به نام خداوند بخشنده مهربان

میکروپروسسور ۸۰۸۶

استاد قرائی

تهیه کننده : حامد مظاهری

دانشگاه آزاد اسلامی - واحد تهران جنوب
دانشکده فنی

(بخش دوم)

فصل ششم

«دستورات ریاضی و منطقی»

در این فصل دستوراتی را آزمایش و بررسی می کنیم که در مجموعه دستورات ۸۰۸۶/۸۸ وجود دارند و عملیات ریاضی و منطقی را انجام می دهند. دستورات ریاضی عبارتند از جمع، تفریق، ضرب، تقسیم، مقایسه، منفی کردن، افزودن و کاستن. دستورات منطقی در برگیرنده AND، OR، XOR، NOT، شیفت ها، چرخش ها و دستورات مقایسه منطقی بنام TEST می باشد.

ما همچنین سعی می کنیم دستورات مقایسه ای رشته ای را نیز معرفی کنیم. زیرا این دستورات برای نمونه برداری و مقایسه دیتاهای جدولی خیلی مفید هستند برای مقایسه دیتاهای دو ناحیه از حافظه برای تطبیق و یا عدم تطبیق شرط کارآئی خوبی دارند.

۱-۶- دستورات جمع، تفریق و مقایسه :

در بین دستورات ریاضی هر ریزپردازنده ای دستور جمع و تفریق و مقایسه بچشم می خورد. ریزپردازنده ۸۰۸۶ نیز از این قاعده مستثنی نیست. در این بخش می خواهیم این دستورات را معرفی کنیم و نحوه دستکاری دیتا در ثبات ها و حافظه ها را نمایش دهیم.

۱-۱-۶- دستور جمع (ADD):

دستور جمع در ریزپردازنده ۸۰۸۶ صورتهای مختلفی دارد. در این بخش ما جزئیات کاربرد دستور جمع ۸ بیتی و ۱۶ بیتی بعلاوه دستور افزودن به دیتا را بررسی می کنیم. و در بخش ۳-۶ صور دیگر دستور جمع مثل جمع BCD و ASCII بررسی می شود. جدول ۱-۶ مدهای آدرس دهی مختلفی که برای جمع استفاده می شوند را نشان می دهد و از آنجائیکه متجاوز از ۱۰۰۰ دستور جمع مختلف برای ریزپردازنده ۸۰۸۶ وجود دارد بدیهی است که نمایش تمام صور آن دستورات در این جدول امکان پذیر نیست.

مطلبی که اینجا لازم است یادآوری شود، اینکه محتوای ثباتهای سگمنت را نمی توان در جمع شرکت داد. و یا یک محل از حافظه را با محل دیگر آن نمی توان جمع کرد. قبلاً هم اعلام شد که محتوای ثبات های سگمنت را فقط ممکن است با دستور MOV و PUSH و POP انتقال داد.

جدول ۱-۶- دستورات جمع

دستورات	جدول ۱-۶ دستورات جمع شرح عملیات
ADD AL,BL	AL←AL+BL
ADD CX,DI	CX←CX+DI
ADD BL,44H	BL←BL+44H
ADD BX,345FH	BX←BX+345FH
ADD [BX],AL	M[DS×10H+BX] ←M[DS×10H+BX]+AL
ADD CL,[BP]	CL←M[SS×10H+BP]+CL
ADD BX,[SI+2]	BX ←BX+M[DS×10H+SI+2]
ADD CL,TEMP	CL ←CL+M[DS×10H+TEMP]
ADD BX,TEMP[DI]	BX ←BX+M[DS×10H+TEMP+DI]
ADD [BX+DI],DL	M[DS×10H+BX+DI] ←M[DS×10H+BX+DI]+DL

۱-۱-۱-۶- جمع ثباتی :

مثال ۱-۶ یک نمونه برنامه کوچکی را ارائه می کند که دستورات جمع را برای بعضی از ثبات ها نشان می دهد. دقت کنید این قطعه برنامه محتوای DX, CX, BX را جمع می کند و حاصل آن را در AX می ریزد. همچنین دقت دارید که بعد از هر جمع ریزپردازنده محتوای ثبات پرچم را تازه می کند. این مطلب مهمی است که همواره بیاد داشته باشید که دستورات ریاضی و منطقی همیشه محتوای ثبات پرچم را تغییر می دهند.

دستور جمع از هر نوعی که باشد روی پرچمهای علامت (SIGN)، نقلی (Carry)، صفر (Zero)، نقلی کمکی (Auxilary carry) و برابری (Parity) و سررفتگی (Overflow) تاثیری می گذارد.

```
Example    6-1
           ADD    AX,BX
           ADD    AX,CX
           ADD    AX,DX
```

۲-۱-۱-۶- جمع فوری :

در مثال ۲-۶ که یک جمع ۸ بیتی را نشان می دهد، وضعیت بیتهای پرچم را نیز مشخص کرده است. در ابتدا عدد 12H درون ثبات DL قرار داده می شود سپس با 33H جمع می شود (از مد

آدرس دهی فوری استفاده می کند) و حاصل جمع درون DL قرار می گیرد. مانند هر جمعی پرچمها تغییر می کنند و حاصل تغییر در این مثال به شرح زیر خواهد بود.

S=0, A=0, C=0, Z=0, O=0, P=0

```
Example     6-2
            MOV     DL,12H
            ADD     DL,33H
```

۳-۱-۱-۶- جمع ثبات با محتوای حافظه :

فرض کنید لازم باشد در یک کاربرد خاصی محتوای محلی از حافظه را با ثبات AL جمع کنیم، مثال ۳-۶ چنین مسئله ای را نشان می دهد :

```
Example     6-3
            MOV     DI,OffsetData
            MOV     AL,0
            ADD     AL,[DI]
            ADD     AL,[DI+1]
```

نحوه عمل به اینصورت است که ابتدا آدرس دیتا توسط شبه دستور OffsetData در DI قرار می گیرد چون DI با DS آدرس دهی می کند، پس دیتا در سگمنت دیتا قرار دارد.

معمولاً همیشه دیتا در سگمنت دیتا قرار دارد مگر اینکه Offset آدرس را از BP یا SP بگیرید یا بوسیله دستور مربوط به فرم پیشوند سگمنت را عوض کنید.

۴-۱-۱-۶- جمع آرایه ای

فرض کنید رشته ای از دیتا داریم بنام ARRAY که ۱۰ بایت حافظه را اشغال کرده است با 0 تا9 نام گذاری شده است.

توسط مثال ۴-۶ می خواهیم محتوای عناصر ۳ و ۵ و ۷ را با هم جمع کنیم، ابتدا درون AL صفر قرار می دهیم و عدد ۳ را در SI می ریزیم. در اولین جمع عنصر سوم با صفر جمع می شود و در دستورات بعدی، سه عنصر مذکور با هم جمع می شوند.

```
Example     6-4
            MOV     AL,0
            MOV     SI,3
            ADD     AL,ARRAY[SI]
            ADD     AL,ARRAY[SI+2]
            ADD     AL,ARRAY[SI+4]
```

۵-۱-۱-۶- جمع با یک (INCEREMENT ADDITION)

جمع کردن مقداری با یک (INC) با هر مد آدرس دهی ممکن است بکار رود بغیر از آدرس دهی ثبات های سگمنت. جدول ۲-۶ لیست تعدادی از دستورات بعلاوه یک را نشان

می دهد. بدیهی است این تمام دستورات موجود در این زمینه نیست. در مثال ۵-۶ مشخص شده که مثال ۳-۶ را می توان با استفاده از دستور INC نیز نوشت.

جدول ۲-۶ دستورات جمع با یک (INCEREMENT)

دستورات		توضیحات
INC	BL	BL←BL+1
INC	SP	SP←SP+1
INC	BYTE PTR[BX]	یکی به محتوای حافظه M[DS×10H+BX]می افزائید
INC	WORD PTR[SI]	یکی به محتوای حافظه ۱۶بیتی M[DS×10H+BX]می افزائید

یعنی با INC DI آدرس بعدی را در DI تولید می کنیم. یک تفاوت ظریفی بین INC و سایر ADD ها وجود دارد و آن اینکه INC روی پرچم نقلی (CARRY) اثر ندارد. در صورتیکه بقیه پرچمها را متاثر می سازد.

```
Example    6-5
           MOV    DI,OffsetData
           MOV    AL,0
           ADD    AL,[DI]
           INC    DI
           ADD    AL,[DI]
```

۲-۱-۶- تفریق (SUBTRACTION)

در این قسمت راجع به دستوراتی که عمل تفریق (SUB) روی دیتاهای ۸ بیتی و ۱۶ بیتی انجام می دهند به تفصیل بحث می شود، همچنین راجع به کاستن از یک دیتا بمیزان یک واحد (DECREMENT) نیز صحبت خواهیم کرد.

در بخش ۴-۶ نیز راجع به تفریق BCD و تفریق در کد ASCII بحث خواهد شد.

در جدول ۳-۶ لیستی از دستورات تفریق (SUB) با مدهای مختلف آدرس دهی ارائه شده است. همانند جمع حدود ۱۰۰۰ دستور در زمینه تفریق نیز وجود دارد که لیست تمامی آنها کار آسانی نیست. آنچه را که نمی شود روی آنها عمل تفریق انجام داد روی محتوای ثبات های سگمنت است، همچنین امکان تفریق دو محل حافظه نیز وجود ندارد.

تفریق نیز روی تمام بیت های پرچم اثر می گذارد.

۱-۲-۱-۶- تفریق محتوای ثبات ها

مثال ۶-۶ یک برنامه ساده ای است که کاربرد دستوراتی که عمل تفریق روی ثبات انجام می دهد را نشان داده است. دقت کنید که محتوای DX, CX از ثبات BX کم می شود پس از هر

عمل تفریق ثبات پرچمها تغییر پیدا می کند یعنی مانند جمع تفریق روی همه پرچمها اثر می گذارد.

```
Example    6-6
           SUB    BX,CX
           SUB    BX,DX
```

جدول ۳-۶ دستورات تفریق

دستورات	شرح عملیات
SUB CL,BL	CL←CL-BL
SUB AX,SP	AX←AX-SP
SUB DH,6FH	DH←DH-6FH
SUB AX,0CCCH	AX←AX-0CCCH
SUB [DI],CH	M[DI] ←M[DI]+CH
SUB CH,[BP]	CH←CH-M[SS× 10H+BP]
SUB AH,TEMP	AH ←AH-M[DS×10H+TEMP]
SUB DI,TEMP[BX]	DI ←DI-M[DS×10H+TEMP+BX]

۱-۲-۱-۶- تفریق محتوای ثبات ها :

مثال ۶-۶ یک برنامه ساده ای است که کاربرد دستوراتی که عمل تفریق روی ثبات انجام می دهد را نشان داده است. دقت کنید که محتوای CX و DX از ثبات BX کم می شود پس از هر عمل تفریق ثبات پرچمها تعییر پیدا می کند یعنی مانند جمع و تفریق روی همه پرچمها اثر می گذارد.

۲-۲-۱-۶- تفریق با آدرس دهی فوری:

مثال ۷-۶ یک تفریق ۸ بیتی با آدرس دهی فوری انجام می دهد. همه پرچم ها را متاثر می کند. ابتدا عدد 22H در CH وارد می شود و در دستور بعدی با 44H وارد تفریق می شود و پس از اجرای تفریق پرچمها بصورت زیر دارای مقدار می شوند :

```
Example    6-7
           MOV    CH,22H
           SUB    CH,44H
```

Z=0, C=1, A=1, S=1, P=1, O=0

توجه دارید که پرچمهای C و A نگهدارنده دو بر یک قرض گرفته شده می باشند همانند رقم نقلی پس از هر جمع بوجود می آیند. مسلماً در عمل تفریق سررفتگی بوجود نمی آید چون اعداد از هم کم می شوند.

۳-۲-۱-۶- کاستن مقدار یک واحد از یک دیتا (DECREMENT)

کاستن به میزان یک واحد از یک دیتا با دستور DEC انجام می شود جدول شماره ۴-۶ لیست تعدادی از دستوراتDEC را نشان می دهد. مجدداً یادآور می شوم که لیست کلیه دستورات در این زمینه ممکن نیست. زیرا تعداد آنها زیاد است.

