فصل سوم

ساختار ریزپردازنده و دستورالعمل‌های ۸۸/۸۰۸۶

سرفصل مطالب

در این فصل ساختار **CPU** در ریزپردازنده های ۸۰۸۶ و ۸۰۸۸ و نیز مجموعه دستورالعمل‌های آنها را بررسی می‌کنیم. موضوعات اصلی مطرح شونده در این فصل عبارتند از:

1. ساختار **CPU** های ۸۰۸۶ و ۸۰۸۸
2. حافظه‌ی قسمت‌بندی شده (**Segmented**)
3. مدهای آدرس‌دهی

دستورالعمل‌ها (مطالعه توسط دانشجو)

1. دستورالعمل‌های انتقال داده
2. دستورالعمل‌های رشته ای (**String**)
3. دستورالعمل‌های منطقی
4. دستورالعمل‌های ریاضی
5. دستورات انتقال کنترل برنامه
6. دستورالعمل‌های کنترل پردازنده

مقدمه

- **در هنگام مطالعه ی ریزپردازنده‌ها دو ملاحظه اساسی باید در نظر گرفته شود:**
  - ساختار CPU، شامل معماری داخلی متشکل از ثبات‌ها و پرچم های داخلی و نیز مجموعه دستورالعمل‌های شناخته شده و قابل اجرا در آن ریزپردازنده است.
  - مدارات واسط الکتریکی که شامل باس‌های داده، آدرس و کنترل، مدار بازنشانی، مولد پالس ساعت و ... است.

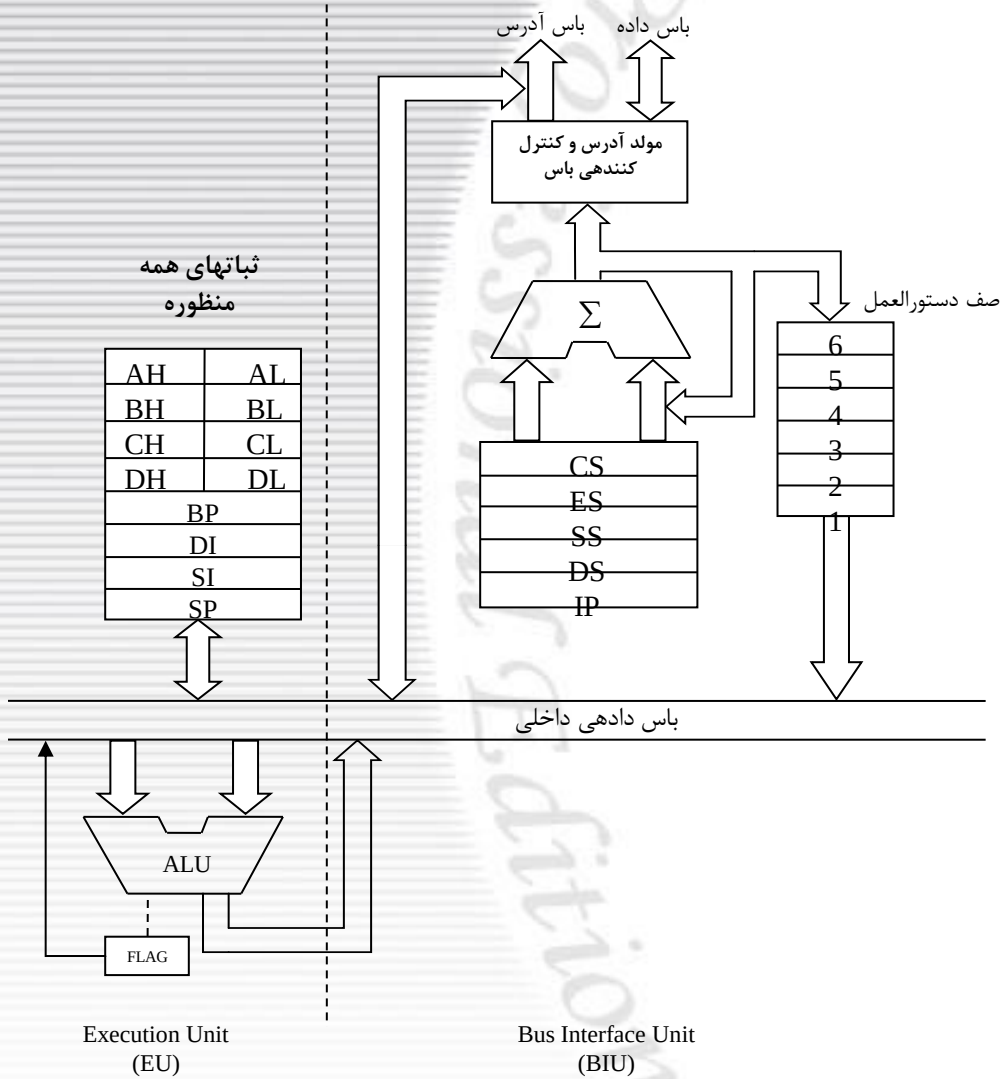
ساختار CPU های ۸۸/ ۸۰۸۶

- در یک ریزپردازنده وظیفه CPU تولید کردن همه سیگنال‌های زمان‌بندی سیستم و همزمان‌سازی انتقال داده بین حافظه، دستگاه های ورودی/خروجی و خود CPU است.
- ریزپردازنده همه این عملیات را از طریق ساختاری با سه باس انجام می‌دهد.
- ریزپردازنده عملکرد نرم‌افزاری نیز دارد.
- ریزپردازنده باید دستورات برنامه را که از حافظه واکشی می شود، شناسایی، دیکود و اجرا کند.
- اجرای دستورات به یک واحد محاسباتی برای انجام عملیات ریاضی و منطقی نیاز دارد که آنرا ALU نامیدیم.

ساختار CPU در ۸۸/ ۸۰۸۶

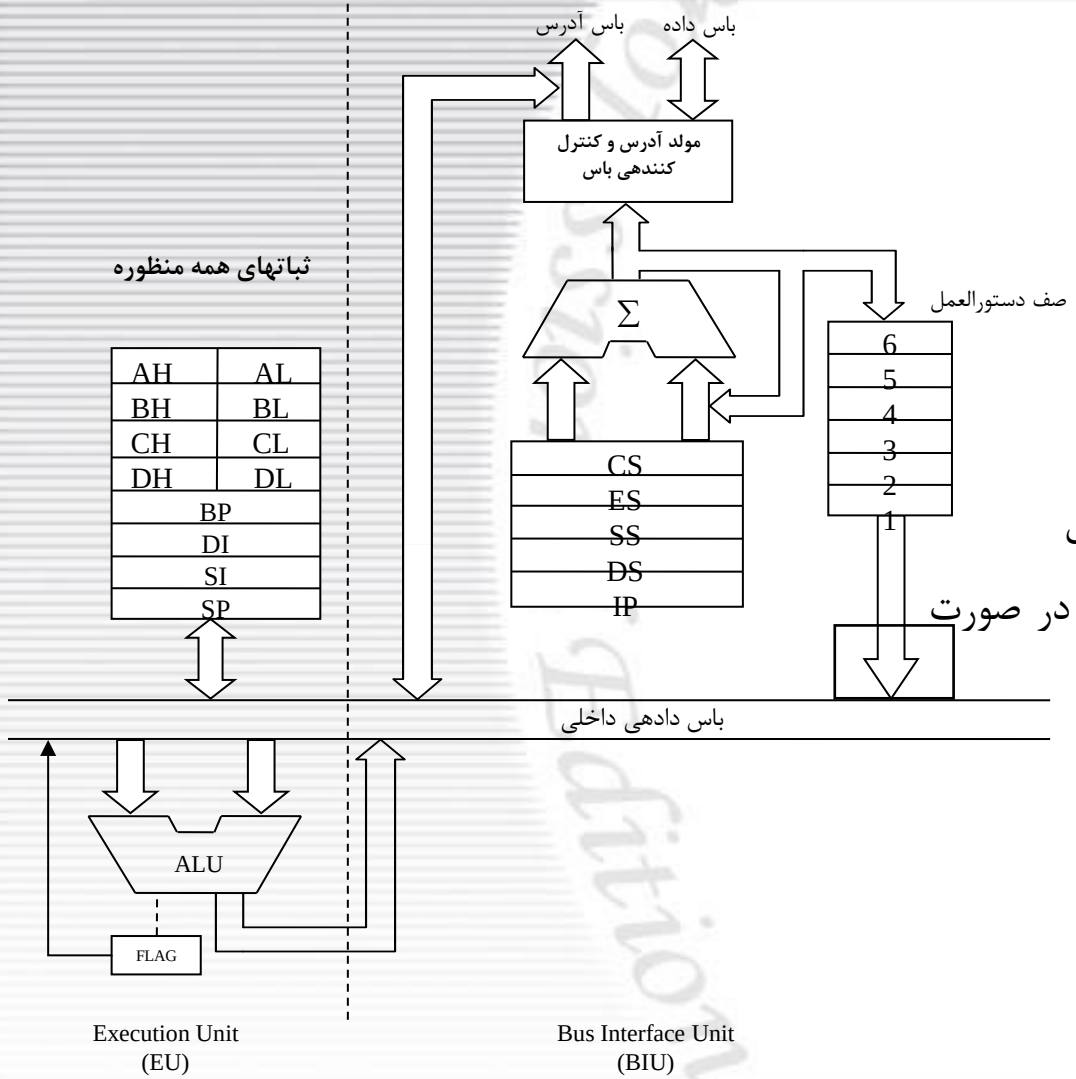
اجزاء CPU:

- واحد BIU
- واحد EU



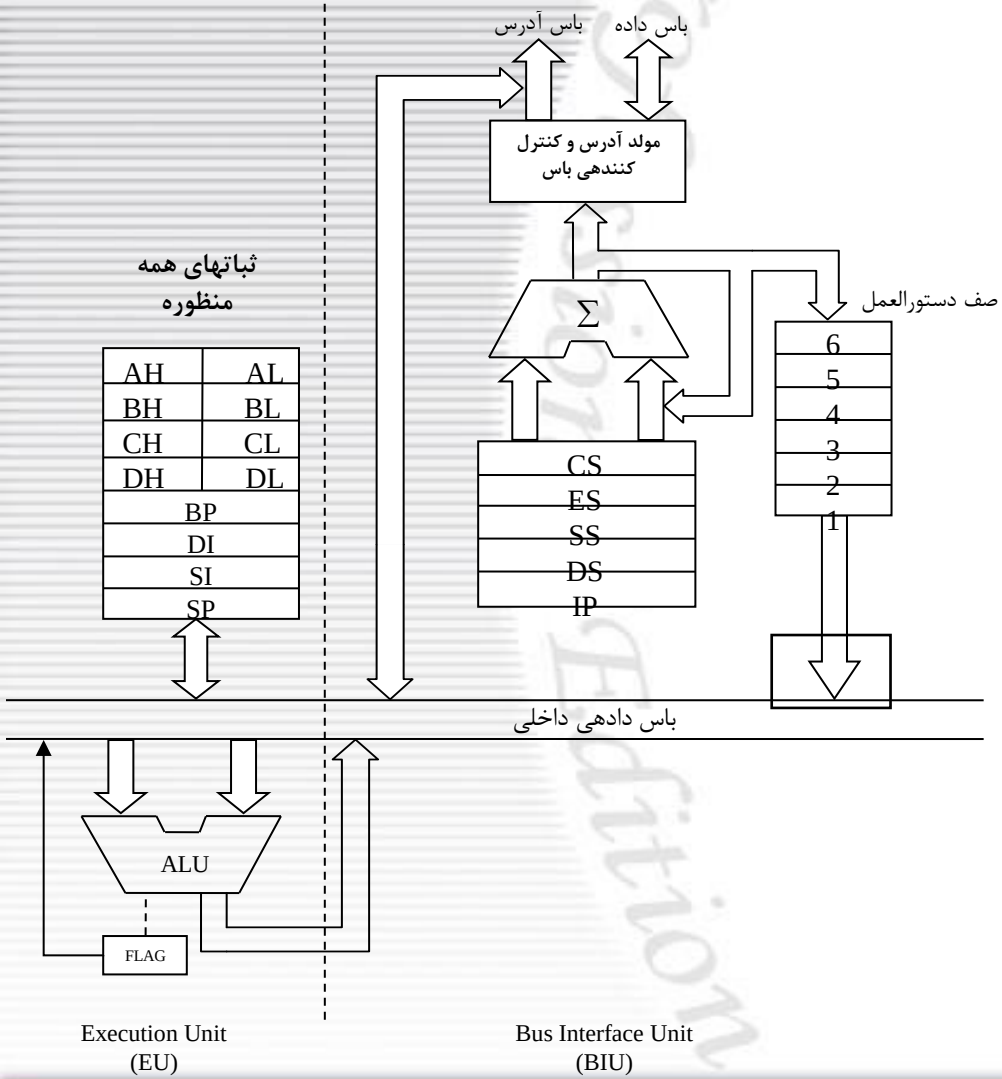
شکل ۱

ساختار CPU های ۸۸/ ۸۰۸۶



- وظایف واحد BIU:
  - تولید آدرس‌های فیزیکی
  - گذاشتن آنها روی باس آدرس
  - تولید سیگنال‌های کنترلی
  - انتقال دستورالعمل‌ها به درون ریزپردازنده
  - گذاشتن دستورالعمل‌ها در صف دستورالعمل
  - بازنشانی (ریست کردن) صف دستورالعمل در صورت لزوم

ساختار CPU های ۸۸/ ۸۰۸۶

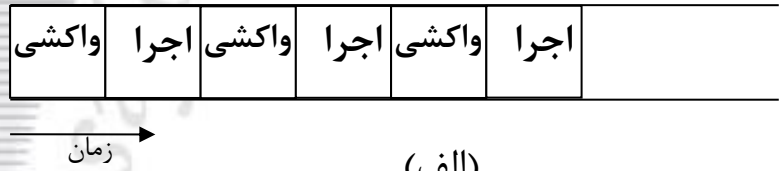


- وظایف واحد EU:
  - کد دستورالعمل‌های برنامه و نیز داده‌ها را از BIU دریافت می‌کند
  - دستورات را اجرا می‌کند
  - نتایج را در ثبات‌های عمومی ذخیره می‌کند
  - با برگرداندن این نتایج به واحد BIU، می‌توان آن را در مکان‌های حافظه یا پورت های خروجی نوشت.
- واحد EU هیچ ارتباطی با باس‌های سیستم ندارد و همه داده‌ها را از طریق BIU دریافت می‌کند یا می‌فرستد .
- با اجرای دستور العمل‌ها پرچم‌ها را متاثر یا از وضیت پرچم‌ها در اجرای دستورات شرطی استفاده می‌کند.

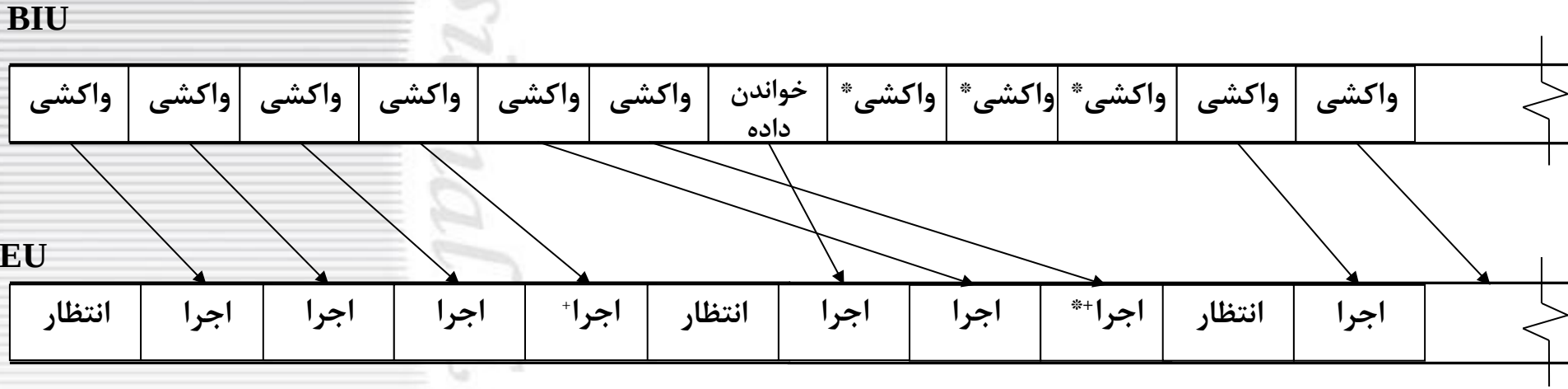
ساختار CPU های ۸۸/ ۸۰۸۶

- تفاوت موجود بین ریزپردازنده‌های ۸۰۸۶ و ۸۰۸۸ در واحد BIU است:
  - در ۸۰۸۸ عرض باس داده BIU، هشت بیتی و در ۸۰۸۶ این باس ۱۶ بیتی است.
  - صف دستورالعمل در ۸۰۸۸ به جای شش بایت، چهار بایتی است.

واکشی و اجرای دستورالعمل



(الف)



* : این بایت‌ها دور ریخته می‌شوند.

+: این دستور درخواست داده‌ای را دارد که در صف موجود نیست.

+*: دستور پرش اتفاق افتاده است.

(ب)

واکشی و اجرای دستورالعمل

دو حالت وجود دارند که واحد EU را به مد انتظار می‌برند:

- اولین حالت زمانی اتفاق می‌افتد که دستور اجراشونده نیاز به دسترسی به یک خانه‌ی حافظه برنامه دارد که در صف موجود نیست.

- حالت دوم زمانی اتفاق می افتد که EU بخواهد یک دستور پرش (jump) را اجرا کند. در این حالت کنترل برنامه به یک آدرس جدید که در ترتیب قبلی قرار ندارد، منتقل می‌شود.

یک حالت وجود دارد که باعث می‌شود واکشی دستورات در BIU به تعویق افتد و آن زمانی است که یک دستور کند در EU اجرا می‌شود.

- مثل دستور AAM که به ۸۳ پالس کلاک نیاز دارد تا کامل شود. بعد از واکشی دستور، صف کاملا پر می‌شود و لذا در حین اجرای این دستور عملیات واکشی به حالت تعویق در می‌آید.

ثبات‌های ۸۰۸۶

برنامه نویس ۸۸/۸۰۸۶ باید با ثبات‌های متنوع به کار رفته در واحدهای BIU و EU آشنا باشد.

این ثبات‌ها در چند گروه قرار داده شده‌اند.

نام ثبات	نام کلی	بایت کم ارزش	بایت پر ارزش	نام گروه
Accumulator	AX	AL	AH	گروه داده
Base	BX	BL	BH	
Count	CX	CL	CH	
Data	DX	DL	DH	
Stack pointer	-	SP		گروه اشاره گر و اندیس
Base pointer	-	BP		
Source index	-	SI		
Destination index	-	DI		
Instruction pointer		IP		

ثبات‌های ۸۰۸۶

گروه سگمنت		ES CS DS SS	- - - -	Extra Code Data Stack
پرچم کنترل و وضعیت	FLAGS H	FLAGS L	-	Status and Control flags

ثبات‌های ۸۰۸۶

• گروه **"ثبا های داده"** شامل ثبات‌های AX (آکومولاتور)، BX، CX و DX است که همگی ۱۶ بیتی هستند.

• هر کدام از این ثبات‌ها به عنوان یک بایت یا یک کلمه قابل دسترسی هستند.

• BX به همه ۱۶ بیتِ ثبات اشاره می کند در حالیکه BH فقط به هشت بیتِ پرارزشِ ثبات BX اشاره دارد.

• ثبات‌های داده برای ذخیره موقتی نتایجی به کار می رود که متعاقبا در دستورات بعدی به کار گرفته می شوند.

•

ثبات‌های ۸۰۸۶

- گروه"ثباتهای **اشاره‌گر شامل SP، BP و IP**" همگی ثبات‌های ۱۶ بیتی هستند (نمی توان به هشت بیت پرارزش یا کم ارزش آنها به تنهایی دسترسی داشت).

- عملکرد ثبات IP اشاره کردن به دستورالعمل بعدی است که BIU باید واکشی کند.

- ثبات IP از نظر سخت‌افزاری بخشی از BIU است و بر خلاف دیگر ثبات‌های اشاره‌گر، تحت کنترل مستقیم برنامه نویس نیست.

- ثبات های SP و BP برای اشاره به پشته استفاده می شوند.
  - معمولا SP برای اشاره به بخشی از پشته که برای ذخیره آدرس بازگشت از زیرروال‌ها و وقفه‌ها در نظر گرفته شده بکار می‌رود.
  - BP برای به اشاره بخشی از پشته که برای ذخیره داده بکار می‌رود استفاده می‌شود. مثل ذخیره محتوای ثبات‌ها در بخش داده پشته، بلافاصله بعد از ورود به زیر روال یا روال وقفه و بازیابی آنها قبل از خروج از زیرروال یا وقفه.

ثبات‌های ۸۰۸۶

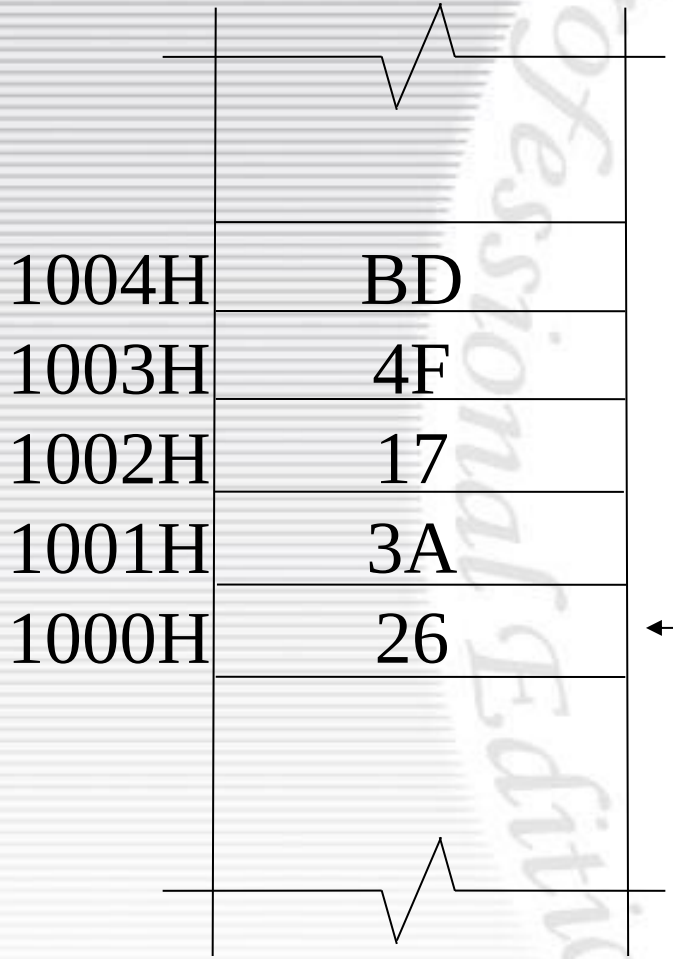
•گروه "ثباتهای **اندیس SI و DI**" همگی ثبات‌های ۱۶ بیتی هستند (نمی توان به هشت بیت پرارزش یا کم ارزش آنها به تنهایی دسترسی داشت).

