ثبات‌ها استفاده می‌کنند بسیار سریع‌تر از دستوراتی که عملوندهای آن‌ها در حافظه‌ی اصلی است اجرا می‌شوند.
- گذرگاه داده و آدرس : گذرگاه داده‌ی داخلی مجموعه‌ای از سیم‌های موازی است که داده‌ها را بین بخش‌های مختلف پردازنده انتقال می‌دهد. وقتی داده‌ها از حافظه‌ی اصلی خوانده می‌شوند ، واحد کنترل آدرس آن را محاسبه کرده آن را در گذرگاه آدرس قرار می‌دهد . واحد حافظه گذرگاه آدرس را می‌خواند و داده‌ی نظیر آن را در گذرگاه داده قرار می‌دهد ، سپس سیگنالی به پردازنده‌ی مرکزی می‌فرستد مبنی بر اینکه داده‌ی درخواست شده آماده است.

[1] ALU (Arithmetic & Logical Unit)
[2] Rotate
[3] CU (Control Unit)
[4] Parity
[5] Registers

تعاریف مربوط به حافظه

- بیت : کوچکترین واحد حافظه است که می‌تواند صفر یا یک باشد.
- بایت: مجموعه‌ی ۸ بیت را یک بایت می‌گویند.
- کلمه : مجموعه‌ای از دو بایت را کلمه می‌گویند.
- کلمه‌ی مضاعف: مجموعه‌ای از دو کلمه (۴ بایت) را کلمه‌ی مضاعف می‌گویند.
- چهار کلمه‌ای : مجموعه‌ای از ۴ کلمه (۸ بایت)را چهارکلمه‌ای می‌گویند.
- پاراگراف: مجموعه‌ی ۱۶ بایت را پاراگراف می‌گویند.
- کیلوبایت (KB) : هر ۱۰۲۴=2^{10} بایت را یک کیلوبایت گویند.
- مگابایت (MB) : هر ۱۰۴۸۵۷۶=2^{20} بایت را یک کیلوبایت گویند.
- گیگا بایت (GB) : هر 2^{30} بایت را یک گیگابایت گویند.
- ترابایت (TB) : هر 2^{40} بایت را یک ترابایت گویند.

نکته : با n بیت آدرس ، می‌توان 2^n بایت را آدرس دهی کرد.

مثال: با یک رجیستر ۱۶ بیتی ، چه مقدار حافظه را می‌توان آدرس دهی کرد؟

$2^{16} = 2^6 \times 2^{10} = 2^6 KB = 64\ KB$

مثال : برای آدرس دهی ۵۱۲ مگابایت به چند بیت آدرس نیاز داریم ؟

$512\ MB = 512 \times 2^{20} = 2^9 \times 2^{20} = 2^{29}$ ➔ ۲۹ بیت آدرس نیاز داریم

ثبات‌های پردازنده ی ۱۶ بیتی

درون پردازنده ی ۸۰۸۶ تعدادی ثبات ۱۶ بیتی وجود دارد که هرکدام دارای نام مشخصی بوده و برای مقاصد مختلفی مورد استفاده قرار می‌گیرند. برای مراجعه به هر ثبات از نام آن استفاده می‌شود. ثبات ها به چند دسته تقسیم می‌شوند که عبارتند از : ثبات‌های عمومی ، ثبات‌های سگمنت ، ثبات‌های اندیس و ثبات‌های وضعیت و کنترل.

ثبات‌های عمومی : عبارتند از AX , BX, CX, DX .

ثبات AX : به انباشتگر معروف است و در اعمالی که نیاز به ورودی-خروجی و محاسبات زیاد است مورد استفاده قرار می‌گیرد. این ثبات به دو قسمت ۸ بیتی با نام‌های AL و AH تقسیم می‌شود.

ثبات BX : به ثبات پایه معروف است و به عنوان اندیس برای توسعه آدرس مورد استفاده قرار می‌گیرد. علاوه بر این در محاسبات نیز می‌توان از این ثبات استفاده کرد. این ثبات به دو قسمت ۸ بیتی با نام‌های BL و BH تقسیم می‌شود.

ثبات CX : این ثبات ، ثبات شمارنده نامیده شده است و برای کنترل تعداد دفعات تکرار حلقه و تعداد شیفت‌ها در عمل شیفت مورد استفاده قرار می‌گیرد. همچنین در انجام محاسبات می‌توان از این ثبات استفاده کرد. این ثبات نیز به دو بخش ۸ بیتی CL و CH تقسیم شده است.

ثبات DX : به ثبات داده معروف است و در انجام اعمال ضرب و تقسیم که با اعداد بزرگ سروکار دارند مورد استفاده قرار می‌گیرد. در محاسبات نیز می‌توان از این ثبات استفاده کرد. این ثبات به دو بخش ۸ بیتی DL وDH تقسیم شده است.

ثبات‌های سگمنت :

تعریف سگمنت : سگمنت ناحیه‌ای از حافظه است که آدرس شروع آن بر ۱۶ قابل تقسیم بوده و اندازه‌ی آن می‌تواند تا ۶۴ کیلوبایت باشد.

هر برنامه‌ی اسمبلی از چندین سگمنت ساخته می‌شود که این سگمنت‌ها می‌توانند ۴ نوع داشته باشند: سگمنت کد[1] ، سگمنت داده‌ها[1] ، سگمنت پشته[2] و سگمنت اضافی[3].

[1] Code segment (CS)

سگمنت کد : دستورالعمل‌های زبان ماشین که باید اجرا شوند ، به ترتیب در این سگمنت قرار می‌گیرند. اگر کد برنامه از یک سگمنت بزرگتر باشد ، برنامه می‌تواند چندین سگمنت کد داشته باشد.

سگمنت داده : قسمتی از برنامه است که داده‌ها (متغیرها) در آنجا قرار دارند. یک برنامه می‌تواند چندین سگمنت داده داشته باشد.

سگمنت پشته : حاوی آدرس برگشت از زیر برنامه‌ها است. همچنین می‌توان با دستورات Push و Pop داده‌هایی را در این ناحیه ذخیره کرد یا از آن خواند.

سگمنت اضافی: این سگمنت برای انجام امور خاص نظیر نوشتن در حافظه‌ی گرافیک سیستم و انجام عملیات بر روی رشته‌ها و ... به کار می‌رود.

ثبات‌های سگمنت عبارتند از: CS ، DS ، SS و ES که هرکدام به ترتیب آدرس شروع سگمنت کد ، داده ، پشته و اضافی را نگهداری می‌کنند.

ثبات‌های اندیس : عبارتند از BP ، SP ، SI و DI

ثبات BP[۴] : این ثبات حاوی آفستی از ثبات SS است . چنانچه پارامترهای زیر برنامه‌ها از طریق پشته ارسال شوند ،این پارامترها از طریق ثباتBP قابل بازیابی خواهند بود.

ثبات SP[۵] : حاوی آفست بالای پشته است. در واقع این ثبات مشخص می‌کند که پشته تا کجا پر شده‌است.

ثبات‌های SI[۶] و DI[۷] : این ثبات‌ها توسط دستورات کپی رشته‌ها مورد استفاده قرار می‌گیرند ، ثبات SI آدرس رشته‌ی منبع و ثبات DI آدرس رشته‌ی مقصد را نگهداری می‌کند. به همین دلیل به آن‌ها ثبات اندیس منبع و ثبات اندیس مقصد می‌گویند.

ثبات‌های وضعیت و کنترل: عبارتند از ثبات IP و ثبات فلگ‌ها

---

[1] Data segment (DS)
[2] Stack segment (SS)
[3] Extra segment (ES)
[4] Base Pointer
[5] Stack Pointer
[6] Source Index
[7] Destination Index

ثبات IP : حاوی آفست دستوری است که باید اجرا شود (در سگمنت کد). هر دستوری که اجرا می‌شود ، مقدار ثبات IP به اندازه‌ی طول آن دستور افزایش می‌یابد تا به این ترتیب به دستور بعدی اشاره نماید. در دستورات پرش و انشعاب ، مقدار IP به اندازه‌ی پرش مورد نظر کاهش یا افزایش می‌یابد.

ثبات فلگ‌ها : دارای بیت‌هایی است که وضعیت پردازنده‌ی مرکزی یا نتیجه‌ی محاسبات را نشان می‌دهند. برخی از این بیت‌ها عبارتند از :

بیت C : در صورتی که انجام محاسبات منجر به تولید رقم نقلی شود ، این بیت برابر ۱ خواهد شد.

بیت D : برای تعیین جهت در اعمال رشته‌ای مثل مقایسه یا انتقال و ... به کار می‌رود. اگر این بیت ۱ باشد ، اعمال مذکور از راست به چپ و در غیر این صورت از چپ به راست انجام می‌شوند.

بیت P : بیت توازن است.

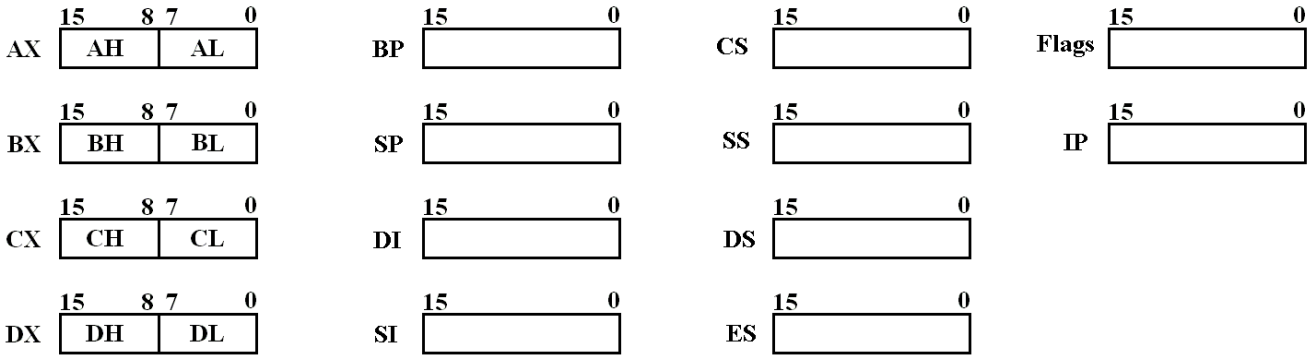
بیت A : در صورتی که در انجام محاسبات ۸ بیتی ، رقم نقلی در بیت چهارم ایجاد شود ، این بیت برابر یک خواهد شد ، به این بیت نقلی کمکی می‌گویند.

بیت Z : چنانچه نتیجه ی انجام محاسبات صفر باشد ، این بیت برابر یک خواهد شد.

بیت S : بیت علامت است. در صورتی که نتیجه‌ی محاسبات عددی منفی باشد ، این بیت برابر یک خواهد شد.

بیت I: چنانچه این بیت برابر ۱ باشد ، سیستم به وقفه[1]ها پاسخ خواهد داد. در غیر این صورت وقفه‌ها نادیده گرفته می‌شوند.

بیت O : در صورت رخداد سرریز در انجام محاسبات ، این بیت برابر یک خواهد شد.



ثبات‌های ۱۶ بیتی

[1] Interrupt

تمارین

۱- مراحل اجرای یک دستورالعمل را توضیح دهید .
۲- به نظر شما ، چرا طول هر سگمنت نهایتا می‌تواند ۶۴ کیلوبایت باشد؟
۳- برای آدرس دهی ۱ مگابایت حافظه ، به چند بیت آدرس نیاز داریم ؟
۴- برای آدرس دهی ۴ گیگا بایت حافظه ، به چند بیت آدرس نیاز داریم ؟
۵- با ۱۲ بیت چند مگابایت حافظه قابل آدرس دهی است؟
۶- پردازنده ۸۰۸۶ توانایی آدرس دهی یک مگابایت حافظه را دارد. با توجه به ثبات‌های ۱۶ بیتی این پردازنده ، تحقیق کنید که چگونه می‌تواند این حجم حافظه را آدرس دهی کند.
۷- پردازنده‌ی ۸۰۳۸۶ که از نسل‌های بعدی ۸۰۸۶ است ، یک پردازنده‌ی ۳۲ بیتی می‌باشد. در مورد ثبات‌های این پردازنده تحقیق کنید، نام و طول هریک را بنویسید.

جلسه دوم : شروع برنامه‌نویسی

شناسه

به عناصر برنامه‌ی اسمبلی نظیر برچسب‌ها ، نام زیر برنامه‌ها ، نام متغیرها و ... شناسه می‌گویند. برای نامگذاری شناسه می‌توان از ترکیب حروف a تا z ، A تا Z ، ارقام ۰ تا ۹ و کاراکترهایی مثل ? ، $ ، زیرخط و نقطه استفاده نمود به طوری که ارقام و نقطه در ابتدا نیایند. مثال : Sum ، $Price ، t_1

قالب دستورات

قالب کلی دستورات اسمبلی به فرم زیر است

[توضیحات ;] [عملوندها] دستورالعمل [شناسه]

شناسه می‌تواند آدرس یک قلم داده یا یک دستورالعمل ، زیربرنامه یا یک سگمنت باشد. شناسه‌ها برای ارجاع مورد استفاده قرار می‌گیرند. مثلا در دستورات پرش یا فراخوانی زیربرنامه‌ها. آوردن شناسه اختیاری است و در صورت نیاز می‌توان از آن استفاده نمود.