جدول ۴-۶ دستورات کاستن (DECEREMENT)

دستورات	توضیحات
DEC BH	BH←BH-1
DEC BP	BP←BP-1
DEC WORD PTR[DI]	از محلی از حافظه که کلمه ۱۶ بیتی است و در DS قرار داردDIآدرس می دهد یک واحد کاسته می شود.
DEC WORD PTR[SI+2]	از محل ۱۶ بیتی از حافظه در DSکه بوسیله SI+2آدرس دهی می شود یک واحد کاسته می شود

۴-۲-۱-۶- جمع با رقم نقلی (ADD WITH CARRY)

جمع دو دیتا بعلاوه Carry زمانی کاربرد دارد که نیاز به جمع دو عدد بیش از ۱۶ بیت باشد. جدول شماره ۵-۶ تعدادی از دستورات جمع با رقم نقلی را لیست کرده است (ADC).
شرح هر دستور در مقابل آن آورده شده است. دستور ADC نیز مانند ADD روی تمام پرچمها تاثیر می گذارد.

جدول ۵-۶- جمع دو دیتا با رقم نقلی

دستورات	شرح
ADC AL,AH	AL ←AL+AH+C
ADC CX,BX	CX ←CX+BX+C
ADC [BX],AL	M[DS×10H+BX] ←M[DS×10H+BX]+AL+C
ADC BX,[BP+2]	BX ←BX+ M[SS×10H+BP+2]+C

فرض کنید عدد ۳۲ بیتی که در ثباتهای BX, AX نگهداری می شود را بخواهیم با عدد ۳۲ بیتی دیگری که در ثباتهای DX, CX نگهداری می شود جمع کنیم. در اینگونه موارد ADC کارساز می شود. مثال ۸-۶ این عمل را انجام می دهد. دقت کنید ابتدا بخش کم ارزش دیتاها یعنی DX, BX با هم جمع می شوند سپس بخش با ارزش دیتاها با هم و با رقم نقلی احتمالی پیش آمده از جمع بخش کم ارزش جمع می گردد. و حاصل جمع ۳۲ بیتی در ثباتهای AX, BX قرار می گیرد.

```
Example   6-8
          ADD   BX,DX
          ADC   AX,CX
```

۵-۲-۱-۶- تفریق با رقم قرض گرفته شده :

دستور تفریق با رقم قرض گرفته شده بیت موجود در فلیپ فلاپ Cary را از اپرند دیتا کم می کند این بیت همان رقم قرض گرفته شده است، این دستور کاربردش زمانی است که دو عدد بیش از ۱۶ بیت را بخواهیم از هم کم کنیم. جدول ۶-۶ لیستی از دستورات تفریق با رقم قرض گرفته شده را (SBB) نشان می دهد. همانند دستور SUB این دستور نیز روی تمامی پرچمها (FLAGS) اثر می گذارد. اگر بخواهیم ۳۲ بیت دیتائی که در DI, SI قرار دارند را از ۳۲ بیت دیگر که در AX, BX قرار دارند کم کنیم. پس از اینکه ۱۶ بیت اول را کم کردیم شاید رقم قرض گرفته شده داشته باشیم ۱۶ بیتی دوم را باید با دستور SBB کم می کنیم که تاثیر رقم قرض گرفته شده برای ۱۶ بیت اولیه را هم دخالت دهد. مثال ۹-۶ را ملاحظه کنید این عمل را نشان می دهد، همانگونه که ملاحظه می فرمائید ۱۶ بیت کم ارزش را نیاز ندارد که با SBB از هم کم کنید.

جدول ۶-۶- تفریق با رقم قرض گرفته شده

دستورات	شرح
SBB AH,AL	AH ←AH-AL-C
SBB AX,BX	AX ←AX-BX-C
SBB CL,3	CL ←CL-3-C
SBB [DI],AL	M[DS×10H+DI] ←M[DS×10H+DI]-AL-C
SBB DI,[BP+2]	DI ←DI-M[SS×10H+BP+2]-C

Example 6-9

```
SUB  BX,SI
SBB  AX,DI
```

۳-۱-۶- مقایسه (COMPARISON)

دستور مقایسه (CMP)، عملاً یک تفریقی است که هیچکدام از دیتاها را تغییر نمی دهد، فقط پرچمها را متاثر می سازد، مهمترین کاربرد آن برای بررسی محتوای ثبات یا یک محل حافظه است که آیا دارای مقدار مشخصی هست یا خیر، معمولاً بدنبال این دستور یک JMP شرطی خواهیم داشت که با بررسی پرچمها تحقق پیدا خواهد کرد.

جدول ۷-۶ تعدادی از دستورات CMP را لیست کرده است که مثل دستورات جمع و تفریق از مدهای مختلف آدرس دهی استفاده می کنند. تنها مد آدرس دهی که مجاز نیست مقایسه دو محل حافظه با هم و یا مقایسه محتوای ثبات سگمنت ها می باشد.

در مثال ۱۰-۶ نشان داده شده که کاربرد CMP چگونه است. اگر محتوای AL برابر 10H باشد حاصل تفریق انجام شده صفر می شود و پرچم Z=1 خواهد شد. دستور شرطی سطر دوم (در آینده مطالعه خواهیم کرد) یک دستور شرطی روی پرچم Z است می گوید اگر Z=1 شده به محلی که "CNTEN" مشخص می کند پرش کن والا ادامه بده.

جدول ۷-۶- دستورات مقایسه

دستورات	شرح
CMP CL,BL	BL وCL را مقایسه می کند بدون اینکه محتوای هیچکدام تغییر کند.
CMP AX,SP	SP وAX را مقایسه می کند بدون اینکه محتوای هیچکدام تغییر کند.
CMP AX,0CCCH	0CCCH وAX را مقایسه می کند بدون اینکه محتوای AX تغییر کند.
CMP [DI],CH	CHوM[DS× 10+DI] را مقایسه می کند بدون اینکه محتوایCH یا حافظه تغییر کند.
CMP CH,[BP]	CHوM[SS× 10+BP] را مقایسه می کند بدون اینکه محتوایCH یا حافظه تغییر کند
CMP AH,TEMP	AHوM[DS× 10+TEMP] را مقایسه می کند بدون اینکه محتوایAH یا حافظه تغییر کند
CMP DI,TEMP[BX]	DIوM[DS× 10+BX+TEMP] را مقایسه می کند بدون اینکه محتوایDI یا حافظه تغییر کند

```
Example   6-10
          CMP    AL,10H
          JZ     CNTEN
```

اگر محتوای AL دیتائی بغیر از 10H باشد حاصل تفریق صفر نمی شود و پرچم Z برابر صفر خواهد شد، شرط تحقق نمی یابد و برنامه را ادامه می دهد.

۲-۶- ضرب و تقسیم :

ریزپردازنده های ۸ بیتی قادر نیستند عملیات ضرب و تقسیم را مستقیماً انجام دهند. بلکه باید این عملیات را بصورت برنامه ای درآورد آنوقت اجرا نمود، اما از ریزپردازنده های ۱۶ بیتی مثل ۸۰۸۶ اجرای این دستورات ممکن گردید. زیرا مدارات ضرب و تقسیم در آنها پیش بینی شد و اجرای این دستورات میسر گردید. ریزپردازنده ۸۰۸۶/۸۰۸۸ قادر است هم ضرب و تقسیم اعداد ۸ بیتی و ۱۶ بیتی و بعلاوه هم می تواند عملیات را روی دیتاهای علامت دار و بدون علامت انجام دهد. بدیهی است برنامه هائی که برای اینگونه ریزپردازنده ها نوشته می شود کوتاهتر و سریعتر از برنامه های ریزپردازنده های قبلی اجرا می شود.

۱-۲-۶- دستور ضرب (MULTIPLICATION)

دستور ضرب می تواند روی دیتاهای ۸ بیتی یا ۱۶ بیتی و همچنین روی دیتاهای علامت دار با دستور IMUL و یا روی دیتای بدون علامت با دستور MUL اجرا شود. دقت داشته باشید که اگر

دو عدد n بیتی را در هم ضرب کنیم جواب 2n بیت می شود یعنی دو عدد ۸ بیتی اگر در هم ضرب شوند جواب ۱۶ بیتی خواهد شد، و همچنین حاصلضرب دو عدد ۱۶ بیتی ۳۲ بیت خواهد شد. بعضی از پرچمها (مانند C وOV) متاثر می شوند و مابقی تغییر نمی یابند، ولی مقدار آنها نامشخص است. اگر ۸ بیت با ارزش نتیجه صفر باشد هر دو پرچم C وOV صفر خواهد بود در غیر اینصورت هر دو برابر یک (SET) می شوند، و در ضرب دو عدد ۱۶ بیتی اگر ۱۶ بیت با ارزش نتیجه صفر باشد هر دو پرچم OV, C صفر می شوند ولی اگر ۱۶ بیت با ارزش دارای مقدار باشد هر دوی آنها برابر یک (SET) می شوند. بغیر از مد آدرس دهی فوری تمام مدهای آدرس دهی که برای جمع و تفریق ذکر شد برای دستور ضرب نیز قابل اجرا هستند.

۱-۱-۲-۶- ضرب اعداد ۸ بیتی :

در ضرب اعداد ۸ بیتی اعم از اینکه علامت دار باشد یا بدون علامت، مضروب همواره در ثبات AL است. برنامه نویس فقط یک اپرند می تواند در دستور تعریف کند و آنهم مضروب فیه است، مانند دستور MUL BL. این دستور AL را در BL ضرب می کند و حاصلضرب را در AX قرار می دهد. جدول ۸-۶ لیست بعضی از دستورات ضرب ۸ بیتی را نشان می دهد.

جدول ۸-۶- دستورات ضرب ۸ بیتی

دستورات	شرح
MUL CL	عدد بدون علامت موجود در AL در عدد بدون علامت CL ضرب و جواب در AX قرار می گیرد.
IMUL DH	عدد علامت دار موجود در AL در عدد DH ضرب و جواب در AX خواهد بود.
IMUL BYTE PTR[BX]	عدد علامت دار موجود در AL در یک بایت عدد موجود در DS که بتوسط BX آدرس دهی می شود ضرب می شود و جواب در AX است.
MUL TEMP	عدد بدون علامت موجود در AL در عدد موجود در DS که با TEMP آدرس هدی شده ضرب و جواب در AX است.

تصور کنید که BL و CL هر کدام دارای یک عدد بدون علامت هستند و در هم ضرب شده اند و جواب در DX باشد. البته این کار با یک دستور امکان پذیر نیست. یعنی چندین دستور باید بکار گرفته شود تا مطلب بالا محقق گردد. مثال ۱۱-۶ طی قطعه برنامه این کار را انجام داده است البته در مثال مقادیر اولیه را نیز در CL, BL قرار داده است.

```
Example   6-11
          MOV   BL,5
          MOV   CL,10
          MOV   AL,CL
          MUL   BL
          MOV   DX,AX
```

اگر ضرب علامت دار انجام دهید، حاصلضرب ممکن است مثبت باشد ممکن است منفی شود اگر منفی شود باید بصورت متمم ۲ باشد. البته که هر عددی در ۸۰۸۶ اگر منفی باشد باید بصورت متمم ۲ در سیستم وجود داشته باشد. در مثال ۱۱-۶ اگر قرار بود دو عدد علامت دار را در هم ضرب کنیم فقط کد عملیاتی (OP-CODE) دستور ضرب عوض می شود، یعنی MUL باید با IMUL جایگزین می گردد.

۲-۱-۲-۶- ضرب اعداد ۱۶ بیتی :

این ضرب خیلی شبیه ضرب اعداد ۸ بیتی است. ثبات AX همواره حامل مضروب ۱۶ بیتی خواهد بود و ثبات های DX, AX دارای حاصلضرب ۳۲ بیتی خواهند بود. توجه داشته باشید که همواره ۱۶ بیت با ارزش باید در DX قرار بگیرد. جدول ۹-۴ لیست بعضی از دستورات ضرب ۱۶ بیتی را نشان می دهد.