•این ثباتها برای اشاره کردن به خانه های حافظه به کار گرفته می شوند.

• دستور MOV AH, [SI] به صورت "بایتی که آدرس آن در ثبات SI قرار دارد را به ثبات AH منتقل کن" تفسیر می شود. بنابراین SI به خانه‌ی حافظه‌ی مطلوب اشاره می‌کند.

•علامت [ ] در دو طرف SI بیانگر آدرس‌دهی غیر مستقیم است یعنی محتوای خانه حافظه‌ای که SI به آن اشاره می‌کند و نه خود SI.

ثبات‌های ۸۰۸۶



مثال : اگر محتوای ثبات SI برابر با 1000H باشد، بعد از اینکه دستور MOV AH,[SI] اجرا شد، محتوای ثبات AH چقدر خواهد بود؟

حل: با توجه به شکل، چون محتوای آدرس 1000H از حافظه، مقدار 26H است، محتوای ثبات AH مقدار 26H خواهد بود.

مثال: بعد از اینکه دستور MOV AX, [SI] اجرا شود، محتوای ثبات AX چقدر خواهد بود؟

حل: مقدار 3A26H

۸۸/۸۰۸۶ همواره کلمات را به گونه‌ای در حافظه ذخیره می‌کند که بایت پرارزش در آدرس پرارزش‌تر کلمات حافظه قرار گیرد.

ثبات پرچم‌ها

- جدول زیر چگونگی تعریف بیت‌ها برای ثبات ۱۶ بیتی پرچم را نشان می دهد.

- شش بیت از این ثبات شاخص‌های وضعیت هستند که ویژگی‌های نتیجه آخرین محاسبات ریاضی و منطقی انجام شده را بیان می کند:

Flags H								Flags L							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

بیت	نام پرچم	عملکرد
0	CF	پرچم **Carry**: اگر بر بیت پر ارزش نتیجه، **Carry** یا **Borrow** اتفاق افتد، این پرچم **1** شده و در غیر اینصورت **0** خواهد بود .
2	PF	پرچم **Parity**: این پرچم **1** می شود اگر تعداد بیت‌های ۱ در هشت بیت مرتبه پایین نتیجه، زوج باشد. در غیر اینصورت **0** می شود.
4	AF	اگر از چهار بیتِ کم ارزش **AL**، **Carry** یا **Borrow** اتفاق افتد، مقدار این پرچم برابر**1** و گرنه **0** می شود.
6	ZF	پرچم **Zero**: اگر نتیجه صفر باشد، این پرچم **1** وگرنه **0** می شود
7	SF	پرچم **Sign**: این پرچم مقدار پرارزش ترین بیت نتیجه را می گیرد. (بیت علامت)
8	TF	پرچم **Single-step**: وقتی این پرچم **1** باشد، بعد از اجرای دستورالعمل بعدی، یک وقفه **single-step** اتفاق می افتد. با به وجود آمدن وقفه ی **single-step**، این بیت **0** می شود
9	IF	پرچم **Interrupt-enable**: وقتی این پرچم **1** شود، وقفه های قابل **mask** شدن باعث می شوند که **CPU**، کنترل برنامه را به مکان بردار وقفه منتقل کند.
10	DF	پرچم **Direction**: **1** بودن این پرچم، موجب می گردد که دستورات رشته ای به طور خودکار ثبات اندیس مربوطه را کاهش دهند، اگر **0** باشد، افزایش خودکار صورت می گیرد.
11	OF	پرچم **Overflow**: اگر نتیجه محاسبات علامت دار انجام شده قابل قرار گرفتن در تعداد بیت‌های عملوند مقصد نباشد (سرریز رخ دهد)، این پرچم **1** می شود.

پرچم ها

پرچم ها بعد از اجرای دستور ADD AL, 1 با فرض AL=7FH

AL = 80H	7FH + 1 = 80H
CF = 0	در اثر عملیات جمع، بیت Carry به وجود نیامده است
PF = 0	عدد 80H تعداد فردی از بیت 1 دارد.
AF = 1	یک بیت Carry از بیت 3 به بیت 4 وارد شده است.
ZF = 0	نتیجه صفر نیست.
SF = 1	مقدار بیت هفتم، 1 است.
OF = 1	با فرض علامتدار بودن محتوای AL، با اضافه کردن ۱ به 7FH، محتوا از محدوده یک عدد ۸ بیتی مثبت خارج می شود و اصولا یک عدد منفی می‌شود.

پرچم‌ها

- برنامه نویس می‌تواند سه تا از بیت‌های پرچم را مستقیما مقداردهی کند و با آنها عملکرد ریزپردازنده را کنترل کند. این پرچم ها TF، IF و DF هستند.

- مقدار 1 در **پرچم TF**, عملکرد پردازنده را در مد **تک‌گامی (single step)** قرار می‌دهد.
  - این مد برای عیب‌یابی برنامه‌ها بسیار مفید است.

- در اینصورت کنترل برنامه بعد از اجرای هر دستورالعمل، به مکان خاصی از حافظه که برنامه نویس قبلا آدرس آن را مشخص کرده است منتقل می شود.

- معمولا برنامه ای برای نمایش دادن همه ثبات‌ها و بیت های CPU در آن مکان ذخیره شده است.

- بدین ترتیب بعد از اجرای هر دستور برنامه‌نویس می‌تواند مقادیر ثبات‌ها و پرچم ها را بررسی کند.

پرچم ها

• هنگامی که بیت **پرچم IF** (پرچم وقفه) مقدار 1 داشته باشد، خط ورودی مربوط به وقفه‌های خارجی (INTR) فعال می‌شود.

•**پرچم DF** با دستورات "انتقال بلوکی" (که انتقال رشته ای "String" نیز نامیده می‌شوند) به کار می‌رود.

•هنگامی که پرچم DF در وضعیت 1 قرار دارد، اشاره گرِ حافظه بلوکی به صورت خودکار کاهش می‌یابد و اگر در وضعیت 0 باشد افزایش می‌یابد.

```
MOVSB       ;     (ES(0)+DI) ← (DS(0)+SI)
            ;     Increment or decrement SI
            ;     Increment or decrement DI
```

ثبات‌های سگمنت

- آخرین گروه از ثبات‌ها، گروه ثبات های "Segment" است.

- واحد BIU این ثبات‌ها را به کار می گیرد تا خروجی آدرس حافظه ای که CPU در هنگام کار با حافظه مشخص کرده است، را تعیین کند.

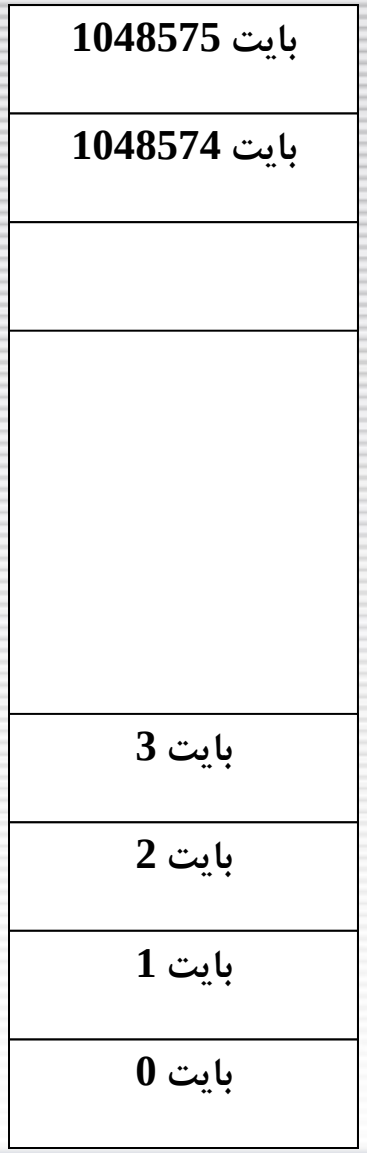
- قبل از بررسی بیشتر این واحد، ابتدا چگونگی تقسیم‌بندی حافظه به بخش‌های مختلف را بررسی می‌کنیم.

حافظه قسمت‌بندی شده

• اگرچه ۸۰۸۶ یک ریزپردازنده ۱۶ بیتی است (عرض باس داده آن ۱۶ بیتی است)، حافظه آن همچنان به صورت بایتی در نظر گرفته می‌شود. این مسئله چند مزیت دارد:

- اول اینکه پردازنده می‌تواند بر روی بایت ها یا کلمه ها کار کند.
- این ویژگی خصوصا هنگام کار با وسایل I/O جانبی مثلِ چاپگر، مودم و ... که همگی با داده های ۷ یا ۸ بیتی در قالب کدهای ASCII کار می کنند، مهم می شود.
- دوم اینکه بسیاری از دستورات ۸۰۸۶ و ۸۰۸۸، تک بایتی هستند. دیگر دستورات ممکن است بین دو تا هفت بایت را اشغال کنند.
- توانایی دسترسی تک بایتی، کنترل دستوراتی که تعداد فردی از بایت‌ها را اشغال می‌کنند را آسان می‌سازد.

حافظه‌ی قسمت بندی شده



•۸۰۸۶ دارای ۲۰ خط آدرس است و توانایی آدرس دهی 1MB مکان حافظه متفاوت را دارد.

•۸۰۸۶ با خواندن همزمان یک بایت با آدرس فرد و بایت دیگری با آدرس زوج، یک داده‌ی ۱۶بیتی را از حافظه می‌خواند.

•به همین دلیل حافظه ی ۸۰۸۶، به دو بانک داده با آدرس های زوج و فرد تقسیم شده است.

• ۸۰۸۶ سیگنال هایی را از طریق باس کنترل فراهم می‌کند که در واحد حافظه دیکود می شود و تعیین می‌کنند که یک بایت یا یک کلمه از حافظه باید خوانده شود.

ثبات‌های سگمنت

- درون فضای حافظه ۱ مگابایتی ۸۸/۸۰۸۶، چهار بلوک ۶۴ کیلوبایتی به اسامی **سگمنت‌های کد، داده، پشته و اضافی** تعریف شده‌اند. CPU هرکدام از این **سگمنت‌ها** را به گونه‌ای متفاوت به کار می‌برد.

- کاربرد سگمنت‌ها:
  - **سگمنت کد** دستورات برنامه را در خود نگه می‌دارد.
  - **سگمنت داده**, داده‌های برنامه را در خود نگه می‌دارد.
  - **سگمنت اضافی** در واقع یک سگمنت داده اضافی است که اغلب برای داده‌های اشتراکی به کار می‌روند.
  - **سگمنت پشته** برای ذخیره کردن وقفه‌ها و آدرس‌های بازگشت از زیر روتین‌ها به کار می‌رود.

ثبات‌های سگمنت

•چهار ثبات سگمنت به نام های CS، DS، ES و SS برای اشاره کردن به مکان صفر (آدرس پایه) هر کدام از بلوک های فوق بکار می‌روند.

•چون این ثبات‌ها ۱۶ بیتی هستند ولی آدرس های حافظه ۲۰ بیتی، واحد BIU چهار مقدار منطقی 0 به بیت‌های کم ارزش این ثبات ضمیمه می‌کند که این کار مقدار عدد این ثبات را در ۱۶ ضرب می‌کند.

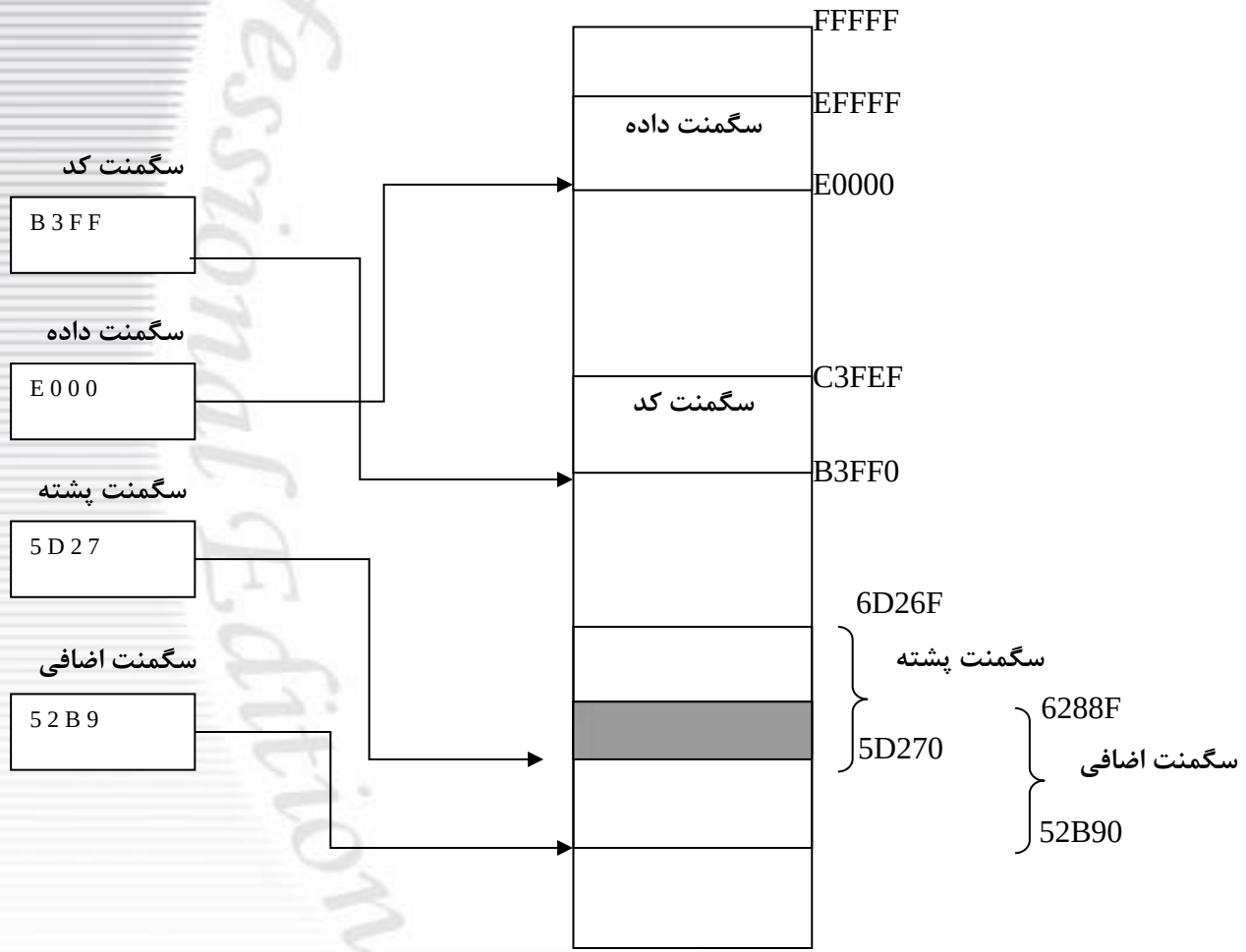
•بعنوان مثال ثبات CS حاوی B3FFH است ولی معادل آدرس B3FF0H در نظر گرفته می‌شود: CS(0)

•نکته مهم این است که آدرس شروع هر سگمنت اختیاری نیست و باید عددی بخش پذیر بر ۱۶ باشد یعنی آنکه چهار بیت کم ارزش آدرس باید 0 باشد.

• لازم نیست که همه چهار سگمنت جدای از هم تعریف شوند. مثلا در شکل اسلاید بعد سگمنت‌های پشته و اضافی، هم‌پوشانی دارند.

•حتی همه سگمنت‌ها هم می‌توانند هم پوشانی داشته باشند.

مثالی از آدرس پایه بلوک‌های ۶۴ کیلو بایتی و محدوده‌های آنها که توسط ثبات‌های سگمنت مشخص شده‌اند.



ثباتهای سگمنت

مثال: اگر محتوای ثبات DS برابر با E000H باشد، آدرس شروع و پایان سگمنت داده را بیابید ؟

حل: آدرس شروع با افزودن چهار مقدار 0 به انتهای ثبات DS حاصل می شود و برابر است با E0000H. آدرس انتها با جمع کردن عدد FFFFH (که معادل ۶۴ کیلوبایت است) به آدرس شروع حاصل می شود EFFFFH.

مکان‌هایی از حافظه که در یکی از سگمنت‌های جاری قرار نمی‌گیرند، قابل استفاده ۸۸/۸۰۸۶ نیستند، تا زمانی‌که یکی از ثبات‌های سگمنت آن مکان را شامل شوند.

در هر لحظه، تنها ۲۵۶ کیلوبایت (چهار بلوک ۶۴ کیلوبایتی) از حافظه ۱ مگابایتی قابل استفاده است.

محتوای ثبات‌های سگمنت تنها از طریق نرم افزار تعیین می‌شوند و بنابران دستوراتی که این ثباتها را مقداردهی می‌کنند در اولین بخش هر برنامه ی ۸۸/۸۰۸۶ قرار می‌گیرند.

آدرس های فیزیکی و منطقی

- چون حجم هر سگمنت ۶۴ کیلوبایت است، محدوده آدرس‌های آن از 0000H تا FFFFH است.

- یک آدرس درون سگمنت را **یک آدرس منطقی یا افست** گویند. مثلا آدرس منطقی 0005H درون سگمنت کد، نشان داده شده در شکل قبل واقعا مربوط به آدرس B3FF0H+0005H است. این آدرس واقعی را **آدرس فیزیکی** گویند.

- طول آدرس فیزیکی ۲۰ بیت است و مربوط به آدرس واقعی است که واحد BIU بر خطوط باس آدرس قرار می دهد. آدرس منطقی یک افست از آدرس صفر که مربوط به ابتدای یک سگمنت است را نشان می دهد.

مثال: آدرس فیزیکی مربوط به آدرس منطقی D470H در سگمنت اضافی و نیز آدرس 2D90H در سگمنت پشته را محاسبه کنید.

حل:

برای سگمنت پشته آدرس پایه 52B90H است لذا آدرس فیزیکی مطلوب = 52B90H + D470H 60000H است.

آدرس پایه ی سگمنت پشته، 5D270H است، پس آدرس فیزیکی خواسته شده = 5D270H + 2D90H 60000H است.

آدرس های فیزیکی و منطقی

- **آدرس های منطقی متفاوت ممکن است به آدرس‌های فیزیکی یکسانی نگاشته شوند.**
- این مساله می‌تواند بسیار دردسرساز باشد، چون ممکن است بعنوان مثال، **داده‌های برنامه** بر روی **داده‌های پشته** که آدرس های بازگشت از زیربرنامه‌ها است نوشته شود و کل برنامه مختل شود.
- اگر مقدار ثبات IP=1000H باشد، دستور بعدی از کجای حافظه واکشی می شود؟
- تمام دستوراتی که به حافظه مراجعه می کنند، یک ثبات **سگمنت پیش‌فرض** دارند.
- **واکشی دستور فقط از سگمنت کد** انجام می شود که **آدرس منطقی** آن را ثبات IP تعیین می‌کند.
- به طور مشابه ثبات SP به عنوان اشاره گر پیش فرض برای **سگمنت پشته** به کار می رود.

ثبات‌های سگمنت پیش فرض و جایگزین

آدرس منطقی	سگمنتهای جایگزین	سگمنت پیش فرض	نوع رجوع به حافظه
IP	ندارد	CS	واکشی دستورالعمل
SP	ندارد	SS	عملیات پشته
آدرس موثر	CS, ES,SS	DS	داده عمومی
SI	CS, ES,SS	DS	منبع رشته
DI	ندارد	ES	مقصد رشته
آدرس موثر	CS, ES,SS	DS	BX بعنوان اشاره گر
آدرس موثر	CS, ES,DS	SS	BP بعنوان اشاره گر

- جدول فوق در واحد BIU برنامه ریزی شده است.

- مثلا اگر ثبات‌های IP=1000H و CS=B3FFH باشند، آدرس فیزیکی ساخته شده در BIU برابر است با B3FF0H + 1000H = B4FF0H.

آدرس موثر

- آدرس‌های موثر به ترتیب زیر قابل تولید هستند:

آدرس موثر					مود آدرس‌دهی
ثبات شاخص		ثبات پایه		جابجایی	مد آدرس دهی
ندارد SI یا DI	+ +	BX یا BP ندارد	+ +	ندارد ندارد	ثباتی غیر مستقیم
SI یا DI	+	ندارد	+	۱۲۸- تا ۱۲۷+	شاخص دار
ندارد	+	BX یا BP	+	۱۲۸- تا ۱۲۷+	آدرس پایه
SI یا DI	+	BX یا BP	+	ندارد	آدرس پایه و شاخص دار
SI یا DI	+	BX یا BP	+	۱۲۸- تا ۱۲۷+	آدرس پایه و شاخص دار به همراه جابجایی

آدرس های فیزیکی و منطقی

مثال: اگر مقدار ثبات BP = 2C30H باشد، در اجرای دستورالعمل MOV [BP], AL کدام آدرس فیزیکی حافظه به کار می رود؟ می‌دانیم که SS=5D27H.

حل: جدول اسلاید قبل بیان می کند که در اجرای چنین دستوری **سگمنت پشته** به کار می رود.

آدرس فیزیکی به کار رفته برابر است با 5D270H + 2C30H = 5FEA0H.

- امکان تغییر دادن تعریف سگمنت نشان داده شده در جدول اسلاید قبل وجود دارد.

- مثلا ثبات BP را می‌توان به عنوان اشاره گر به سگمنت های کد، داده و اضافی نیز به کار برد. در حالیکه واکشی دستور تنها از سگمنت کد امکان پذیر است

تعریف مکان های حافظه

اگر بخواهیم یک کلمه از حافظه را برچسب دهیم از عملگر DB استفاده می‌کنیم.

```
DATA SEGMENT
MEMBDS DB 50H
```

این دنباله مقدار 50H را در محلی از حافظه با برچسب MEMBDS اختصاص می‌دهد.

- اگر بخواهیم یک کلمه از حافظه را برچسب دهیم از عملگر DW استفاده می‌کنیم.

•مثال:

```
MEMWDS DW 1234H
```

- •عملگر DD برای تعریف یک کلمه دوتایی (۳۲ بیتی) به کار می رود.
- عملگر DQ برای تعریف کلمه چهارتایی (هشت بایتی)
- عملگر DT، ده بایت را تعریف می کند.
- تعریف آرایه ای از داده ها بدون مقدار اولیه به صورت ? DUP 100 است.

•دستور آخر ۱۰۰ بایت داده متوالی را در حافظه رزرو می کند. برای مقداردهی همه ۱۰۰ بایت آرایه به مقدار اولیه ی 0 از دستور 100 DUP(0) استفاده می کنیم.

•دستور DB “THIS IS A MESSAGE” باعث می شود که اسمبلر ۱۷ بایت متوالی از حافظه را رزرو کند و مقدار کد اسکی کاراکترهای قرار گرفته بین دو علامت “” را در آن بخش از حافظه قرار دهد.