دستورالعمل مشخص می‌کند که کامپیوتر چه کاری را انجام دهد. (پرش ، مقایسه ، جمع ، ضرب و ...)

نکته : برخی از دستوراتی که در یک برنامه اسمبلی می‌آیند ، *راهنمای اسمبلر*[1] هستند و به کد زبان ماشین ترجمه نمی‌شوند.

عملوند : برخی دستورات اسمبلی دارای عملوند نمی‌باشند. در صورتی که دستورالعمل دارای عملوند باشد ، این عملوندها در قسمت مشخص شده می‌آیند. برخی دستورات دارای یک عملوند و برخی دیگر دارای دو عملوند می‌باشند. به طور کلی در دستورات دو عملوندی ، ابتدا عملوند مقصد و سپس عملوند مبدا می‌آید.

[توضیحات ;] <عملوند مبدا> , <عملوند مقصد> دستورالعمل [شناسه]

توضیحات : این قسمت با علامت سمی‌کالن[2] شروع شده و آوردن آن نیز اختیاری است. آوردن توضیحات تنها به خوانایی برنامه کمک می‌کند. از آنجا که دستورات اسمبلی دارای خوانایی کمی هستند می‌توانید با نوشتن توضیحات کافی و مناسب در مقابل دستورات به خوانایی آن‌ها کمک کنید. توضیحات را می‌توانید در هر جایی از برنامه بنویسید. (حتی در یک خط جدید)

---

[1] Directive
[2] Semicolon

تعریف سگمنت‌ها

همانطور که در جلسه‌ی اول گفته شد ، هر برنامه اسمبلی از چندین سگمنت تشکیل شده است. این سگمنت‌ها با استفاده از راهنمای اسمبلر Segment و به شکل زیر تعریف می‌شوند.

نام سگمنت Segment پارامترها

...

نام سگمنت ends

نام سگمنت می‌تواند هر نام معتبری که از قوانین نامگذاری شناسه‌ها تبعیت می‌کند باشد.

پارامترهایی که در دستور Segment استفاده می‌شوند بر سه نوع‌اند: پارامتر تنظیم ، پارامتر ترکیب و پارامتر کلاس. ترتیب آمدن پارامترها به شکل زیر است:

نام سگمنت Segment [پارامتر تنظیم] [پارامتر ترکیب] ['پارامتر کلاس']

پارامتر تنظیم : مرزی را که سگمنت باید از آنجا شروع شود را مشخص می‌کند و می‌تواند یکی از مقادیر Byte ، Word ، Para و Page را اختیار کند. چنانچه از نوشتن این پارامتر صرف نظر کنید اسمبلر آن را Para در نظر می‌گیرد.

پارامتر ترکیب : مشخص می‌کند که آیا این سگمنت با سگمنت‌های دیگری که پس از ترجمه برنامه به آن پیوند داده می‌شوند ترکیب شود یا خیر و می‌تواند یکی از مقادیر None ، Public ، Stack ، Common و AT را اختیار کند.

پارامتر کلاس : می‌تواند مقادیر ’Code‘ ، ’Data‘ و ’Stack‘ را بپذیرد و مشخص کند که محتوی آن سگمنت چه جیزی است. مثلا با آمدن مقدار ’Code‘ پیوند دهنده انتظار دارد که آن سگمنت حاوی کد باشد.

مثال : تعریف سگمنت‌های یک برنامه :

```
DataS   SEGMENT 'DATA'
  …
DataS   ENDS

StackS   SEGMENT STACK   'STACK'
  …
StackS   ENDS

CodeS   SEGMENT 'CODE'
CodeS   ENDS
```

تعریف رویه[1] در سگمنت کد

سگمنت کد از یک یا چند رویه تشکیل می‌شود. در واقع برای نوشتن دستورات ، باید آن‌ها را درون یک رویه نوشت. هر برنامه حداقل دارای یک رویه‌ی اصلی است، اجرای برنامه از اولین دستورالعمل موجود در این رویه آغاز می‌شود. این رویه را می‌توان مشابه با تابع main() در زبان C دانست. این رویه دارای نام مشخصی نیست و می‌تواند هر نامی را اختیار کند. به نظر شما اسمبلر چگونه این رویه را از سایر رویه‌ها تشخیص خواهد داد؟ پاسخ این سوال این است که برنامه‌نویس باید با استفاده از راهنمای اسمبلر end این روال را معرفی نماید. برای تعریف رویه از راهنماهای اسمبلر Proc و Endp استفاده می‌شود که به ترتیب شروع و خاتمه‌ی رویه را نشان می‌دهند. آنچه گفته شد در شکل زیر آمده است.

```
CodeSeg Segment 'Code'
main proc far
…                          تعریف رویه اصلی
main endp
Codeseg ends               تعیین رویه اصلی
    end main
```

نکته:رویه‌ی اصلی برنامه باید از نوع far تعریف شود.

تعریف داده‌ها در سگمنت داده

در زبان اسمبلی ۵ نوع داده می‌توان تعریف نمود که عبارتند از بایت ، کلمه ، کلمه مضاعف ، چهار کلمه‌ای و ده بایتی و به ترتیب با راهنماهای اسمبلر DB[2] ، DW[3] ، DD[4] ، DQ[5] و DT[6] تعریف می‌شوند. با چند مثال نحوه‌ی تعریف داده را نشان می‌دهیم :

مثال ۱ : تعریف متغیری از نوع بایت با نام X و مقدار اولیه‌ی ۱۰: `X DB 10`

مثال ۲ : تعریف متغیری از نوع کلمه با نام Y و مقدار اولیه‌ی نامشخص: `Y DW ?`

---

[1] Procedure
[2] Define Byte
[3] Define Word
[4] Define Double
[5] Define Quad Word
[6] Define Ten bytes

مثال ۳: تعریف آرایه‌ای از نوع بایت به طول ۳ با نام Arr Arr DB 10,20,30

مثال ۴ : تعریف آرایه‌ای از نوع بایت به طول ۲۰ با نام Buffer که تمام عناصر آن مقدار صفر دارند

Buffer DB 20 dup(0)

نکته : راهنمای اسمبلر Dup برای تکرار یک مقدار مشخص (مقدار ذکر شده درون پرانتز مقابل آن) به تعداد دفعات مورد نظر (عددی که قبل از آن آمده) مورد استفاده قرار می‌گیرد.

مثال ۶ : تعریف یک رشته با نام Str که از ۲۶ ستاره و ۲۵ خط فاصله در بین ستاره‌ها تشکیل شده‌است

Str DB 25 dup('*-'),'*'

دستورMOV

این دستور برای انتقال داده‌ها از یک محل به محل دیگر به کار می‌رود.

Mov مقصد ، مبدا

مثال ۱ : انتقال محتویات ثبات Ax به ثبات Bx : Mov Bx,Ax

مثال ۲: انتقال محتویات متغیر X به ثبات Ax : Mov Ax,X

مثال ۳: انتقال عدد 10h به ثبات Ch : Mov Ch,10h

چند نکته :

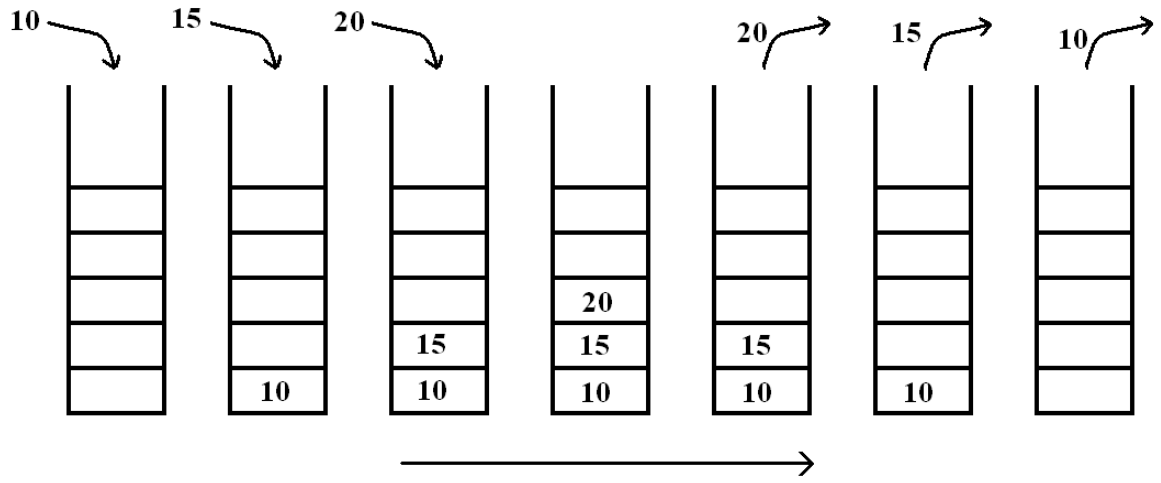
- *انتقال از حافظه به حافظه ممکن نیست* یعنی مستقیما نمی‌توان مقدار یک متغیر را به متغیر دیگر انتقال داد و باید از ثبات‌ها کمک گرفت به این ترتیب که مقدار متغیر اول را به یک ثبات منتقل کرد و سپس مقدار این ثبات را به متغیر دوم منتقل نمود.
- *انتقال از یک ثبات سگمنت به ثبات سگمنت دیگر ممکن نیست*
- *انتقال یک مقدار ثابت به ثبات‌های سگمنت ممکن نیست*
- *طول مبدا و مقصد باید یکسان باشد* یعنی نمی توان یک بایت را به یک کلمه یا یک کلمه را به یک بایت منتقل نمود و این کار منجر به خطا خواهد شد. در مثال ۲ ، باید متغیر X از نوع کلمه باشد چرا که طول AX ۱۶ بیت است.

دستورالعمل XCHG

این دستور دارای دو عملوند بوده و محتوی عملوندهای خود را با یکدیگر عوض می‌کند. داده‌ها می‌توانند از ثبات به حافظه ، از حافظه به ثبات و از ثبات به ثبات جابجا شوند. (یعنی حداقل یکی از عملوندها باید ثبات باشد). این دستور روش مناسبی برای تعویض دو عملوند است زیرا نیاز به حافظه یا ثبات کمکی ندارد و به همین دلیل به سرعت اجرای برنامه کمک می‌کند. (مثلا در مرتب‌سازی)

دستورات Push و Pop

پشته : ساختمان داده‌ای است که مقادیر به ترتیب عکس ورودشان به آن از آن خارج می‌شوند. به این‌چنین ساختمان داده‌ای LIFO[1] می‌گویند. فرض کنید اعداد ۱۰ ، ۱۵ و ۲۰ به ترتیب وارد پشته شده‌اند ، حال ترتیب خارج شدنشان از پشته به این صورت است : ابتدا ۲۰ ، سپس ۱۵ و در آخر ۱۰.



دستور Push برای قرار دادن یک عنصر به داخل پشته و دستور pop برای خارج کردن یک داده از پشته مورد استفاده قرار می‌گیرند. عملوندهای Push و Pop باید ۱۶ بیتی یا ۳۲ بیتی باشند.

[1] Last In First Out

تمارین

۱- دستوراتی بنویسید که اعمال زیر را انجام دهند

a. متغیری از نوع بایت با مقدار ۱۳۶ تعریف کنید
b. رشته‌ای با محتوی "this is a test" تعریف کنید
c. آرایه‌ای تعریف کنید که شامل ۲۵ علامت ستاره باشد.
d. رشته‌ای تعریف کنید که شامل ۵ بار تکرار نام خودتان باشد.

۲- در سگمنت داده‌ها متغیرهای x و y را از نوع کلمه و به ترتیب با مقادیر اولیه‌ی ۱۰۰۰ و ۲۰۰۰ تعریف کنید. یک بار فقط با استفاده از دستور mov و بار دیگر فقط با استفاده از دستور xchg محتوی آن‌ها را با یکدیگر عوض کنید.

۳- درست یا نادرست بودن دستورات زیر را تعیین کنید. علت نادرست بودن را ذکر کنید.

a. Mov al,bx
b. Mov ax,X ; X is a Word
c. Xchg X,Y ; X and Y are Words
d. Mov X,Y ; X and Y are Words
e. Mov cs,10h
f. Mov cs,ax
g. X DB 520
h. Mov al,300
i. Mov ax,10
j. Xchg al,150
k. Mov ah,150

۴- مقادیر دلخواه به AX و BX نسبت دهید و سپس با استفاده از پشته ، مقدار این دو ثبات را با یکدیگر تعویض نمایید.