جدول ۹-۶- دستورات ضرب ۱۶ بیتی

دستورات	شرح
MUL CX	عدد بدون علامت موجود در AXضربدر عددCXشده وجواب در DX:AXقرار می گیرد.
IMUL DI	عدد علامت دور موجود در AXدرمحتوای DIضرب می شود و حاصل در DX:AXقرار می گیرد.
MUL WORD PTR[SI]	عدد بدون علامت AXدر محتوای حافظه ای در DSکه بوسیله SIآدرس دهی می شود ضرب و جواب در DX:AX قرار می گیرد.

۲-۲-۶- دستور تقسیم (DIVISION):

دستور تقسیم مانند دستور ضرب هم روی دیتاهای ۸ بیتی و هم روی دیتاهای ۱۶ بیتی کاربرد دارد. هم دیتاها می توانند علامت دار باشند (IDIV) و هم ممکن است بدون علامت باشند (DIV). اعداد همیشه به میزان دو برابر عریض می شوند، بعنوان مثال در یک تقسیم ۸ بیتی مقسوم در یک ظرف ۱۶ بیتی قرار خواهد گرفت و برای تقسیم ۱۶ بیتی مقسوم در یک ظرف ۳۲ بیتی قرار می گیرد. دستور تقسیم نیز همانند دستور ضرب از مد آدرس دهی فوری نمی تواند استفاده کند. وضعیت هیچکدام از پرچم ها تعریف نشده است. اگر در تقسیم خارج قسمت بیش از حد بزرگ شود سیستم اعلام خطا می کند. یا اگر عددی تقسیم بر صفر شود، یک اینتراپت در سیستم اتفاق می افتد بنام تقسیم بر صفر (DIVIDED-BY-ZERO). در آینده بیشتر راجع به اینتراپت صحبت خواهد شد.

۱-۲-۲-۶- دستور تقسیم ۸ بیتی :

همانطور که ذکر شد، مقسوم ۸ بیتی باید در ظرف ۱۶ بیتی قرار بگیرد لذا مقسوم در AX و مقسوم علیه عددی است که با یکی از مدهای آدرس دهی برای دستور انتخاب می شود. حاصل تقسیم دو عدد ۸ بیتی دو مقدار بنام خارج قسمت و باقیمانده است که خارج قسمت در AL و باقیمانده در AH قرار می گیرد. در تقسیم اعداد علامت دار همواره علامت باقیمانده مثل علامت خارج قسمت می باشد. جدول شماره ۱۰-۶ لیست بعضی از دستورات ۸ بیتی را نشان می دهد.

جدول ۱۰-۶- دستورات تقسیم ۸ بیتی

دستورات	شرح
DIV CL	عدد بدون علامت موجود در AX بر CL تقسیم می شود و خارج قسمت در AL و باقیمانده در AH قرار می گیرد.
IDIV BL	عدد علامت دار درون AX بر BL تقسیم و خارج قسمت در AL و باقیمانده در AH قرار می گیرد.
DIV BYTE PTR[BP]	عدد بدون علامت ۸ بیتی موجود در AX بر یک بایت موجود در SS که با BP آدرس دهی می شود تقسیم و خارج قسمت در AL و باقیمانده در AH خواهد بود

در یک تقسیم ۸ بیتی عدد ۸ بیتی باید به ۱۶ بیت تبدیل شود. یک دستور در ریزپردازنده وجود دارد بنام CBW که یک بایت را به WORD تبدیل می کند. تبدیل توسط این دستور انجام می گیرد. اگر یک عدد علامت دار در AL داشته باشیم وقتی تبدیل به ۱۶ بیت می شود بصورت ۱۶ بیتی علامت دار در AX قرار می گیرد. این دستور همواره قبل از تقسیم ۸ بیتی بکار می رود. مثال ۱۲-۶ قطعه برنامه ای را نشان می دهد که عدد بدون علامت 12H را بر 3 تقسیم می کند. توجه دارید که دستور CBW در این مثال بکار نرفته است. به جای آن از دستور MOV AH,0 استفاده شده است. اگر از دستور تبدیل هم استفاده می شد در AH مقدار 00H قرار می گرفت. پس از اجرای تقسیم عدد 04H در AL و 00 در AH خواهد بود

```
Example    6-12
           MOV    AL,12H
           MOV    CL,3
           MOV    AH,0
           DIV    CL
```

۲-۲-۲-۶- دستور تقسیم ۱۶ بیتی :

تقسیم ۱۶ بیتی نیز مثل ۸ بیتی است با این تفاوت که مقسوم ۱۶ بیتی به ۳۲ بیت تبدیل می شود و بخش با ارزش تر درون DX و کم ارزشتر درون AX خواهد بود. بعد از اجرای دستور خارج قسمت در AX و باقیمانده در DX قرار خواهد گرفت.

جدول ۱۱-۶ بعضی از دستورات مهم تقسیم اعداد ۱۶ بیتی را نشان می دهد. همانگونه که در تقسیم ۸ بیتی یک دستور وجود داشت تا عمل تبدیل بایت را به ۱۶ بیت انجام دهد، در این جا نیز دستور CWD عمل تبدیل ۱۶ بیت به ۳۲ بیت را انجام می دهد (CONVERT WORD TO DOUBLE WORD)

جدول ۱۱-۶- دستورات تقسیم ۱۶ بیتی

دستورات	شرح
DIV CX	عدد بدون علامت موجود در DX:AX برCXتقسیم می شود و خارج قسمت درAXوباقیمانده درDXخواهد بود
IDIV SI	عدد علامت دار موجود در DX:AXبرSIتقسیم و خارج قسمت در AXوباقیمانده درDX است.
DIV DATA	عدد موجود در DX:AXبریک دیتا۱۶بیتی در حافظه که DATAنشان می دهد تقسیم و خارج قسمت درAXو باقیمانده درDXخواهد بود.

مثال ۱۳-۶ تقسیم دو عدد علامت دار را نشان می دهد که عدد 100- را در AX و عدد 9 را در CX قرار داده است. از دستور CWD نیز استفاده شده است.

```
Example    6-13
           MOV   AX,-100
           MOV   CX,9
           CWD
           IDIV  CX
```

۳-۶- دستورات ریاضی ASCII, BCD

ریزپردازنده ۸۰۸۶ قادر است که در هر دو کد BCD (Binary Code Decimal) و ASCII (American Standard Code for Information Interchange) عملیات ریاضی انجام دهد و نتیجه را در کد مربوطه اعلام نماید.

کاربرد عملیات در کد BCD در مواردی است که محاسبات کمی دارند، مثل ترمینالهای فروش فروشگاهها. اما کد ASCII در سیستمهائی کاربرد دارد که برای نمایش کاراکترها و دیتا از کد ASCII استفاده می کنند و دیتا بصورت ASCII در سیستم ذخیره شده و پردازش می شود.

۱-۳-۶- دستورات عملیات BCD

چهار دستور در عملیات BCD وجود دارد و آنها عبارتند از :

الف- DAA(Decimal Adjast after Adition) تنظیم دیتا در پایه ۱۰ بعد از انجام عمل جمع

ب- DAS(Decimal Adjast after Subtraction) تنظیم دیتا در پایه ۱۰ بعد از انجام عمل تفریق

ج- AAM(Adjust result of BCD Multiplication) تنظیم حاصلضرب بعد از عمل ضرب

د- AAD(Adjust befor BCD Division) تنظیم دیتا در کد BCD قبل از عمل تقسیم

سه دستور AAM, DAS, DAA بعد از اینکه عمل جمع یا تفریق یا ضرب انجام شد کاربرد دارند اما دستور AAD قبل از عمل تقسیم کاربرد دارد. در عملیات جمع و تفریق دیتاهائی که باید در جمع یا تفریق شرکت کنند دو رقم در هر بایت در AL دخیره می شوند، که به این عمل دو رقم در بایت کدBCD فشرده شده (Packed BCD format) می گویند. بعنوان مثال اگر یک بایت شامل 34H باشد و آنرا بصورت کد BCD نمایش دهند دارای دو رقم 3 و 4 خواهد بود که این از نظر ارزش با 34H متفاوت است.

در تقسیم و ضرب، اعداد بصورت یک رقم در بایت ذخیره می شوند که بصورت کد BCD فشرده شده نیست. بعنوان مثال اگر در AL مقدار 01 باشد یعنی معادل 1 کد BCD است. که در بایت 01 نوشته می شود، بنام کد BCD فشرده نشده (Unpacked BCD Format) می گویند.

۱-۱-۳-۶- دستور DAA :

این دستورالعمل پس از جمع دو عدد در کد BCD برای تبدیل حاصل جمع به همان فرم BCD بکار می رود این دستور در صورتیکه 4 بیت کم ارزش AL بیش از ۹ باشد و یا AF=1، ۶ واحد به آن می افزاید. سپس ۶ واحد به چهار بیت بالائی AL خواهد افزود البته به شرطی که مقدار آن از 9 بزرگتر باشد و یا CF=1 باشد، به مثال ۱۴-۶ توجه کنید.

```
Example   6-14
          MOV   AL,47H     ;  AL=01000111
          ADD   AL,38H     ;  AL=47H+38H=7FH        درBCDمعنی ندارد
          DAA              ;  F+6=15 ⇒  5 & 7 + 1=8 ⇒ 85
```

چون رقم اول بیش از 9 است با 6 جمع می شود Carry حاصله با رقم بعدی جمع خواهد شد. مثال شماره ۱۵-۶ را دقت کنید ازآنجائیکه دستور DAA فقط روی AL عمل تنظیم دیتا را انجام می دهد. لذا یک انتقال از BL به AL داریم. عیناً مثل BH به AL لازم به ذکر است که دستور DAA روی پرچمها اثر می گذارد.

```
Example   6-15
          MOV   DX,1234H
          MOV   BX,3099H
          MOV   AL,BL
          ADD   AL,DL
          DAA
          MOV   CL,AL
          MOV   AL,BH
          ADC   AL,DH
          DAA
          MOV   CH,AL
```

۲-۱-۳-۶- دستور DAS

این دستور تنظیم دهدهی پس از تفریق است. شبیه DAA عمل می کند. فقط عمل تنظیم را روی AL انجام می دهد. همانگونه که DAA بعد از یک دستور جمع اعمال می شد این دستور بعد از عمل تفریق اجرا می شود. مثال ۱۶-۶ عملاً مثل مثال ۱۵-۶ کار می کند.

نحوه عمل به این شکل است که اگر چهار بیت اول AL بزرگتر از 9 باشد و یا AF=1 عدد 6 را از رقم چهار بیت اول کم می کند و همچنین اگر ۴ بیت بالاتر AL رقمی بیش از 9 داشته باشد و یا CF=1 باشد عدد ۶ را از رقم چهار بیت بالا کم می کند.

```
Example   6-16
          MOV   DX,1234H
          MOV   BX,3099H
          MOV   AL,BL
          SUB   AL,DL
          DAS
          MOV   CL,AL
          MOV   AL,BH
          SBB   AL,DH
          DAS
          MOV   CH,AL
```

لازم به ذکر است که تمام پرچمها را متأثر می کند ولی OF نامشخص است.

۳-۱-۳-۶- دستور AAM

دستور AAM بعد از ضرب دو عدد تک رقم در کد BCD که بصورت BCD فشرده نشده است بکار می رود مانند مثال ۱۷-۶ اگر AL=05 و CL=05 باشد، ثبات AX شامل عدد 0019H می شود حال باید با دستور AAM آنرا بصورت 0205 یا بصورت 2 رقم BCD فشرده نشده مثل 25 درآید.

در واقع AAM فقط AL را اصلاح می کند و هر رقم بزرگتر از 9 در AH ذخیره می شود.

```
Example   6-17
          MOV   AL,5
          MOV   CL,5
          MUL   CL
          AAM
```

۴-۱-۳-۶- دستور AAD

دستور AAD قبل از دستور DIV بکار می رود در صورتیکه سه دستور قبلی بعد از عمل جمع و تفریق و ضرب بکار می رفتند و حاصل عملیات را بصورت BCD در می آوردند. این دستور ثبات AX را به دو رقم BCD فشرده نشده تبدیل می کند که با ارزش ترین رقم در AH و کم ارزش ترین در AL قرار می گیرد.