مدهای آدرس‌دهی

- همان گونه که قبلا عنوان شد، دستورالعمل‌های کامپیوتری از یک کد عملیاتی (op-code) به همراه هیچ، یک یا دو عملوند تشکیل شده‌اند.

- op-code عملیاتی که باید اجرا شود را تعیین می‌کند و عملوندها منبع و مقصد داده‌هایی هستند که این عملیات بر روی آنها اجرا می‌شود.

- عملوندها می‌توانند یکی از ثبات‌های CPU، یکی از مکان‌های حافظه در یکی از سگمنت‌ها یا یک پورت I/O باشند.

- روش‌های متفاوتی که یک ریزپردازنده آدرس این عملوندها را تعیین می‌کند، مدهای آدرس‌دهی نامیده می‌شوند. جدول اسلاید بعدی، نُه روش متفاوت آدرس‌دهی که برای ۸۸/۸۰۸۶ موجود است را نشان می‌دهد.

- در مثال‌های بعدی فرض شده است، MEMBDS به مکان 1000H درون سگمنت داده اشاره می‌کند.

- برای تمیز دادن بایت از کلمه از BYTE PTR و WORD PTR استفاده شده است.

مد آدرس‌دهی بلافصل

- مقدار جابجایی که در بعضی دستورات استفاده شده است به عنوان یک عدد باینری مکمل دو، به ثبات پایه یا اشاره گر اضافه می‌شود.

- دستوراتی که مد آدرس‌دهی بلافصل را به کار می‌گیرند، داده مورد استفاده شان را به عنوان بخشی از خود دستورالعمل دریافت می‌کنند.

- این مد حالت خاصی است که در آن داده (که جزء خود دستورالعمل است) به جای ذخیره شدن در سگمنت داده در سگمنت کد قرار می گیرد.

- برای دستور MOV AX, 1000H بخش داده دستورالعمل، فورا بعد از op-code این دستور که B8 است قرار می‌گیرد.

- مد آدرس‌دهی بلافصل بیشتر برای مقداردهی اولیه به ثبات‌ها و خانه های حافظه به کار می رود.

- یک محدودیت موجود آن است که **ثبات‌های سگمنت را نمی توان بدین طریق مقداردهی کرد.**

مد آدرس‌دهی ثباتی

• برخی از دستورات مستقیما بر روی مقادیر داده های موجود در ثباتهای CPU کار می کنند یا داده ها را بین ثبات‌ها جابجا می‌کنند.

• دستور MOV DX, CX یک کپی از محتوای ثبات CX را به DX منتقل می کند.

• INC BH یک واحد به محتوای ثبات BH می افزاید.

• مد آدرس‌دهی ثبات می تواند با مد بلافصل ترکیب شود و یک مقدار عددی را در ثبات سگمنت بارگذاری کند.

• مثلا:

```
MOV AX, 8010H
MOV DS, AX
```

مد آدرس‌دهی مستقیم

• در مد آدرس‌دهی مستقیم، آدرس مکان حافظه، مستقیما به عنوان بخشی از دستورالعمل قرار می‌گیرد.

• در اغلب موارد برچسب‌هایی به این آدرس‌ها داده می‌شود تا برنامه‌نویس مستقیما با مقادیر عددی درگیر نباشد.

• دستور MOV AH, [MEMBDS] چنین تفسیری دارد: "محتوای مکان حافظه MEMBDS را در ثبات AH ذخیره کن".

مد آدرس دهی	کد شیء (Object Code)	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
بلافصل	B8 00 10	MOV AX, 1000H	کد	AH←10H; AL←00	منبـــــع داده درون دستورالعمل است
ثبات	8B D1	MOV DX, CX	درون CPU	DX ← CX	منبـع و مقصـد داده، ثباتهای CPU است.
مستقیم	8A 26 00 10	MOV AH, [MEMBDS]	داده	AH ← [1000H]	آدرس حافظـــه درون دستورالعمل است.
غیر مستقیم ثباتی	8B 04 FF 25 FE 46 00 FF 0F	MOV AX, [SI] JMP [DI] INC BYTE PTR[BP] DEC WORD PTR[BX]	داده داده پشته داده	AL←[SI]; AH←[SI+1] IP←[DI+1:DI] [BP] ←[BP]+1 [BX+1:BX]←[BX+1:BX]-1	آدرس حافظه در یـک ثبات شاخص یا اشاره-گر قرار دارد.
شاخص دار	8B 44 06 FF 65 06	MOV AX, [SI+6] JMP [DI+6]	داده داده	AL←[SI+6]; AH←[SI+7] IP←[DI+7:DI+6]	آدرس حافظه مجموع ثبات شـاخص و یـک جابجایی درون دستور است.
آدرس پایه	8B 46 02 FF 67 02	MOV AX, [BP+2] JMP [BX+2]	پشته داده	AL←[BP+2]; AH←[BP+3] IP←[BX+3:BX+2]	آدرس حافظه مجموع ثباتهای پایه (BP یـا BX) و یک جابجـایی درون دستور است.
آدرس پایه و شاخص دار	8B 00	MOV AX, [BX+SI]	داده	AL←[BX+SI] AH←[BX+SI+1]	آدرس حافظه مجموع ثباتهـــای شـــاخص و ثبات پایه است.
	FF 21 FE 02 FF 0B	JMP [BX+DI]	داده	[BP+DI+1:BP+DI] ← [BP+DI+1:BP+DI]-1	
		INC BYTE PTR[BP+SI]	پشته		
		DEC WORD PTR[BP+DI]	پشته		
آدرس پایه و شاخص دار به همراه جابجایی	8B 40 05	MOV AX, [BX+SI+5]	داده	AL←[BX+SI+5] AH←[BX+SI+6]	آدرس حافظه مجموع ثبات شاخص، پایـه و جابجایی درون دستور است.
	FF 61 05	JMP [BX+DI+5]	داده	IP←[BX+DI+6:BX+DI+5]	
	FE 42 05	INC BYTE PTR[BP+SI+5]	پشته	[BP+SI+5]←[BP+SI+5]+1	
	FF 4B 05	DEC WORD PTR[BP+DI+5]	پشته	[BP+DI+6:BP+DI+5]← [BP+DI+6:BP+DI+5]-1	
رشته	A4	MOVSB	داده، اضافی	[ES:DI]←[DS:SI] اگر DF=0 سپس: SI←SI+1; DI←DI+1	آدرس حافظه‌ی مبـدا، ثبـــــات SI درون سگمنت داده اسـت، آدرس حافظه‌ی مقصد

مد آدرس‌دهی غیر مستقیم

- مد آدرس دهی مستقیم برای دسترسی های غیر متداول به حافظه مناسب است ولی زمانی که یک مکان حافظه بارها نوشته یا خوانده می شود، استفاده از این روش راندمان برنامه را کاهش می‌دهد چون یک آدرس دو بایتی باید مرتبا واکشی شود.

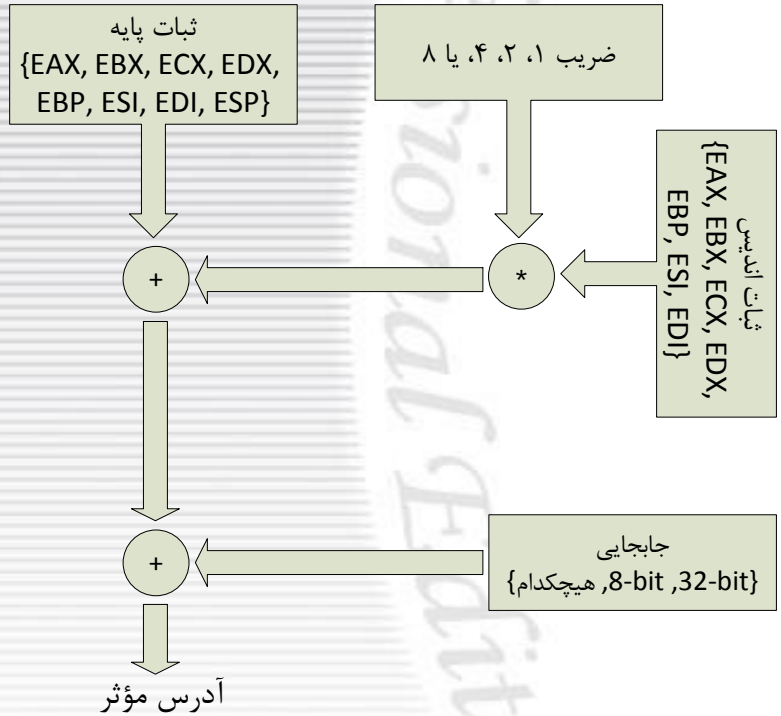
- مد آدرس دهی **غیر مستقیم** این مشکل را حل می کند. در این روش آدرس حافظه در یک ثبات اشاره‌گر یا اندیس (BX، BP، SI و یا DI) قرار می‌گیرد.

- جدول اسلاید بعدی ترکیبات متنوع ممکن را نشان می هد.

مدهای آدرس‌دهی

آدرس موثر					
ثبات شاخص		ثبات پایه		جابجایی	مد آدرس دهی
ندارد SI یا DI	+ +	BX یا BP ندارد	+ +	ندارد ندارد	ثباتی غیر مستقیم
SI یا DI	+	ندارد	+	۱۲۸- تا ۱۲۷+	شاخص دار
ندارد	+	BX یا BP	+	۱۲۸- تا ۱۲۷+	آدرس پایه
SI یا DI	+	BX یا BP	+	ندارد	آدرس پایه و شاخص دار
SI یا DI	+	BX یا BP	+	۱۲۸- تا ۱۲۷+	آدرس پایه و شاخص دار به همراه جابجایی

تولید آدرس موثر در پنتیوم



مثال:

MOV EAX, [EBX + ECX * 4 + 6]

مدهای آدرس‌دهی

- به طور کلی می توان جابجایی را به یک ثبات پایه اضافه کرد و نتیجه را به یک ثبات شاخص اضافه نمود.

- این حالت یک مد تجملی است که مد "**آدرس پایه و شاخص دار به همراه جابجایی**" نامیده می شود.

- در این میان مدهای اقتصادی تری وجود دارد که تنها یکی از ثباتهای شاخص یا پایه را با یا بدون جابجایی به کار می گیرند.

- آدرس بدست آمده را معمولا **آدرس موثر** می نامند.

- مقدار جابجایی به یک بایت محدود می شود و این محدودیت باعث می شود که آدرس موثر بتواند از ۱۲۷+ بایت تا ۱۲۸- بایت نسبت به آدرس موجود در اشاره گر پایه جابه جا شود.

مدهای آدرس دهی

- گاهی عملوند مد آدرس دهی غیر مستقیم به صورت MOV AX, TABLE[SI] دیده می شود.

- TABLE برچسبی است که به یک مکان حافظه داده شده است (احتمالا آدرس پایه یک جدول داده).

- این دستور را می توان به صورت MOV AX, [TABLE+SI] نوشت.

- به هرحال قالب اول بیان می کند که SI به عنوان شاخصی درون جدول TABLE به کار رفته است.

- بنابراین اگر SI=5 باشد، پنجمین عنصر جدول به کار گرفته می شود. اکثر اسمبلر ها هر دو قالب دستور را می پذیرند.

مدهای آدرس‌دهی

مثال : مدآدرس دهی را برای هر کدام از دستورات زیر تعیین کنید.

```
MOV AH, 47H
MOV AH, [BP+2]
MOV AH, [BP+SI]
MOV AH, [XRAY]
MOV AH, TEMP[BX]
```

XRAY و TEMP دو مکان حافظه را تعیین می کنند.

مدهای آدرس دهی

•دو شبه عملگر BYTE PTR و WORD PTR به کمک اسمبلر می آیند تا تعیین کنند که دسترسی به یک بایت یا یک کلمه از حافظه مورد نیاز است.

•مثلا دستور INC [BP] را به این دو صورت می توان تفسیر کرد که "یک کلمه از حافظه" یا "یک بایت از حافظه" را که BP بدان اشاره می کند "یک واحد افزایش بده".

• هنگامی که دستور دو عملوند را تعیین می کند, **عملوند ثباتی** نوع دسترسی به بایت یا کلمه را تعیین می کند. مثلا دستور MOV AX, [BP] چنین تفسیر میشود که "محتوای کلمه ای از حافظه که ثبات BP بدان اشاره می کند را در AX کپی کن»، چون AX یک ثبات دو بایتی است .

• سگمنت پیش فرض برای تمامی مدهای آدرس دهی غیر مستقیم، وقتی که ثبات BP به کار رفته است، **سگمنت پشته** است و هنگامی که ثبات BX، SI یا DI به کار رفته باشد، سگمنت داده است.

مد آدرس دهی رشته ای

- در کامپیوتر، رشته دنباله ای از بایت ها یا کلمات ذخیره شده در حافظه است.

- پردازشگرهای واژه، متن را به صورت رشته های طولانی از کاراکترهای ASCII ذخیره می کنند.

- یک جدول داده مثال دیگری از رشته است.

- به خاطر اهمیت رشته ها، ۸۸/۸۰۸۶ چندین دستورالعمل برای کنترل کردن رشته هایی از کاراکترها دارد.

این دستورات مدهای آدرس دهی ویژه ای برای خود دارند و از DS:SI برای اشاره کردن به منبع رشته و ES:DI برای اشاره کردن به رشته ی مقصد استفاده می کنند.

MOVSB بایت داده مبدا را به مقصد منتقل می کند. نکته اینکه SI و DIبه صورت خودکار با توجه به مقدار DF کاهش یا افزایش می یابند. دستورات رشته ای در ادامه به تفصیل بررسی می شوند.

تغییر ثباتهای سگمنت پیش فرض

- هر دستورالعملی که به حافظه دست می یابد، از یک ثبات سگمنت ویژه استفاده می کند تا آدرس موثر را از آدرس منطقی محاسبه کند.

- اختصاصاتِ پیش فرض ارائه شده برای بعضی از انواع مراجعات حافظه قابل لغو شدن و تغییر یافتن است.

- جدول اسلاید بعدی مثال هایی از به کار بردن پیشوند قبل از دستورالعمل ها برای لغو کردن سگمنت پیش فرض را نشان می دهد.

- در هر بار عملیات لغو کردن سگمنت پیش فرض، تنها برای همان دستور اعمال می شود و در صورت لزوم برای دستورات بعدی باید تکرار شود.

- یک کاربرد نوعی برای لغو کردن سگمنت پیش فرض، به وجود آوردن امکان ذخیره شدن داده در سگمنت کد است.
- مثال:

```
CODE SEGMENT
COUNT DB 0FFH
MOV AL,CS:COUNT
```

تغییر ثباتهای سگمنت پیش فرض

• متغیر COUNT در سگمنت کد تعریف شده است و لذا در دستور MOV لازم است از پیشوند :CS استفاده کنیم تا سگمنت پیش فرض برای داده که DS است لغو و با CS جایگزین شود.

• بعضی از برنامه های اسمبلر آنقدر زیرک هستند که این مساله را تشخیص دهند

• در چنین برنامه هایی چون متغیر COUNT با استفاده از عملگر DB در سگمنت کد تعریف شده است، به طور خودکار در هنگام استفاده تصحیح سگمنت انجام می شود.

• استفاده از پیشوند :CS در چنین برنامه هایی آسیبی به برنامه نمی رساند.

تغییر ثباتهای سگمنت پیش فرض

مثال‌های برنامه‌نویسی

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
	CS:	2E A1 00 10 2E 89 4E 00	MOV AX, CS:MEMWCS[1] MOV CS:[BP], CX	کد کد	AX←CS:[1001H:1000H] CS:[BP] ←CX	سگمنت پیش فرض برای مبدا یا مقصد حافظه لغو شده و یکی از سگمنت های کد، داده، پشته یا اضافی به جای آن قرار گرفته است.
	ES:	26 A1 00 10 26 89 4E 00	MOV AX, ES:MEMWES MOV ES:[BP], CX	اضافی اضافی	AX←ES:[1001H:1000H] ES:[BP] ←CX	
	DS:	3E 89 4E 00	MOV DS:[BP], CX	داده	DS:[BP] ←CX	
	SS:	36 A1 00 10 36 89 0F	MOV AX, SS:MEMWSS MOV SS:[BP], CX	پشته پشته	AX←SS:[1001H:1000H] SS:[BP] ←CX	

[1]اسمبلر به طور خودکار عملیات لغو کردن سگمنت پیش فرض را انجام می دهد اگر این کلمات حافظه قبلا در سگمنت مناسب تعریف شده باشند. در این جا فرض شده است هر کدام از کلمات در آدرس 1000H از سگمنت مربوطه تعریف شده اند.

تغییر ثباتهای سگمنت پیش فرض

مثال: عملیات اجرا شونده در برنامه ی زیر راتشریح کنید.

```
MOV CX, CS;
MOV DS, CX;
MOV AL, COUNT;
```

حل: دو دستور اول محتوای DS را با CS برابر می کنند.

آخرین دستور ثبات AL را با محتوای COUNT در سگمنت داده مقداردهی می کند.

از آنجا که DS = CS است، متغیر COUNT از آدرس فیزیکی مربوط به فضای برنامه خوانده می شود و نیازی به عملیات لغو سگمنت پیش فرض نیست.

دستورات انتقال داده

- اگر تمامی op-code ها را با مدهای مختلف آدرس دهی ترکیب کنیم، بیش از ۳۰۰۰ دستورالعمل متفاوت به وجود می آیند.

- دستورالعمل های گروه انتقال داده که در اینجا بررسی می شوند، برای CPU امکان برقراری ارتباط با دنیای خارج را فراهم می آورند.

- دیگر کاربرد این دستورات، انتقال داده بین ثباتهای CPU و حافظه ی سیستم است.

دستور MOV

- عملوندهای این دستور همواره در قالب "منبع، مقصد MOV" قرار دارد.
- داده انتقالی می تواند یک بایت یا یک کلمه باشد.
- اگر چه ثباتهای ایندکس، اشاره گر و سگمنت فقط به صورت کلمه ای قابل دستیابی هستند. در این دستور نمی‌توان منبع و مقصد را هر دو از مکان های حافظه انتخاب کرد.
- این گروه از دستورالعمل ها، پرچم های وضعیت را تغییر نمی دهند.
- برای دستور MOV ترکیب های بسیاری ممکن می باشد. مثلا:

```
MOV AX, [BX]
MOV AX,[BX+4]
MOV AX,[SI]
MOV AX,[SI-6]
MOV AX,[BP+SI]
MOV AX,[BP+SI+3EH]
```

این ها همگی دستورات انتقال از حافظه به ثبات هستند که در قالب های متنوعی از مدهای آدرس دهی غیر مستقیم ارائه شده اند.

عملیات سمبولیک	سگمنت مربوطه	عبارت یادآور	کد شیء	مد آدرس دهی
AX←BX	درون CPU	MOV AX, BX	8B C3	مبدا, مقصد MOV
AH ← BL	درون CPU	MOV AH, BL	8A E3	
AL ← [1000H] AH ← [1001H]	داده	MOV AX, MEMWDS	A1 00 10	
AL ← [1002H]	داده	MOV AL, MEMBDS	A0 02 10	
[1000H] ← BL [1001H] ← BH	داده	MOV MEMWDS, BX	89 1E 00 10	
[1002H] ← BL	داده	MOV MEMBDS, BL	88 1E 02 10	
[1000H] ← 34H [1001H] ← 12H	داده	MOV MEMWDS, 1234H	C7 06 00 10 34 12	
[1002H] ← 34H	داده	MOV MEMBDS, 34H	C6 06 02 10 34	
AL ← 10H	کد	MOV AL, 10H	B0 10	
AL ← 00H, AH ← 10H	کد	MOV AX, 1000H	B8 00 10	
DS ← AX	درون CPU	MOV DS, AX	8E D8	
DX ← ES	درون CPU	MOV DX, ES	8C C2	
ES ← [1001H:1000H]	داده	MOV ES, MEMWDS	8E 06 00 10	
[1001H:1000H] ← CS	داده	MOV MEMWDS, CS	8C 0E 00 10	

دستور MOV

•فرض بر آن است که MEMWDS به کلمه ای از حافظه در آدرس 1000H و MEMBDS به بایتی در آدرس 1002H از سگمنت داده اشاره می کنند.

مثال : فرض کنید مکانی از حافظه به نام COUNT به طول یک بایت، تعداد دفعاتی که عملیات خاصی انجام می شود را در خود نگه می دارد. برنامه ای بنویسید که CH را با مقدار 00H و CL را با محتوای COUNT پر کند.

حل:

مقدار 0 را به ثبات CH منتقل کن MOV CH, 00H

محتوای COUNT را به ثبات CL منتقل کن; MOV CL, COUNT

دستورات انتقال داده ی خاص

- دستورات انتقال داده ی دیگری نیز وجود دارند که از عبارت MOV استفاده نمی کنند.
- دستور XCHG محتوای دو ثبات یا یک ثبات و یک خانه ی حافظه را با هم عوض می کند و معادل عملکرد سه دستور MOV است.
- هشت بیت کم ارزش پرچم را می توان در ثبات AH ذخیره کرد یا اینکه از آن بارگذاری کرد:

LAHF
SAHF

دستورات انتقال داده ی خاص

- دستورات IN و OUT برای تبادل داده ها با دنیای خارج هستند.
- با این دو دستور یک بایت یا کلمه می تواند وارد یا خارج شود ولی باید حتما از آکومولاتور بگذرد.
- برای عملیات I/O هشت بیتی ثبات AL به کار می رود.
- آدرس وسایل I/O که **آدرس پورت** نامیده می شود، می تواند در محدوده ی 0000H تا FFFFH قرار گیرد و بنابراین تا ۶۵۵۳۶ پورت I/O متفاوت امکان پذیر است.
- آدرس پورت به دو روش قابل تعیین است:
  - در مد مستقیم آدرس در خود دستور قرار می گیرد که در محدوده ی 0 تا 255 است (یک بایت).
  - در مد غیر مستقیم ثبات DX آدرس ۱۶ بیتی پورت را نگه می دارد و امکان دسترسی به همه ی ۶۵۵۳۶ پورت وجود دارد.