جلسه سوم : اعمال محاسباتی دودویی

دستورات جمع و تفریق دودویی

Add عملوند دوم ، عملوند اول

Sub عملوند دوم ، عملوند اول

این دستورات برای انجام جمع و تفریق دودویی مورد استفاده قرار میگیرند. عملوند اول این دستورات حتما باید متغیر یا ثبات (بجز ثباتهای سگمنت) باشد و عملوند دوم میتواند ثبات ، متغیر یا یک مقدار ثابت باشد. توجه کنید که دو عملوند در یک دستور نمیتوانند متغیر باشد. عملوندهای این دستورات میتواند بایت ، کلمه یا کلمه مضاعف (در پردازنده ی ۳۲ بیتی) باشد.

تاثیر این دستورات بر فلگ‌ها : اگر حاصل این دستورات برابر صفر باشد ، فلگ ZF برابر یک می‌شود. اگر منفی باشد ، فلگ SF برابر یک می‌شود. در عمل جمع ، چنانچه نتیجه‌ی عملیات بقدری بزرگ باشد که در عملوند اول جا نشود ، CF برابر یک می‌شود. در عمل تفریق هنگامی‌که عملوند اول از عملوند دوم کوچکتر باشد ، CF برابر یک خواهد شد.

دستورات ADC و SBB

این دستورات برای محاسبات دودویی مضاعف بکار برده میشوند. ADC جمع به کمک بیت نقلی است. در این دستور ، رقم نقلی یعنی CF با عملوند اول جمع شده و سپس عملوند دوم به این حاصل اضافه میشود. فرض کنید میخواهیم دو عدد ۳۲ بیتی را با استفاده از ثباتهای ۱۶ بیتی جمع بزنیم ، برای اینکار چه باشد کرد؟ برای اینکار باید ابتدا ۱۶ بیت کم ارزش دو عدد ۳۲ بیتی را با استفاده از دستور Add و سپس ۱۶ بیت با ارزش اعداد ۳۲ بیتی را با استفاده از دستور ADC جمع بزنیم. با اینکار ، چنانچه جمع زدن ۱۶ بیت کم ارزش منجر به تولید رقم نقلی شده باشد ، این رقم نقلی به ۱۶ بیت بالا مرتبه منتقل خواهد شد. برای محاسبه ی حاصلجمع اعداد بزرگتر (مثلا ۶۴ بیتی) نیز میتوان از دستور ADC به صورت زنجیره‌ای استفاده نمود.

دستور SBB مشابه دستور ADC است با این تفاوت که برای تفریق استفاده می‌شود. به عنوان مثال برای تفریق کردن دو عدد ۳۲ بیتی ، می‌توان ابتدا ۱۶ بیت کم ارزش این اعداد را با استفاده از دستور sub از یکدیگر کم نمود و سپس ۱۶ بیت با ارزش آن‌ها را با دستور SBB تفریق کرد. چنانچه در دستور تفریق اول ، نیاز به رقم قرضی پیدا شده باشد ، دستور SBB این موضوع را در تفریق ۱۶ بیت با ارزش تاثیر خواهد داد.

مثال : می‌خواهیم حاصل جمع اعداد 0123BC62h و 0012553Ah را محاسبه نماییم.

برای اینکار ابتدا ۱۶ بیت کم ارزش این اعداد یعنی BC62h و 553Ah را با استفاده از دستور ADD جمع می‌زنیم و بلافاصله پس از آن با دستور ADC ، ۱۶ بیت بالا را جمع می‌زنیم.

```
Dseg segment ‘data’
  …
  Word1H dw 0123h
  Word1L dw 0BC62h
  Word2H dw 0012h
  Word2L dw 553Ah
  ResultH dw ?
  ResultL dw ?
  …
Dseg ends
```

```
Cseg segment ‘code’
  …
  Mov ax,Word1L
  Add ax,Word2L
  Mov ResultL,ax
  Mov ax,Word1H
  Adc ax,Word2H
  Mov ResultH,ax
  …
Cseg ends
```

دستور NEG

Neg عملوند

این دستور علامت عملوند خود را تغییر می‌دهد. (از مثبت به منفی و از منفی به مثبت) درواقع این دستور متمم دوی عملوند خود را محاسبه می‌کند. عملوند این دستور می‌تواند ثبات یا حافظه باشد.

دستور NOT

Not عملوند

این دستور تمام بیت‌های عملوند خود را از صفر به یک و از یک به صفر تبدیل می‌کند. (متمم یک)

ضرب اعداد دودویی

برای ضرب اعداد بدون علامت از دستور MUL و برای ضرب اعداد علامتدار از دستور IMUL استفاده می‌شود. این دستورات به شکل زیر مورد استفاده قرار می‌گیرند.

Mul عملوند IMul عملوند

عملوند هر دو دستور می‌تواند ثبات یا حافظه باشد. عمل ضرب در پردازنده‌های ۸۰۲۸۶ و به پایین می‌تواند بایت در بایت یا کلمه در کلمه باشد اما در پردازنده‌های ۸۰۳۸۶ و به بالا ، علاوه بر این‌ها ، ضرب کلمه‌ی مضاعف در کلمه مضاعف نیز امکان‌پذیر است.

عمل ضرب بین دو عدد انجام می‌شود و یک نتیجه تولید می‌کند. اما این دستورات دارای یک عملوند هستند، پس عملوند دیگر کجاست و نتیجه در چه مکانی ذخیره می‌شود؟ پاسخ این است که در این دستورات ، همیشه یکی از عملوندها ثبات Al یا Ax است و نتیجه در AX یا DX:AX ذخیره می‌شود. چنانچه عملوند جلوی این دستورات ، ۸ بیتی باشد ، عملوند دوم باید در ثبات Al باشد و نتیجه در ثبات Ax ذخیره خواهد شد و چنانچه ۱۶ بیتی باشد ، عملوند دوم باید در Ax باشد و نتیجه در Dx:Ax ذخیره خواهد شد.

مثال: دستور Mul bl را در نظر بگیرید. از آنجا که ثبات bl هشت بیتی است ، پردازنده ضرب al*bl را انجام خواهد داد و نتیجه را در Ax ذخیره خواهد کرد.

نکته : منظور از Dx:Ax ، یک حافظه‌ی ۳۲ بیتی است که از دو قسمت ۱۶ بیتی تشکیل شده است که ۱۶ بیت کم ارزش همان ثبات Ax و ۱۶ بیت با ارزش ثبات Dx می‌باشد.

تقسیم اعداد دودویی

برای انجام تقسیم بدون علامت از دستور Div و برای تقسیم داده‌های علامتدار از دستور IDiv استفاده می‌شود. برنامه نویس باید با توجه به داده‌های برنامه ، دستور مناسب را انتخاب نماید.

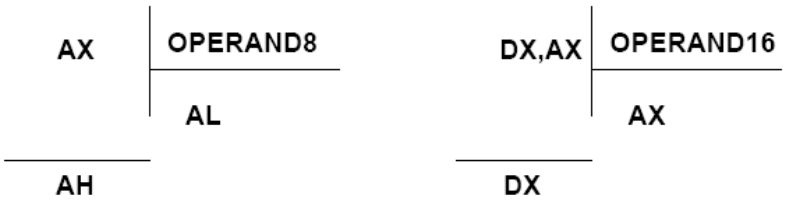
Div عملوند

IDiv عملوند

عملوند این دستورات می‌تواند ثبات یا حافظه باشد. این عملوند نماینده‌ی مقسوم‌علیه در دستور تقسیم می‌باشد. در پردازنده‌ی ۸۰۸۶ می‌توان کلمه را بر بایت و کلمه‌ی مضاعف را بر کلمه تقسیم نمود ، ولی در پردازنده‌ی ۸۰۳۸۶ می‌توان علاوه بر این‌ها ، تقسیم چهارکلمه‌ای بر کلمه‌مضاعف را نیز انجام داد.

تقسیم کلمه بر بایت: در این تقسیم ، مقسوم در ثبات AX و مقسوم علیه یک بایت از حافظه یا یک ثبات ۸بیتی است. نتیجه‌ی تقسیم که شامل باقیمانده و خارج قسمت می‌باشد به ترتیب در ثبات‌های Ah و Al ذخیره می‌شوند.

تقسیم کلمه‌ی مضاعف بر کلمه : در این تقسیم ، مقسوم در ثبات‌های Dx:Ax و مقسوم‌علیه در یک کلمه از حافظه یا یک ثبات ۱۶بیتی است. پس از انجام تقسیم ، باقیمانده در ثبات Dx و خارج قسمت در ثبات Ax قرار می‌گیرند.



دستورات INC وDEC :

این دستورات به فرم زیر به کاربرده می‌شوند:

INC عملوند DEC عملوند

عملوند این دستورات می‌تواند حافظه یا ثبات باشد. دستور INC به عملوند خودش یک واحد اضافه کرده و دستور DEC یک واحد از عملوند خودش کم می‌کند.

تمارین

۱- برنامه‌ای بنویسید که متمم دوی مقدار ۳۲ بیتی Dx:Ax را محاسبه نماید (مشابه دستور NEG)

۲- برنامه‌ای بنویسید که یک مبلغ مشخص را به سکه‌های ۲۵ ، ۱۰ ، ۵ و ۱ تومانی خرد نماید. برای این کار یک متغیر به نام Price برای مبلغ اولیه و ۴ متغیر با نام‌های C25 ، C10 ، C5 و C1 به ترتیب برای تعداد سکه‌های ۲۵ ، ۱۰ ، ۵ و ۱ تومانی تعریف کنید. برنامه‌ی خود را برای مبالغ ۲۶۷ تومان و ۲۵۵۷۷ تومان تست نمایید. در صورتی که پاسخ صحیح دریافت نکردید ، علت را بررسی کرده و برنامه‌ی خود را اصلاح نمایید

۳- برنامه‌ای بنویسید که درجه حرارت را از فارنهایت به سلسیوس تبدیل نماید.برای انجام اینکار از معاله‌ی C=(F-32)*(5/9) استفاده کنید. در این برنامه متغیرهایی با نام‌های F و C تعریف کنید تا درجه‌ی حرارت فارنهایت و معادل سلسیوس آن را نگهداری کنند. دقت در حد درجه قابل قبول است و نیاز به محاسبات اعشار نیست. برای تست برنامه ورودی‌های زیر را به برنامه داده و خروجی آن را بررسی نمایید. (ورودی ۳۲ ، خروجی صفر - ورودی ۱۴۰ ، خروجی ۶۰)

۴- برنامه‌ای بنویسید که با داشتن نمرات (صحیح) ۵ درس A ، B ، C ، D و E ، معدل آن‌ها را تا ۲ رقم اعشار محاسبه نماید. برای قسمت صحیح معدل متغیری با نام AVG و برای قسمت اعشار معدل متغیری با نام AVGD تعریف نمایید. (برای اینکار از تقسیمات متوالی باقیمانده تا دو مرحله استفاده نمایید)

۵- در مورد دستورات CBW و CWD تحقیق نمایید.

جلسه چهارم : ساختارهای تصمیم و حلقه‌های تکرار

در جلسات گذشته برنامه‌هایی نوشتیم که تمام دستورات آن‌ها به ترتیب اجرا می‌شدند. اما در همه ی مسائل اینکار امکان‌پذیر نیست. گاهی ممکن است بخواهیم در شرایط خاص تعدادی از دستورات اجرا شوند و تعدادی دیگر اجرا نشوند. ساختارهایی که این امکان را فراهم می‌کنند ساختارهای تصمیم نامیده می‌شوند. و گاهی ممکن است بخواهیم تا یک قسمت از برنامه چندین بار تکرار شود. حلقه‌های تکرار این امکان را فراهم می‌آورند.

برای پیاده‌سازی ساختارهای تصمیم و حلقه‌ها ، نیاز به انتقال کنترل برنامه داریم تا به این ترتیب از اجرای ترتیبی دستورات جلوگیری کنیم. یکی از مواردی که در انتقال کنترل برنامه از نقطه‌ای به نقطه‌ی دیگر باید مورد توجه قرار گیرد ، شیوه‌های آدرس دهی است. سه نوع آدرس در زبان اسمبلی وجود دارد :

آدرس کوتاه[1] : در این روش آدرس دهی می‌توان فاصله‌ی ۱۲۸- تا ۱۲۷بایت را آدرس دهی کرد.

آدرس نزدیک[2]: در این روش فاصله ی ۳۲۷۶۸- تا ۳۲۷۶۷ بایت را در داخل یک سگمنت می‌توان آدرس‌دهی کرد.

آدرس دور[3] : در این روش با استفاده از آدرس‌های ۳۲ بیتی (۱۶ بیت آدرس سگمنت و ۱۶ بیت آفست) می‌توان کل فضای حافظه ی ۴ گیگا بایتی را آدرس دهی کرد.