پس ابتدا AAD بکار می رود پس از آن دستور تقسیم مورد استفاده قرار می گیرد و خارج قسمت در AL و باقیمانده در AH قرار می گیرد. مثال شماره ۱۸-۶ چگونگی تقسیم عدد 72در کد BCD را بر 9 نشان می دهد. که چگونه عدد 8 بدست می آید. عدد 72 در AX ریخته می شود و عدد 9 هم در BL قرار می گیرد. دستور AAD عدد 0702H را به 0048H تبدیل می کند (توجه دارید که این دستور عدد BCD فشرده نشده را به باینری تبدیل می کند) دستور DIV دیتای باینری (48H) را بر 9 تقسیم می کند خارج قسمت در AL و باقیمانده در AH قرار می گیرد.

```
Example     6-18
            MOV     AX,0702H
            MOV     BL,9
            AAD
            DIV     BL
```

۲-۳-۶- دستور محاسباتی در کد ASCII

دستورات محاسباتی ASCII بدستوراتی اطلاق می شود که روی دیتاهائی که در کد ASCII هستند عمل پردازش انجام دهند و نتیجه را نیز در کد ASCII ارائه نمایند. رنج دیتاهائی که در کد ASCII هستند عبارت است از 30H تا 39H و برای نمایش ارقام دسیمال 0 تا 9 مورد استفاده قرار می گیرند. دو دستور داریم که روی داده های در کد ASCII کاربرد دارند، اول دستور AAA (Adjust for Ascii Addition) و دوم دستور AAS (Adjust for Ascii Subtraction) این دستورات همیشه روی ثبات AX بمنزله منبع اطلاعات قبل از اعمال دستور لازم و همچنین مقصد اطلاعات تنظیم شده عمل می نمایند.

۱-۲-۳-۶- دستور AAA

اولاً این دستور روی پرچمهای CF, AF اثر می گذارد و مابقی نامشخص می باشند، ثانیاً این دستور بعد از یک دستور ADD که دو رقم اسکی را جمع کرده بکار می رود که عمل جمع دیتا در کد اسکی را بدون دخالت رقم دوم که 3 است فراهم نماید. این دستور فقط ثبات AL را اصلاح می کند، بعنوان مثال اگر عدد 31H با عدد 39H جمع شود نتیجه 6AH خواهد بود. جمع دو عدد در کد اسکی مثل 31H و 39H باید جوابی هم در کد اسکی ارائه نماید. این جواب دو رقمی و معادل 10 دسیمال است. که در کد اسکی بصورت 31H و 30H خواهد بود. نکته ای که باید دقت کرد اینکه اگر حاصل جمع کمتر از 10 باشد باید AH=0 شود و اگر حاصل جمع بیش از 9 باشد باید AH=1H شود.

اگر AAA بدنبال یک دستور جمع بیاید مثال بالا را اینگونه اصلاح می کند که در AH عدد 01H و در AL عدد 00H قرار می گیرد. البته دقت دارید که هنوز این جواب در کد اسکی نیست.

براحتی مـی تـوان آن را بـه اسـکی تبدیـل کـرد. اگـر عـدد 3030H را بـه AX اضافـه کنیـم حـاصل در کد اسکی خواهد بـود. زیربرنامـه ای کـه عملیـات بـالا را انجـام مـی دهـد در مثـال شـمار ۱۹-۶ نشـان داده شـده اسـت. دقـت داریـد کـه اعـداد بـاید در AL جمـع شـوند تـا AAA آنـرا تنظیـم کنـد بعـد از اعمال دسـتور AAA محتـوای AX=0100H خواهـد شـد. جـواب دارای دو رقـم بدسـت آمـده اسـت البتـه جواب بدست آمده در کد اسـکی نیسـت. زیربرنامـه زمـانی کـامل مـی شـود کـه دسـتور جمـع ADD AX,3030H را اجرا کنیم. در نتیجه جـواب 3130H مـی شـود کـه معـادل 10 دسـیمال اسـت.

```
Example     6-19
            MOV     AX,0031H
            ADD     AL,39H
            AAA
            ADD     AX,3030H
```

۲-۲-۳-۶- دسـتور AAS

مـانند دسـتور AAA، دسـتور AAS عمـل تنظیـم AX را بعـد از یـک تفریـق در کـد اسـکی بعـهده دارد. بعنـوان مثـال تصـور کنیـد کـه عـدد 35H از عـدد 39H در کـد اسـکی تفریـق شـود، حـاصل عـدد 4 خواهد بـود. کـه نیـاز بـه تصحیـح هـم نـدارد. لـذا شـما بـاید مواظـب عـدد درون ثبـات AH باشـید. بعبـارت دیگـر اگـر 37H-38H کنیـم درون AL مقـدار 09H را خواهیـم داشـت و عـدد درون AH بـاید یکی کاسته شود، این کم کردن از AH اجازه می دهـد کـه چنـد رقـم اسـکی را از هـم کـم کنیـم.

```
Example     6-20
            MOV     AL,32H      ;   عـدد اسـکی ۲ را در ثبـاتAL مـی گـذارد
            MOV     DH,37H      ;   عـدد اسـکی ۷ را در ثبـاتDH مـی گـذارد
            SUB     AL,DH       ;   متمـم دو عـدد ۵- اسـتAL-DH=32-37=FBH
            AAS                 ;   حـال AL=5 و CF=1 می شود
```

مجـدداً اگـر قـرار اسـت بصـورت اسـکی درآیـد بـاید محتـوای AL بـا عـدد 30H جمـع شـود. یـا بـا عدد 30H با هم OR مـی شـود.

۴-۶- دستورات پایـه منطقـی :

منظور از دستورات پایـه منطقـی NOT, XOR, OR, AND اسـت. بعـلاوه اینکـه در ایـن بخـش راجـع بـه دسـتور TEST (AND مخصـوص) و NEG نـیز صحبـت خواهـد شـد. البتـه دسـتور NEG واقعـاً یـک دسـتور محاسـباتی ریـاضی اسـت امـا چـون شـبیه دسـتور NOT منطقـی عمـل مـی کنـد در ایـن بخـش مورد بحـث قـرار مـی گـیرد. کـاربرد دسـتورات منطقـی معمـولاً بـرای کنـترل یـک بیـت در نـرم افـزار سـطح پـایین (Low Level Software) مـی باشـند. مثـلاً بـرای آن کـه یـک بیـت خـاصی در یـک ثبـات را 0 یا 1 یا متمم کرد بکار مـی رونـد، تمـام دسـتورات منطقـی روی پـرچـم هـا اثـر مـی گذارنـد و رقـم

نقلی (Carry) و پرچم سررفتگی (OverFlow) را صفر می کنند بنابر پرچم ها بگونه ای تغییر می یابندکه وضعیت عمل منطقی را نشان دهند.

۱-۴-۶- دستور AND

دستور AND روی دو داده بیت با بیت متناظر مطابق جدول صحت زیر عمل می کند و حاصل AND را اگر با X نشان دهیم $X_i=A_i \wedge B_i$ و می توان گفت هر گاه یکی از ورودیها صفر باشد حاصل AND هم صفر است.

A_i	B_i	X_i
0	0	0
0	1	0
1	0	0
1	1	1

مهمترین کاربرد دستورات منطقی دستکاری یک بیت در یک کلمه است. بعنوان مثال اگر بخواهیم بیت سوم دیتائی را یا محتوای ثباتی را صفر کنیم باید ثبات یا دیتای مورد نظر را با عدد مثل 11111011، AND کرد. و حاصل را در آن ثبات ریخت تا بیت سوم آن صفر شود. این عمل را پوشاندن بیت (Bit Masking) گویند. عکس عمل فوق را یعنی قرار دادن یک '1' در بیت خاصی (Bit Inserting) می گویند. در شکل شماره ۱-۶ یک نمونه از عمل پوشاندن بیت را نشان می دهد. توجه دارید منظور از X یعنی هرچه باشد صفر یا یک وقتی که با 1 ، AND شود تغییری نمی کند ولی اگر با صفر AND حاصل صفر خواهد شد.

دیتای ناشناخته ای	X	X	X	X	X	X	X	X
الگوی پوشاننده	0	0	0	0	1	1	1	1
حاصل بدست آمده	0	0	0	0	X	X	X	X

شکل شماره ۱-۶- چگونگی پوشاندن بیت یا بیت های یک دیتا

برای عمل AND هم از هر مد آدرس دهی بغیر از ثبات های سگمنت و AND کردن حافظه با حافظه مجاز است. جدول ۱۲-۶ لیست بعضی از دستورات AND آورده شده است.

یک دیتا در کد ASCII توسط عمل AND به آسانی قابل تبدیل به BCD می باشد. با پوشاندن ۴ بیت با ارزش هر عددی در کد ASCIIآن عدد به معادل BCD آن تبدیل می شود مثال شماره ۲۱-۶ را دقت کنید.

جدول شمار ۱۲-۶ دستورات AND

دستورات	شرح
AND AL,BL	AL ←AL∧BL
AND CX,DX	CX ←CX∧DX
AND CL,33H	CL ←CL∧33H
AND DI,4FFFH	DI ←DI∧4FFFH
AND AX,[DI]	AX ←AX∧M[DS×10H+DI]
AND Arry[SI],AL	AL با محلی از حافظه که آدرس آن برابر DS×10H+Array+SI باشد AND می شود و حاصل در همان محل ریخته خواهد شد

```
Example     6-21
            MOV   DX,3135H
            MOV   AX,BX
            AND   AX,0F0FH
```

دقت کنید حاصل برنامه در ثبات AX بصورت 0105 خواهد بود که دو رقم BCD هستند.

۲-۴-۶- دستور OR

دستور OR در واقع جمع منطقی است. که کمی با جمع ریاضی فرق می کند و آن فرق در این است که در دستور OR حاصل زمانی صفر می شود که هر دو متغیر صفر باشند، در غیر اینصورت حاصل یک خواهد بود. لذا حاصل از جدول صحت زیر تبعیت می کند که A و B ورودیها و حاصل یا خروجی X می باشد.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

کاربرد مهم OR مانند AND است با اندکی تفاوت زمانی که بخواهیم یک بیت از دیتا را یک کنیم می توانیم آن دیتا را با عددی که همه بیت های آن صفر است و فقط آن بیت متناظر یک است OR کنیم و در آن ظرف بریزیم عمل مورد نظر انجام می شود به این عمل اعمال یک '1' در دیتا گفته می شود (Bit Inserting). شکل ۲-۶ عمل اعمال یک بیت یا چند بیت درون دیتائی را نشان می دهد.

دیتای ناشناخته ای	X	X	X	X	X	X	X	X
الگوی پوشاننده	0	0	0	0	1	1	1	1
حاصل بدست آمده	X	X	X	X	1	1	1	1

شکل شماره ۲-۶- اعمال چهار بیت ۱ درون لغت

دستور OR نیز می تواند با هر مد آدرس دهی مورد استفاده قرار بگیرد. بغیر از ثبات سگمنت و OR کردن دو محل حافظه در بقیه موادر مثل AND عمل می کند.

فرض کنید دو عدد را در کد BCD ضرب کنید بعد با دستور AAM در کد BCD تنظیم نمائید حال اگر بخواهیم آنرا در کد ASCII نمایش دهید با دستور OR خیلی کار آسان می شود

مثال ۲۲-۶ جدول ۱۳-۶ تعدادی از دستورات OR را آورده که می توانید ملاحظه فرمائید :

جدول شمار ۱۳-۶ دستورات OR

دستورات	شرح
OR AH,BL	AH ←AH∨BL
OR SI,DX	SI ←SI∨DX
OR DH,A3H	DH ←DH∨A3H
OR SP,990DH	SP ←SP∨990DH
OR DX,[BX]	DX ←DX∨M[DS×10H+BX]
OR DATE[DI+2],AL	M[DATE+DS×10H+DI+2] ←M∨AL

Example 6-22

```
MOV   DX,05
MOV   BL,07
MUL   BL
AAM
OR    AX,3030H
```

۳-۴-۶- دستور XOR

دستور OR انحصاری (EXCLUSIVE-OR) مطابق جدول صحت زیر عمل می کند :

$$X=A\oplus B$$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

با دقت به جدول صحت آن متوجه انحصاری شدن آن می شوید. انحصار در حالت 1 و 1 می باشد که در XOR برابر صفر می شود. از طرفی به آن دروازه نامتعادل گویند زیرا وقتی ورودی های آن نابرابر باشند خروجی اش یک خواهد بود، و هنگامیکه ورودیها برابر باشند خروجی صفر می شود.