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
XCHG	مبدا ،مقصد	93 86 C7 87 14	XCHG AX, BX XCHG AL, BH XCHG [SI], DX	درون CPU درون CPU داده		محتوای کلمه یا بایت مبدا را با مقصد جابجا می کند. بیت های پرچم تغییری نمی یابند.
LAHF		9F	LAHF	درون CPU	AH ← Flags1	بایت کم ارزش پرچم را در AH کپی می کند.
SAHF		9E	SAHF	درون CPU	Flags1 ← AH	AH را در بایت کم ارزش پرچم کپی می کند.
IN	پورت، آکومولاتور	E4 26 E5 26 EC ED	IN AL, 26H IN AX, 26H IN AL, DX IN AX, DX		AL ← port 26H AL ← port 26H; AH ← port 27H AL ← port DX AL ← port DX; AH ← port DX+1	یک بایت یا کلمه را از پورت های I/O مستقیم با آدرس 0 تا 255 وارد می کند. یک بایت یا کلمه را از پورت های I/O غیر مستقیم با آدرس 0 تا 65535 وارد می کند. آدرس پورت در DX قرار دارد. بیت های پرچم تغییری نمی یابند.
OUT	پورت، آکومولاتور	E6 26 E7 26 EE EF	OUT 26H, AL OUT 26H, AX OUT DX, AL OUT DX, AX		port 26H ← AL port 26H ← AL; port 27H ← AH port DX ← AL port DX ← AL; port DX+1 ← AH	یک بایت یا کلمه را در پورت های I/O مستقیم با آدرس 0 تا 255 می نویسد. یک بایت یا کلمه را در پورت های I/O غیر مستقیم با آدرس 0 تا 65535 می نویسد. آدرس پورت در DX قرار دارد. بیت های پرچم تغییری نمی یابند.
LEA	مبدا ،مقصد	8D 1E 00 10	LEA BX, MEMBDS	داده	BL ← 00; BH ← 10H	آدرس موثر عملوند مبدا، به عملوند مقصد منتقل می شود. بیت های پرچم تغییری نمی یابند.
LES	مبدا ،مقصد	C5 1C	LDS BX, DWORD PTR[SI]	داده	BL←[SI]; BH←[SI+1]; DS←[SI+3:SI+2]	چهار بایت متوالی از حافظه را که آدرس شروع آن در متغیر اشاره گر قرار دارد، به ثبات مقصد و یکی از ثبات های DS یا ES منتقل می کند.
	مبدا ،مقصد	C4 1C	LES BX, DWORD PTR[SI]	داده	BL←[SI]; BH←[SI+1]; ES←[SI+3:SI+2]	
XLAT		D7	XLAT	داده	AL←[BX+AL]	بایت AL را با یک بایت از جدول ۲۵۶ بایتی که آدرس شروع آن BX است و مقدار افست آن را AL تعیین می کند جایگزین می کند.

دستورات انتقال داده ی خاص

- دستورات IN و OUT هیچ کدام از ثباتهای سگمنت را به کار نمی گیرند.

- در مورد دستورات LDS و LES، مقصد باید یکی از ثبات های CPU باشد.

- **مثال:** برنامه ای برای ۸۸/۸۰۸۶ بنویسید تا کلمه ی موجود در ثبات BX را در پورت های 8004H و 8005H بنویسد.

حل:

```
MOV DX, 8004H;
MOV AX, BX;
OUT DX, AX;
```

دستورات انتقال داده خاص

- **تمامی مدهای آدرس دهی غیر مستقیم، نیاز به یک ثبات ایندکس یا اشاره گر برای نگه داری آدرس حافظه دارند.**

- این ثبات را می توان با مد آدرس دهی بلافصل مقدار دهی کرد. مثلا: MOV BX, 1000H.

- ولی این کار نیازمند آن است که برنامه نویس آدرس مکان حافظه هدف را بداند که این مساله با هدف به کارگیری زبان برنامه نویسی اسمبلی در تناقض است. تکنیک بهتری برای این کار دستور LEA است.

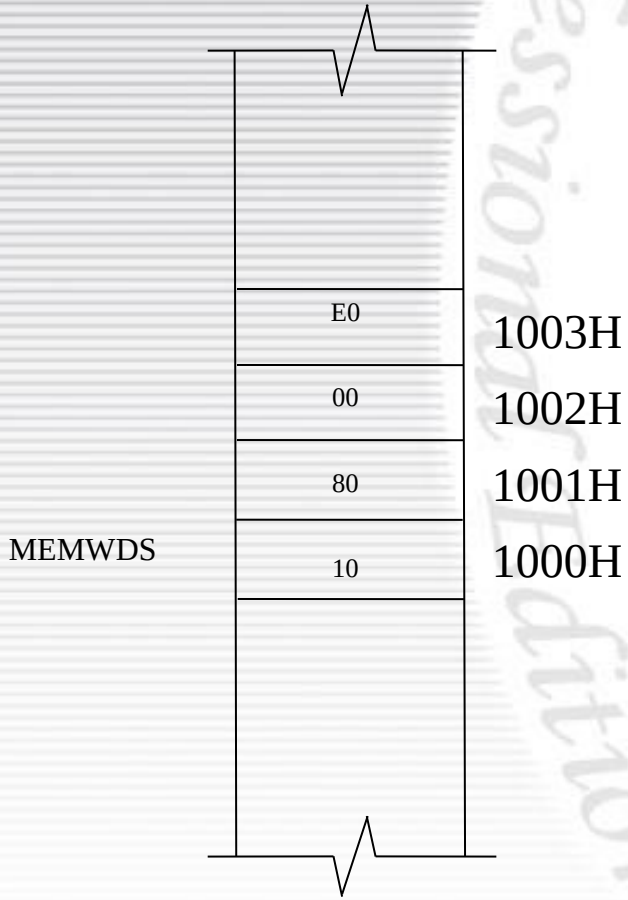
- مثلا فرض کنید به آدرس 1000H برچسب MEMBDS اختصاص داده شده است.

- دستورالعمل LEA BX, MEMBDS، ثبات BX را با آدرس موثر MEMBDS مقداردهی می کند. با این کار دیگر نیازی نیست که برنامه نویس آدرس را بداند، بلکه این اسمبلر است که آدرس موثر را پیدا می کند و در object code مربوط به LEA قرار می دهد.

- در بعضی از موارد، یک آدرس کاملا جدید (شامل ثبات سگمنت هم می شود) باید تعیین شود. دستورات LDS و LESبه این منظور به کار می روند. این دستورات ثبات ۱۶ بیتی مقصد و ثبات سگمنت DS یا ES را با محتوای دو کلمه ای عملوند حافظه مقداردهی می کنند.

مثال: فرض کنید MEMWDS یک کلمه ی دوتایی را در سگمنت داده تعریف می کند که آدرس شروع آن 1000H است . بعد از اجرای دنباله دستورات زیر، ثبات BX به چه آدرس فیزیکی اشاره می کند؟

```
LEA SI, MEMWDS
LDS BX, DWORD PTR[SI]
```



حل: اولین دستور ثبات SI را با آدرس MEMWDS که 1000H است، مقداردهی می کند.

دستور دوم ثبات های BX و DS با محتوای کلمه ی دوتایی که SI به آن اشاره می کند، مقداردهی می کند. در این مورد BX = 8010H و DS = E000H.

از آنجا که به طور پیش فرض ثبات BX به سگمنت داده اشاره می کند، آدرس فیزیکی آن E0000H+8010H = E8010H است.
به جای دو دستور بالا می توان نوشت: LDS BX, MEMWDS.

دستورات رشته ای

•قبلا گفته شد که ۸۸/۸۰۸۶ چندین دستورالعمل برای انتقال بلوک های بزرگ داده یا همان رشته ها دارد. جدول اسلاید دوصفحه بعد دستورات رشته ای، پنج دستورالعمل کار با رشته ها را معرفی می کند.

• برای همه ی این دستورالعمل ها حافظه ی منبع DS:SI و حافظه ی مقصد ES:SI است. لغو کردن سگمنت پیش فرض تنها قابل اعمال به آدرس منبع است و در مورد مقصد باید همان سگمنت اضافی را به کار برد.

• اشاره گرهای آفست حافظه در این دستورات که DI و SI هستند، با توجه به وضعیت پرچم DF، به طور خودکار یک واحد برای بایت و دو واحد برای کلمه، کاهش یا افزایش می یابند.

• از آنجا که این دستورات ثبات های تخصیص یافته برای تعیین کردن مبدا و مقصد داده به کار می گیرند، عملوند مقصد لازم نیست.

دستورات رشته ای

- دستورات STOS (store string byte or word) و LODS (load string byte or word) یک بایت یا کلمه را از **حافظه به اکومولاتور** یا بالعکس منتقل می کنند.

- دستور MOVS (move sring byte or word) این دو عملیات را با هم ترکیب می کند و یک بایت یا کلمه را از **حافظه مبدا به حافظه مقصد** منتقل می کند.

- دستورات SCAS (scan string byte or word) و CMPS (compare string byte or word) بایت یا کلمه مقصد را با **آکومولاتور**(SCAS) یا با **حافظه مبدا** (CMPS) **مقایسه** می کنند.

- بعد از اجرای دستور، **پرچم ها تغییر می کنند** و رابطه مبدا و مقصد را بیان می کنند.

- **دستور پرش شرطی** را می توان بعد از اجرای این دستورات به کار برد تا تصمیماتی نظیر **"پرش کن اگر ثبات AL از بایت حافظه بزرگتر است"** یا **"پرش کن اگر کلمه حافظه مقصد برابر است با کلمه حافظه ی مبدا"** گرفت.

مثال های برنامه نویسی						
توصیف	عملیات سمبولیک	سگمنت مربوطه	عبارت یادآور	کد شیء	عملوند	op-code
یک بایت یا کلمه را از رجیستر AL یا AX به عنصر رشته ای که آدرس آن در ثبات DI قرار دارد و متعلق به سگمنت اضافی است، منتقل می کند. اگر DF=0 باشد، DI یک واحد اضافه می شود و گرنه یکی کاسته می شود. بیت های پرچم تغییری نمی یابند.	ES:[DI] ←AL اگر DF=0 آنگاه: DI←DI+1 اگر DF=1 آنگاه DF←DF-1	اضافی	STOSB	AA	ندارد	STOSB
	ES:[DI] ← AL, ES:[DI+1] ←AH اگر DF=0 آنگاه DI←DI+2 اگر DF=1 آنگاه DI←DI-2	اضافی	STOSW	AB	ندارد	STOSW
	همانند STOSB است اگر DI = MEMBES. همانند STOSW است اگر DI=MEMWES.	اضافی اضافی	STOS MEMBES STOS MEMWES	AA AB	مقصد	STOS
یک بایت یا کلمه را از عنصر رشته ای که آدرس در ثبات SI قرار دارد و متعلق به سگمنت داده است به AL یا AX منتقل می کند. اگر DF=0 باشد، ثبات SI یک واحد و در غیر اینصورت کاهش می یابد. بیت های پرچم تغییری نمی یابند.	AL ← DS:[SI] اگر DF=0 آنگاه SI←SI+1. اگر DF=1 آنگاه SI←SI-1.	داده	LODSB	AC	-	LODSB
	AL←DS:[SI], AH←DS:[SI+1] اگر DF=0 آنگاه SI←SI+2. اگر DF=1 آنگاه SI←SI-2.	داده	LODSW	AD	-	LODSW
	همانند LODSB است اگر SI = MEMBDS. همانند LODSW است اگر SI = MEMWDS.	داده داده	LODS MEMBDS LODS MEMWDS	AC AD	مبدا	LODS
یک بایت یا کلمه را عنصر رشته ای که آدرس آن در SI قرار دارد و متعلق به سگمنت داده است، به عنصر رشته ای که آدرس آن در DI است و متعلق به سگمنت اضافی است، منتقل می کند. اگر DI=0 باشد، DI و SI یک واحد افزوده و گرنه کاسته می شوند. بیت های پرچم تغییری نمی یابند.	ES:[DI] ←DS:[SI] اگر DF=0 آنگاه DI←DI+1 , SI←SI+1. اگر DF=1 آنگاه DI←DI-1 , SI←SI-1.	اضافی، داده	MOVSB	A4	ندارد	MOVSB
	ES:[DI] ←DS:[SI] ; ES:[DI+1] ←DS:[DI+1] اگر DF=0 آنگاه DI←DI+2 , SI←SI+2. اگر DF=1 آنگاه DI←DI-2 , SI←SI-2	اضافی، داده	MOVSW	A5	ندارد	MOVSW
	همانند MOVSB است اگر SI=MEMBDS و DI = MEMBES باشد. همانند MOVSW است، اگر SI=MEMWDS و DI = MEMWES باشد.	داده، اضافی داده، اضافی	MOVS MEMBES, MEMBDS MOVS MEMWES, MEMWDS	A4 A5	مبدا، مقصد	MOVS
بایت یا کلمه ی موجود در عنصر رشته ای به آدرس DI متعلق به سگمنت اضافی، را از AL یا AX کم می کند. بیت های پرچم تغییر می یابند و ارتباط عملوند مبدا و مقصد را بیان می کنند. اگر DF=0 باشد، DI یک واحد افزوده و در غیر اینصورت کاسته می شود.	به روزرسانی بایت پرچم AL-ES:[DI]; اگر DF=0 آنگاه: DI←DI+1 اگر DF=1 آنگاه DF←DF-1	اضافی	SCASB	AE	-	SCASB

SCASW	–	AF	SCASW	اضافی	به روزرسانی بایت پرچم -AX ES:[DI+1:DI]; اگر DF=0 آنگاه: DI←DI+2 اگر DF=1 آنگاه DI←DI-2	
SCAS	مقصد	AE AF	SCAS MEMBES SCAS MEMWES	اضافی اضافی	همانند SCASB اگر DI = MEMBES باشد. همانند SCASW اگر DI = MEMWES باشد.	
CMPSB	–	A6	CMPSB	داده، اضافی	به روزرسانی بایت پرچم-DS:[SI] ES:[DI]; اگر DF=0 آنگاه DI←DI+1 , SI←SI+1. اگر DF=1 آنگاه DI←DI-1 , SI←SI-1	بایت یا کلمه رشته مقصد، به آدرس DI در سگمنت اضافی را از بایت یا کلمه رشته مبدا، به آدرس SI در سگمنت داده کم می کند. بیتهای پرچم تغییر می یابند و ارتباط عملوند مبدا و مقصد را بیان میکنند. افزایش یا کاهش DI و SI با توجه به مقدار DF انجام میشود.
CMPSW	–	A7	CMPSW	داده، اضافی	DS:[SI+1:SI]-ES:[DI+1:DI]; به روزرسانی بایت پرچم. اگر DF=0 آنگاه DI←DI+2 , SI←SI+2. اگر DF=1 آنگاه DI←DI-2 , SI←SI-2.	
CMPS	مبدا، مقصد	A6 A7	CMPS MEMBES, MEMBDS CMPS MEMWES, MEMWDS	داده، اضافی داده، اضافی	همانند CMPSB اگر SI=MEMBDS و DI=MEMBES. همانند CMPSW اگر SI=MEMWDS و DI=MEMWES.	

- در این جدول، و در مواردی که عملوند های مبدا یا مقصد وجود دارند، این عملوندها باید عنصر رشته را به صورت یک بایت یا کلمه تعیین کنند. اختصاص دادن چنین عملوند هایی فقط برنامه را واضح تر می کند ولی کد شیء نهایی ایجاد شونده تغییری نسبت به حالت بدون عملوند ندارد.

پیشوند تکرار

- ممکن است بپرسید که دستورات رشته ای معرفی شده چگونه می توانند بر بلوک های بزرگ داده اعمال شوند. تا کنون همه دستورات بر روی یک بایت یا کلمه عمل می کنند. پیشوند تکرار REP برای این منظور به کار گرفته می شود.

- استفاده از REP قبل از دستورات STOS یا MOVS باعث می شود که این دستورات به تعداد محتوای ثبات CX تکرار شوند.

- با مقداردهی ثبات CX به اندازه ی تعداد بایت ها یا کلماتی که باید منتقل شوند، یک دستور رشته ای به همراه پیشوند REP، می تواند حداکثر 65536 بایت را جابجا کند.

پیشوند تکرار

مثال: یک برنامه ی 88/8086 بنویسید که ۱۰۰۰ خانه ی حافظه در سگمنت اضافی را با داده ی 20H پر کند. آدرس شروع بلوک داده در BLOCK قرار دارد.

حل:

```
MOV AL, 20H  ;
LEA DI, BLOCK  ;
MOV CX, 03E7H ;
REP STOSB ;
```

آخرین دستور ۱۰۰۰ بار تکرار می شود. در این برنامه فرض DF=0 در نظر گرفته شده است.

پیشوند تکرار

قالب های دیگری از پیشوند REP برای به کارگیری با دستورات SCAS و CMPS در نظر گرفته شده اند. این قالب ها عبارتند از:

REPE/REPZ (repeat while equal or zero)
REPNE/REPNZ (repeat while not equal or not zero)

این دو قالب اجازه می دهند که دستورات رشته ای تا زمانی که شرط برابری یا نابرابری برقرار است تکرار شوند.

این قالب دستورات در هنگام کار با جداول مراجعه یا بررسی یکسان بودن دو رشته حائز اهمیت خواهند بود.

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
REP		F3 AA F3 AB F3 A4 F3 A5	REP STOSB REP STOSW REP MOVSB REP MOVSW	اضافی اضافی داده، اضافی داده، اضافی	تکرار تا CX=0 شود;CX←CX-1 STOSB; تکرار تا CX=0 شود;CX←CX-1 STOSW; تکرار تا CX=0 شود;CX←CX-1 MOVSB; تکرار تا CX=0 شود;CX←CX-1 MOVSW;	دستور رشته ای آورده شده بعد از REP آنقدر تکرار می شود که CX صفر شود.
REPE/REPZ		F3 AE F3 AF F3 A6 F3 A7	REPZ SCASB REPZ SCASW REPZ CMPSB REPZ CMPSW	اضافی اضافی داده، اضافی داده، اضافی	SCASB; CX←CX-1 تکرار اگر ZF=1 و CX≠0 مانند بالا برای SCASW مانند بالا برای CMPSB مانند بالا برای CMPSW	دستور رشته ای را تا زمانی که شروط ZF=1 و CX≠0 برقرار باشند، تکرار می کند. کاهش CX بایت پرچم را تغییر نمی دهد.
REPNE/REPNZ		F2 AE F2 AF F2 A6 F2 A7	REPNE SCASB REPNE SCASW REPNE CMPSB REPNE CMPSW	اضافی اضافی داده، اضافی داده، اضافی	SCASB; CX←CX-1 تکرار اگر ZF=0 و CX ≠0 مانند بالا برای SCASW مانند بالا برای CMPSB مانند بالا برای CMPSW	دستور رشته ای را تا زمانی که شروط ZF=0 و CX≠0 برقرار باشند، تکرار می کند. کاهش CX بایت پرچم را تغییر نمی دهد.

دستورات منطقی

- دستورات منطقی به گروهی از دستورات که **توابع منطقی بولی** را اجرا می کنند اطلاق می شود.

- این توابع شامل AND، OR، NOT، XOR و دستورات شیفت و چرخش می‌گردند.

- این دستورات همگی در واحد ALU انجام می شوند و معمولا بر روی همه ی پرچم ها تاثیر می گذارند.

مثال: محتوای ثبات AL و وضعیت پرچم ها را بعد از اجرای دستورات زیر تعیین کنید.

```
MOV AL, 6DH;
MOV BH, 40H;
AND AL, BH;
```

حل: محتوای دو ثبات به صورت باینری چنین خواهند بود:

01101101 (AL)
. 01000000 (BH)
01000000 = 40H (AL)

وضعیت بیت های پرچم بعد از اجرای دستورات چنین خواهد شد:

```
CF = 0      ; با اجرای دستور AND، بیت CF ریست می شود
PF = 0      ; 40H تعداد فردی 1 دارد
AF = X      ; AF تعریف نشده است
ZF = 0      ; نتیجه غیر صفر است
SF = 0      ; بیت هفتم در وضعیت ریست است
OF = 0      ; با اجرای دستور AND، بیت OF ریست می شود
```

توابع منطقی

- می توان با دستور AND بیت های خاصی از عملوند مقصد را مقداردهی 0 کرد.
- برای این کار کافیست بیت های مطلوب را در عملوند مبدا صفر قرار داد.
- دیگر کاربرد AND در نظر گرفتن عملوند مبدا به عنوان یک **ماسک** برای آزمودن بیت های خاصی از عملوند مقصد است.
- در مثال قبل همه بیت ها به جز بیت ششم صفر شده اند، لذا اگر این بیت 0 باشد, نتیجه نیز 0 و در غیر اینصورت نتیجه غیر صفر خواهد بود. بدین ترتیب می توان مقدار هر بیت را بررسی کرد.
- دنباله ی دستورات زیر کنترل برنامه را به شرط صفر بودن بیت ششم ثبات AL، به مکان START منتقل می کند (با فرض اینکه BH=40H).

```
AND AL, BH
JZ START
```

توابع منطقی

- استفاده از دستور AND موجب خراب شدن الگوی بیتی عملوند مقصد می شود چون محتوای عملوند مقصد تغییر می کند. دستور TEST همان عملکرد را دارد ولی عملوند های مبدا یا مقصد را تغییر نمی دهد.

```
TEST AL, 40H
JZ START
```

- دستور OR را می توان برای مقداردهی 1 به بیت های عملوند مقصد به کار گرفت.

- به عنوان مثال دستور OR AL, 80H بیت هفتم از ثبات AL را مقدار 1 می دهد، بدون اینکه دیگر بیت ها تغییری یابند.

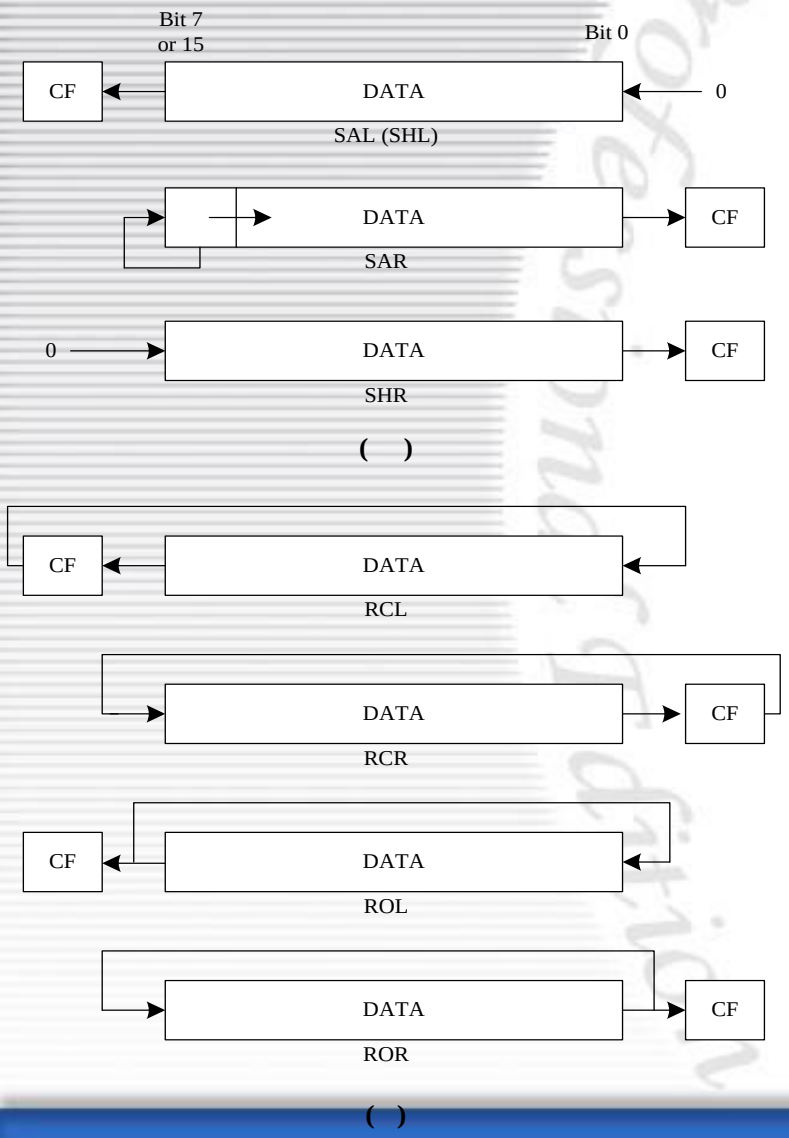
- به طور مشابه دستور XOR برای مکمل کردن بیت ها قابل استفاده است.