پرش‌های غیرشرطی

به عمل انتقال کنترل برنامه از یک نقطه به نقطه‌ی دیگر ، *پرش* می‌گویند. پرش‌ها به دو دسته ی شرطی و غیر شرطی تقسیم می‌شوند. پرش‌های شرطی در صورتی رخ می‌دهند که شرط خاصی برقرار باشد در غیر این صورت پرش انجام نخواهد شد اما پرش‌های غیر شرطی همیشه انجام می‌شوند و شرط خاصی ندارند.

در زبان اسمبلی ، پرش‌های غیرشرطی با دستور JMP پیاده‌سازی می‌شوند.

JMP <برچسب دستور>

[1] Short
[2] Near
[3] Far

اجرای این دستور باعث می‌شود کنترل برنامه به دستوری برود که برچسب آن مشخص شده است.

مثال : قطعه برنامه‌ای که به طور نامحدود ادامه می‌یابد و دائما به ثبات Ax یک واحد یک واحد می‌افزاید.

```
Lable1: Inc Ax
        Jmp  Lable1
```

پرش‌های شرطی

پرش‌های شرطی براساس نتیجه‌ی یک مقایسه انجام می‌شوند. انجام مقایسه توسط دستور CMP و به شکل زیر صورت می‌پذیرد.

Cmp <عملوند ۱>,<عملوند ۲>

عملوندهای این دستور می‌توانند به صورت زیر باشند :

عملوند ۱	ثبات	حافظه	ثبات یا حافظه	مقدار ثابت
عملوند ۲	حافظه	ثبات	مقدار ثابت	ثبات یا حافظه

دستور Cmp مشابه با دستور Sub عمل می‌کند یعنی عملوند ۲ را از عملوند ۱ کم می‌کند با این تفاوت که نتیجه را در عملوند ۱ قرار نمی‌دهد و تنها با انجام این عمل ، تغییراتی را در ثبات فلگ‌ها ایجاد می‌کند. (مثل ZF ، CFو....) دستورات پرش شرطی نیز بر اساس مقدار فلگ‌ها تصمیم می‌گیرند که پرش انجام شود یا خیر. به همین دلیل است که باید دستورات پرش‌شرطی را بلافاصله بعد از دستور Cmp آورد. در غیر این صورت ممکن است دستوراتی که در این بین انجام می‌شوند ثبات فلگ را تغییر دهند و در نتیجه پرش به نادرستی انجام شود.

مثال : مقایسه ی ثبات dx با عدد صفر Cmp dx,0

مثال: مقایسه ی ثبات‌های ax و dx Cmp ax,dx

از آنجا که تفسیر اعداد علامت‌دار و بدون علامت متفاوت است ، دودسته دستورات پرش شرطی وجود دارند. یک دسته برای اعداد علامت‌دار و دسته‌ی دیگر برای اعداد بدون علامت.

سوال: از اعداد باینری ۸ بیتی مقابل کدامیک بزرگ‌تر است؟ a=11010100 , b=01101110

پاسخ : اگر اعداد فوق بدون علامت در نظر گرفته شوند آنگاه : a=212 و b=110 لذا a بزرگتر است اما اگر این اعداد علامتدار فرض شوند آنگاه a=-44 و b=110 پس b بزرگتر است.

نتیجه : در مقایسه باید دقت‌کرد که اعداد علامتدار هستند یا بدون علامت.

دستورات پرش شرطی برای داده‌های بدون علامت :

دستور	توضیحات	فلگ هایی که تست می شوند
JE یا JZ	Jump if ZF , Jump if Equal	ZF
JNE یا JNZ	Jump if Not ZF , Jump if Not Equal	ZF
JAE یا JNB	Jump if Not Below Jump if Above or Equal	CF , ZF
JA یا JNBE	Jump if Not Below or Equal Jump if Above	CF
JB یا JNAE	Jump if Not Above or Equal Jump if Below	CF
JBE یا JNA	Jump if Not Above Jump if Below or Equal	CF

برای اینکه دستورات را ساده‌تر به خاطر بسپارید به قسمت توضیحات توجه نمایید.

دستورات پرش شرطی برای داده‌های علامتدار :

دستور	توضیحات	فلگ هایی که تست می شوند
JE یا JZ	Jump if ZF , Jump if Equal	ZF
JNE یا JNZ	Jump if Not ZF , Jump if Not Equal	ZF
JG یا JNLE	Jump if Not Less than or Equal Jump if Greater than	ZF,SF,OF
JGE یا JNL	Jump if Not Less than Jump if Greater than or Equal	SF , OF
JL یا JNGE	Jump if Not Greater than or Equal Jump if Less than	SF , OF
JLE یا JNG	Jump if Not Greater than Jump if Less than or Equal	ZF,SF,OF

دستور Loop

این دستور به فرم مقابل استفاده می‌شود : Loop <برچسب>

این دستور یک واحد از ثبات CX کم کرده و چنانچه مقدار این ثبات به صفر نرسد به برچسب مورد نظر پرش می‌کند. در غیر این صورت پرش انجام نخواهد شد.

مثال :

```
Mov CX,10
L1:
    بدنه حلقه
Loop  L1
```

قطعه کد مقابل یک حلقه تعریف می‌کند که بدنه ی آن ۱۰ مرتبه اجرا می‌شود.

بجز دستورات پرش گفته شده ، تعدادی دستورات پرش خاص نیز وجود دارند. این دستورات در جدول زیر آمده‌اند.

دستور	توضیحات	فلگ هایی که تست می شوند
JCXZ	Jump if CX is Zero	*
JC	Jump if Carry	CF
JNC	Jump if Not Carry	CF
JO	Jump if Overflow	OF
JNO	Jump if Not Overflow	OF
JP یا JPE	Jump if Parity (Even) , Jump if Parity	PF
JNP یا JPO	Jump if Parity (Odd) , Jump if Not Parity	PF
JS	Jump if Sign	SF
JNS	Jump if Not Sign	SF

تمارین

۱- برنامه‌ای بنویسید که با استفاده از عمل جمع ، حاصلضرب A×B را محاسبه نماید.

۲- برنامه‌ای بنویسید که تابع زیر را پیاده‌سازی کند. برای اینکار متغیرهای X و Y را تعریف نموده و با مقدار دهی X مقدار Y را محاسبه نمایید.

$$y = \begin{cases} 5x + 10 & ; \quad x < 0 \\ 2x + 10 & ; \quad 0 \leq x < 5 \\ x^2 + 2 & ; \quad x \geq 5 \end{cases}$$

۳- برنامه‌ای بنویسید که بزرگترین مقسوم‌علیه مشترک دو عدد X و Y را محاسبه نماید. برای اینکار از روش نردبانی استفاده نمایید. قطعه کد زیر اینکار را انجام می‌دهد.

```
int gcd(int a,int b)
{ int c;
While (a!=0)
   {  c = a mod b;
      a = b;
      b = c;
   }
return b;
}
```

جلسه پنجم : عملوندهای حافظه

پردازنده‌ی اینتل انواع گوناگونی از عملوندهای حافظه را به کار می‌برد تا با آرایه‌ها و سایر ساختمان‌های داده به خوبی کار کند. شش نوع عملوند حافظه وجود دارند که در ادامه به معرفی آن‌ها می‌پردازیم.

۱- عملوندهای مستقیم : به کمک این عملوند می‌توان مستقیما به محل خاصی از حافظه اشاره نمود.

مثال۱ : قراردادن محتوی آفست 3521h از سگمنت داده‌ها در ثبات cx mov cx,ds:[3521h]

مثال۲ : قراردادن محتوی آفست 1A20h از سگمنت اضافه[1] در ثبات ax mov ax,es:[1A20h]

مثال۳ : قراردادن محتوی ثبات al در ابتدای سگمنت‌داده‌ها mov ds:[0000h],al

نکته : در مثال‌های ۱ و ۲ از آنجا که عملوند دوم (سمت راست) دستور mov به محلی از حافظه اشاره می‌کند عملوند اول حتما باید از ثبات‌ها باشد و نمی‌تواند متغیری در حافظه باشد. در مورد مثال ۳ نیز باید بگوییم که عملوند دوم نمی‌تواند حافظه باشد چراکه عملوند اول حافظه است.

۲- عملوندهای مستقیم – آفست : در این روش آدرس دهی ، آفست با یک مقدار ثابت (که به آن تفاوت مکان می‌گویند) جمع می‌شود. این مقدار ثابت می‌تواند یک عدد منفی نیز باشد.

مثال : آرایه‌ای با نام list در سگمنت داده‌ها تعریف شده‌است.

```
List db 10,20,30,40
...
```

دستور mov اول مقدار ۱۰ (اندیس صفر آرایه) را به ثبات al نسبت خواهد داد

```
Mov al,list
```

دستور mov دوم مقدار ۲۰ (اندیس یک آرایه) را به ثبات bl نسبت خواهد داد

```
Mov bl,list+1
```

دستور mov سوم مقدار ۳۰ (اندیس دوی آرایه) را به ثبات cl نسبت خواهد داد

```
Mov cl,list+2
...
```

۳- عملوند غیر مستقیم : عملوند غیر مستقیم ، ثباتی است که حاوی آفست داده در حافظه است. وقتی آفست یک متغیر در یک ثبات قرار می‌گیرد ، آن ثبات اشاره‌گری به آن متغیر خواهد بود. این روش برای دسترسی به عناصر یک آرایه و یا آدرسی از حافظه که در حین اجرای برنامه قابل محاسبه است ، مناسب می باشد. ثبات‌های si ، di ، bx و bp می‌توانند به عنوان عملوند غیرمستقیم مورد استفاده قرار بگیرند.

[1] Extra segment (ES)

مثال : x آرایه‌ای ۴ عنصری است که در ثبات داده‌ها تعریف شده‌است.

```
x db 1,2,3,4
...
lea bx,x;        ←——————  قرار دادن آفست x در ثبات bx
mov [bx],5       ←——————  قرار دادن مقدار ۵ در اولین عنصر آرایه‌ی x
inc bx           ←——————  افزایش bx برای دسترسی به عنصر دوم آرایه
mov [bx],10      ←——————  قرار دادن مقدار ۱۰ در دومین عنصر آرایه‌ی x
...
```

آفست‌هایی که توسط عملوندهای غیر مستقیم ایجاد می‌شوند ، از ثبات DS منظور خواهند شد ، مگر اینکه ثبات bp مورد استفاده قرار گیرد که در این صورت ثبات SS به عنوان پایه در نظر گرفته خواهد شد. اگر هدف برنامه‌نویس چیزی غیر از این باشد ، باید سگمنت مورد نظر ذکر شود.

مثال :

```
Mov dl,[di]       ;offset from ds
Mov al,[bp]       ;offset from ss
Mov dl,es:[di]    ;offset from es
Mov al,ds:[bp]    ;offset from ds
Mov ax,[bx]       ;offset from ds
```

۴- عملوندهای اندیس یا ثبات پایه : در این روش آدرس دهی ، ثبات پایه یا ثبات اندیس به یک تفاوت‌مکان اضافه می‌شود. تفاوت مکان ، یک مقدار ثابت است که می‌تواند آفست یک متغیر باشد. تفاوت بین پایه و اندیس این است که ثبات‌های bx و bp ثبات‌های پایه و ثبات‌های si و di ثبات‌های اندیس هستند. شکل‌های مختلفی از این روش آدرس دهی در زبان اسمبلی قابل استفاده است. در زیر نمونه‌هایی از این روش را می‌بینید.

```
Var db 'A','B','C','D','E','F'
...
Mov bx,0
Mov al,var[bx]          ;al='A'
Mov dl,[var+bx]         ;dl='A'
Mov cl,var[bx+4]        ;cl='E'
Lea di,Var
Mov bl,2[di]            ;bl='C'
Mov bh,3[Var]           ;bh='D'
```

۵- عملوندهای پایه – اندیس : در این نوع آدرس دهی ثبات پایه به ثبات اندیس اضافه می‌شود تا آفست حافظه‌ای به دست آید. این نوع عملوند برای دستیابی به عناصر آرایه ی دو بعدی مفید است. به این صورت که ثبات پایه حاوی آفست سطر و ثبات اندیس حاوی آفست ستون باشد.

مثال : mov al,[bx+si]

نکته : دستور فوق را می‌توان به این فرم نیز نوشت mov al,[bx][si]

نکته : دوثبات پایه یا دو ثبات اندیس را نمی‌توان با یکدیگر ترکیب نمود.

۶- عملوندهای پایه-اندیس و تفاوت مکان : در این روش ثبات پایه و ثبات اندیس و تفاوت مکان با هم ترکیب می‌شوند تا آدرس موثر به دست آید.

مثال : mov dx,list[bx][si]

Mov dx ,[bx+si+list]

دستور LEA : این دستور دو عملوند دارد که عملوند دوم باید یک متغیر باشد. این دستور آفست این متغیر را به عملوند اول خود که باید از رجیسترها باشد نسبت می‌دهد.

عملگر Offset : آفست یک متغیر را محاسبه می‌نماید.