جدول شماره ۱۴-۶ بعضی از دستورات XOR را آورده است. بدیهی است که تمام دستورات نیست، XOR نیز همانند OR, AND برای تغییر یک بیت انتخاب شده در یک دیتا کاربرد دارد اگر بیت معینی با صفر XOR شود هیچ تغییری نمی کند، و اگر با یک XOR شود متمم می شود. شکل ۳-۶ چگونگی متمم کردن بیت خاص در یک لغت را نشان می دهد :

جدول شمار ۱۴-۶ دستورات XOR

دستورات	شرح
XOR CH,DH	CH ←CH ⊕ DH
XOR SI,BP	SI ←SI ⊕ BP
XOR AH,0EEH	AH ←AH ⊕ 0EEH
XOR SI,00DDH	SI ←SI ⊕ 00DDH
XOR BX,[SI]	BX ←BX ⊕ M[DS×10H+SI]
XOR DATE[DI+2],AL	M[DATE+DS×10H+DI+2] ←M⊕AL

ملاحظه می فرمائید که چهار بیت سمت راست معکوس می شوند چون دیتا با چهار عدد یک XOR می شود، و بقیه بیت ها تغییر نمی کنند چون با صفر XOR می شوند :

دیتای ناشناخته ای	X	X	X	X	X	X	X	X
الگوی پوشاننده	0	0	0	0	1	1	1	1
حاصل بدست آمده	X	X	X	X	X	X	X	X

شکل شماره ۳-۶- متمم کردن بیت های مورد نظر با XOR

فرض کنید ۱۰ بیت اول ثبات AX را بخواهیم متمم کنیم و مابقی بیت ها بدون تغییر بمانند یک دستور XOR بصورت زیر مورد نیاز است :

XOR AX,03FFH

در واقع ده بیت اول الگو یک هستند و مابقی صفر طبق خواسته ما عمل می کنند، یعنی ۱۰ بیت متمم می شوند و مابقی بدون تغییر می مانند.

۶-۴-۴- دستور TEST

دستور TEST عمل AND را روی دو دیتای داده شده پیاده می کند، اما با AND فرقی دارد و آن این است که AND هم روی دیتا و هم روی پرچمها اثر می گذارد ولی دستور TEST فقط پرچمها را متأثر می سازد و هیچکدام از اپرندها تغییر نمی کنند. در دستور TEST هم مثل AND ثبات های سگمنت نمی توانند اپرند باشند و از تمام مدهای آدرس دهی که AND استفاده می کرد می تواند استفاده نماید.

دستور TEST اغلب بهمان سبکی که دستور CMP بکار برده می شود مورد استفاده قرار می گیرد. اما در این دستور بررسی یک بیت آسان تر صورت می گیرد. بعنوان مثال TEST یک انتخاب منطقی برای بررسی بیت سمت راست ثبات AX می باشد، زیرا اگر از دستور TEST AX,1 استفاده کنید بیت سمت راست AX با یک مقایسه می شود اگر پرچم Z=1 شود بیت سمت راست AX برابر صفر است و در غیر اینصورت بیت سمت راست AX برابر یک است.

۶-۴-۵- دستورات NOT, NEG

عمل معکوس سازی منطقی (NOT) یا متمم یک و معکوس سازی عدد علامت دار (NEG) یا متمم دو دو دستور آخر منطقی هستند البته بجز دستورات جابجائی (Sift) و چرخشی (Rotate) این دو دستور، روی یک اپرند عمل می کنند. در جدول ۱۵-۶ دستورات NOT, NEG را نشان می دهد. برخلاف سایر دستورات منطقی این دو دستور بر روی پرچمها اثر می گذارد. دستور NOT یا متمم یک تمام بیت های یک عدد را تغییر می دهد . لذا عدد 00H را به FFH تبدیل می کند. دستور NEG یا متمم دو یک عدد دقیقاً متمم یک بعلاوه عدد یک است. اگر بخواهیم عدد 88H را تحت تأثیر دستور NEG قرار دهیم اول NOT می کنیم، 88H خواهد شد 77H و پس از آن 77H+1 مقدار 78H بدست می آید.

جدول شمار ۱۵-۶ دستورات NEG, NOT

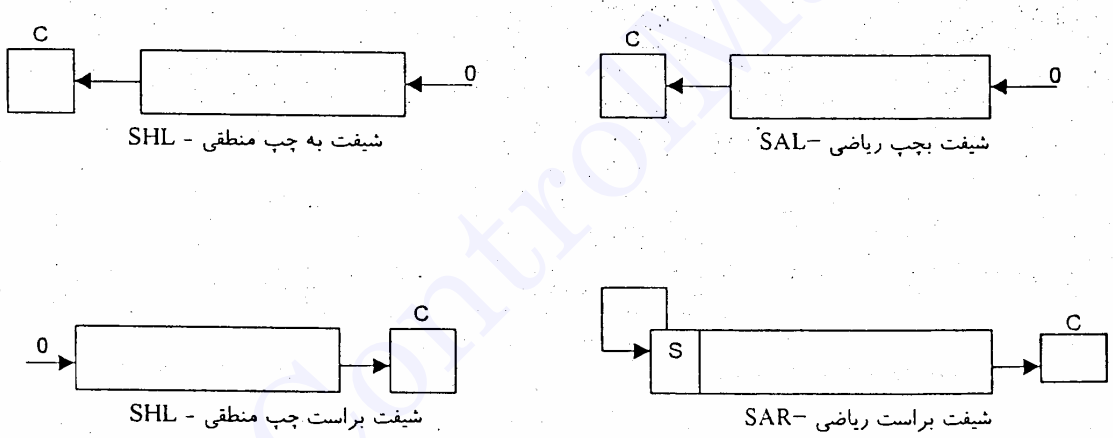
دستورات	شرح
NOT CH	CHرا متمم یک می کند.
NEG CH	CHرا متمم دو می کند.
NEG AX	مقدارAXمتمم دو می شود.
NOT TEMP	حافظه به آدرسTEMPمتمم یک می شود.
NOT BYTE PTR[BX]	یک محل حافظه به مقدار یک بایت به آدرسOffsetثباتBXمتمم یک می شود

۵-۶- دستورات جابجائی (SHIFT) یا چرخشی (ROTATE)

دستورات شیفت و چرخشی اطلاعات باینری را به اندازه یک بیت جابجا می کنند. کاربرد آنها بیشتر در کنترلهای سطح پایین ورودی و خروجی سیستم است. ریزپردازنده ۸۰۸۶ یک مجموعه کاملی از این نوع دستورات دارد که به شرح آنها می پردازیم.

۱-۵-۶- دستورات جابجائی (SHIFTS)

این دستورات اطلاعات موجود در حافظه و یا ثبات را جابجا می کنند. این دستورات همچنین قادرند عملیات ریاضی ساده مثل ضرب در 2^n (با عمل شیفت به راست) را انجام دهند، ریزپردازنده ۸۰۸۶ دارای چهار نوع مختلف شیفت است. دو نوع شیفت منطقی دارد و دو نوع شیفت حسابی (ریاضی) دارد. که چهار نوع آن در شکل ۵-۶ نشان داده شده است. توجه داشته باشید که در شکل ۵-۶ دو نوع مختلف شیفت به راست و دو نوع هم شیفت به چپ است.



شکل ۵-۶ چهار نوع شیفت (چپ و راست – ریاضی و منطقی)

ملاحظه می فرمائید در شیفت منطقی اعم از اینکه به چپ باشد یا به راست از سمت دیگر طرف دیتائی که شیفت داده می شود صفر وارد خواهد شد.

اما در شیفت ریاضی وقتی که به چپ شیفت داده می شود از سمت راست صفر وارد می شود اما زمانی که شیفت به راست داده می شود باید برای عدد مثبت صفر و برای عدد منفی یک وارد شود. خوشبختانه وقتی عدد مثبت است بیت علامت 0 و وقتی که عدد منفی است بیت علامت یک است. لذا می توان گفت بیت علامت را داخل ظرف سیستم سرایت می دهیم.

ضمناً دقت دارید که یک بار شیفت ریاضی بچپ عدد ضربدر دو می شود و اگر دوباره شیفت به چپ تکرار شود مجدداً ضربدر ۲ می شود، در مجموع یعنی ضربدر 2^2 می شود و لذا می گوئید با

شیفت بچپ می توان عدد را ضربدر 2^n کرد. در مورد شیفت به راست عدد مورد نظر تقسیم بر دو و با تکرار تقسیم بر 2^2 و ..... خواهد شد. جدول ۱۶-۶ دستورات شیفت ریاضی و منطقی را همراه با بعضی از مدهای آدرس دهی مجاز نشان می دهد. دقت کنید اگر در دستور شیفت ثبات CL را نیز وارد کنیم CL می تواند تعداد دفعات شیفت دادن را نگهدارد. مثال ۲۴-۶ چگونگی شیفت دادن ثبات DX را به تعداد ۱۴ دفعه نشان می دهد.

جدول شمار ۱۶-۶ – دستورات شیفت ریاضی و منطقی

دستورات	شرح
SHL BL	ثبات BL را یک بیت به سمت چپ شیفت (منطقی) می دهد
SHR AX	ثبات AX را یک بیت به سمت راست شیفت (منطقی) می دهد
SAL BYTE PTR[BX],CL	یک محل ۸ بیتی از حافظه را که BX آدرس می دهد به اندازه عدد موجود در CL شیفت به چپ(ریاضی)می دهد
SAR SI,CL	ثبات SI را به اندازه عدد موجود درون CL شیفت(ریاضی) به راست می دهد

```
Example   6-24
          MOV   CL,14
          SHL   DX,CL
```

فرض کنید محتوای AX را بخواهیم ضربدر ۱۰ کنیم. می توانیم مطابق مثال ۲۵-۶ عمل کنیم.

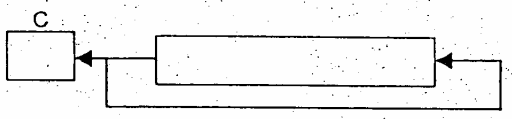
```
Example   6-26
          SHL   AX        ;  AX=2*AX
          MOV   BX,AX     ;  BX=AX
          SHL   AX        ;  AX=4AX
          SHL   AX        ;  AX=8AX
          ADD   AX,BX     ;  AX=8AX+2AX=10AX
```

۲-۵-۶- دستورات چرخشی (ROTATE)

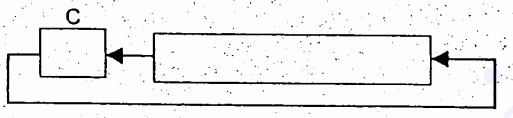
دستور چرخشی دیتای موجود در یک ثبات یا یک محل حافظه را به اندازه یک بیت جابجا می کند. و بیت خارج شده از یک طرف را به سمت دیگر وارد می کند. این عمل بدون قرار دادن فلیپ فلاپ Carry در حلقه چرخش و یا با قرار دادن آن در حلقه چرخش انجام می شود. شکل ۶-۶ هر دو صورت آن را نشان می دهد. برنامه نویس باید دقت داشته باشد که کدام فرم چرخشی را استفاده می کند.