- مثلا دستور XOR AL, 80H بیت هفتم AL را مکمل می کند و باقی بیت ها بدون تغییر می مانند.

مثال های برنامه نویسی						
op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
NOT	مقصد	F7 D3 F6 14	NOT BX NOT BYTE PTR[SI]	درون CPU داده	BX← BX ! [SI]←[SI]!	همه بیت های کلمه یا بایت عملوند را مکمل می کند. بایت پرچم تغییر نمی کند.
AND	مبدا، مقصد	23 CA 22 3C 25 00 80	AND CX, DX AND BH, BYTE PTR[SI] AND AX, 8000H	درون CPU داده کد	CX←CX . DX BH←BH . [SI] AX←AX . 8000H	بیت به بیت عملوند مبدا را با مقصد AND می کند و در مقصد ذخیره می کند. بیت های پرچم تغییر می یابند (AF تعریف نشده است).
OR	مبدا، مقصد	0B CA 0A 3C 0D 00 80	OR CX, DX OR BH, BYTE PTR[SI] OR AX, 8000H	درون CPU داده کد	CX←CX+DX BH←BH+[SI] AX←AX+8000H	همانند قبلی است، با این تفاوت که عملیات OR را انجام می دهد.
XOR	مبدا، مقصد	33 CA 32 3C 35 00 80	XOR CX, DX XOR BH, BYTE PTR[SI] XOR AX, 8000H	درون CPU داده کد	8000H	همانند قبلی است، با این تفاوت که عملیات XOR را انجام می دهد.
TEST	مبدا، مقصد	85 D1 84 3C A9 00 80	TEST CX, DX TEST BH, BYTE PTR[SI] TEST AX, 8000H	درون CPU داده کد	تغییر پرچم;CX . DX تغییر پرچم;BH . [SI] تغییر پرچم;AX . 8000H	همانند دستور AND است ولی عملوند ها بدون تغییر می یابند (نتیجه ذخیره نمی شود).

- لازم به ذکر است تمامی مدهای آدرس دهی قابل استفاده هستند. با اجرای هر یک از دستورات AND، OR، XOR یا TEST، پرچم های CF و OF ریست می شوند.

دستورات شیفت و چرخش



- چندین دستور شیفت و چرخش وجود دارند.

- داده ای که چرخش می یابد، می تواند یک ثبات ۸ یا ۱۶ بیتی CPU یا یکی از مکان های حافظه باشد.

دستورات شیفت و چرخش

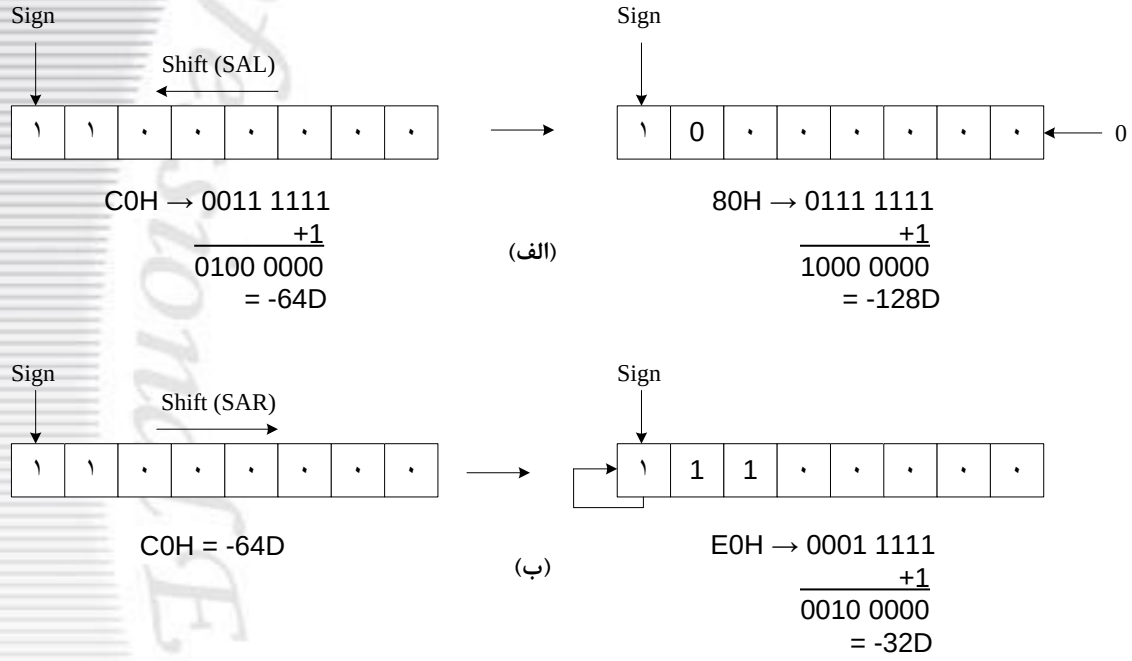
- تفاوت اصلی دستور شیفت و چرخش در این است که بیت های شیفت داده شده از انتهای ثبات بیرون می افتند در حالیکه بیت های چرخش یافته از دیگر انتهای ثبات مجددا به ثبات وارد می شوند.

- دستورات شیفت در دو گروه "ریاضی" و "منطقی" قرار می گیرند.

- شیفت های ریاضی (SAL, SAR) چنان عمل می کنند که بیت علامت عدد (بیت ۷ یا ۱۵) در حین شیفت تغییر نکند.

•شکل تاثیر دستورات SAL و SAR را بر بایت داده C0H نشان می دهد.

• از این شکل چنین برمی آید که دستور SAL (با حفظ علامت) داده را در ۲ ضرب و SAR بر ۲ تقسیم می کند. در صورتیکه عدد شیفت داده شده قابل نمایش در ۸ بیت یا ۱۶ بیت نباشد, پرچم OF مقدار 1 خواهد گرفت.



• دستورات شیفت و چرخش را می توان تا ۲۵۵ بار تکرار کرد. این کار با مقداردهی ثبات CL به اندازه ی تعداد دفعات مطلوب انجام می شود. مثلا برای انجام پنج بار عملیات شیفت چپ بر ثبات DX از میان Carry، دستورات زیر به کار می رود:

```
MOV CL, 5
RCL DX, CL
```

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
مثال های برنامه نویسی						
SAL/SHL	تعداد، مقصد	D1 E0 D3 E0	SAL AX, 1 SAL AX, CL	CPU درون CPU درون		کلمه یا بایت عملوند را یک بار یا به تعداد CL بار به چپ یا راست شیفت می دهد. بیت های پرچم تغییر می یابند (AF تعریف نشده است). برای شیفت های تک بیتی در صورتی که علامت عدد تغییر یابد، پرچم OF مقدار 1 می یابد.
SAR SHR	تعداد، مقصد تعداد، مقصد	D0 F8 D2 F8 D1 2C D2 2C	SAR AL, 1 SAR AL, CL SHR WORD PTR[SI], 1 SHR BYTE PTR[SI], 1	CPU درون CPU درون داده داده		
RCL	تعداد، مقصد	D1 D3 D3 D3	RCL BX, 1 RCL BX, CL	CPU درون CPU درون		کلمه یا بایت عملوند را یک بار یا به تعداد CL بار به چپ یا راست می چرخاند. فقط پرچم های CF و OF تاثیر می یابند. در مورد چرخش های تک بیتی در صورتی که علامت عدد عوض شود، پرچم OF مقدار 1 می یابد.
RCR	تعداد، مقصد	D0 DB D2 DB	RCR BL, 1 RCR BL, CL	CPU درون CPU درون		
ROL	تعداد، مقصد	D1 04 D2 04	ROL WORD PTR[SI], 1 ROL BYTE PTR[SI], 1	داده داده		
ROR	تعداد، مقصد	D1 0E 00 10 D2 0E 04 10	ROR MEMWDS, 1 ROR MEMBDS, CL	داده داده		

مثال: محتوای ثبات های AX، BX و CX را بعد از اجرای دستورات زیر تعیین کنید.

```
MOV CL, 3
MOV AX, 7FH
MOV BX, 0505H
ROL AX, CL
AND AH, BH
OR BL, AL
```

حل: نتیجه اجرای دستورات مرحله به مرحله در جدول زیر آورده شده است.

CX	BX	AX		
??03	????	????	CL,3	MOV
??03	????	007F	AX, 7FH	MOV
??03	0505	007F	BX, 0505H	MOV
??03	0505	03F8	AX, CL	ROL
??03	0505	01F8	AH, BH	AND
??03	05FD	01F8	BL, AL	OR

دستورات ریاضی

دستورات ریاضی توانایی های محاسباتی ریزپردازنده را ارائه می کنند.

بر خلاف پردازنده های ۸ بیتی اولیه، دستورات ریاضی ۸۸/۸۰۸۶ محدود به جمع و تفریق اعداد ۸ بیتی درون آکومولاتور نیستند.

۸۸/۸۰۸۶ می تواند اعداد ۸ یا ۱۶ بیتی را در هر کدام از ثباتهای همه منظوره ی CPU جمع یا تفریق کند.

۸۸/۸۰۸۶ ثباتهای تخصیص یافته ای برای عملیات ضرب و تقسیم با علامت یا بدون علامت دارد.

دستورات جمع و تفریق

عملوند های مقصد و مبدا می توانند ثبات و ثبات، ثبات و حافظه، حافظه و ثبات، داده ی بلافصل و ثبات، یا داده بلافصل و حافظه باشند.

دو قالب از دستورات جمع و تفریق وجود دارند. یکی قالب carry را شامل می شود و دیگری نمی شود.

توانایی شامل کردن carry اجازه ی کنترل کردن اعداد با دقت مضاعف را ممکن می سازد.

به عنوان مثال فرض کنید می خواهیم یک عدد ۳۲ بیتی در ثباتهای BX:AX را به ثباتهای DX:CX اضافه کنیم و نتیجه را در DX:CX ذخیره کنیم. عملیات جمع به صورت زیر خواهد بود:

```
  BX   AX
+ DX   CX
  DX   CX
```

دستورات جمع و تفریق

- اگرچه دستور جمع ۳۲ بیتی وجود ندارد، این مساله با به کارگیری دستور ADC (جمع با carry) به سادگی حل می شود:

```
ADD CX, AX          ;CX ← CX + AX
ADC DX, BX          ;DX ← DX + BX + CF
```

- اولین دستور، جمع معمولی است که carry را شامل نمی شود. اگر این عملیات جمع پرچم CF را مقدار 1 دهد، دستور دوم آن را به حاصل جمع DX و BX می افزاید.

- دستورات SUB (subtract) و SBB (subtract with borrow) نیز به طریق مشابه عمل می کنند. در این حالت CF بیانگر وضعیت borrow در عملیات تفریق اول است.

- برای افزودن یا کاستن یک واحد از اشاره گرهای حافظه یا متغیرهای شمارنده، دستورات INC (increment) و DEC (decrement) را باید به کار برد.

- برای این دستورات می توان محتوای یک مکان حافظه را به عنوان عملوند مقصد قرار داد.

- دستور NEG (negate) مکمل ۲ عملوند مقصد را تشکیل می دهد و در نتیجه این کار بیت علامت عدد تغییر می یابد. این کار با کم کردن عملوند از 0 انجام می شود.

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف

ADD	مبدا، مقصد	03 F2
		00 2F
		81 C7 00 80
		81 06 00 10 00 80
			ADD SI, DX
			ADD BYTE PTR[BX], CH
			ADD DI, 8000H
			ADD MEMWDS, 8000H
				درون CPU داده
				درون CPU داده
					SI← SI+DX
					[BX] ← [BX] + CH
					DI←DI+8000H
					[1001H:1000H]←[1001H:1000H]+8000H
						بایت یا کلمه ی مقصد را با مجموع عملوندهای مبدا و مقصد جایگزین می کند. همه ی پرچم ها به هنگام می شوند.

ADC	مبدا، مقصد	13 F2 10 2F 81 D7 00 80 81 16 00 10 00 80	ADC SI, DX ADC BYTE PTR[BX], CH ADC DI, 8000H ADC MEMWDS, 8000H	درون CPU داده درون CPU داده	SI←SI+DX+CF [BX] ←[BX]+CH+CF DI←DI+8000H+CF [1001H:1000H]←[1001H:1000H]+8000H+CF	بایت یا کلمه ی مقصد را با مجموع عملوندهای مبدا و مقصد و Carry جایگزین می کند. همه ی پرچم ها به هنگام می شوند.

SUB	مبدا، مقصد	2B F2
		28 2F
		81 EF 00 80
		81 2E 00 10 00 80
			SUB SI, DX
			SUB BYTE PTR[BX], CH
			SUB DI, 8000H
			SUB MEMWDS, 8000H
				درون CPU داده
				درون CPU داده
					SI←SI-DX
					[BX] ←[BX]-CH
					DI←DI-8000H
					[1001H:1000H]←[1001H:1000H]-8000H
						بایت یا کلمه ی مقصد را با تفاضل عملوندهای مبدا و مقصد جایگزین می کند. همه ی پرچم ها به هنگام می شوند.

SBB	مبدا، مقصد	1B F2 18 2F 81 DF 00 80 81 1E 00 10 00 80	SBB SI, DX SBB BYTE PTR[BX], CH SBB DI, 8000H SBB MEMEWDS, 8000H	درون CPU داده درون CPU داده	SI←SI-DX-CF [BX] ←[BX]-CH-CF DI←DI-8000H-CF [1001H:1000H]←[1001H:1000H]-8000H-CF	بایت یا کلمه ی مقصد را با تفاضل عملوندهای مبدا و مقصد و Carry جایگزین می کند. همه ی پرچم ها به هنگام می شوند.
INC	مقصد	FE C3 FF 05 FE 06 04 10	INC BL INC WORD PTR[DI] INC MEMBDS	درون CPU داده داده	BL←BL+1 [DI+1:DI]←[DI+1:DI]+1 [1004H] ←[1004H]+1	یک واحد به بایت یا کلمه ی مقصد اضافه می کند. همه ی پرچم ها به جز CF به هنگام می شوند.
DEC	مقصد	FE CB FF 0D FE 0E 04 10	DEC BL DEC WORD PTR[DI] DEC MEMBDS	درون CPU داده داده	BL←BL-1 [DI+1:DI]←[DI+1:DI]-1 [1004H] ←[1004H]-1	یک واحد از بایت یا کلمه ی مقصد می کاهد. همه ی پرچم ها به جز CF به هنگام می شوند.
NEG	مقصد	F6 DB F7 1D F6 1E 04 10	NEG BL NEG WORD PTR[DI] NEG MEMBDS	درون CPU داده داده	BL←0-BL [DI+1:DI]←0-[DI+1:DI] [1004H] ←0-[1004H]	مکمل ۲ بایت یا کلمه ی مقصد را محاسبه می کند. همه ی پرچم ها به هنگام می شوند. (CF=1 به جز وقتی که عملوند صفر است)

CMP	مبدا، مقصد	3A C4
		39 0D
		81 3E 00 10 00 80
		81 FF 00 80
			CMP AL, AH
			CMP [DI], CX
			CMP MEMWDS, 8000H
			CMP DI, 8000H
				درون CPU داده
				داده
				درون CPU
					تغییر پرچم ;AL-AH
					تغییر پرچم;[DI+1:DI]-CX
					تغییر پرچم-[1001H:1000H]8000H
					تغییر پرچم ;DI-8000H
						عملوند مبدا را از مقصد کم می کند و پرچم ها به هنگام می شوند تا رابطه ی این دو را نشان دهند. عملوند ها بدون تغییر باقی می مانند.

• دستورات جمع و تفریق

• در جدول اسلاید قبلی فرض شده MEMWDS به یک کلمه از حافظه، با آدرس شروع 1000H درون سگمنت داده و MEMBDS به یک بایت از سگمنت داده در آدرس 1004H اشاره می کند.

•برای دستورات INC، DEC و NEG نمی توان از مد آدرس دهی بلافصل استفاده کرد.

•- دستور CMP (compare) برای تعیین کردن اندازه ی نسبی دو عملوند مناسب است. به طور معمول با یک دستور پرش شرطی مثل "پرش کن اگر برابر باشند" دنبال می شود.

دستورات جمع و تفریق

مثال: مقدار ثبات AL و نیز وضعیت پرچم ها را بعد از اجرای دستورات زیر تعیین کنید.

```
MOV   AL, 5
NEG   AL
```

حل: AL←00H-05H = FBH = -5 D وضعیت پرچم ها چنین خواهد بود:

CF = 1 ; NEG پرچم CF را ست می کند به جز وقتی عملوند 0 است

PF = 0 ; FBH تعداد فردی 1 دارد

AF = 1 ; یک borrow از بیت ۴ اتفاق افتاده است

ZF = 0 ; نتیجه صفر نیست

SF = 1 ; مقدار بیت هفتم 1 است

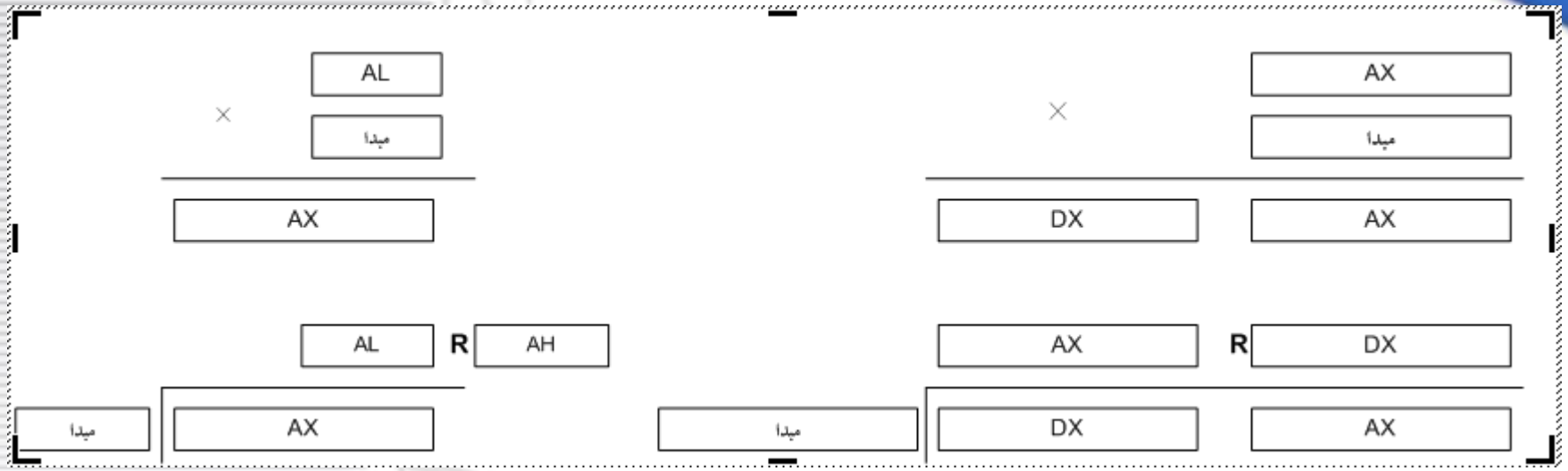
OF = 0 ; سرریز رخ نداده است

دستورات ضرب و تقسیم

آنچنان که در جدول دیده می شود، عملیات ضرب و تقسیم را می توان بر اعداد با علامت یا بدون علامت انجام داد.

عملوند مبدا می تواند یکی از ثباتهای CPU یا یک مکان حافظه باشد ولی عملوند مقصد باید AX (و DX برای نتایج ۳۲ بیتی) باشد.

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
MUL	مبدا	F6 E3 F7 E1 F6 27 F7 26 00 10	MUL BL MUL CX MUL BYTE PTR[BX] MUL MEMWDS	درون CPU درون CPU داده داده	AX← AL * BL DX:AX←AX*CX AX←AL*[BX] DX:AX←AX*[1001H:1000H]	حاصل ضرب بدون علامت عملوند مبدا در آکومولاتور را در AX ذخیره می کند. اگر نتیجه ضرب دو کلمه ای باشد، در DX:AX ذخیره می شود. اگر در ضرب بایتی، نتیجه دو بایتی یا در ضرب کلمه ای نتیجه دو کلمه ای شود، CF و OF مقدار 1 و در غیر اینصورت 0 می گیرند. باقی بیت های پرچم تعریف نشده اند.
IMUL	مبدا	F6 EB F7 E9 F6 2F F7 2E 00 10	IMUL BL IMUL CX IMUL BYTE PTR[BX] IMUL MEMWDS	درون CPU درون CPU داده داده	AX← AL * BL DX:AX←AX*CX AX←AL*[BX] DX:AX←AX*[1001H:1000H]	همانند دستور MUL است، با این تفاوت که اعداد با علامت را می توان به کار گرفت. اگر حاصل ضرب را نتوان در ثبات مرتبه ی پایین نتیجه نمایش داد، CF و OF مقدار 1 وگرنه مقدار 0 خواهند گرفت. علامت عدد به ثبات مرتبه ی بالا گسترده می شود. دیگر پرچم ها تاثیری نمی یابند.
DIV	مبدا	F6 F3 F7 F1 F6 37 F7 36 00 10	DIV BL DIV CX DIV BYTE PTR[BX] DIV MEMWDS	درون CPU درون CPU داده داده	AL← AX / BL AX← DX:AX /CX AL←AX/[BX] AX←DX:AX /[1001H:1000H]	عملیات تقسیم بدون علامت AX (برای مقسوم علیه تک بایتی) و DX:AX (برای مقسوم علیه یک کلمه ای) بر مقسوم علیه را انجام می دهد و در مورد تک بایتی نتیجه را در AL و باقیمانده را در AH ذخیره می کند، برای یک کلمه ای، نتیجه در AX و باقیمانده را در DX ذخیره می کند. اگر خارج قسمت بیشتر از ظرفیت ثبات مقصد شود، یک وقفه ی نوع 0 تولید می شود. همه ی پرچم ها تعریف نشده اند.
IDIV	مبدا	F6 FB F7 F9 F6 3F F7 3E 00 10	IDIV BL IDIV CX IDIV BYTE PTR[BX] IDIV MEMWDS	درون CPU درون CPU داده داده	AL← AX / BL AX← DX:AX /CX AL←AX/[BX] AX←DX:AX /[1001H:1000H]	همانند DIV است با این تفاوت که تقسیم با علامت هم می توان انجام داد.