مثال : نتیجه ی دستورات زیر یکسان است .

Lea dx,Var

Mov dx,offset Var

تمارین

۱- یک رشته‌ی منتهی به $ با طول دلخواه در سگمنت داده‌ها تعریف نمایید. قطعه برنامه‌ای بنویسید که طول این رشته را محاسبه و در متغیر Len قرار دهد.

۲- آرایه ای ۱۰ عضوی از اعداد صحیح دوبایتی در سگمنت داده‌ها تعریف نمایید. برنامه‌ای بنویسید که میانگین ، مینیمم و ماکزیمم این آرایه را محاسبه نماید. برای میانگین ، مینیمم و ماکزیمم متغیرهایی با نام مناسب تعریف نمایید.

۳- برنامه‌ای بنویسید که آرایه ی تمرین ۲ را به صورت نزولی مرتب نماید.

جلسه ششم :وقفه‌های خروجی صفحه نمایش

در این جلسه برآنیم تا با برخی از وقفه‌های مربوط به صفحه نمایش آشنا شویم. (وقفه‌هایی برای انتقال مکان نما ، پاک‌کردن صفحه نمایش ، چاپ رشته‌ها و کاراکترها و ...) قبل از پرداختن به این موضوع بهتر است اندکی در مورد وقفه‌ها صحبت کنیم.

وقفه[1] چیست؟ سیگنالی است از طرف یک ابزار جانبی یا برنامه درحال اجرا که سرویس خاصی را درخواست می کند.

برای توضیح بیشتر به یک مثال می‌پردازیم ، یک بیمارستان با تعدادی بیمار را در نظر بگیرید. وظیفه ی پرستار بخش این است که به بیماران خدمات بدهد. پرستار چگونه می‌تواند از نیازهای تمام بیماران مطلع گردد؟ یک روش برای انجام اینکار این است که پرستار هر چند دقیقه همه‌ی کارهای خود را رها کرده و کل اتاق‌های بخش را سرکشی کند و ببیند کسی به خدمات نیاز دارد یا خیر. در این روش ، پرستار وقت زیادی از ساعات کاری خود را برای سرکشی از دست می‌دهد حتی اگر کسی به خدمات نیاز نداشته باشد. آیا روش بهتری وجود دارد ؟ بلی ، روش بهتر این است که در کنار تخت هر بیمار یک دکمه ی زنگ قرار دهیم تا هر بیماری که نیاز به خدمات دارد با فشردن آن ، پرستار بخش را مطلع سازد. در این روش تا زمانی که یک بیمار نیاز به خدمات نداشته باشد ، پرستار از سایر امور خود دست نخواهد کشید.

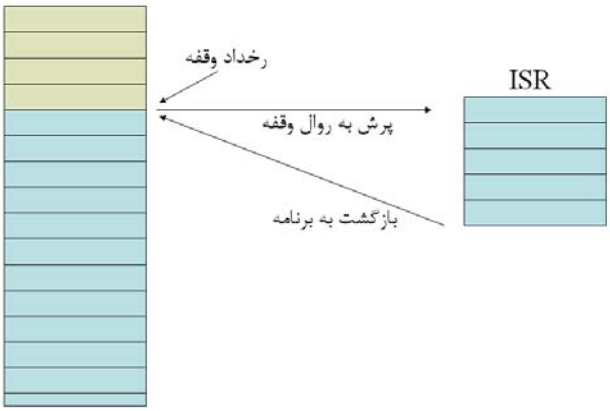
در مثال فوق ، پرستار را CPU و بیماران را قطعات و برنامه‌هایی در نظر بگیرید که نیاز به CPU دارند. یک روش پیاده سازی ، روش سرکشی است که CPU با سرکشی به قطعات جانبی بررسی کند که آن قطعات نیاز به کار پردازشی دارند یا خیر. به این روش *سرکشی* می‌گویند. روش دیگر استفاده از وقفه است. چنانچه یک ابزار جانبی یا یک برنامه ی درحال اجرا نیاز به کار پردازشی خاصی پیدا کند ، با ارسال یک سیگنال به CPU ، درخواست خود را اعلام می‌کند. به این روش ، روش وقفه و به سیگنال ارسالی از جانب خدمات گیرنده ، وقفه می‌گویند.

بازده‌ی روش سرکشی پایین بوده اما پیاده‌سازی آسانی دارد و در مقابل پیاده‌سازی روش وقفه پیچیده‌تر است اما کارایی بالایی دارد.

انواع وقفه‌ها : سخت‌افزاری (داخلی یا خارجی) و نرم‌افزاری (DOS و BIOS)

[1] Interrupt

مراحل اجرای یک وقفه



۱. ذخیره CS:IP و ثبات پرچم
۲. پرش به ISR مربوطه (هر وقفه دارای یک ISR است)
۳. اجرای ISR[1]
۴. بازیابی CS:IP و ثبات پرچم
۵. بازگشت به برنامه اصلی

نکته : زمانی که یک وقفه می‌آید , CPU دستور در حال اجرا را کامل کرده و بعد مراحل فوق اتفاق می‌افتند. بعنوان مثال فرض کنید در زمانی که CPU در حال انجام عمل ضرب است یک سیگنال وقفه دریافت کند, این سیگنال به صف انتظار می‌رود یا زمانی که CPU دستور ضرب را به پایان برساند. در این لحظه CPU به وقفه پاسخ خواهد داد.

جدول بردار وقفه

در ابتدای حافظه ی اصلی ، یعنی از آدرس 0000:0000 تا آدرس 0000:03FF ، یک جدول یک کیلوبایتی وجود دارد که در آن آدرس روال‌های ۲۵۶ وقفه ی مختلف ذخیره شده است. در واقع این جدول دارای ۲۵۶ رکورد ۴ بایتی است که هر رکورد آدرس روال سرویس‌دهنده یک وقفه ی خاص را نگهداری می‌کند. برای یافتن آدرس روال سرویس‌دهنده به وقفه ی n باید به رکورد nام این جدول مراجعه کرد. CPU با دریافت یک وقفه ، ابتدا به این جدول مراجعه می‌کند و پس از یافتن آدرس روال پاسخگو به این وقفه ، به این آدرس رفته و دستورات آن را اجرا می‌نماید.

نکته : چنانچه بخواهید ، می‌توانید یک روال برای پاسخ‌گویی به یک وقفه ی خاص بنویسید و با ایجاد تغییر در جدول بردار وقفه کاری کنید تا با رویداد آن وقفه ، روال طراحی شده توسط شما اجرا گردد.

اجرای وقفه در زبان اسمبلی

در زبان اسمبلی ، وقفه‌ها با دستور int اجرا می شوند. به شکل زیر :

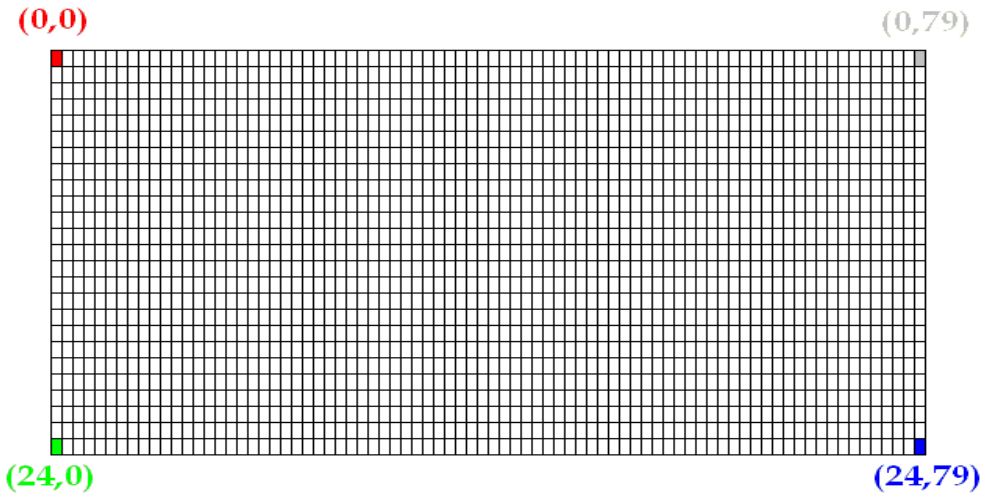
Int شماره ی وقفه

[1] Interrupt service routine

برخی وقفه‌ها دارای چند تابع هستند که باید شماره ی نابع مورد نظر در ah قرار گیرد.

آشنایی با صفحه نمایش

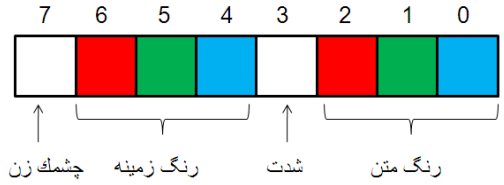
صفحه نمایش دارای ۲۵ سطر و ۸۰ ستون است. سطرها از ۰ تا ۲۴ و ستون ها از ۰ تا ۷۹ شماره‌گذاری شده اند. مختصات هر سلول صفحه به شکل (y,x) بیان می شود که y شماره ی سطر و x شماره ستون را مشخص میکند.



در حافظه ی سیستم فضایی به نام حافظه نمایش وجود دارد که اطلاعات موجود در آن در صفحه نمایش ظاهر می شوند. در صفحه نمایش رنگی ، حافظه نمایش معمولا از B8000h شروع میشود و 16k را اشغال می کند. آدرس صفحه نمایش ممکن است در تمام کامپیوترها یکسان نباشد. برای بدست آوردن آن از وقفه ها 10h و 21h استفاده نمایید. حافظه نمایش به قطعات کوچکتری به نام صفحه تقسیم می‌شود ، چهار صفحه به شماره های ۰ ، ۱ ، ۲ و ۳ که به طور پیش فرض صفحه ی ۰ فعال است.

صفات کاراکترها

در حافظه‌ی نمایش ، هر کاراکتر دو بایت از حافظه را اشغال می کند. یک بایت کد کاراکتر و یک بایت صفت آن. (صفات کاراکتر شامل : رنگ متن ، رنگ زمینه ، شدت رنگ ، چشمک زن یا ثابت)



شماره رنگ	رنگ	آبی	سبز	قرمز	شدت
۰	سیاه	۰	۰	۰	۰
۱	آبی	۱	۰	۰	۰
۲	سبز	۰	۱	۰	۰
۳	کبود	۱	۱	۰	۰
۴	قرمز	۰	۰	۱	۰
۵	بنفش	۱	۰	۱	۰
۶	قهوه ای	۰	۱	۱	۰
۷	سفید	۱	۱	۱	۰
۸	خاکستری	۰	۰	۰	۱
۹	آبی روشن	۱	۰	۰	۱
۱۰	سبز روشن	۰	۱	۰	۱
۱۱	کبود روشن	۱	۱	۰	۱
۱۲	قزمر روشن	۰	۰	۱	۱
۱۳	بنفش روشن	۱	۰	۱	۱
۱۴	زرد	۰	۱	۱	۱
۱۵	سفید با شدت بالا	۱	۱	۱	۱

جدول ترکیب رنگ

مثال : چنانچه صفت کاراکتری 14h باشد ، به چه شکلی دیده خواهد شد؟

14h= 00010100b → متن قرمز و زمینه آبی ثابت

پاک کردن صفحه نمایش

تابع 06H وقفه 10H برای پاک کردن صفحه نمایش استفاده می شود. پارامترهای این تابع به صورت زیر می باشد :

- AH: 6H
- AL: تعداد خطوطی که باید پاک شوند
- CH: شماره ی سطر گوشه بالا سمت چپ
- CL:شماره ستون گوشه بالا سمت چپ
- DH:شماره سطر گوشه پایین سمت راست
- DL:شماره ستون گوشه پایین سمت راست
- BH:صفت محدوده ای که باید پاک شود

انتقال مکان نما

برای انتقال مکان نما در صفحه نمایش از تابع 02H وقفه 10H استفاده می کنیم. پارامترهای این تابع به صورت زیر می باشد:

- AH: 2H
- DH: شماره سطر مورد نظر
- DL: شماره ستون مورد نظر
- BH: شماره صفحه حافظه نمایش

مثال : برنامه‌ای بنویسید که مکان نما را به سطر ۱۰ و ستون ۱۵ منتقل نماید.

```
mov ah,2h
mov dh,10      سطر
mov dl,15      ستون
mov bh,0       شماره صفحه
int 10h
```

چاپ رشته‌ها در صفحه نمایش

برای این منظور از تابع 9h وقفه 21h استفاده می نماییم. پارامتر های این تابع به صورت زیر می باشد:

- AH: 9h
- DX: آفست رشته حاوی پیام (این رشته باید منتهی به $ باشد)

مثال : برنامه‌ای که پیام hello world را روی مانیتور می‌نویسد :

```
DSEG    SEGMENT ‘DATA’
      Message DB 'Hello World$‘
DSEG    ENDS
CSEG    SEGMENT 'CODE'
  START   PROC    FAR
      …
      Mov dx,offset Message      ; or    Lea dx,Message
      Mov ah,9h
      Int    21h
      …
  START   ENDP
CSEG    ENDS
```

چاپ یک کاراکتر در صفحه نمایش

برای این منظور از تابع 2h وقفه 21h استفاده می نماییم. پارامتر های این تابع به صورت زیر می باشد:

- AH: 2h
- DX: کد اسکی کاراکتر مورد نظر

با استفاده از توابع 9h و 0Ah از وقفه 10h نیز می توان کاراکتری را در صفحه نمایش چاپ کرد. در تابع 9h میتوان صفتی را برای کاراکتر تعیین کرد اما در تابع 0Ah از صفت فعلی برای چاپ استفاده می شود. تفاوت این دو تابع با تابع 2h وقفه 21h این است که توسط این توابع میتوان یک کاراکتر را به تعداد دلخواه چاپ نمود.