ROL و ROR دو دستور چرخشی به چپ و راست هستند که C در حلقه چرخش نیست. البته بیت خارج شده از یک طرف علاوه بر اینکه به طرف دیگر وارد می شود به C نیز وارد خواهد شد. دو دستور RCR, RCL عمل چرخش را همراه با قرار دادن C در حلقه چرخش انجام می دهند. در دستورات چرخشی نیز ثبات CL به منزله شمارنده تعداد دفعات چرخش مورد

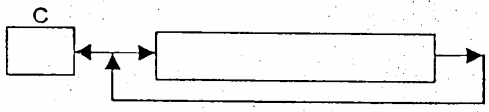
استفاده قرار می گیرد. جدول شماره ۱۷-۶ دستورات چرخشی را با بعضی از مدهای آدرس دهی نشان می دهد.



شیفت چرخشی به چپ بدون ROL-C



شیفت چرخشی به چپ با RCL-C



شیفت چرخشی به راست بدون ROR-C

شیفت چرخشی به راست با RCR-C

شکل ۶-۶ چهار نوع چرخش به راست و چپ بدون یا با فیلیپ فلاپ Carry

اگر بخواهیم عددی را که تعداد بیت های آن بیش از یک کلمه سیستم باشد شیفت چرخشی به چپ یا به راست بدهیم می توانیم از قطعه برنامه مثال ۲۶-۶ استفاده کنیم. ۴۸ بیت دیتا داریم در سه ثبات DX, BX, AX قرار دارند، این دیتای ۴۸ بیتی یک بیت شیفت داده می شود.

```
Example     6-26
            SHL     AX
            RCL     BX
            RCL     DX
```

۶-۶- دستورات مقایسه ای رشته ای (String Comparisios):

همانگونه که در فصل ۵ مشاهده کردید که دستورات رشته ای ریزپردازنده ۸۰۸۶ چقدر قوی عمل می کنند و چگونه یک بلوک دیتا را از محلی به محل دیگر حافظه منتقل می کنند، در این قسمت دو دستور دیگر که عمل مقایسه یا جستجوی دیتائی را انجام می دهند معرفی می کنیم. در واقع این دو دستور عمل جستجوی یک دیتای خاصی را در یک بخش و یا عمل مقایسه محتوای دو بخش از حافظه که آیا با هم تطبیق دارند یا خیر را انجام می دهند و این دستورات عبارتند از: SCAS = Scan String, Cmps= Compare String

۱-۶-۶- دستور SCAS

دستور SCAS، ثبات AL را با یک بلوک حافظه بصورت بایت به بایت و یا ثبات AX را با یک بلوک از حافظه بصورت کلمه (۱۶ بیتی) به کلمه مقایسه می کند به این ترتیب که عمل تفریق روی AL و حافظه یا AX و حافظه انجام می دهد بدون اینکه تأثیری روی ثبات AX یا AL و حافظه داشته باشد. اگر AL را با یک محل حافظه ۸ بیتی مقایسه کنید SCASB بکار می رود ولی اگر AX را با یک کلمه ۱۶ بیتی مقایسه کند SCASW مورد استفاده قرار می گیرد. دقیقاً مثل STOS, LODS, MOVS که در فصل ۵ مطالعه شد.

دستور SCAS نیز از پرچم D (Direction Flag) جهت خود افزایشی یا خود کاهشی ثبات های SI و DI استفاده می کند. همچنین اگر از پیشوند REP استفاده شود این دستور نیز قابل تکرار است. باز یادآور می شوم که DI همیشه دیتا را در بخش اکسترا (ES) آدرس دهی می کند و SI در بخش دیتا (DS) آدرس دهی می کند.

فرض کنید بخواهید یک بلوک دیتا به حجم ۱۰۰ بایت را که شروع آن Block می باشد را بررسی که آیا دارای 00H هست یا خیر ؟ قطعه برنامه شماره ۲۷-۶ چگونگی این جستجو یا بررسی را نشان می دهد. یادآور می‌شوم که بایت یا کلمه‌ای باید scan شود باید در اکسترا سگمنت و offset ثبات DI باشد. DI بطور اتوماتیک با توجه به DF تغییر یا کاهش می‌یابد.

Example 6-27

```
MOV     DI,OffsetBlock
CLD
MOV     CX,100
MOV     AL,0
REPNE   SCASB
```

در برنامه بالا زمانی عمل مقایسه را متوقف می کند که یا CX=0 باشد یا محلی از حافظه بیاید که دارای مقدار 00H باشد.

اینگونه دستورات (رشته ای) می توانند با پیشوند REPE یا REPNE, REPZ, REPNZ بکار روند. در مثال ۲۷-۶ عمل SCAS بوسیله تفریق انجام می شود و اگر حاصل تفریق صفر شود پرچم Z=1 می شود یعنی دو دیتا مساوی بوده اند پس هم REPNE و هم می توان REPNZ قرار داد.

۲-۶-۶- دستور CMPS

دستور CMPS همواره دو محل حافظه را یا بصورت بایت به بایت یا بصورت کلمه به کلمه (۱۶ بیتی) با هم مقایسه می کند. برای حالت تطبیق داشتن دو مقدار یا تطبیق نداشتن دو مقدار هیچ تأثیری روی محلهای حافظه نمی گذارد، فقط پرچمها(Flags) متأثر می شوند. البته عمل مقایسه در اینجا هم با تفریق دو مقدار صورت می گیرد. لازم به ذکر است که دو محل حافظه

در این دستور نیز توسط DI در اکستراسگمنت (ES) و SI در دیتا سگمنت (DS) آدرس دهی می شوند و با بکار گرفتن پرچم D جهت افزایش یا کاهش آنها مشخص می شود. همانند دستور SCAS با پیشوندهای مذکور می توان این دستور را تکرار کرد. مثال ۲۸-۶ دو بلوک حافظه را با هم مقایسه می کند در صورتیکه اختلافی مشاهده کرد متوقف می شود.

```
Example    6-28
           MOV    SI,OffsetLine
           MOV    DI,OffsetTable
           MOV    CX,10
           CLD
           REP    CMPSB
```

سؤالات دوره ای فصل ۶

۱-اشکال دستور ADD CL,AX چیست؟

۲-آیا ممکن است CX را با DS جمع کنیم؟

۳-اگر AX=1001H و DX=20FFH باشد پس از اجرای دستور ADD AX,DX مطلوبست محتوای هر یک از ثباتهای DX, AX و پرچمها؟

۴-دستوری انتخاب کنید که DX و BX را با هم جمع کنید و محتوای فیلیپ فلاپ Carry را به آن بیافزاید؟

۵-اشکال دستور INC [BX] چیست؟

۶-قطعه برنامه ای بنویسید که محتوای ثبات BP, SI, DI را از محتوای ثبات AX کم کند و حاصل تفریق را در BX ذخیره کند.

۷-دستوری که بتواند عدد یک را از محتوای BL کم کند کدام است؟

۸-وقتی که دو عدد ۸ بیتی ضرب می شوند حاصلضرب در چه ثباتی است؟

۹-فرق بین دستور MUL و IMUL چیست؟

۱۰- یک قطعه برنامه بنویسید که مکعب عدد ۸ بیتی ای که ثبات DL می باشد را محاسبه کند، بعنوان تمرین فرض کنید DL=5 باشد، تحقیق کنید حاصلضرب عدد ۱۶ بیتی می شود.

۱۱- اگر عمل تقسیم انجام دهیم خارج قسمت در چه ظرفی خواهد بود؟

۱۲- چه دستوراتی بکار می روند تا حاصل عملیات در کد BCD تصحیح شود؟

۱۳- قطعه برنامه ای بنویسید که سه بیت سمت چپ DH بدون تغییر سایر بیت ها صفر شوند.

۱۴- شرح دهید اختلاف دستور AND و TEST چیست؟

۱۵- فرق بین دو دستور NOT و NEG در چیست؟

۱۶- نقش پرچم D چیست؟

۱۷- پیشوند REPE چه عملی انجام می دهد.

۱۸- چه شرطی باعث ادامه و توقف SCASB REPNE می شود؟

۱۹- قطعه برنامه ای بنویسید که در یک بلوک دیتا که با LIST شروع می شود جستجو کند آیا عدد 66 وجود دارد یا خیر ؟

۲۰- عددی ۱۶ بیتی در خانه ۱۰۰۰ حافظه قرار دارد که شماره (آدرس) پورتی است، برنامه ای بنویسید که این عدد را خوانده و اطلاعات خانه های 20000 تا 200FF را روی آن پورت بنویسید.

۲۱- برنامه بنویسید که عدد موجود در خانه 1000 حافظه را که یک عدد باینری است به معادل BCD آن تبدیل و در خانه های 10001 و 10002 قرار دهد.

۲۲- برنامه بنویسید که دو عدد ۳۲ بیتی را که در خانه های 1000:0000~0003 و 1000:0004~0007 می باشد جمع کند و نتیجه را در 1000:0008~000B ذخیره کند.

فصل هفتم

«دستورات کنترل برنامه PROGRAM CONTROL INSTRUCION»

مقدمه :

کامپیوتر زمانی ارزش واقعی خود را پیدا می کند که دستورات کنترل برنامه ای داشته باشد که بتواند آنرا در حین اجرای برنامه راهنمایی کند. در غیر اینصورت باید قدم به قدم به اجرای برنامه بپردازد. اما با این گونه دستورات بطور اتوماتیک از یک بخش برنامه به بخش دیگری از برنامه بدون دخالت اپراتور منتقل می شود و حتی می تواند مجدداً به بخش قبلی برگردد. در این فصل سعی می شود تمام دستورات کنترل برنامه ای معرفی شوند. این دستورات را JUMP, CALL, RETUR, INTRUPT می نامند.

۱-۷- دستورات گروه پرش (JUMP GROUP)

اصلی ترین دستورات کنترل برنامه ای دستورات پرش (JMP) هستند. که اجازه می دهند کنترل ماشین از یک بخش برنامه به بخش دیگری از برنامه پرش کند. یعنی از یک دستور به n دستور بعد یا قبل پرش نماید. و پرش های شرطی اجازه می دهند که برنامه ریز بر اساس یک داده یا نتیجه ای تصمیم گیری نماید. در این قسمت ما راجع به تمام دستورات JMP صحبت خواهیم کرد و کاربرد آنها را تحت عنوان نمونه برنامه هائی نشان خواهیم داد. همچنین سعی می کنیم راجع به دستورات LOOP نیز مرور دیگری داشته باشیم.

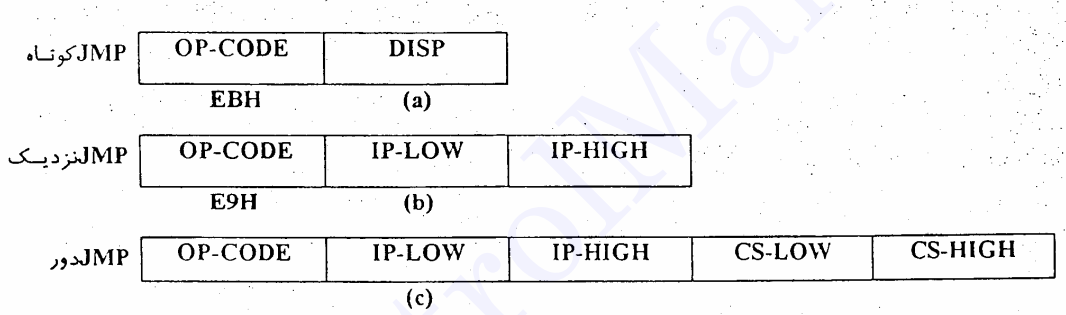
۱-۱-۷- دستورات پرش غیر شرطی (JMP)

با توجه به شکل ۱-۷ در ریزپردازنده ۸۰۸۶ سه نوع JMP وجود دارد. و آن سه نوع عبارتند از JMP کوتاه (Short JMP)، JMP نزدیک (Near JMP) و JMP دور (Far JMP). JMP کوتاه یک دستوری است که دارای دو بایت است که اجازه می دهد برنامه نویس به اندازه 127+ یا 128- محل حافظه را پرش نماید. JMP نزدیک دارای سه بایت است اجازه می دهد کنترل ماشین به هر محلی از همان کد سگمنت که در آن وجود دارد پرش نماید. و بالاخره JMP دور دارای ۵

بایت طول می باشد و اجازه می دهد که کنترل ماشین به هر محلی از حافظه که برنامه نویس بخواهدپرش نماید. معمولاً به JMP کوتاه و نزدیک و یک پرشهای داخل سگمنتInteraSegment و به JMP دورInterSegmentگفته می شود.