AL
×
مبدا
AX
AX
DX
×
AX
مبدا
AL
R
AH
AX
DX
R
DX
AX
مبدا
AX
DX
مبدا
AX

مثال : برنامه ای بنویسید که دو عدد ۸ بیتی را از پورت های A0H و B0H وارد کند و نتیجه حاصل ضرب آن دو را در پورت ۱۶ بیتی 7080H قرار دهد.

حل:

```
IN      AL, 0A0H;
MOV   BL, AL;
IN      AL, 0B0H;
MUL    BL;
MOV   DX, 7080H;
OUT    DX, AX;
```

عملیات تقسیم را می توان بر روی یک کلمه در AX یا یک کلمه ی دوتایی در DX:AX انجام داد. مقسوم علیه می تواند یکی از مکان های حافظه یا ثباتهای CPU، ۸یا ۱۶ بیتی باشد. باقیمانده ی تقسیم در ثبات AH یا DX قرار می گیرد.

مثال: برنامه ای بنویسید که کلمه ای را از پورت I/O غیر مستقیم با آدرس 8000H وارد کند و بر 500 دسیمال تقسیم کند. نتیجه را وقتی داده ی ورودی 56723 است تعیین کنید.

حل: برنامه ی مطلوب چنین است:

```
MOV   DX, 8000H        ;ثبات DX آدرس پورت را نگه  می دارد
MOV   BX, 01F4H        ;ثبات BX مقسوم علیه را نگه  می دارد (500D)
IN      AX, DX         ;وارد کردن کلمه ی ورودی
DIV     BX             ;خارج قسمت در AX و باقیمانده در DX قرار می گیرد
```

- برای ورودی 56723 نتایج عبارتند از:

AX = INT (56723/500) = 113 = 71H
DX = MOD (56723/500) = 223 = 00DFH.

- مشکل خاصی در هنگام انجام تقسیم بر مقسوم علیه های کوچک ممکن است اتفاق افتد. در این موارد ممکن است خارج قسمت از محدوده ی قابل نمایش ثبات تخصیص یافته اش فراتر رود. به عنوان مثال نتیجه ی تقسیم 65000 بر 2 قابل ذخیره شدن در AL نیست. در این موارد به طور خودکار یک وقفه ی نرم افزاری "تقسیم بر صفر" از طرف ۸۸/۸۰۸۶ اتفاق می افتد. کنترل برنامه به آدرسی از حافظه که در مکان های 00000H- 00003H ذخیره شده است، منتقل می شود. وقفه های نرم افزاری در ادامه بررسی می شوند.

- دستورات ضرب و تقسیم صحیح مشابه حالت بدون علامت هستند با این تفاوت که پرارزش ترین بیت بیانگر علامت عدد است. برای تقسیم صحیح بایتی، خارج قسمت محدود به 128- تا 127 و تقسیم صحیح کلمه ای محدود به 32768- تا 32767 است.

- دستورات IDIV و IMUL اغلب با دستورات:

CBW : convert byte to word
CWD : convert word to double word

- بکار می روند.

دستورات تنظیمات حسابی

- گاهی داده ای که با آن سر و کار داریم در قالب ASCII یا BCD کد شده است.

•قالب BCD از چهار بیت برای نمایش ارقام دهدهی 9-0 استفاده می کند و لذا اعداد A تا F نادرست محسوب می شوند.

•بدین ترتیب یک بایت می تواند دو رقم دهدهی را در خود نگه دارد که آن را دسیمال بسته بندی شده گویند.

•به عنوان مثال عدد 1001 1001 (99H) بزرگترین عدد تک بایتی است که در قالب دسیمال بسته بندی شده قابل نمایش است.

- یک عدد دهدهی را نیز می توان به فرم یک رقم به ازای هر بایت، گشود. مثلا 00001001 (09H) بیانگر عدد دسیمال 9 است که از فرم بسته بندی خارج شده است.

- یک عدد ASCII مشابه یک عدد دسیمال باز شده است از این نظر که هر بایت می تواند تنها یک رقم را در خود نگه دارد.

•برای تبدیل یک رقم ASCII به دسیمال کافیست عدد 30H را از آن کم کنیم.

•مثلا عدد 9 در قالب ASCII برابر است با 39H. کم کردن 30H از 39H به 09H منتج خواهد شد که همان قالب باز شده ی عدد دهدهی 9 است.

- نتیجه ی عملیات جمع AL = 7 + 5 = 0CH است که از نظر باینری صحیح و به لحاظ BCD نامعتبر می باشد. در قالب BCD نتیجه ی معتبر 12 است.

- خوشبختانه چندین دستور تنظیم حسابی وجود دارند که می توان آنها را بلافاصله قبل یا بعد از دستور حسابی به کار برد و به نتایج معتبری در قالب BCD دست یافت.

- در مثال قبلی اگر بعد از دستور ADD AL, BL، دستور DAA (decimal adjust for addition) اجرا شود، نتیجه به گونه ای تصحیح می شود که AL = 12H شود.

- دستور DAS (decimal adjust for subtraction) عملیات مشابهی را بعد از تفریق دو عدد دهدهی انجام می دهد.

- برای تبدیل نتایج جمع یا تفریق دهدهی به قالب ASCII از دستورات زیر استفاده می کنیم. این دستورات را فقط باید با اعداد دهدهی باز شده به کار گرفت:

• AAA (ASCII adjust for addition)
• AAS (ASCII adjust for subtraction)

- دستورات AAA و AAS نتایج را واقعا به قالب ASCII تبدیل نمی کنند. ولی به هرحال نتیجه ی موجود در AL (یا AH اگر carry یا borrow اتفاق افتد) یک عدد دهدهی بازشده معتبر خواهد بود. با افزودن عدد 30H به آن، به قالب ASCII تبدیل می شود. به عنوان مثال به برنامه ی جمع دهدهی زیر توجه کنید:

```
MOV    AH, 0          ;  AH = 0 مطمئن شویم
MOV    AL, 7           اولین عدد;
MOV    BL, 5           دومین عدد;
ADD    AL, BL        ;AX = 000CH
AAA                  ;AX = 0102
ADD    AX, 3030H     ;AX = 3132H
```

اکنون نتیجه ی موجود در AX در قالب ASCII کد شده است و بیانگر نتیجه ی دهدهی معتبری برابر با 12 است.

مثال های برنامه نویسی						
توصیف	عملیات سمبولیک	سگمنت مربوطه	عبارت یادآور	کد شیء	عملوند	op-code
AL را به یک جفت رقم دهدهی بسته بندی شده ی معتبر تنظیم می کند و به دنبال عملیات جمع دو عدد دسیمال بسته بندی شده یا باز شده معتبر می آید. همه‌ی پرچم ها جز OF (تعریف نشده) تغییر می یابند.	اگر AF=1یا AL.0F>9 آنگاه AF←1; AL←AL+6 اگر CF=1 یا AL>9F آنگاه CF←1; AL←AL+60H	درون CPU	DAA	27	-	DAA
AL را به یک جفت رقم دهدهی بسته بندی شده ی معتبر تنظیم می کند و به دنبال عملیات تفریق دو عدد دسیمال بسته بندی شده یا باز شده معتبر می آید. همه ی پرچم ها جز OF (تعریف نشده) تغییر می یابند.	اگر AF=1یا AL.0F>9 آنگاه AF←1; AL←AL-6 اگر CF=1 یا AL>9F آنگاه CF←1; AL←AL-60H	درون CPU	DAS	2F	-	DAS
AL را به یک عدد دهدهی بازشده تنظیم می کند و به دنبال جمع دو عدد دهدهی بازشده ی معتبر می آید. نصف بیت مرتبه ی بالای AL صفر و یک واحد به AH می افزاید. همه ی پرچم ها جز AF و CF تعریف نشده اند.	اگر AF=1یا AL.0F>9 آنگاه AL←AL+6; AH←AH+1; AF←1; CF←AF; AL←AL.0F	درون CPU	AAA	37	-	AAA
AL را به یک عدد دهدهی بازشده تنظیم می کند و به دنبال تفریق دو عدد دهدهی بازشده ی معتبر می آید. نصف بیت مرتبه ی بالای AL صفر و یک واحد از AH می کاهد. همه ی پرچم ها جز AF و CF تعریف نشده اند.	اگر AF=1یا AL.0F>9 آنگاه AL←AL-6; AH←AH-1; AF←1; CF←AF; AL←AL.0F	درون CPU	AAS	3F	-	AAS
به دنبال ضرب دو عدد دهدهی بازشده ی معتبر می آید. نتیجه ی موجود در AL را به دو رقم باینری بازشده ی معتبر در AL و AH تبدیل می کند. همه ی پرچم ها جز PF، SF و ZF تعریف نشده اند.	AH←AL/0AH AL←remainder	درون CPU	AAM	D4 0A	-	AAM
قبل از تقسیم AX بر عملوند دهدهی بازشده ی تک رقمی، عدد دورقمی دهدهی بازشده ی موجود در AX را به یک عدد باینری در AL و 0 در AH تبدیل می کند. خارج قسمت تولید شده بعد از عملیات تقسیم یک عدد دهدهی بازشده ی معتبر در AL و باقیمانده در AH است. همه‌ی پرچم ها جز PF، SF و ZF تعریف نشده اند.	AL←(AH*0AH) + AL AH←0	درون CPU	AAD	D5 0A	-	AAD
قبل از تقسیم AX بر عملوند بایتی، بیت علامت مقسوم در AL را به AH گسترش می دهد و لذا AL به یک کلمه ی علامت دار معتبر در AX تبدیل می شود. هیچ کدام از پرچم ها تغییری نمی یابند.	اگر AL<80H آنگاه AH←0 اگر AL>7F آنگاه AH←FFH	درون CPU	CBW	98	-	CBW

دستورات تنظیمات حسابی

- در هنگام ضرب دو عدد دسیمال بازشده، تقریبا همیشه نتیجه ی بدست آمده به لحاظ BCD نامعتبر هستند. مثلا اگر AL = 06H و CL = 09H باشد، دستور MUL CL نتیجه ی 36H را در AL باقی گذارد که از نظر باینری معتبر و به لحاظ BCD نامعتبر است.

- دستور AAM (ASCII adjust for multiplication) نتیجه بدست آمده را به دو عدد دهدهی بازشده معتبر در AL و AH تبدیل می کند. این ارقام را می توان به راحتی به کد ASCII معادل تبدیل کرد.

دستورات تنظیمات حسابی

مثال: برنامه ای بنویسید که دو رقم ASCII موجود در CH و CL را در هم ضرب کند و نتیجه را به ASCII در AH و AL قرار دهد. محتوای AX را بیابید اگر CX = 3639H.

حل: برنامه ی مربوطه چنین است:

```
AND     CX, 0F0FH;      CX=0609H
MOV     AL, CH
AND     CX, 0F0FH;      CX = 0609H
MUL     CL;
AAM
ADD     AX, 3030H;
```

نتیجه را به دو رقم دهدهی باز شده تبدیل می کند ;AX = 0504H
به کد ASCII معادل تبدیل می کند AX = 3534H

دستورات تنظیمات حسابی

- دستور AAM را می توان به عنوان حالت خاصی از دستور DIV در نظر گرفت، چون AL را بر 10 تقسیم می کند، خارج قسمت را در AH و باقیمانده را در AL قرار می دهد.

- هنگام تقسیم یک عدد دهدهی باز شده دو رقمی در AX، بر یک عدد دهدهی باز شده، دستور AAD (ASCII adjust for division) باید قبل از تقسیم اجرا شود. این عملیات AX را به یک عدد باینری در AL و 0 در AH تبدیل می کند (بزرگترین عدد دسیمال دو رقمی 99H است که قابل نمایش به صورت تک بایتی 63H است و لذا AH صفر خواهد شد). سپس نتیجه ی تقسیم دو عدد دهدهی باز شده است که خارج قسمت در AL و باقیمانده در AH قرار دارد.

به عنوان مثال اگر AX = 0607H و CL = 09H باشد، AX را می توان بر CL تقسیم کرد با دستورات زیر:

```
AAD          ;AX به 0043H معادل 67D تبدیل می شود
DIV CL       ; AL = 07 و AH = 04
```

دستورات تنظیمات حسابی

- در هنگام اجرای تقسیم با علامت، ممکن است دستورات CBW و CWD مورد نیاز باشند.

- مثلا اگر بخواهیم دو بایت با علامت را بر هم تقسیم کنیم، مشکلی رخ می دهد. IDIV نیاز دارد که یکی از ارقام در AX باشد (یک ثبات ۱۶ بیتی).

- دستورالعمل CBW این مشکل را با تبدیل کردن بایت موجود در AL به یک کلمه در AX حل می کند.

- این کار با گسترش بیت علامت از AL به AH انجام می شود. بنابراین بعد از دستور CBW عدد FBH(-5) به FFFBH در AX تبدیل می شود.

- CWD عملیات مشابهی را انجام می دهد، وقتی که می خواهیم دو کلمه ی ۱۶ بیتی را تقسیم کنیم. بعد از اجرای CWD، کلمه ی علامت دار موجود در AX، به یک دو کلمه ای علامت دار در DX:AX تبدیل می شود.

دستورات انتقال کنترل

- کامپیوتر با برنامه ی ذخیره شده در حافظه، مرتبا دنباله "واکشی دستورالعملی که آدرس آن را IP تعیین می کند، افزایش IP و اجرای دستورالعمل" را تکرار می کند.

- در واقع برنامه به گونه ای ترتیبی اجرا می شود. گاهی لازم است کنترل برنامه به مکانی منتقل شود که دستورالعمل بعدی نیست.

- مجموعه دستوراتی که لازم است چندین بار اجرا شوند، گروهی از دستورات که در برنامه به اشتراک گذاشته شده اند (زیر برنامه ها)، انتقال های شرطی بر اساس وضعیت پرچم ها و وقفه های نرم افزاری مثال هایی از چنین برنامه هایی هستند.

- در ادامه دستوراتی را که کنترل برنامه را از حالت ترتیبی خارج می کنند بررسی می کنیم.

دستورات پرش غیر مشروط

- سه دستور اول غیر مشروط هستند بدین معنی که عملیات پرش بدون توجه به وضعیت پرچم های پردازنده انجام می شود. این قالب پرش برای مواردی که لازم است گروهی از دستورات تکرار شوند به کار می رود.
- به عنوان مثال حلقه برنامه زیر را در نظر بگیرید.

آدرس	کد تولید شده	برچسب	op-code	عملوند	توضیحات
0000	B3 04		MOV	BL, 04H	مقسوم علیه در BL قرار می گیرد.
0002	E5 06	REPEAT:	IN	AX, 06H	داده ی کلمه ای وارد می شود.
0004	F6 F3		DIV	BL	تقسیم بر 4
0006	E7 9A		OUT	9AH, AX	نتیجه در خروجی قرار می گیرد
0008	E9 F7 FF		JMP	REPEAT	سیکل تکرار می شود.
000B					

دستورات پرش غیر مشروط

- این برنامه مکررا داده پورت 6 و 7 را بر 4 تقسیم می کند و نتیجه را در پورت 9AH قرار می دهد.
- دستور JMP REPEAT کنترل برنامه را به دستور خواندن از ورودی منتقل می کند.
- REPEAT برچسبی است که به یک خانه از حافظه داده شده است. این خانه اولین بایت دستور IN AX, 06H را در خود نگه می دارد.
- در این مورد REPEAT بیانگر آدرس 0002H است و در این آدرس E5 ذخیره شده است. عملوند 06H در مکان 0003H قرار دارد.
- قالب مستقیم دستور پرش نزدیک را **پرش نسبی** نیز می گویند، زیرا کنترل برنامه به آدرس جدیدی نسبت به مقداری که در IP قرار دارد منتقل می شود. دو بایتی که بعد از کد عملیاتی JMP قرار می گیرند به محتوای IP افزوده می شوند تا آدرس هدف را تشکیل دهند.

دستورات پرش غیر مشروط

- اسمبلر دو بایت F7 و FF را به عنوان آدرس هدف REPEAT در نظر گرفته است در حالیکه آدرس شروع دستور IN AX, 06H در واقع 0002H است. همان طور که گفته شد این مقدار باید به محتوای IP افزوده شود تا آدرس صحیح بدست آید. بعد از واکشی دستور پرش، محتوای ثبات IP برابر است با 000BH. با افزودن مقدار FFF7H به آن خواهیم داشت:

```
   000B   H
+ FFF7   H
1   0002    H
```

- با صرفنظر کردن از بیت carry حاصل جمع همان آدرس شروع دستورالعمل مطلوب است. عدد FFF7 در واقع همان 9- دهدهی است که بیانگر تعداد یابت هایی است که برنامه باید به عقب برگردد تا به ابتدای دستور مطلوب برسد.

- وقتی هدف در فاصله ی 127+ تا 128- بایتی از محتوای IP قرار گیرد، با بکارگیری **پرش کوتاه**، می توان یک بایت در کد شیء تولید شده صرفه جویی کرد .

دستورات پرش غیر مشروط

آدرس	کد تولید شده	برچسب	op-code	عملوند	توضیحات
0000	B3 04		MOV	BL, 04H	مقسوم علیه در BL قرار می گیرد.
0002	E5 06	REPEAT:	IN	AX, 06H	داده ی کلمه ای وارد می شود.
0004	F6 F3		DIV	BL	تقسیم بر 4
0006	E7 9A		OUT	9AH, AX	نتیجه در خروجی قرار می گیرد
0008	EB F8		JMP SHORT	REPEAT	سیکل تکرار می شود.
000A					

یک مثال برای پرش کوتاه

دستورات پرش غیر مشروط

• با توجه به اینکه در **پرش کوتاه**، یک بایت کمتر به کار می رود لذا روش مناسب تری است.

• در صورتیکه پرش به کار رفته در فاصله ی 127+ تا 128- بایتی IP باشد، اسمبلر به طور خودکار آن را به **پرش کوتاه** تبدیل می کند.

• لذا استفاده از SHORT اختیاری است ولی بهتر است که آن را به کار گیرید چون ممکن است در بعضی از موارد که پرش کوتاه نیز امکان پذیر است اسمبلر پرش نزدیک را به کار گیرد.

• این مساله در مواردی که اسمبلر هنوز برچسب هدف را ندیده است و طبیعتا بدترین حالت را در نظر می گیرد اتفاق می افتد.

مثال های برنامه نویسی

op-code	عملوند	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
JMP	هدف نزدیک	JMP MEMN JMP [MEMWDS] JMP [BX] JMP AX	کد داده داده درون CPU	IP←MEMN IP←[MEMWDS+1:MEMWDS] IP←[BX+1:BX] IP←AX	کنترل برنامه را مکان هدف (نزدیک) درون سگمنت منتقل می کند. مد آدرس دهی می تواند مستقیم، غیر مستقیم حافظه ای یا ثباتی باشد.
JMP SHORT	هدف	JMP SHORT MEMS	کد	IP←MEMS	کنترل برنامه را به مکان هدف (کوتاه) منتقل می کند. برای این دستور قالب غیر مستقیم وجود ندارد.
JMP	هدف دور	JMP FAR PTR MEMF JMP [MEMWWDS] JMP DWORD PTR[BX]	کد داده داده	IP←0003H; CS←9ED3H IP←[1006H:1005H]; CS← [1008H:1007H] IP←[BX+1:BX]; CS←[BX+3:BX+2]	کنترل برنامه را به مکان هدف دور، خارج از سگمنت منتقل می کند.
Jcond	هدف کوتاه	JNC MEMS	کد	اگر CF=0 آنگاه IP←MEMS	اگر شرط برقرار باشد، کنترل را به آدرس هدف کوتاه منتقل می کند. همهی پرش های شرطی به هدف های کوتاه نیاز دارند.
JCXZ	هدف کوتاه	JCXZ MEMS	کد	اگر CX=0 آنگاه IP←MEMS	اگر CX=0 باشد، کنترل را به آدرس هدف کوتاه منتقل می کند.

دستورات پرش غیر مشروط

در جدول و در تمام مواردی که با علامت -- -- مشخص شده اند، مقداری که به جای آن قرار می گیرد، یک افست مکمل ۲ بین مکان حافظه ی نزدیک MEMN و دستوری است که بلافاصله بعد از دستور CALL آمده است.

مثلا اگر CALL به اندازه ۱۲ مکان حافظه جلوتر است، این مقدار آفست 000CH خواهد بود و اگر ۱۲ مکان حافظه عقب تر باشد، FFF4H خواهد بود.

در این جدول فرض شده است MEMN به یک مکان حافظه ی نزدیک درون سگمنت اشاره می کند.

- MEMWDS به یک کلمه از حافظه درون سگمنت داده که آدرس شروع آن 1000H است، اشاره می کند. استفاده از [ ] اختیاری است چون MEMWDS یک کلمه را تعریف می کند نه یک مکان که بتوان به آن پرش کرد.

دستورات پرش غیر مشروط

- **MEMS** به یک مکان حافظه ی کوتاه اشاره می کند (در فاصله ی 127+ تا 128- بایتی از دستوری که بلافاصله بعد از دستور JMP قرار دارد). اگر MEMS بوسیله ی اسمبلر شناخته شده باشد، استفاده از عملگر SHORT اختیاری است.

- **MEMF** به یک مکان حافظه ی دور اشاره می کند 9ED3H:0003H.

- **MEMWWDS** به یک کلمه ی دوتایی در سگمنت داده، به آدرس شروع 1005H اشاره می کند.

دستورات پرش غیر مشروط

- امتیاز اصلی استفاده از دستور پرش نسبی در آن است که **برنامه ی ایجاد شده در هر کجای سگمنت کد قابل بارگذاری و اجرا است**.

- تغییر دادن مکان برنامه ی نوشته، همه آدرس ها را به یک مقدار ثابت تغییر می دهد و لذا موقعیت نسبی مکانی که پرش به آنجا باید انجام شود، تغییری نمی یابد.

- این امکان وجود دارد که کنترل را به آدرس هدف در یک سگمنت کد جدید منتقل کرد. این نوع را **پرش دور** گویند.

- مجددا قالب های مستقیم و غیر مستقیم امکان پذیر است ولی هیچ کدام نسبی نیستند.

- قالب مستقیم به عملگر اسمبلری FAR PTR نیاز دارد تا برچسب داده شده را در یک سگمنت کد جدید تعیین کند.

- قالب های غیرمستقیم به دو کلمه نیاز دارند تا محتوای جدید CS و IP را تخصیص دهند.

دستورات پرش شرطی

دستورات پرش شرطی بر اساس شرایط پرچم های وضعیت یک پرش کوتاه انجام می دهند.

این دستورات معمولا بعد از دستورات محاسباتی می آیند و کنترل برنامه را با توجه به نتایج بدست آمده منتقل می کنند.