پارامترهای تابع 9h از وقفه ی 10h به شکل زیر است :

- 09H:AH
- AL:کاراکتر
- BL:صفت
- BH:شماره صفحه
- CX: دفعات چاپ

پارامترهای تابع 0Ah از وقفه‌ی 10h به شکل زیر است :

- 0Ah :AH
- AL:کاراکتر
- BH:شماره صفحه
- CX: دفعات چاپ

تمارین :

۱- برنامه ای بنویسید که کاراکتر 'A' را به تعداد ۲۰ مرتبه با رنگ کبود روشن روی زمینه آبی چاپ نماید.

۲- برنامه‌ای بنویسید که کاراکترهای A تا Z را روی مانیتور بنویسد.

۳- برنامه ای بنویسید که کل صفحه نمایش را پاک کند. در این برنامه صفت کاراکترها طوری انتخاب شود که رنگ متن قرمز و رنگ زمینه آبی باشد. سپس نام خود را توسط تابع 9h وقفه ی 21h بنویسید.

۴- یک رشته با طول دلخواه در سگمنت داده‌ها تعریف نمایید. سپس برنامه‌ای بنویسید که این رشته را دقیقا در وسط مانیتور چاپ نماید به گونه‌ای که با تغییر رشته برنامه همچنان درست کار کند. (دقت کنید که طول رشته در این امر تاثیرگذار است)

۵- برنامه‌ای بنویسید که شکل زیر را روی مانیتور چاپ نماید

```
*
**
****
********
**********
```

جلسه هفتم :وقفه‌های ورودی

صفحه کلید ، یکی از مهمترین وسایل ارتباطی بین کاربر و برنامه‌ی رایانه‌ای است. در این جلسه برخی وقفه‌های موجود برای ورود اطلاعات از صفحه کلید را معرفی می‌نماییم.

خواندن کاراکتر از صفحه کلید

به این منظور می‌توان از توابع 01h و 8h وقفه‌ی 21h استفاده نمود. هردوی این توابع یک کاراکتر از صفحه کلید می‌خوانند و تا زمانی که کلیدی فشرده نشده باشد ، منتظر باقی می‌مانند. تفاوت این توابع در این است که تابع 01h ضمن خواندن کاراکتر از صفحه کلید ، آن را روی مانیتور چاپ می‌کند. (مشابه تابع getche() در زبان c) اما تابع 8h کاراکتر خوانده شده را روی مانیتور چاپ نمی‌کند. (مشابه تابع getch() در زبان c)

پس اجرای این توابع ، کد کاراکتر خوانده شده در ثبات al خواهد بود.

کاراکترهای خوانده شده از صفحه کلید

پس از اینکه یک کاراکتر از صفحه کلید خوانده شد ، باید بتوانیم تشخیص دهیم که کدام کلید فشرده شده است. کلیدهای صفحه کلید به دودسته تقسیم می‌شوند. یک دسته کلیدهایی با کد یک بایتی مثل کاراکترهای A-Z یا a-z و ارقام 0-9 و کاراکترهایی نظیر ? ، * ، + ، - ، / ، @ ، (،) و ... . در اینگونه موارد کد اسکی کاراکتری که کلید مربوط به آن فشرده شده است از صفحه کلید خوانده خواهد شد. کلیدهای Enter ، Space و Esc نیز جزء همین دسته هستند و کد آن‌ها به ترتیب ۱۳ ، ۳۲ و ۲۷ می‌باشد. دسته‌ی دوم کلیدهایی با کد توسعه یافته هستند. دراینگونه موارد کد خوانده شده از صفحه کلید یک کد دو بایتی خواهد بود که بایت اول آن صفر است. چنانچه در هنگام خواندن کدها از صفحه کلید ، کد خوانده شده صفر باشد ، بیانگر این موضوع است که یکی از کلیدهای توسعه یافته فشرده شده است که در این صورت باید یک بایت دیگر از بافر صفحه کلید را خواند. بایت دوم است که مشخص می‌کند کدامیک از کلیدهای توسعه یافته فشرده شده اند.

خواندن رشته‌ها از صفحه کلید

از تابع 0Ah وقفه 21h برای این منظور استفاده می کنیم. برای خواندن رشته از ورودی باید لیستی به نام لیست پارامترها تعریف نمود و آفست این لیست را در ثبات dx قرار داد. این لیست با یک label آغاز شده و به شکل زیر تعریف می‌شود:

- بایت اول : حداکثر تعداد کاراکترهایی که از ورودی خوانده می‌شود.
- بایت دوم : تعداد واقعی کاراکترهای خوانده شده.
- بایت سوم : شروع محلی از حافظه برای ذخیره رشته خوانده شده

مثال : تعریف یک لیست برای خواندن یک رشته حداکثر به طول ۲۰ کاراکتر .

```
strlist :
max    db   20
len    db   ?
buffer db   20 dup (‘ ‘)
```

نکته : لیست پارامترها در سگمنت داده‌ها تعریف می‌شود.

مثال : استفاده از تابع 0Ah برای خواندن رشته

```
mov  ah,0aH
lea   dx,strlist
int    21h
```

تمارین

۱- برنامه ای بنویسید که رشته ای به طول حداکثر ۲۰ کاراکتر را از ورودی خوانده و تمام کاراکترهای بزرگ آن را به کاراکترهای کوچک تبدیل نموده و آن را توسط تابع 9h وقفه 21h روی مانیتور چاپ نماید.

۲- برنامه‌ای بنویسید که حداکثر ۱۵ کاراکتر را از صفحه کلید خوانده و آن‌ها را در یک رشته ذخیره نماید. این برنامه را به گونه‌ای بنویسید که تنها کاراکترهای A-Z و a-z را بپذیرد و در صورت فشرده شدن کلید Enter کار خواندن از ورودی را خاتمه دهد. رشته‌ی خوانده شده را روی مانیتور چاپ نمایید.

۳- برنامه‌ی تمرین ۲ را به گونه‌ای بنویسید که حد اکثر ۵ کاراکتر از کاراکترهای 0-9 را بپذیرد.

۴- تحقیق کنید که کد کلیدهای چهارگانه‌ی مکان‌نما  چیست. برنامه‌ای بنویسید که با فشردن هریک از این ۴ کلید ، پیام مناسبی روی مانیتور چاپ نماید. (به عنوان مثال با فشردن کلید → پیام right به نمایش درآید)

راهنمایی : کد این کلیدها ۲ بایتی است. نرم افزار Emu8086 این کلیدها را به درستی شبیه سازی نمی‌کند. برای اینکه برنامه‌ی خود را تست کنید پس از این‌که در محیط Emu8086 کلید F5 را فشردید ، از منوی External گزینه‌ی Run (external) را انتخاب نمایید.

جلسه هشتم: ورودی و خروجی اعداد

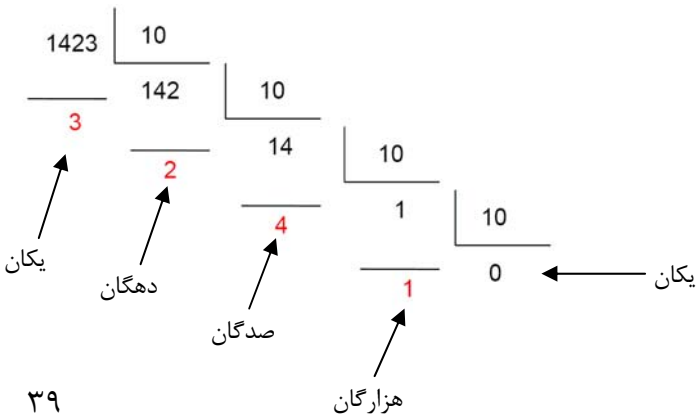
در جلسات ششم و هفتم نوشتن رشته‌ها و کاراکترها در مانیتور و خواندن آن‌ها از صفحه‌کلید را بررسی کردیم. اگر دقت کرده باشید وقفه‌هایی که آموختیم تنها می‌توانند رشته و کاراکتر بخوانند یا بنویسند. آیا همیشه ورودی‌ها و خروجی‌های یک برنامه رشته‌ای هستند؟ مسلما پاسخ منفی‌است. بسیاری از برنامه‌های کامپیوتری ورودی‌هایی عددی دارند و نتیجه‌ی پردازش آن‌ها نیز اعداد هستند و باید این اعداد برای کاربر به نمایش درآیند.

برای چاپ یک عدد روی مانیتور لازم است که ابتدا آن عدد را به یک رشته تبدیل نمایید سپس این رشته را به مانیتور چاپ کنید. بعنوان مثال فرض کنید حاصل محاسبات در ثبات ax قرار دارد. باید با دنبال کردن یک الگوریتم خاص ، این عدد را به یک رشته قابل چاپ در مانیتور تبدیل نموده و سپس آن‌را چاپ نمایید. در مورد ورودی اعداد نیز عکس این قضیه صادق است. آنچه از صفحه کلید خوانده می‌شود یا رشته است یا کاراکتر ، باید با استفاده از یک الگوریتم خاص، این ورودی‌ها را به عدد تبدیل نمایید.

تبدیل اعداد به رشته

برای شروع با این سؤال آغاز می‌کنیم. به نظر شما تفاوت عدد 125 با رشته‌ی '125' در چیست؟ مسلما پاسخ شما این است که عدد 125 ، یک عدد یک بایتی است که نمایش مبنای ۲ آن به صورت 01111101b می‌باشد در حالی که رشته‌ی '125' یک رشته‌ی سه بایتی است که هر بایت آن کد اسکی یک کاراکتر است. بایت اول کد اسکی کاراکتر '1' ، بایت دوم کد اسکی کاراکتر '2' و بایت سوم کد اسکی کاراکتر '5' است. به نظر شما چگونه می‌توان یک عدد را به رشته‌ی معادل با آن تبدیل کرد؟ در پاسخ باید گفت که در مرحله‌ی اول باید ارقام یکان ، دهگان ، صدگان و ... را از عدد جدا نمود و سپس این ارقام را به کاراکتر معادل آن تبدیل کرد و این کاراکتر را در محل مناسبش در یک بافر رشته‌ای قرار داد. برای تفکیک ارقام یک عدد می‌توان از تقسیمات متوالی بر ۱۰ استفاده نمود. به این ترتیب باقیمانده‌ی هر مرحله یکی از ارقام عدد اصلی خواهد بود. شرط خاتمه‌ی روند تقسیمات متوالی این است که حاصل تقسیم صفر شود.

مثال : تفکیک ارقام عدد ۱۴۲۳:



تبدیل ارقام به کاراکترهای مربوطه

برای تبدیل هر رقم به کاراکتر معادل با آن کافی است که آن رقم را با کاراکتر '0' یا عدد 30h جمع بزنیم. همانطور که می‌دانید کداسکی کاراکترهای '0' تا '9' پشت سرهم قرار دارند. چنانچه به کد اسکی کاراکتر '0' ، یک واحد بیافزاییم به کد اسکی کاراکتر '1' می‌رسیم و اگر ۲ واحد بیافزاییم به کد اسکی کاراکتر '2' خواهیم رسید و در مورد سایر ارقام نیز به همین شکل است.

قرار دادن کاراکترها در رشته

حال که ارقام عدد تفکیک شده و به کاراکتر هم تبدیل شده‌اند نوبت آن است که این کاراکترها را در یک رشته قرار دهیم تا رشته‌ی معادل با عدد اولیه حاصل شود. برای انجام اینکار توجه به این نکته ضروری است که در اولین مرحله‌ی تقسیم ، رقم یکان بدست می‌آید و این رقم باید در انتهای بافر رشته‌ای قرار گیرد و ارقام صاحل از مراحل بعدی به سمت ابتدای رشته قرار می‌گیرند. این بدان معناست که رشته از انتها ساخته می‌شود. چنانچه می‌خواهید این رشته را با تابع 09h از وقفه ی 21h چاپ نمایید باید به انتهای آن یک '$' هم بیافزایید.