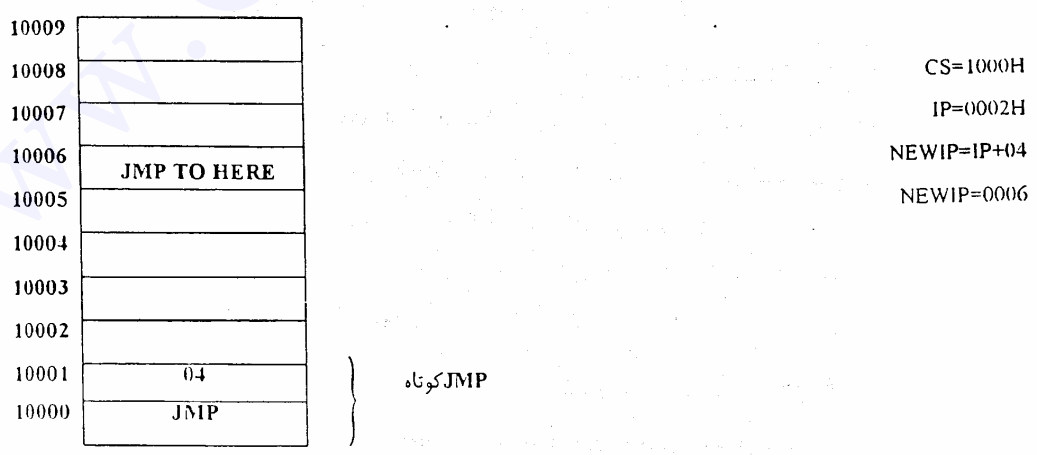
۱-۱-۱-۷- پرش کوتاه (ShortJMP)

دستور JMP کوتاه را معمولاً پرش (JMP) نسبی نیز می گویند زیرا محل پرش از همان جائی محاسبه می شود که دستور JMP را از حافظه خوانده ای. لذا به اندازه ۱۲۷ محل بعد یا به اندازه ۱۲۸ محل به قبل می تواند برگردد. در واقع آدرس با OP-CODE ذخیره نمی شود بلکه عددی ذخیره می شود (یک بایت) که می تواند مثبت یا منفی باشد. این عدد با IP جمع می شود و IP جدید را برای خواندن دستورالعمل جدید ارائه می نماید.



شکل ۱-۷- انواع دستور پرش : (a) پرش کوتاه، (b)پرش نزدیک، (c)پرش دور

اگر ریزپردازنده یک دستور JMP کوتاه اجرا نماید. مطابق شکل ۲-۷ محتوای محل بعد از کد عملیاتی (DISP) به مقدار IP اضافه می شود. تا آدرس جدید را تولید کند.



شکل ۲-۷- نمایش یک پرش کوتاه (ShortJMP) به ۴ محل بعد از دستور بعدی

آدرس جدید تولید شده همان آدرس JMP است که باید از آنجا به اجرای برنامه بپردازد. مثال ۱-۷ نیز چگونگی استفاده از این دستور را در برنامه نمایش می دهد. در این مثال همچنین نحوه استفاده از Label جدید را آورده است.

```
Example     7-1
    START:  MOV   AX,1
            ADD   AX,BX
            JMP   NEXT
            :
            :
    NEXT:   MOV   BX,AX
            JMP   START
```

در زبان اسمبلی برچسبی (Label) که در محلی بکار می رود، دقیقاً آدرس آن محل حافظه است. دو راه برای معرفی Label برای زبان اسمبلی ۸۰۸۶ وجود دارد.

۱-برچسبی که بیانگر Label نزدیک است.

۲-برچسبی که بیانگر Label دور است.

برچسب نزدیک برای ShortJMP و NearJMP مورد استفاده قرار می گیرد. برچسب دور ر برای FarJMP کاربرد دارد. فرق برچسب نزدیک با برچسب دور در این است که برچسب نزدیک به ستون (COLON) : ختم می شود مثل LABEL:, START:, NEXT: نمونه هایی از برچسب نزدیک هستند. ولی برچسب دور دارای : نیست این مطلب وجه تمایز دو برچسب است. بخاطر داشته باشید که فقط : زمانی بعد از یک برچسب قرار می گیرد که بمنزله آدرس مورد استفاده قرار گرفته است، نه وقتی که بصورت یک اپرند آورده می شود. دستور JMP LABEL ایجاد خطا می کند، زیرا اینجا LABEL یک اپرند است.

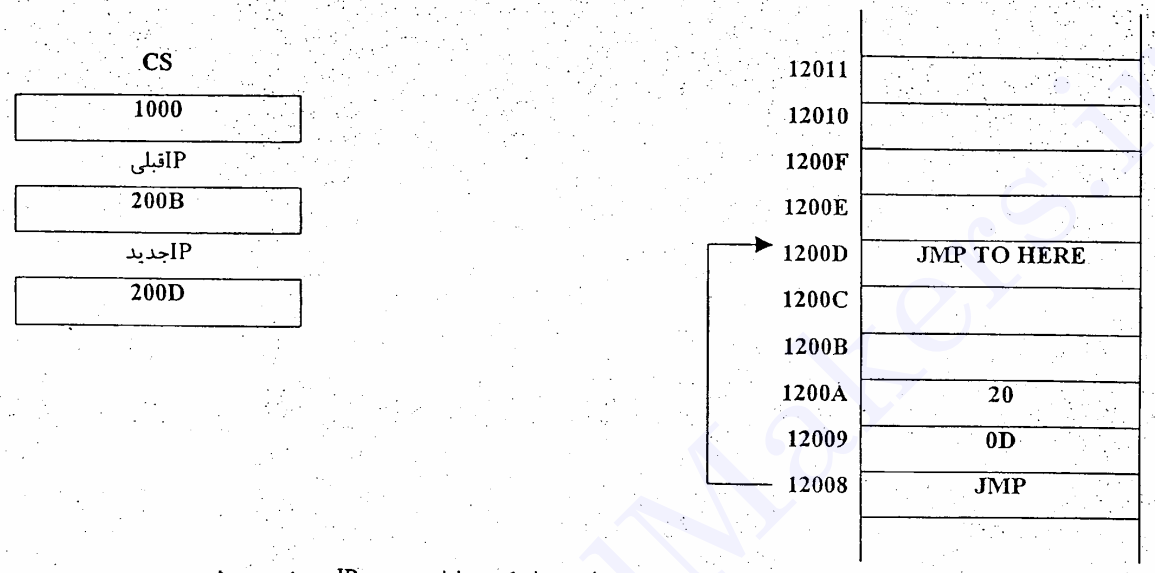
۲-۱-۱-۷- پرش نزدیک (NEAR JMP)

پرش نزدیک هم شبیه پرش کوتاه است با این تفاوت که فاصله پرش بیش از قبلی است یعنی در این پرش دو بایت فضای بعد از کد عملیاتی می تواند هر آدرسی را در همان کد سگمنت قبلی ارائه نماید. وقتی که دستور JMP اجرا شود دو بایت بعد از کد عملیاتی درون IP ریخته می شود. شکل ۳-۷ نحوه اجرای پرش نزدیک را نشان می دهد.

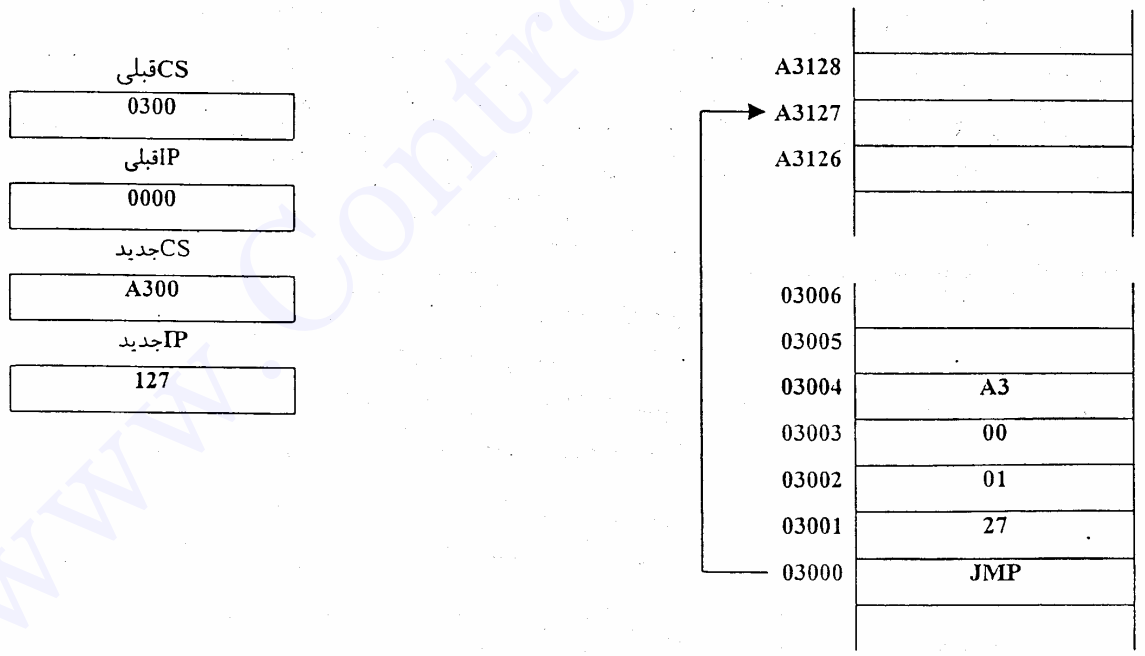
۳-۱-۱-۷- پرش دور (FAR JMP)

در پرش دور، پرش به هر محلی از حافظه امکان پذیر است زیرا دستور ۵ بایت طول دارد هم سگمنت جدید را معرفی می کند و هم IP جدید را ارائه می نماید. شکل ۴-۷ چگونگی اجرای

آن را نشان مـی دهـد. برخـلاف پـرش کوتـاه و نزدیـک در پـرش دور از بـرچسـبی اسـتفاده مـی شـود که : نـدارد.



شکل ۳-۷- یک پرش نزدیک محتوی دو محل بعد از کد عملیاتی درون IP ریخته می شود.



شکل ۴-۷- یک پرش دور که هم محتوای IP و CS را عوض می کند.

۴-۱-۱-۷- پرش غیر مستقیم ثباتی

در دستور پرش یک ثبات می تواند بعنوان اپرندی که آدرس را داراست استفاده شود. برای این دستور یک آدرس دهی غیر مستقیم برای پرش نزدیک استفاده شده است. آدرس مورد نظر درون ثباتی که در مقابل دستور JMP قرار می گیرد. مثل JMP AX محتوای چنین ثباتی دورن IP ریخته خواهد شد. این نوع پرش وقتی که در JMP TABLE استفاده می شود کاربرد زیادی دارد، زیرا سرعت اجرا زیاد می شود. مثال شماره ۲-۷ یک جدول پرش را تنظیم کرده است. که می توان به یکی از موارد دورن جدول منتقل شد. فرض کنید محتوای ثبات BX از یک منوی دیتا دریافت شود. اپراتور عدد ۲ را می نویسد و نرم افزار آنرا از صفحه کلید می خواند و به باینری تبدیل می کند و در BX قرار می دهد. و یکی از آن کم می شود و دوباره عدد یک که محتوای BX است به SI افزوده می شود.

بعد از آن دوباره افزوده شدن در SI مقدار TABLE+2 را داریم و با دستور MOV AX,[SI] آدرسTWOدرون AX قرار می گیرد و با JMP AX محتوای AX که TWO است دورن IPریخته می شود و کنترل برنامه به آنجا منتقل می شود.

```
Exampel     7-2
            MOV   SI,OFFSET TABLE
            DEC   BX
            ADD   SI,BX
            ADD   SI,BX
            MOV   AX,[SI]
            JMP   AX
              .
              .
              .
              .
TABLE:      DW    ONE
            DW    TWO
            DW    THREE
```

۵-۱-۱-۷- پرش غیر مستقیم حافظه ای

اگر قرار باشد از پرش غیر مستقیم حافظه استفاده کنیم باید از تعاریف NEAR PTR و یا FAR PTRاستفاده کرد. مثال ۳-۷ نشان می دهد که چگونه می توان آدرس دهی غیر مستقیم را استفاده کرد. در JMP اول که آدرس ADDR1 را می دهد۴ بایت آدرس ذخیره کرده چون DDتعریف کرده است یعنی Double Word که هم CS و هم IP بعدی را خواهد داشت.