عبارت یادآور	شرایط	
	عملیات با علامت	
JG/JNLE	((SF xor OF)+ZF) = 0	**بزرگتر/نه کوچکتر نه مساوی**
JGE/JNL	(SF xor OF)=0	**بزرگتر یا مساوی/نه کوچکتر**
JL/JNGE	(SF xor OF)=1	**کوچکتر/نه بزرگتر نه مساوی**
JLE/JNG	((SF xor OF)+ZF) = 1	**کوچکتر یا مساوی/نه بزرگتر**
JO	**سرریز OF=1**	
JS	**علامت SF=1**	
JNO	**عدم سرریز OF=0**	
JNS	**مکمل علامت SF=0**	

عبارت یادآور	شرایط	
	عملیات بدون علامت	
JA/JNBE	(CF xor ZF)=0	بالاتر/ نه پایین تر نه مساوی
JAE/JNB	بالاتر یا مساوی/ نه پایین تر CF=0	
JB/JNAE	پایین تر/ نه بالاتر نه مساوی CF=1	
JBE/JNA	(CF xor ZF)=1	پایین تر یا مساوی/نه بالاتر

عبارت یادآور	شرایط
	هر دو عملیات
JC	Carry (CF=1)
JE/JZ	مساوی/صفر ZF=1
JP/JPE	توازن/توازن زوج (PF=1)
JNC	NOT Carry (CF=0)
JNE/JNZ	نه مساوی/نه صفر (ZF=0)
JNP/JPO	مکمل توازن/ توازن فرد (PF=0)

دستورات پرش شرطی

مثال: عملیات برنامه ی زیر را توضیح دهید.

```
MOV BL, 47H
IN AL, 36H
CMP AL, BL
JE MATCH
JA BIG
JMP SMALL
```

حل: این برنامه یک بایت داده را از پورت ورودی 36H می خواند و آن را با 47H مقایسه می کند. اگر برابر بودند به مکان MATCH پرش می کند. در صورتیکه داده ی ورودی از 47H بیشتر باشد، پرشی به آدرس BIG صورت می‌گیرد و اگر هیچ کدام از دو شرط برقرار نبود به مکان SMALL پرش می کند. نکته اینکه MATCH، BIG و SMALL باید در فاصله ی 127+ تا 128- بایتی از دستور پرش شرطی قرار داشته باشند.

دستورات پرش شرطی

- دستور JCXZ یک **پرش شرطی ویژه است که وضعیت پرچم ها را بررسی نمی کند**.
- این دستور محتوای CX را می آزماید و اگر 0 باشد کنترل را به آدرس هدف منتقل می کند.
- کاربرد آن در دستورات حلقه ای است.
- دستورات شرطی از جمله ی مهم ترین دستورات ریزپردازنده ای هستند که به سیستم کامپیوتری اجازه می دهند بر اساس شرایط برنامه تصمیم گیری کند.

دستورات حلقه

- یکی از مسائل معمول در برنامه نویسی، برپا کردن مجموعه ای از دستورات است که باید چندین بار اجرا شوند.

- یکی از ثباتهای CPU با شمارنده حلقه بارگذاری می شود و در انتهای هر بار اجرای حلقه یک واحد از آن کاسته می‌شود.

- دستور JNZ در انتهای حلقه، کنترل برنامه را به ابتدای حلقه منتقل می کند اگر شمارنده صفر نباشد.

- دستورات حلقه ای ۸۸/۸۰۸۶ دقیقا برای انجام این عملیات طراحی شده اند.

- عملیات کاستن از شمارنده حلقه و پرش به ابتدای حلقه را در یک دستور انجام می دهد.

- اسلاید صفحه بعد دستورات حلقه سه قالب ممکن را نشان می دهد.

مثال های برنامه نویسی

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
LOOP	هدف کوتاه	E2 --	LOOP MEMS	کد	CX←CX-1 اگر CX≠0 آنگاه IP←MEMS	یک واحد از CX می کاهد و اگر CX≠0 کنترل برنامه را به هدف کوتاه منتقل می کند.
LOOPE /LOOPZ	هدف کوتاه	E1 --	LOOPZ MEMS	کد	CX←CX-1 اگر (CX≠0).(ZF=1) آنگاه IP←MEMS	یک واحد از CX می کاهد و اگر CX≠0 و آخرین دستور تاثیر گذار بر پرچم ها نتیجه ی 0 داشته (ZF=1)، کنترل برنامه را به هدف کوتاه منتقل می کند.
LOOPNE /LOOPNZ	هدف کوتاه	E0 --	LOOPNZ MEMS	کد	CX←CX-1 اگر (CX≠0).(ZF=0) آنگاه IP←MEMS	یک واحد از CX می کاهد و اگر CX≠0 و آخرین دستور تاثیرگذار بر پرچم ها نتیجه ی غیرصفر داشته (ZF=0)، کنترل برنامه را به هدف کوتاه منتقل می کند

- در این جدول فرض شده است که MEMS به یک مکان حافظه ی کوتاه در فاصله ی 127+ تا 128- بایتی از دستورالعملی که بلافاصله بعد از دستور پرش قرار گرفته اشاره می کند.

دستورات حلقه

مثال: با استفاده از دستورات حلقه، برنامه ای بنویسید که ۲۵۶ بایت داده را از یک جدول که آدرس شروع آن در TABLE قرار دارد خوانده و در پورت خروجی A0H قرار دهد.

حل:

LEA SI, TABLE	; *SI* را با آدرس پایه ی TABLE بارگذاری می کند
MOV CX, 0100H	;
AGAIN: LODSB	; بایت داده در AL قرار گرفته، SI افزایش می یابد.
OUT 0A0H, AL	; بایت داده در خروجی قرار می گیرد
LOOP AGAIN	; در این برنامه DF=0 فرض شده است.

در این برنامه DF=0 فرض شده است.

دستورات حلقه

- مثال قبل نشان می دهد که دستورات رشته ای و حلقه در حقیقت هر کدام دو عملیات را انجام می دهند.

- دستور LODSB معادل دستورات MOV AL, [SI] و INC SI است.

- دستور LOOP AGAIN هم معادل DEC CX و JNZ AGAIN است.

- دستورات LOOPE یا LOOPZ (تکرار حلقه به شرط تساوی یا صفر بودن) و LOOPNE یا LOOPNZ (تکرار حلقه به شرط عدم تساوی یا صفر نبودن)، پرچم های CX و ZF را می آزمایند.

- مثلا در دستور LOOPE (تکرار حلقه مادامی که تساوی برقرار است) اگر CX≠0 و ZF=1 باشد حلقه تکرار می شود.

- دستور LOOPNE (تکرار حلقه مادامی که تساوی برقرار نیست) نیز به طریق مشابه عمل می کند. این دو قالب برای مقایسه کردن دو رشته یا جستجو کردن محتوای یکی از ثباتهای CPU در یک جدول داده مناسب است.

- همه ی قالب های دستورات حلقه ای تکرار می شوند تا زمانیکه CX=0 شود. اگر مقدار اولیه ی CX=0 باشد، حلقه 65536 بار تکرار خواهد شد.

دستورات PUSH و POP

- دستورات Push و Pop واقعا از نوع دستورات انتقال کنترل برنامه نیستند، ولی آنها فضای پشته از حافظه را به کار می گیرند که لازم است قبل از مطالعه ی وقفه های نرم افزاری و دستورات CALL و RET، مورد بررسی قرار گیرند.

- یادآوری می شود که حافظه پشته یک سگمنت ۶۴ کیلوبایتی از حافظه است که آدرس پایه ی آن را ثبات سگمنت SS تعیین می کند. دو ثبات CPU معمولا به این مکان اشاره می کنند. این دو ثبات SP و BP هستند.

- حافظه پشته یک سگمنت داده جدید نیست ولی پردازنده از آن برای ذخیره ی داده های موقتی استفاده می کند. آنچه پشته را منحصر به فرد می کند، ویژگی LIFO بودن است. وارد شدن داده به پشته از طریق دستور "مبدا PUSH" انجام می شود. دستور "مقصد POP" آخرین داده ی وارد شده به پشته را می خواند.

دستورات PUSH و POP

- شکل موجود در اسلایدهای در چند اسلاید بعد سگمنت پشته را نشان می دهد به همراه ثبات SP که در حال حاضر به بالای پشته (1000H) اشاره می کند. فرض کنید دستور PUSH CX اتفاق افتد. دو واحد از SP کاسته می شود و CL و CH در پشته ذخیره می شوند. اکنون آدرس جدیدی که بالای پشته را تعیین می کند SP’ (0FFEH) است. همان گونه که دیده می شود آدرس بالای پشته در اثر دستورات متوالی PUSH، به سمت پایین رشد می کند. بعد از اجرای دستور PUSH بعدی، بالای پشته در آدرس 0FFCH (SP’’) است.

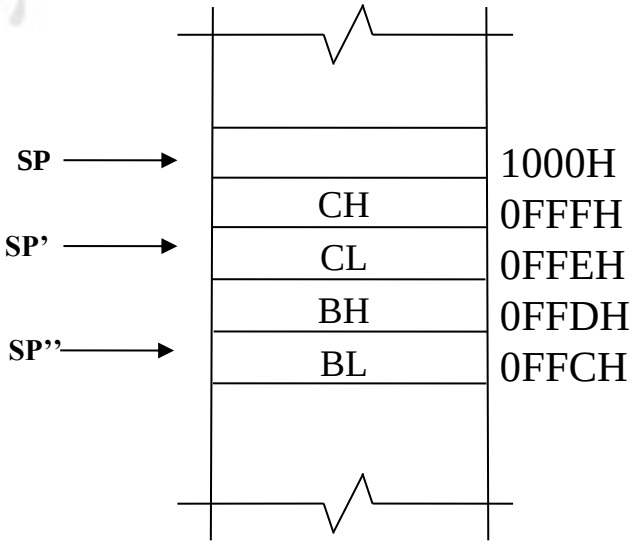
- حال که محتوای دو ثبات BX و CX به درستی در پشته ذخیره شد، می توان از این دو ثبات برای اهداف دیگری بهره جست. اگر بخواهیم محتوای قبلی این ثباتها را بازیابی کنیم، از دستور POP استفاده می کنیم ولی باید این نکته را در نظر داشته باشیم که ابتدا محتوای BX و سپس CX بازیابی می شود.

دستورات PUSH و POP

- برپایه ی مشاهدات قبلی می توان یک قانون کلی برای استفاده از پشته ارائه کرد. ترتیب بازیابی ثباتها از پشته برعکس ترتیب قرار گرفتن آنها باید انجام می شود. در مثال عنوان شده دستور POP BX محتوای آدرس های 0FFCH و 0FFDH را می خواند و در BL و BH قرار می دهد. اکنون محتوای ثبات CX در بالای پشته است و اجرای یک دستور POP آن را می خواند و در CX قرار می دهد.

- قبل از استفاده از پشته، لازم است ثبات SP مقدار دهی اولیه شود. انجام این کار با یک دستور MOV و مد آدرس دهی بلافصل مثل MOV SP, 1000H انجام می شود بالای پشته را در مکان 1000H از سگمنت پشته قرار می‌دهد. از آنجا که پشته معمولا به سمت پایین رشد می کند، SP را در یک مکان با آدرس بزرگ مقدار دهی می کنند تا پشته بتواند براحتی بزرگ شود.

دستورات PUSH و POP



- موقعیت پشته در حافظه به خصوص وقتی که سگمنت های داده ، کد و پشته بر یکدیگر هم پوشانی دارند، بسیار مهم است. اگر مراقب نباشید، محتوای پشته ممکن است بر روی داده های برنامه و یا حتی خود برنامه نوشته شود. عیب یابی چنین برنامه ای مشکل است چون دیگر برنامه ای باقی نمانده است.

مثال های برنامه نویسی						
op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
PUSH	**مبدا**	**51** **1E** **FF 75 02**	**PUSH CX** **PUSH DS** **PUSH [DI+2]**	**پشته** **پشته** **پشته، داده**	SP←SP-2; [SP+1] ← CH; [SP] ← CL; SP←SP-2; [SP+1:SP] ←DS; SP←SP-2; [SP+1] ←[DI+3]; [SP] ←[DI+2];	**دو واحد از SP می کاهد و کلمه ی عملوند مبدا را به بالای پشته که اکنون SP بدان اشاره می کند، منتقل می کند.**
POP	**مقصد**	**59** **1F** **8F 45 02**	**POP DX** **POP DS** **POP [DI+2]**	**پشته** **پشته** **پشته، داده**	CL←[SP]; CH←[SP+1]; SP←SP+2; DS←[SP+1:SP]; SP←SP+2; [DI+3] ←[SP+1]; [DI+2] ←[SP]; SP←SP+2;	**کلمه ی بالای پشته را که SP بدان اشاره می کند، به عملوند مقصد منتقل می کند. دو واحد به SP می افزاید.**
PUSHF	**ندارد**	**9C**	**PUSHF**	**پشته**	SP←SP-2; [SP+1:SP] ←flags	**کلمه ی ۱۶ بیتی پرچم را در بالای پشته قرار می دهد.**
POPF	**ندارد**	**9D**	**POPF**	**پشته**	flags←[SP+1:SP]; SP←SP+2	**کلمه ی بالای پشته را به ثبات پرچم منتقل می کند.**

- در انتها ذکر این نکته ضروری است که دستورات PUSHF و POPF به پردازنده اجازه می دهند که محتوای پرچم های وضعیت را در پشته ذخیره و بازیابی کند. این امکان در بکارگیری وقفه ها و زیر روتین ها بسیار مفید است چون وضعیت ثباتها و پرچم های CPU ذخیره می شود و بعد از بازگشت از زیربرنامه مجددا بازیابی می شود.

دستورات Call و Return

• دستورات CALL و RET به برنامه نویس اجازه می دهند گروهی از دستورات را به عنوان یک زیربرنامه یا زیر روال به کار گیرد و در بخش های مختلف برنامه ی اصلی فراخوانی کند. به عنوان مثال دستورات زیر یک تاخیر کوتاه ایجاد می کنند:

```
DELAY    PROC NEAR ; یک زیرروال  نزدیک تعریف می کند
         PUSH AX;AX را در پشته ذخیره می کند
         MOV AX, 0000;شمارنده را برای یک سیکل ۶۵۵۳۶تایی مقداردهی می کند
REPEAT:  DEC AX; از مقدار شمارنده یکی می کاهد
         JNZ REPEAT;۶۵۵۳۶ بار فرآیند را تکرار می کند
         POP AX;مقدار قبلی AXرا بازیابی می کند
         RET;به برنامه ی اصلی برمی گردد
DELAY    ENDP;
```

دستورات Call و Return

• اولین خط برنامه، گروه دستورات را به عنوان یک زیربرنامه ی نزدیک تعریف می کند (نزدیک بودن به معنی قرار داشتن درون سگمنت است) و نام آن را DELAY قرار می دهد. این زیربرنامه با تکرار دستورات DEC AX و JNZ REPEAT به تعداد ۶۵۵۳۶ بار، زمان صرف می کند. این عملیات در یک سیستم با پالس ساعت 5MHz به ۲۵۰میلی ثانیه زمان نیاز دارد. به محض انجام این عملیات، مقدار قبلی AX بازیابی می شود و کنترل برنامه از طریق دستور RET به مکان قبلی اش بازمی گردد. آخرین سطر برنامه به اسمبلر می گوید که اینجا انتهای زیربرنامه است.

• زیربرنامه ی DELAY توسط هر برنامه ای که به ۲۵۰ میلی ثانیه زمان تاخیر نیاز دارد قابل فراخوانی است. به عنوان مثال فرستادن داده به یک دستگاه کند که نمی تواند به سرعت ریزپردازنده داده را بپذیرد نیازمند ایجاد تاخیر در برنامه ی ریزپردازنده است. فراخوانی زیربرنامه در برنامه ی اصلی می تواند به صورت زیر باشد:

برنامه اصلی
CALL DELAY
ادامه ی برنامه ی اصلی

.
.

مثال های برنامه نویسی						
op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
Call	هدف نزدیک	E8 -- --	CALL MEMN	کد	SP←SP-2; [SP+1:SP] ←IP; IP←MEMN;	IP در بالای پشته قرار می گیرد و کنترل برنامه به آدرس هدف نزدیک درون سگمنت منتقل می شود.
		FF 16 00 10	CALL [MEMWDS]	داده	SP←SP-2; [SP+1:SP] ←IP; IP←[1001H:1000H]	
		FF 15	CALL [DI]	داده	SP←SP-2; [SP+1:SP] ←IP; IP←[DI+1:DI]	
		FF D7	CALL DI	درون CPU	SP←SP-2; [SP+1:SP] ←IP; IP←DI	
CALL	هدف دور	9A 00 10 D3 09	CALL FAR PTR[MEMF]	کد	SP←SP-2; [SP+1:SP] ←CS; CS←09D3H; SP←SP-2; [SP+1:SP] ←IP; IP←1000H	CS و IP در بالای پشته قرار می گیرند و کنترل برنامه به آدرس هدف دور در سگمنت جدید منتقل می شود.
		FF 1E 00 10	CALL [MEMWWDS]	داده	همانندبالا به جز اینکه: CS←[1003H:1002H]; IP←[1001H:1000H]	
		FF 1D	CALL DWORD PTR[DI]	داده	همانند بالا به جز اینکه: CS←[DI+3:DI+2]; IP←[DI+1:DI]	
RET	(نزدیک) n	C3	RET	پشته	IP←[SP+1:SP]; SP←SP+2	کلمه ی بالای پشته خوانده شده و در IP قرار می گیرد و کنترل برنامه به این آدرس جدید منتقل می شود. به طور معمول دستور RET برای بازگشت از زیربرنامه ها به کار می رود. در صورت وجود n مقدار آن به SP افزوده می شود.
		C2 08 00	RET 8	پشته	IP←[SP+1:SP]; SP←SP+2+8	
RET	(دور)n	CB	RET	پشته	IP←[SP+1:SP]; SP←SP+2; CS←[SP+1:SP]; SP←SP+2	همانند قبل به جز اینکه دو کلمه ی بالای پشته به IP و CS منتقل می شوند و کنترل برنامه را به آدرس دور منتقل می کنند.
		CA 08 00	RET 8	پشته	IP←[SP+1:SP]; SP←SP+2; CS←[SP+1:SP]; SP←SP+2+8	

دستورات Call و Return

-در سطر اول جدول، مقداری که به جای -- -- قرار می گیرد، یک افست مکمل ۲ بین مکان حافظه ی نزدیک MEMN و دستوری است که بلافاصله بعد از دستور CALL آمده است. مثلا اگر CALL به اندازه ی ۱۲ مکان حافظه جلوتر است، این مقدار آفست 000CH خواهد بود و اگر ۱۲ مکان حافظه عقب تر باشد، FFF4H خواهد بود.

-مقدار MEMN به یک مکان حافظه ی نزدیک (درون سگمنت) اشاره می کند.
MEMWDS به کلمه ای با آدرس شروع 1000H در سگمنت داده اشاره می کند. علامت [ ] اختیاری است چون MEMWDS یک کلمه را تعریف می کند و نه یک مکان که بتوان به آن پرش کرد.

-عبارت FAR PTR بین می کند که MEMF در سگمنت کد متفاوتی واقع است. در این مورد فرض شده است که به مکان 09D3H:1000H اشاره می کند.

-MEMWWDS به دو کلمه با آدرس شروع 1000H در سگمنت داده اشاره می کند.
علامت [ ] اختیاری است چون MEMWWDS دو کلمه را تعریف می کند و نه یک مکان که بتوان به آن پرش کرد.

- برای هر دو نوع بازگشت دور و نزدیک یک عبارت یادآور به کار می رود. اسمبلر با توجه به چگونگی تعریف زیربرنامه (دور یا نزدیک) کد مناسب را تولید می کند.

دستورات Call و Return

- دستور CALL شبیه دستور پرش غیر مشروط است ولی با این تفاوت که مقدار ثبات IP (که اکنون به دستور بعد از CALL اشاره می کند) در پشته ذخیره می شود. سپس کنترل برنامه به DELAY (در اینجا) منتقل می شود.

- بعد از اجرای زیربرنامه ی DELAY لازم است یک دستور RET در انتهای زیربرنامه وجود داشته باشد. این دستور محتوای بالای پشته را در IP قرار می دهد و لذا کنترل برنامه به دستوری که بعد از CALL بود منتقل می شود.

- جدول قبل قالب های مختلف دستورات CALL و RET را توصیف می کند. دستوری که مورد بحث قرار گرفت، به CALL نزدیک معروف است زیرا زیربرنامه در همان سگمنت کد فعلی است. این امکان هم وجود دارد که یک دستور CALL دور در سگمنت کد متفاوتی انجام دهیم.

- تفاوت CALL دور در این است که مقدار ثبات CS هم علاوه بر IP در پشته ذخیره می شود. به همین دلیل هنگام بازگشت از این زیر برنامه باید دستور RET دور را اجرا کرد تا علاوه بر IP مقدار CS هم از پشته بازیابی شود. تشخیص نوع دستور RET بر عهده ی اسمبلر است که با توجه چگونگی تعریف زیربرنامه (دور یا نزدیک) انجام می دهد.

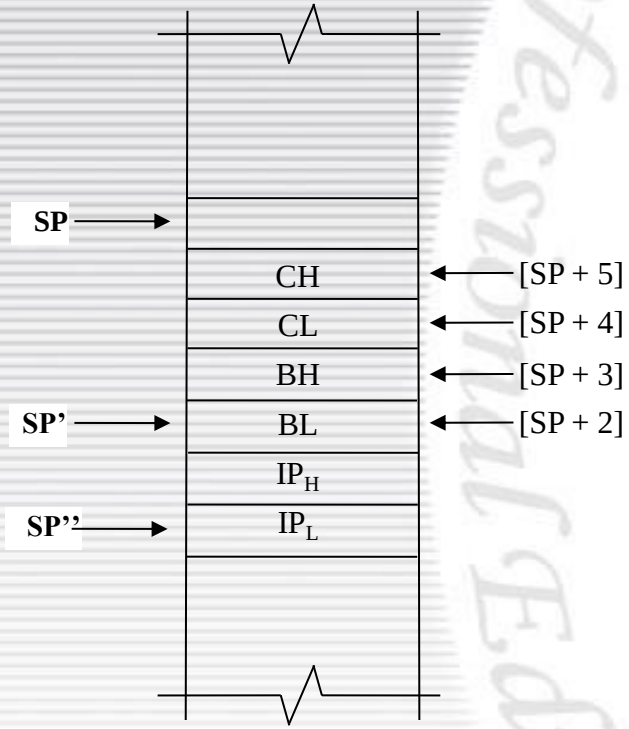
دستورات Call و Return

- تشخیص نوع دستور RET بر عهده ی اسمبلر است که با توجه چگونگی تعریف زیربرنامه (دور یا نزدیک) انجام می دهد.