روال تبدیل اعداد ۱۶ بیتی بدون علامت به رشته

۱. یک بافر رشته‌ای به نام STR شامل ۵ کاراکتر[1] Space و یک '$' در انتهای آن تعریف نمایید.
۲. عدد مورد نظر را در ثبات ax قرار دهید و dx را صفر نمایید.
۳. ثبات bx را برابر با ۴ قرار دهید.
۴. dx:ax را بر ۱۰ تقسیم نمایید.
۵. باقیمانده را (که مطمئنیم از ۱۰ کوچکتر است و در dl قرار دارد) به کاراکتر تبدیل نمایید.
۶. کاراکتر حاصل از مرحله‌ی قبل را در STR[bx] قرار دهید
۷. bx=bx-1
۸. dx=0
۹. چنانچه ax نامساوی صفر است به مرحله‌ی ۴ برو.

[1] بزرگترین عدی که در ثبات ۱۶ بیتی جای خواهد گرفت ، بیشتر از ۵ رقم ندارد.

خواندن اعداد از ورودی

آنچه از ورودی خوانده می‌شود یا کاراکتر است یا رشته. برای خواندن اعداد از ورودی دو رویکرد وجود دارد. رویکرد اول این است که یک رشته از ورودی خوانده شود و سپس این رشته با یک رویه‌ی خاص تبدیل به عدد شود. رویه‌ی انجام اینکار عکس رویه تبدیل اعداد به رشته‌هاست. ایراد وارد به این روش این است که کاربر در هنگام ورود رشته می‌تواند هریک از کاراکترهای موجود در صفحه کلید را وارد نماید و این کار منجر به ایجاد خطا در مرحله‌ی تبدیل رشته به عدد خواهد شد. اگر بتوانیم به نوعی از ورود کاراکترهایی غیر از کاراکترهای '0' تا '9' جلوگیری کنیم ، کار کامل‌تری انجام داده‌ایم. این رویکرد دوم در ورود اعداد است. در این رویکر با استفاده از توابع مربوط به خواندن کاراکتر از صفحه کلید اقدام به خواندن یک کاراکتر می‌کنیم. چنانچه این کاراکتر یکی از کاراکترهای '0' تا '9' باشد ، آن‌را می‌پذیریم در غیر این صورت از آن کاراکتر صرف نظر می‌کنیم. چگونه می‌توان کاراکترهای وارد شده را به یک عدد تبدیل نمود؟

فرض کنید کاربر ابتدا کاراکتر '9' ، سپس کاراکتر '1' و بعد از آن کاراکتر '5' را وارد نماید و در آخر کلید Enter را بزند. عدد خوانده شده چند است؟ پاسخ ۹۱۵ است. برای تبدیل کاراکترهای خوانده شده به یک عدد از رویه‌ی زیر استفاده می‌کنیم.

۱. N=0
۲. یک کاراکتر از ورود بخوان
۳. چنانچه کاراکتر خوانده شده Enter است برو به ۹
۴. چنانچه این کاراکتر مجاز نیست برو به ۲
۵. کاراکتر خوانده شده را به عدد تبدیل کن (برای اینکار کافی است از کد اسکی کاراکتر ، کد اسکی کاراکتر '0' را کم نمود.)
۶. N=N*10
۷. حاصل بدست آمده از مرحله‌ی ۵ را به N اضافه کن
۸. برو به ۲
۹. پایان

تمارین

۱- رویه تبدیل اعداد ۱۶ بیتی بدون علامت به رشته را نوشته و توسط آن یک عدد دلخواه را به رشته تبدیل کنید. سپس این رشته را با استفاده از تابع 9h وقفه ی 21h روی مانیتور چاپ نمایید.

۲- رویه تمرین ۱ را به گونه‌ای تغییر دهید که بتواند اعداد علامت دار را نیز به رشته تبدیل کند.

۳- رویه شرح داده‌شده در صفحه‌ی ۴۱ (خواندن اعداد از ورودی با استفاده از خواندن کاراکتر) را پیاده‌سازی نمایید.

۴- با استفاده از رویه‌های طراحی شده در تمارین قبلی ، برنامه‌ای بنویسید که عدد x را از ورودی خوانده و پس از محاسبه‌ی مقدار y از رابطه‌ی زیر ، آن را روی مانیتور چاپ نماید.

y=5*(x+10)

*۵- رویه‌ی طراحی شده در تمرین ۱ را به گونه‌ای تغییر دهید که به کاربر اجازه‌ی تصحیح نیز بدهد. بعنوان مثال کاربر می‌خواهد عدد ۱۵۶ را وارد کند اما کاراکتر دوم را به اشتباه به‌جای ’5‘ ، ’2‘ وارد می‌کند. این امکان را فراهم آورید که او بتواند با استفاده از کلید Backspace اصلاح مورد نظر را انجام دهد. لازم است نتیجه‌ی اصلاحات انجام شده روی مانیتور نیز قابل رویت باشد. (یعنی ۵ پاک‌شده و ۲ جایگزین آن‌شود)

جلسه نهم: زیربرنامه‌ها

برای حل مسائل بزرگ ، بهتر است که آن‌ها را به چند زیر مسئله تقسیم کرد و برای حل هر یک از این زیر مسائل از یک برنامه استفاده نمود. به این برنامه که یک کار خاص را انجام می‌دهد ، زیربرنامه می‌گوییم.

استفاده از زیربرنامه‌ها دارای مزایای زیر است :

۱- افزایش خوانایی برنامه ، زیرا برنامه به قسمت‌های کوچکتری تبدیل می‌شود که قابل فهم‌ترند.
۲- امکان انجام کار گروهی ، به این شکل که هر عضو گروه مسئول نوشتن تعدادی از زیر برنامه‌ها شده و در نهایت تمام زیربرنامه‌ها در کنار هم به کار برده می‌شوند.
۳- امکان استفاده آسان از کد دیگران.
۴- کاهش حجم کد نهایی . به این ترتیب که چنانچه عمل خاصی باید n مرتبه انجام شود ، لازم نیست n مرتبه کد مربوط به آن نوشته شود و کد آن تنها یکبار نوشته شده و در مواقع نیاز فراخوانی می‌شود.
۵- افزایش قابلیت اعتماد برنامه. چنانچه یک زیربرنامه تست شد و صحت آن مشخص گردید، در مواقع استفاده از آن نگران وجود خطا در آن نخواهیم بود.
۶- امکان ایجاد کتابخانه ای از توابع پرکاربرد.
۷- افزایش سرعت طراحی برنامه

تعریف زیربرنامه

برای استفاده از یک زیر برنامه ، ابتدا باید آن را تعریف نمود. برای تعریف زیر برنامه در زبان اسمبلی ، از راهنماهای PROC و ENDP به شکل زیر استفاده می‌شود.

```
نام زیربرنامه  proc  نوع زیربرنامه

        بدنه‌ی زیربرنامه

        ret/retf [مقدار]

endp نام زیربرنامه
```

نام زیربرنامه: این نام باید از قوانین نام‌گذاری شناسه‌ها تبعیت کند و از آن برای فراخوانی تابع استفاده می‌شود.

نوع زیر برنامه: یک زیربرنامه می‌تواند داخلی (near) یا خارجی (far) باشد. چنانچه یک زیربرنامه در کدسگمنت اصلی برنامه باشد به آن زیربرنامه داخلی و در غیر این صورت زیربرنامه خارجی می‌گوییم. زیربرنامه‌هایی که به صورت far نوشته می‌شوند قابل فراخوانی توسط برنامه‌های خارج از کدسگمنت می‌باشند.

بدنه زیربرنامه : قسمت اصلی یک زیربرنامه را تشکیل می‌دهد. در این قسمت بسته به اینکه این زیربرنامه می خواهد چه کاری انجام دهد ، دستورات لازم را می‌نویسیم. به عنوان مثال چنانچه بخواهیم زیربرنامه‌ای برای پاک کردن صفحه نمایش بنویسیم ، در این قسمت دستورات مربوط به پاک‌کردن صفحه نمایش را قرار می‌دهیم.

بازگشت از زیربرنامه : پس از اتمام کار زیربرنامه ، باید کنترل به نقطه‌ی فراخوانی این زیربرنامه بازگردد. برای این منظور از دستور ret برای توابع داخلی و از دستور retf برای توابع خارجی استفاده می‌شود. مقداری که جلوی این دستورات نوشته می‌شود ، تعداد بایت‌هایی است که باید حین بازگشت از پشته حذف گردد. در ادامه راجع به این مقدار بیشتر خواهیم گفت. (می‌توان این مقدار را ننوشت، در این صورت چیزی جز آدرس بازگشت از پشته حذف نخواهد شد)

مثال : تعریف تابعی با نام MyProc به صورت خارجی (far)

```
MyProc proc far
        …
        Retf
MyProc endp
```

فراخوانی زیربرنامه

پس از آنکه یک تابع تعریف شد ، باید در مواقع نیاز فراخوانی شود تا عمل مورد نظر را انجام دهد. فراخوانی توابع با دستور call و به صورت زیر انجام می‌شود :

```
Call   نام زیربرنامه
```

هنگامی که دستور call اجرا می‌گردد اعمال زیر رخ می‌دهند :

۱- آدرس برگشت در پشته ذخیره می‌گردد. منظور از آدرس برگشت ، آدرس دستور بعد از دستور call است که پس از اتمام کار تابع فراخوانی شده ، اجرای برنامه باید از آن ادامه یابد. این آدرس برای زیربرنامه‌های near دو بایت (IP) و برای زیر برنامه‌های far چهار بایت (CS:IP) می‌باشد.
۲- کنترل به تابع فراخوانده شده انتقال می‌یابد تا دستورات آن را انجام دهد. در انتهای زیربرنامه ، دستور ret/retf آدرس برگشت که در مرحله‌ی قبل در پشته ذخیره شده بود را بازیابی کرده و کنترل به آن آدرس باز می‌گردد.

چند نکته :

۱- در ابتدای هر زیربرنامه ، تمام ثبات‌هایی که توسط آن زیربرنامه مورد استفاده قرار می‌گیرند را در پشته قرار دهید و در انتهای زیربرنامه آن‌ها را فراخوانی کنید. به این ترتیب زیربرنامه‌ی شما مقادیر ثبات‌ها را تغییر نخواهد داد.
۲- بهتر است در ابتدای زیربرنامه توضیحاتی راجع به آن بنویسید تا به این ترتیب مشخص گردد که این زیربرنامه چه کاری انجام می‌دهد.

انتقال پارامترها

برخی از زیربرنامه‌ها ممکن است نیاز به پارامتر ورودی داشته باشند ، مثل زیر برنامه‌ای که می‌خواهد مکان‌نما را به نقطه‌ی خاصی از صفحه نمایش انتقال دهد ، مسلما باید به این زیر برنامه بگویید که مکان نما را به کدام نقطه انتقال دهد ، برای اینکار باید پارامترهایی برای این زیربرنامه تعریف نمایید. گاهی نیز ممکن است زیربرنامه بخواهد مقداری را به عنوان خروجی به برنامه‌ی فراخوان منتقل نماید. مثل تابعی که جذر یک عدد را محاسبه می‌نماید ، مسلما مقدار حاصل از محاسبات باید به برنامه‌ی فراخوان بازگردد. برای اینکار نیز از پارامترهای خروجی استفاده می‌کنیم. پس به طور کلی پارامترها به دو دسته‌ی ورودی و خروجی تقسیم بندی می‌شوند. در ادامه به نحوه‌ی انجام این اعمال می‌پردازیم.

به طور کلی سه روش انتقال پارامتر داریم :

۱- استفاده از متغییرهای تعریف شده در سگمنت‌داده‌ها: این روش انتقال پارامتر روش ساده‌ایست اما روش مناسبی نیست. بعنوان مثال فرض کنید می‌خواهیم زیربرنامه‌ای بنویسیم که یک عدد به عنوان ورودی گرفته و جذر آن را به‌عنوان خروجی تولید کند. برای این منظور می‌توانید متغیری با نام N برای ورودی و متغیری با نام Sq برای خروجی تعریف نمایید و در زیربرنامه‌ی خود از این متغیرها استفاده نمایید. محاسبات را روی N انحام داده و حاصل را در Sq قراردهید. به این ترتیب برنامه‌ی فراخوان باید عدد مورد نظر را در N قرار داده و پس از فراخوانی زیربرنامه ، جذر این عدد را در Sq بیاید. تنها مزیت این روش سادگی آن است. اما دارای این معایب است :

الف) زیربرنامه‌ی فراخوانی شده و زیربرنامه‌ی فراخوان باید هر دو از یک سگمنت‌داده‌ی مشترک استفاده نمایند.

ب)حتی اگر زیربرنامه فراخوانی نشود ، متغیرهای تعریف شده در سگمنت‌داده‌ها فضایی را اشغال می‌کنند.

ج) امکان نوشتن زیرنامه‌های بازگشتی وجود ندارد.