- دستور CALL نزدیک همانند دستورات پرش نزدیک می توانند مستقیم یا غیرمستقیم باشد. قالب های مستقیم نسبی هستند و به زیربرنامه اجازه می دهند که در فاصله ی 32767+ تا 32768- بایتی از آدرس موجود در IP قرار گیرند. قالب کوتاه وجود ندارد. قالب غیرمستقیم آدرس دقیق را در یک مکان حافظه یا ثبات CPU تعیین می کنند. دستور CALL دور به عملگر FAR PTR نیاز دارد تا به اسمبلر اعلام کند که زیربرنامه در سگمنت دیگری قرار دارد. قالب های غیرمستقیم به دو کلمه برای تعیین مقادیر جدید CS و IP نیاز دارند.
همچنان که در جدول مشخص است دستورات RET می توانند یک مقدار *pop* اختیاری نیز داشته باشند. این امکان اجازه می دهد که ارسال داده به زیربرنامه به جای ثباتهای CPU از طریق پشته باشد. به عنوان مثال:

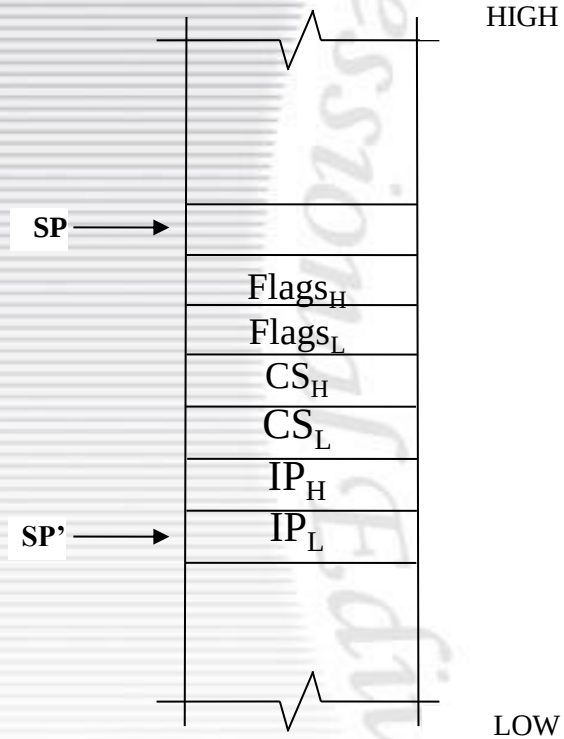
```
PUSH CX
PUSH BX
CALL SUB
SUB:    .
        .
        .
RET 4
```

دستورات Call و Return



• برنامه ی اصلی چهار بایت (ثباتهای CX و DX) را قبل از فراخوانی زیربرنامه در پشته قرار می دهد. این داده ها بعدا در زیر برنامه با استفاده از ثبات BP و اشاره به مکان های SP+2 تا SP+5 (شکل روبرو) قابل دستیابی هستند. بعد از اجرای زیربرنامه که داده های موجود در پشته بی اهمیت هستند، دستور RET 4 مقادیر قبلی IP را بازیابی کرده و موقعیت بالای پشته را در وضعیت SP' قرار می دهد؛ مقدار ۴ نیز به آن افزوده می شود و نهایتا آن را وضعیت SP قرار می دهد و لذا اثر داده ها در پشته از بین می رود.

وقفه های نرم افزاری

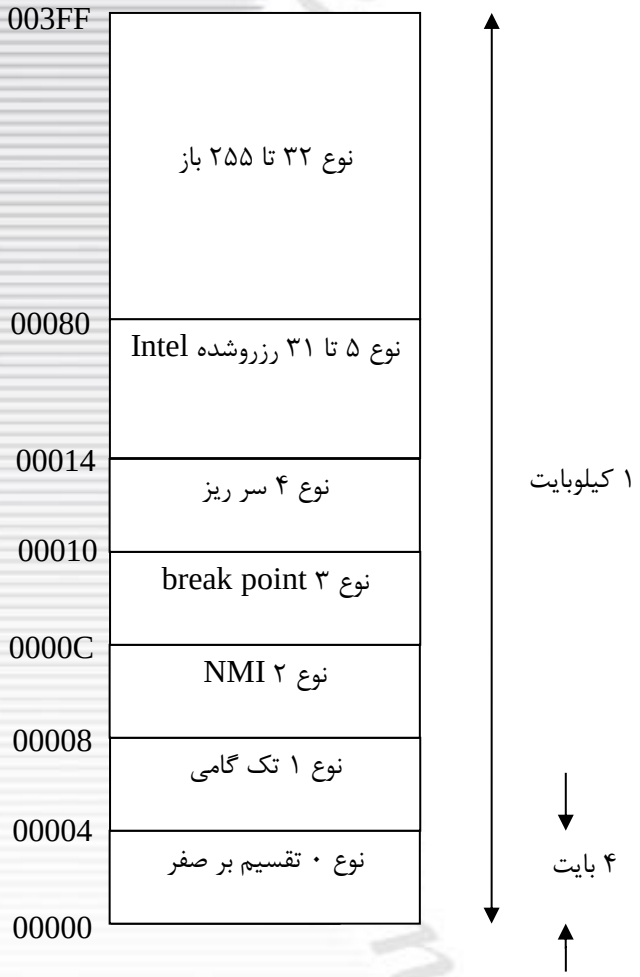


وقفه یک درخواست است که به CPU داده می شود تا برنامه ی فعلی خود را به تعویق اندازد و کنترل برنامه را به برنامه ی جدیدی که روتین اجرای وقفه (ISR) نامیده می شود منتقل کند.

درخواست وقفه می تواند سخت افزاری یا نرم افزاری باشد. برای ۸۸/۸۰۸۶ اعمال سطح منطقی ۱ به پایه ی INTR یا NMI از خطوط ورودی، درخواست وقفه می کند.

وقفه های سخت افزای در بخش های بعدی بررسی می گردند

وقفه های نرم افزاری



- وقفه های نرم افزاری با دستور "نوع INT" شروع می شوند.
- پردازنده در اجرای این دستور محتوای IP و CS و پرچم های پردازنده را در پشته قرار میدهد.
- مقادیر جدید IP و CS از جدول پرش وقفه که در آدرس مطلق 00000 تا 003FFH قرار دارد، خوانده میشود.
- این یک کیلوبایت ۲۵۶ نوع وقفه سخت افزاری متفاوت را ممکن می سازد که در شکل روبرو نشان داده شده است.

وقفه های نرم افزاری

• اگر دستور INT 23H اجرا شود، پردازنده 23 را در 4 ضرب کرده و آدرس مربوط به این وقفه را در جدول پرش وقفه پیدا می کند.

00100011*4=10001100=0008CH

• ثباتهای IP و CS با کلمه دوتایی موجود در آدرس 0008CH تا 0008FH (CS در آدرس پرارزش و IP در آدرس کم ارزش) مقداردهی میشوند.

•در واقع INT 23H یک CALL دور به مکانی از حافظه که آدرس آن در دو کلمه به آدرس شروع 0008CH قرار دارد انجام می دهد.

• وقتی روتین سرویس وقفه تمام شد، دستور IRET باید اجرا شود.

•این دستور شبیه دستور بازگشت از زیربرنامه است ولی علاوه بر CS و IP پرچم ها را نیز از پشته فراخوانی می کند.

•برنامه اصلی با دستوری که بلافاصله بعد از دستور INT 23H قرار دارد دنبال می شود.

مثال های برنامه نویسی						
op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
INT	نوع	CD 23	INT 23H	پشته و جدول پرش وقفه در آدرس 00000 تا 003FFH	SP←SP-2; [SP+1:SP] ←flags; IF←0; TF←0; SP←SP-2; [SP+1:SP] ←CS; CS←[0008FH:0008EH]; SP←SP-2; [SP+1:SP] ←IP; IP←[0008DH:0008CH]	محتوای ثباتهای IP،CS و پرچم را در پشته ذخیره کرده و کنترل را به آدرس دوری که در مکان حافظه به آدرس شروع "نوع*۴" ذخیره شده، منتقل می‌کند.
INTO	ندارد	CE	INTO	پشته و جدول پرش وقفه در آدرس 00000 تا 003FFH	اگر OF=1 آنگاه SP←SP-2; [SP+1:SP] ←flags; IF←0; TF←0; SP←SP-2; [SP+1:SP] ←CS; CS←[00013H:00012H]; SP←SP-2; [SP+1:SP] ←IP; IP←[00011H:00010H]	اگر سرریز رخ داده باشد (OF=1) آنگاه یک وقفه نوع ۴ اجرا میشود.
IRET	ندارد	CF	IRET	پشته	IP←[SP+1:SP]; SP←SP+2; CS←[SP+1:SP]; SP←SP+2; flags←[SP+1:SP]; SP←SP+2	کنترل برنامه را به مکانی که وقفه در آنجا اتفاق افتاده است، برمیگرداند. ثباتهای IP، CS و پرچم را از پشته بازیابی میکند. IRET معمولا برای خروج از هر زیربرنامه وقفه ای (سخت افزاری یا نرم افزاری) به کار میرود.

وقفه های نرم افزاری

- از آنجا که یک وقفه ی سخت افزاری نیز می تواند INT 23H را تولید کند، یکی از کاربردهای وقفه های نرم افزاری **عیب یابی روتین سرویس وقفه** بدون نیاز به ایجاد وقفه به صورت سخت افزاری است.

- همان طور که در شکل قبل دیده می شود، بعضی از مکان های جدول پرش وقفه برای کاربردهای خاص رزرو شده اند.

- به عنوان مثال وقتی عملیات تقسیم بر صفر اتفاق می افتد، دو کلمه ی موجود در آدرس 00000H-00003H به عنوان تعیین کننده آدرس روتین پاسخ دهی به وقفه به کار می رود.

- به طور مشابه اگر TF=1 شود، کنترل برنامه بعد از اجرای هر یک از دستورات برنامه ی اصلی، به آدرسی که در مکان 00004H-00007H تعیین شده است منتقل می شود.

وقفه های نرم افزاری

•دستور INTO (interrupt on overflow) از این نظر خاص است که برای تعیین نوع، نیازی به آوردن شماره ی وقفه نیست. بعد از اجرای یک عملیات ریاضی با علامت، اجرای دستور INTO در صورتیکه سرریز رخ داده باشد وقفه ی نوع 4 ایجاد می کند.

• نکته اینکه اجرای دستور وقفه (نرم افزاری یا سخت افزاری) پرچم های IF و TF را مقدار 0 می دهد.

•این کار مطمئن می سازد که روتین اجرای وقفه در مد تک گامی نیست و نیز وقفه های خارجی (قابل پوشش) بر پایه INTR را غیرفعال می کند.

• ولی وقفه های غیر قابل پوشش بر پایه ی NMI و نیر وقفه های نرم افزاری همچنان پذیرفته می شوند.

•اجرای دستور IRET پرچم ها را به وضعیت قبل از وقفه برمی گرداند. بنابراین نیازی نیست در مورد مقادیر IF و TF قبل از وقفه نگران بود.

دستورات کنترل پردازنده

دستورات کنترل پروسسور آخرین گروه دستوراتی هستند که عملکرد پردازنده را کنترل می کنند و شاخص های وضعیت را مقداردهی می کنند.

پرچم های توازن (CF)، جهت (DF) و وقفه (IF) را هر کدام می توان مقداردهی 0 یا 1 کرد.

پرچم CF را می توان مکمل کرد.

به باقی پرچم های وضعیت (PF، AF، ZF و OF) نمی توان از طریق دستورات کنترلی ویژه دسترسی داشت.

op-code	عملوند	کد شیء	عبارت یادآور	سگمنت مربوطه	عملیات سمبولیک	توصیف
STC	ندارد	F9	STC	درون CPU	CF←1	مقدار پرچم Carry را 1 می کند.
CLC	ندارد	F8	CLC	درون CPU	CF←0	مقدار پرچم Carry را 0 می کند.
CMC	ندارد	F5	CMC	درون CPU		Carrt را مکمل می کند.
STD	ندارد	FD	STD	درون CPU	DF←1	پرچم جهت را 1 می کند.
CLD	ندارد	FC	CLD	درون CPU	DF←0	پرچم جهت را 0 می کند.
STI	ندارد	FB	STI	درون CPU	IF←1	پرچم وقفه را 1 می کند.
CLI	ندارد	FA	CLI	درون CPU	IF←0	پرچم وقفه را 0 می کند.
HLT	ندارد	F4	HLT	درون CPU	ندارد	توقف
WAIT	ندارد	9B	WAIT	درون CPU	ندارد	به وضعیت انتظار می‌رود اگر TEST !=1
LOCK	دستور	F0 A1 00 10	LOCK MOV AX, MEMWDS	داده	ندارد	در حالیکه دستور بعد از LOCK انجام میشود، خروجی خط LOCK! را برابر 0 قرار میدهد. این کارمعمولا برای جلوگیری از دسترسی coprocessor به باس در حین اجرای دستورات خاص است.
NOP	ندارد	90	NOP	داده	ندارد	عدم اجرای عملیات
ESC	مبدا، شماره	DE 0E 00 10	ESC 31H, MEMWDS	داده	Data Bus← [MEMWDS]	محتوای عملوند مبدا حافظه را بر باس داده قرار میدهد و یک NOP اجرا میکند. عملوند اول یک دستور escape ویژه را تعیین میکند که coprocessor باید انجام دهد.

دستورات کنترل پردازنده

- پرچم هایDF، IF و TF در حقیقت بیت های کنترل پردازنده هستند نه شاخص های وضعیت.

- پرچم DF با گروه دستورات رشته ای به کار می رود تا تعیین کند ثبات های اشاره گر باید کاسته شوند یا افزایش یابند.

- دستورات STD و CLD برای مقدار دهی این پرچم به کار گرفته می شوند.

- دستورات STI(set interrupt enable flag) و CLI(clear interrupt enable flag) برای فعال یا غیرفعال کردن وقفه های قابل استتار بر روی خط ورودی INTR به کار می روند. 0

- قرار دادن این بیت، همه ی وقفه ها را بر پورت INTR غیرفعال کرده و در واقع این ورودی را مستتر می کند.

دستورات کنترل پردازنده

- بیت TF (trap flag) وقتی 1 شود، بعد از اجرای هر دستور یک وقفه ایجاد می شود.

•دستورالعملی برای مقدار دهی به این پرچم وجود ندارد ولی به عنوان مثال گروه دستورات زیر برای مقداردهی به این پرچم قابل استفاده است.

```
PUSHF                          ;پرچم ها را در پشته کپی می کند
MOV BP, SP                     ;BP به بالای پشته اشاره می کند
OR BYTE PTR[BP+1], 01H         ;بیت صفرم که همان TF است را مقدار 1 می دهد
POPF                           ;پرچم ها را بازیابی می کند
```

•دستور HALT پردازنده را متوقف می کند و آن را به حلقه ی بیکار وارد می کند.

•بعد از متوقف شدن پردازنده با اعمال ریست یا وقفه ی سخت افزاری مجددا شروع به کار می کند.

•به همین دلیل استفاده از HALT در انتهای برنامه ایده خوبی نیست چون راه اندازی مجدد کامپیوتر ممکن است برنامه را پاک کند.

دستورات کنترل پردازنده

- سه دستور کنترل پردازنده برای کار با coprocessor های ویژه در نظر گرفته شده اند.

- **دستور WAIT برای سنکرون کردن** ۸۸/۸۰۸۶ با ۸۰۸۷ (پردازنده داده های عددی) از طریق سیگنال ورودی TEST به کار می رود.

- وقتی ۸۰۸۷ در حال انجام کاری است و نمی تواند داده یا دستور جدیدی را از ۸۸/۸۰۸۶ دریافت کند، این ورودی را در وضعیت High قرار می دهد.

- با قرار دادن دستور WAIT قبل از هر دستوری که نیاز به دریافت داده از ۸۰۸۷ دارد، ۸۸/۸۰۸۶ مطمئن می شود که داده بین این دو پردازنده گم نخواهد شد.

- وقتی لازم است مطمئن شویم هیچ پردازنده دیگری کنترل باس سیستم را در دست ندارد، از دستور LOCK استفاده می کنیم.

- تاثیر LOCK فقط برای یک دستور بعدی است.

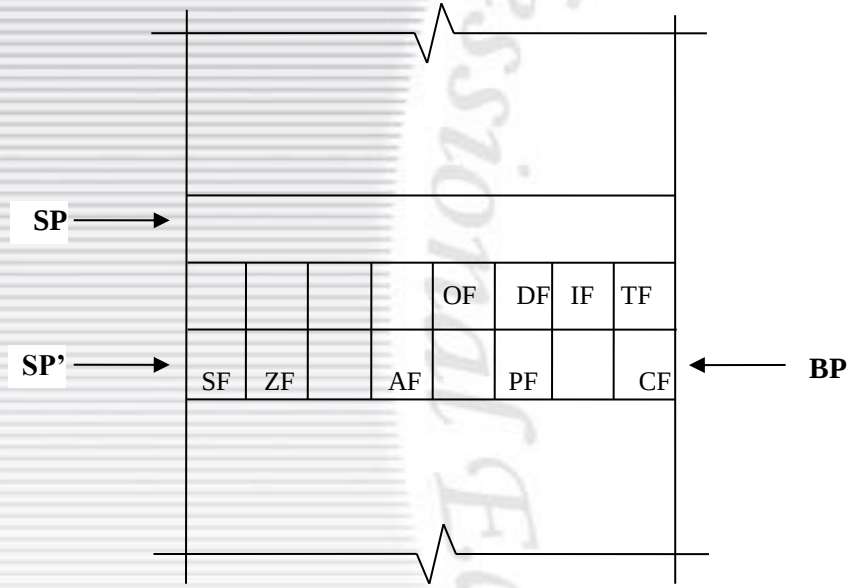
- این دستور مانع از آن می شود که coprocessor به یک مکان حافظه که ۸۸/۸۰۸۶ می خواهد دست یابد، دسترسی داشته باشد یا تغییری در آن ایجاد کند.

دستورات کنترل پردازنده

دستور ESC (escape) به عنوان پیشوندی برای دستورات coprocessor به کار می رود.

۸۸/۸۰۸۶ عملوند مبدا را بر باس داده قرار می دهد ولی هیچ عمل دیگری انجام نمی دهد.

coprocessor که دائما محتوای باس داده را بررسی می کند، با پیشوند ESC فعال می شود، دو عملوند را می خواند و شروع به اجرای دستور می کند.



چگونگی قرار گرفتن ثبات پرچم ها در پشته

مسائل

۱- کدامیک از آدرس‌های زیر می‌توانند به‌عنوان آدرس پایه یک سگمنت از حافظه برای ثبات‌های سگمنت باشند؟
الف- FFFFFH ب- C0FF0H ج-90055H د-10040H

۲- آدرس منطقی مناسب برای حافظه پشته درحالی‌که آدرس فیزیکی، 8B3F8H باشد، چیست؟ فرض کنید که SS=5C00H.

۳- اگر CS=143A و IP=32B4باشند، آدرس منطقی و فیزیکی و محدوده ی بالا و پایین قطعه کد را مشخص کنید.

۴- مزیت حالت آدرس‌دهی غیرمستقیم ثباتی نسبت به حالت آدرس‌دهی مستقیم چیست؟

۵- عبارت یادآوری را بنویسید که کلمه‌ای که ثبات SI با جابه‌جایی 33H به آن اشاره می‌کند را به ثبات CX منتقل کند.

۶- تحت چه شرایطی دستورالعمل REPNE CMPSB کنترل برنامه را به دستورالعمل بعدی منتقل می‌کند؟ (دو شرط)

۷- زیر روالی بنویسید که یک بایت داده را از درگاه I/O به شماره 9200H وارد کرده و بیت ۵ آن را تست کند . اگر این بیت 1 بود، عدد 21H را در همین درگاه بنویسد.

8- بعد از اجرای دستورات زیر محتوای ثبات BL چه مقداری خواهد بود؟

```
MOV BL, 0B2H
MOV CL, 2
SAR BL, CL
```

مسائل

۹- محتوای ثبات AL را بعد از اجرای دستورات زیر تعیین کنید.

```
MOV AL, 3EH
MOV CH, 0A0H
ADD CH, 16H
ADD AL, CH
NEG AL
INC AL
```

۱۰- پنج دستورالعملی را که می‌توان برای انتقال کنترل برنامه به مکان جدیدی خارج از ترتیب به‌کار برد، را نام ببرید.

۱۱- کدام دستورالعمل ۸۸/۸۰۸۶ معادل مجموعه‌ی چهار دستور زیر است؟

```
PUSH BX
PUSH AX
POP BX
POP AX
```

۱۲- برنامه‌ی زیر برای مقایسه دو رشته‌ی ذخیره شده در حافظه نوشته شده است. حداقل دو عیب در این برنامه وجود دارد. آن‌ها را بیابید.

```
LEA SI, STRING1
LEA DI, STRING2
CLD
MOV CX, NO_OF_BYTES
REPNE  SCASB
JNZ ERROR
JMP OK
```

مسائل

۱۳-محتوای ثبات‌های AL و BLو پرچم‌ها را در انتهای برنامه زیر مشخص نمائید.

```
MOV BL, 0C2H
MOV CL, 3
SAR BL, CL
MOV AL, 4AH
MOV CH, 0B9H
ADD AL, CH
NEG AL
DEC AL
SBB AL, 3EH
XOR BL, BL
MOV [SI], BL
```

۱۴- دنباله‌ای از دستورات را بنویسید که به ترتیب محتوای BX را با AX و SI را با DI معاوضه کنند.

۱۵- یک زیربرنامه‌ی دور بنویسید که ۵ کلمه موجود در حافظه‌ی CS:DATA1به بعد را را در ثبات‌های AX، BX، CX، DX و SI کپی کند.

۱۶- دنباله‌ای از دستورات را بنویسید که مقدار 90H را در یک بخش 100H بایتی از حافظه داده با عنوان LIST جستجو کند.

۱۷- دنباله‌ی کوتاهی از دستورات را بنویسید که مقدار 00H را در 155H بایت حافظه در مکان DATA که در سگمنت اضافی قرار دارد، بنویسد. برای سادگی این کار لازم است از دستور LOOP استفاده کنید.

۱۸- زیرروال نزدیکی بنویسید که محتوای ثبات CX را به توان ۳ رسانده و در CX ذخیره کند. این برنامه نباید محتوای هیچ ثبات دیگری به‌جز CX را تغییر دهد.

مسائل

19- توضیح دهید که برنامه زیر چه کاری را انجام می‌دهد:
توجه: فراخوانی روتین PRINT_DL کاراکتر با کد اسکی که در ثبات DL است را چاپ می‌کند.

```
L00: MOV CX, 16
L01: RCL AX, 1
L02: ADC DL, 30H
L03: CALL PRINT_DL
L04: LOOP L01
```

20- تفاوت بین دو دستور LEA BX, NUMBDS و MOV BX, NUMBDS را توضیح دهید.

21- نتیجه اجرای دستورات زیر بر روی مقدار AX را بگویید:

```
L00: JMP L04
L01: MOV AX, 1
L02: CLC
L03: JMP L08
L04: LAHF
L05: XOR AH, 1
L06: SAHF
L07: XOR AX, AX
L08: JC L01
```

مراجع

John Uffenbeck, “The 8086/8088 Family: Design, Programming and Interfacing, Prentice Hall International, Ch. 1, pp. 1-27, 1987.