۲- استفاده از ثبات‌ها : این روش نیز مشابه روش قبلی است. به این ترتیب که برای پارامترهای ورودی و خروجی یک قرار داد می‌کنیم. مثلا اینکه ورودی باید در ثبات Ax باشد و خروجی در Bx قرار خواهد گرفت. ادامه کار مشابه با روش قبلی است. این روش نسبت به روش قبلی این مزیت را دارد که نیازی نیست تا زیربرنامه‌ی فراخوانده شده و زیربرنامه‌ی فراخوان از یک سگمنت داده‌ی مشترک استفاده کنند. اما دارای معایب زیر است :

الف) تعداد ورودی‌ها و خروجی‌ها محدود می‌شوند. چراکه تعداد ثبات‌ها محدود هستند.

ب) قبل از فراخوانی تابع چنانچه ثبات‌هایی که به عنوان پارامتر مورد استفاده قرار می‌گیرند دارای مقادیر مهم برای زیربرنامه‌ی فراخوان باشند ، باید آن‌ها را در مکانی مناسب (مثلا پشته) ذخیره نمود.

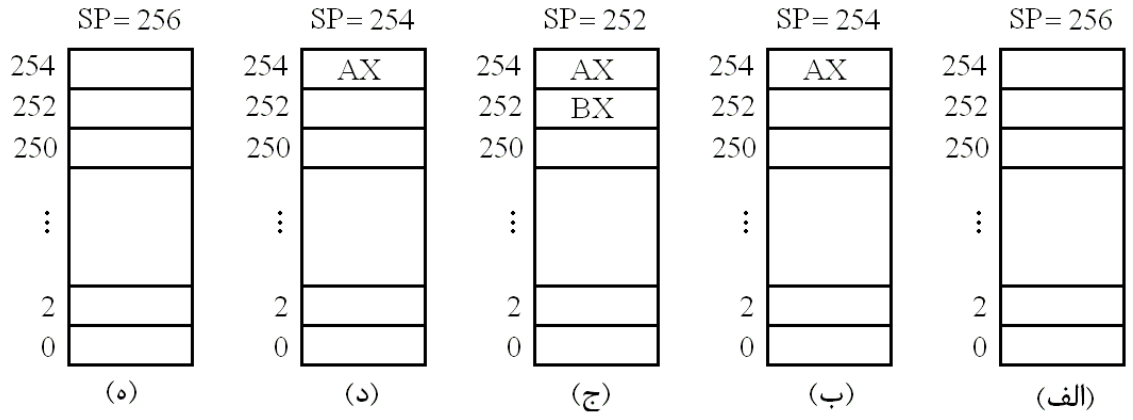
ج) امکان نوشتن زیربرنامه‌های بازگشتی وجود ندارد.

۳- استفاده از پشته : در این روش پارامترهای ورودی با استفاده از دستور push در پشته قرار گرفته و سپس زیربرنامه فراخوانده می‌شود. زیربرنامه‌ی فراخوانده شده ، می‌تواند این پارامترها را از پشته بازیابی کرده و از آن‌ها استفاده نماید. همچنین می‌تواند پارامترهای خروجی را در پشته قرار دهد تا

پس از بازگشت از زیربرنامه ، برنامه‌ی فراخوان از آن‌ها استفاده نماید. این روش هیچکدام از معایب روش‌های قبلی را ندارد. تنها عیب آن نسبت به روش‌های قبلی پیچیدگی پیاده‌سازی آن می‌باشد. در بخش بعدی به تفصیل به این بحث می‌پردازیم.

انتقال پارامتر از طریق پشته

برای این منظور ابتدا باید اندکی با پشته‌ی ۸۰۸۶ آشنا شویم. نکته‌ی قابل ذکر در مورد این پشته این است که از آدرس‌های بالا به سمت آدرس‌های پایین رشد می‌کند. ثبات sp در هر لحظه آفست آخرین داده‌ی push شده به پشته را نسبت به ابتدای پشته نشان می‌دهد و به این ترتیب مشخص می‌کند که پشته تا کجا پر شده است. فرض کنید پشته‌ای به اندازه‌ی ۲۵۶ بایت تعریف کرده‌ایم. قبل از اینکه داده‌ای به پشته وارد شود ، وضعیت پشته و ثبات Sp به فرم نشان داده‌شده در شکل ۹-۱(الف) می‌باشد. پس از آنکه ثبات AX به پشته Push شود وضعیت به شکل ۹-۱(ب) و پس از push کردن ثبات Bx به شکل ۹-۱(ج) تغییر خواهد یافت. شکل‌های ۹-۱(د) و ۹-۱(ه) نیز وضعیت پشته پس از دو pop کردن متوالی را نشان می‌دهد.

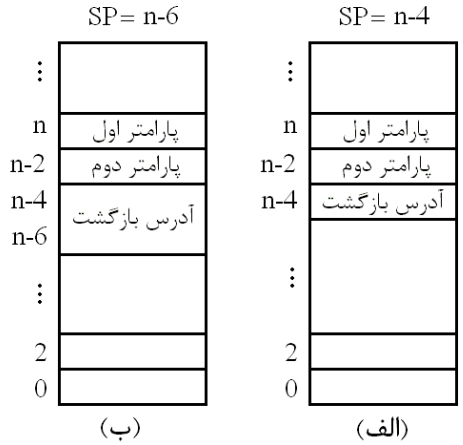


شکل ۹-۱) پشته ی ۸۰۸۶ ، الف)پشته‌ی خالی ، ب)پشته بعد از یک Push ax ، ج) پشته بعد از Push bx
د) پشته بعد از یک pop ، ه)پشته بعد از pop دوم

گفتیم برای انتقال پارامتر از طریق پشته باید ابتدا پارامترها را به ترتیب قرارداد شده در پشته push کرده و سپس زیربرنامه‌را فراخوانی کنیم. به نظر می‌رسد که می‌توان در زیربرنامه‌ی فراخوانی شده ، با استفاده از دستور pop این مقادیر را بازیابی کنیم. اما یک مشکل وجود دارد. مشکل این است که بعد از push کردن پارامترها به پشته ، از دستور call برای فراخوانی زیر برنامه استفاده می‌شود. همانطور که قبلا گفتیم دستور call آدرس بازگشت را به پشته push می‌کند و به این ترتیب بالاترین عضو پشته پس از ورود به

زیربرنامه، آدرس بازگشت خواهد بود نه پارامترها و چنانچه از دستور pop استفاده نمایید بجای پارامترها ، آدرس بازگشت را از پشته بازیابی خواهید کرد.

زیربرنامه‌ای با دو پارامتر ورودی را در نظر بگیرید. شکل ۹-۲ وضعیت پشته پس از push کردن این دو پارامتر به پشته و فراخوانی زیربرنامه را نشان می‌دهد. اگر این زیر برنامه near باشد ، آدرس بازگشت دوبایتی (شکل ۹-۲ الف) و در صورتی که far باشد ۴ بایتی خواهد بود(شکل ۹-۲ ب).



شکل ۹-۲) وضعیت پشته پس از قرار دادن دو پارامتر در آن و فراخوانی زیربرنامه.

الف) زیربرنامه near ب) زیربرنامه far

با توجه به وضعیت تشریح شده ، چگونه می‌توان به پارامترها دست‌یافت؟ پاسخ این است که برای دستیابی به آن‌ها باید بجای استفاده از دستور pop ، از آدرس دهی غیر مستقیم ثباتی استفاده نماییم و چون می‌خواهیم این آدرس دهی را در پشته انجام دهیم ، باید از ثبات BP استفاده کنیم.

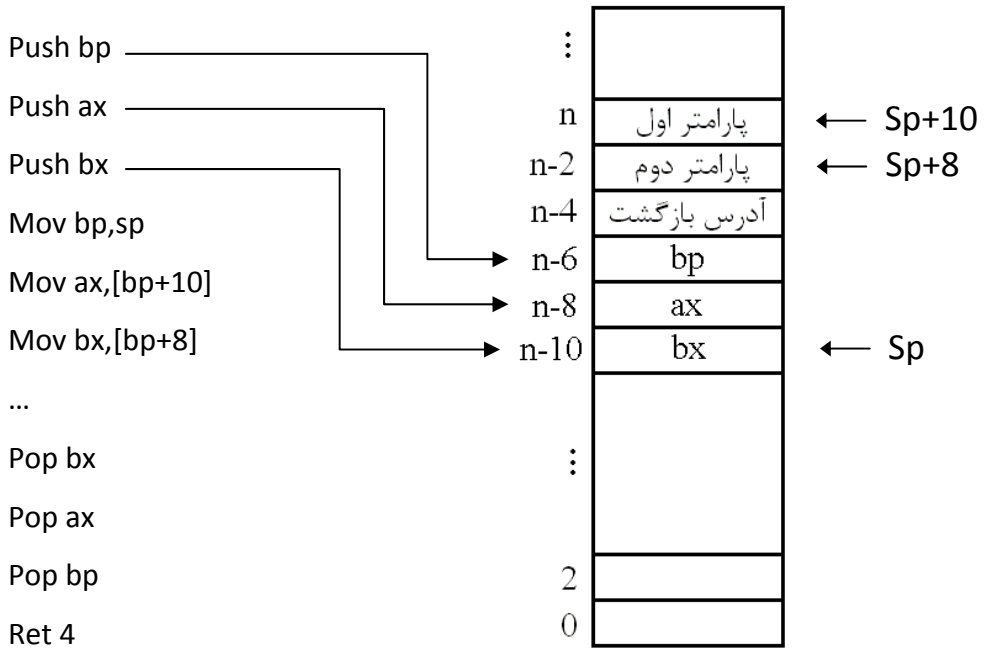
مثال: با در نظر گرفتن وضعیت پشته در شکل (۹-۲ الف) می‌خواهیم مقدار پارامتر اول را به Ax و مقدار پارامتر دوم را به Bx منتقل کنیم. برای اینکار کد زیر مناسب است

```
Mov bp,sp

Mov ax,[bp+4]

Mov bx[bp+2]
```

نکته : چنانچه بخواهیم مقادیر ثبات‌ها پس از خاتمه‌ی زیر برنامه بدون تغییر باشند ، باید قبل از هرچیز ثبات‌های bp ، ax و bx در پشته push می‌شدند که در این صورت باید کد را به صورت زیر می‌نوشتیم:



نکته : پس از خاتمه‌ی کار زیربرنامه ، دستور ret آدرس بازگشت را از پشته بازیابی کرده و کنترل را به زیربرنامه‌ی فراخوان باز می‌گرداند. مقادیر پارامترها همچنان در پشته باقی‌خواهند ماند و این امر ممکن است منجر به ایجاد مشکل در زیربرنامه‌ی فراخوان گردد. برای جلوگیری از این مشکل ، وظیفه‌ی زیربرنامه‌است که به طریقی پارامترهای خود را از پشته خارج کند. این کار با نوشتن تعداد بایت‌های اشغال شده توسط پارامترها در جلوی دستور ret/retf امکان پذیر است. در مثال فوق ، چون زیربرنامه دارای دو پارامتر است، مقدار ۴ جلوی دستور ret نوشته شده است. یعنی دستور ret ۴ بایت بعد از آدرس بازگشت را نیز از پشته حذف می‌کند.

نکته : در انتقال پارامترها از طریق پشته ، پارامترهای را ۱۶ یا ۳۲ بیتی درنظر بگیرید تا در push کردن‌آن‌ها مشکلی نداشته باشید.

رویکرد مشابهی برای انتقال پارامترهای خروجی وجود دارد که خارج از حوصله‌ی این جلسه بوده و انتظار می‌رود دانشجویان عزیز بتوانند با تعمیم موارد مطرح شده ، اینکار را انجام دهند.

تمارین

۱-فرض کنید زیربرنامه‌ای از نوع far با دستورات زیر آغاز می‌شود

```
Push bp
Mov bp,sp
Push ax
Push bx
Push cx
```

این زیربرنامه ۳ پارامتر ورودی دارد که در پشته قرار دارند. وضعیت پشته را رسم کرده و دستورات مناسب برای بازیابی مقادیر این پارامترها از پشته را بنویسید.

۲-زیربرنامه‌ای بنویسید که آفست یک آرایه (در ثبات dx) و تعداد عناصر آن (در ثبات cx) را به عنوان ورودی دریافت کرده و مینیمم این آرایه را در ثبات ax قرار دهد.

۳-زیربرنامه‌ای بنویسید که از طریق پشته ، دو آفست حافظه را بعنوان ورودی گرفته و محتویات این دو محل را با یکدیگر تعویض نماید.

۴-با استفاده از زیربرنامه‌ی طراحی شده در تمرین ۳ ، برنامه‌ای برای مرتب‌سازی یک آرایه بنویسید.

*۵- زیربرنامه‌ای بازگشتی برای محاسبه‌ی فاکتوریل یک عدد بنویسید. (کد زیر به زبان C این کار را انجام می‌دهد)

```
int fact(int n)
{ if(n==0) return 1;
  else return fact(n-1)*n;
}
```