اما در JMP دوم فقط دو بایت که محتوای آن دو بایت IP بعدی خواهد بود.

```
Example     7-3
            JMP   ADDR1
              .
              .
              .
              .
              .
            MOV   DI,Offset ADDR1
            JMP   [DI]
              .
              .
              .
              .
ADDR1:      DD    LOOP        ;  Far ADDRESS
ADDR2:      DW    NEXT        ;  Near ADDRESS
```

۲-۱-۷- پرشهای شرطی (CONDITIONAL JUMP)

دستورات JMP روی پرچمها اثر نمی گذارد. تمام پرشهای شرطی از نوع پرش کوتاه (Short JMP) خواهند بود و حداکثر میزان پرش آنها 127+ یا 128- خواهد بود. شرایط پرش های شرطی را به سه گروه می توان دسته بندی کرد:

۱-پرش با بررسی پرچمها

۲-پرش با مقایسه اعداد بدون علامت

۳-پرش با مقایسه اعداد علامت دار

۱-پرش که روی پرچمها عمل می کند به پرچم خاصی اشاره می کند، اگر مقدار آن یک (TRUE) شد عمل پرش به آن آدرس داده شده انجام می شود در غیر اینصورت به دستور بعد از پرش می رود. جدول ۱-۷ دستورات پرش شرطی نوع اول نشان می دهد.

جدول شماره ۱-۷- لیست دستورات پرش شرطی روی پرچمها

دستور	شرح دستور	شرایط	
JC	JUMP Carry	IF CF=1 THEN	JUMP
JNC	JUMP NoCarry	IF CF=0 THEN	JUMP
JP	JUMP Parity	IF PF=1 THEN	JUMP
JNP	JUMP NoParity	IF PF=0 THEN	JUMP
JZ	JUMP Zero	IF ZF=1 THEN	JUMP
JNZ	JUMP NoZero	IF ZF=0 THEN	JUMP
JS	JUMP Sign	IF SF=1 THEN	JUMP
JNS	JUMP NoSign	IF SF=0 THEN	JUMP
JO	JUMP OverFlow	IF OF=1 THEN	JUMP
JNO	JUMP NoOverFlow	IF OF=0 THEN	JUMP

۱-در پرش نوع دوم ابتدا دو عدد بدون علامت مقایسه می شوند و دو پرچم ZF, CF مورد بررسی قرار می گیرند. نحوه مقایسه به این شکل است که :

CMP DES(مقصد), SORCE(منبع)

سه حالت اتفاق می افتد که بصورت زیر قابل بیان است :

CF	ZF			
0	0	مقصد	$<$	مبدأ
0	1	مقصد	=	مبدأ
1	0	مقصد	$>$	مبدأ

با توجه به نتیجه حاصل دستورالعمل های شرطی آن در جدول شماره ۲-۷ آورده شده است:

جدول شماره ۲-۷- لیست دستورات شرطی که بعد از مقایسه دو عدد بدون علامت می آیند

دستور	شرح دستور	شرایط
JA	JUMP ABOVE	IF CF=0 AND Z=0 THEN JUMP
JAE	JUMP ABOVE OR EQUAL	IF CF=0 THEN JUMP
JB	JUMP BELOW	IF CF=1 THEN JUMP
JBE	JUMP BELOW OR EQUAL	IF CF=1 OR ZF=1 THEN JUMP
JE	JUMP EQUAL	IF ZF=1 THEN JUMP
JNE	JUMP NOTEQUAL	IF ZF=0 THEN JUMP

۳- در پرش نوع سوم که در آن شرط به مقایسه اعداد علامت دار ارجاع داده می شود. در حالت مقایسه اعداد علامت دار، اگر چه از همان دستور CMP DES,SOURCE استفاده می شود ولی پرچمهای بکار رفته برای اعلام نتیجه بقرار زیرند :

مبدأ > مقصد Z=0 یا OF=SF

مبدأ = مقصد ZF=1

مبدأ < مقصد OF معکوس SF

در نتیجه دستورالعمل های پرش شرطی جداگانه ای بکار می رود که به شرح زیر خواهد بود :

جدول شماره ۳-۷- لیست دستورات شرطی که بعد از مقایسه دو عدد علامت دار می آیند

دستور	شرح دستور	شرایط
JG	JUMP Greater or Equal	IF ZF=0 OR OF=SF THEN JUMP
JGE	JUMP Greater or Equal	IF OF=SF THEN JUMP
JL	JUMP Less	IF OF≠SF THEN JUMP
JLE	JUMP Less or Equal	IF ZF=1 or SF≠OF THEN JUMP
JE	JUMP Equal	IF ZF=1 THEN JUMP

یک دستورالعمل پرش شرطی دیگری هم وجود دارد به صورت زیر است :

JCXZ IF CX=0 THEN JUMP

ممکن است سؤال شود اگر بخواهیم پرشی به بیش از محدوده 127+~128- داشته باشیم چه باید کرد؟ بدیهی است باید از دستورات پرش غیر شرطی استفاده کرد. فرض کنید از دستور رشته ای SCASB برای پیدا کردن عدد 0AH در یک بلوک بطول ۱۰۰ بایت استفاده کنیم. (مثال شمار ۴-۷) در دو صورت از حلقه جستجو خارج می شود یکی اینکه دیتای مورد نظر (OAH) را بیابد و دیگر اینکه چنین دیتائی در بلوک پیدا نکند و با CX=0 خارج شود، با استفاده از دستور JCXZ می توان مشخص کرد علت خروج کدام یک از دو حالت فوق بوده است.

```
Example     7-4
  SCAN:     MOV     DI,OFFSET TABLE
            MOV     CX,100
            MOV     AL,0AH
            CLD
            REPNE   SCANSB
            JCXZ    NOT-FOUN
```

۳-۱-۷- دستور LOOP

دستور LOOP ترکیبی است از یک پرش شرطی و دستور کم کردن از CX در واقع یک واحد از CX کم می کند اگر ثبات CX صفر نشده باشد به برچسبی که در مقابل دستور LOOP نوشته شده است پرش می کند اما اگر CX=0 شده باشد به دستور بلافاصله بعد از LOOP رفته و اجرای برنامه ادامه پیدا می کند.

مثال شماره ۵-۷ چگونگی جمع کردن دیتاهائی از یک بخش حافظه (BLOCK1) را با دیتاهائی در بخش دیگر حافظه (BLOCJ2) را نشان می دهد و بکار گرفتن دستور LOOP برای کنترل تعداد جمع ها می باشد.

```
Example     7-5
            MOV     CX,100
            MOV     SI,OFFSET BLOCK1
            MOV     DI,OFFSET BLOCK2
  AGAIN:    LODSW
            SEG ES:
            ADD     AX,[DI]
            STDSW
            LOOP    AGAIN
```

نکته دیگری که در مثال ۵-۷ قابل توجه است تغییر سگمنت است که توسط پیشوند SEG ES: آورده شد، در واقع جمع AX با دیتائی است در اکستراسگمنت که DI آدرس آنرا می دهد.

۱-۳-۱-۷- دستور حلقه شرطی

دستور LOOP صور دیگری نیز دارد که به آنها (Conditional Loop)حلقه شرطی نیز می گویند مثل LOOPNE, LOOPE اولی می گوید حلقه ایجاد کن اگر مساوی هستند یعنی اگر CX برابر صفر نشده و شرط تساوی برقرار است به آدرس داده شده مقابل دستور LOOP می رود در غیر اینصورت از LOOP خارج شده و به دستور بلافاصله بعد از LOOP رفته و برنامه ادامه پیدا می کند. دستور LOOPNE عکس قبلی عمل می کند.

دستور دیگری که در این زمینه وجود دارد LOOPZ است البته همانند دستورات قبلی LOOPNZ نیز وجود دارد، ناگفته نماند که LOOPE همانندLOOPZ و LOOPNE همانند LOOPNZ می باشد.

۲-۷- زیربرنامه (SUBROUTINE)

زیربرنامه یک بخش خیلی مهم هر ساختار نرم افزاری یک کامپیوتر محسوب می شود. یک مجموعه دستوراتی که یک عمل مشخصی را انجام می دهند بصورت زیربرنامه در می آورند. اهمیت آن از این نظر است که ممکن است شما چندین بار این قسمت را بکار بگیرید، یک بار نوشتن این بخش و چند بار استفاده کردن مسلم است که نقش مهمی خواهد داشت. هم حافظه

صرفه جوئی می شود، هم برنامه شما ساده تر می شود. زیربرنامه توسط دستوری بنام CALL مورد خطاب قرار می گیرد و کنترل ماشین توسط دستور RET از زیربرنامه به برنامه اصلی برمی گردد. هنگام رفتن سراغ زیربرنامه از پشته (STACK) جهت ذخیره کردن آدرس محل برگشت به برنامه اصلی استفاده می شود. تا با اجرای دستور RET به دستور بلافاصله بعد از CALL برگردد.

در زبان اسمبلی ۸۰۸۶ قواعد محدودی برای ذخیره کردن زیربرنامه وجود دارد که ذیلاً ذکر می شود. اول اینکه مثل زبان های سطح بالا (بعنوان مثال PASCAL) زیربرنامه را رویه (PROCEDURE) می نامند.

مثال ۶-۷ دو نوع رویه نزدیک و دور را نشان می دهد (Far Procedure-Near Procedure). اگر دستور CALL زیربرنامه ای را صدا بزند که در همان کد سگمنت باشد آنرا رویه نزدیک می گویند. اما اگر رویه ای که دستور CALL آنرا مورد خطاب قرار داده در کد سگمنت دیگری باشد آنرا رویه دور می گویند.

به مثال فوق دقت کنید دو شبه دستور (Pseudo-Opcode) مشاهده می شود که یکی PROC و دیگری ENDP است. PROC برای نشان دادن محل شروع زیربرنامه است که توسط اسمی که برای زیربرنامه انتخاب می کنید برچسب زده می شود. و نوع زیربرنامه بعد از PROC (نزدیک یا دور بودن آن) آورده می شود.

```
Example        7-6
NAME    PROC    FAR          ;   این دستور اسم زیربرنامه و نوع آن و شروع آنرا مشخص می کند
        MOV         AX,BX
        ADD         AX,CX
        RET                  ;   این دستور برگشت به برنامه اصلی است
NAME    ENDP                 :   این دستور به اسمبلر می گوید اینجا پایان زیربرنامه تعریف شده است
        .
        .
        .
LABEL   PROC    NEAR         :   شروع زیربرنامه با نام LABEL و از نوع نزدیک است
        MUL         BX
        MUL         CX
        MUL         DX
        RET
LABEL   ENDP                 :   پایان زیربرنامه LABEL
```

در مثال بالا دو نوع زیربرنامه یکی از نوع نزدیک و یکی از نوع دور معرفی شده است، چگونگی شروع و پایان آنها نیز مشخص گردیده است. اغلب زیربرنامه ها که بخشی از کتابخانه سیستم محسوب می شوند باید بصورت رویه دور (Far Procedure) نوشته شوند. بعلاوه هر JMP درون هر زیربرنامه باید از نوع کوتاه (Short) باشد. برای اینکه بتوانند آنرا براحتی جابجا کنند. زیرا پرش کوتاه است که این خاصیت را دارد. و می توان آنرا به هر برنامه ای اضافه کرد. در یک زیربرنامه عمومی، تمام ثبات هائی که در زیربرنامه بکار گرفته می شوند باید در پشته ذخیره شوند و در انتهای زیربرنامه قبل از برگشت باید از پشته برداشته شوند.

۱-۲-۷- دستورات CALLS

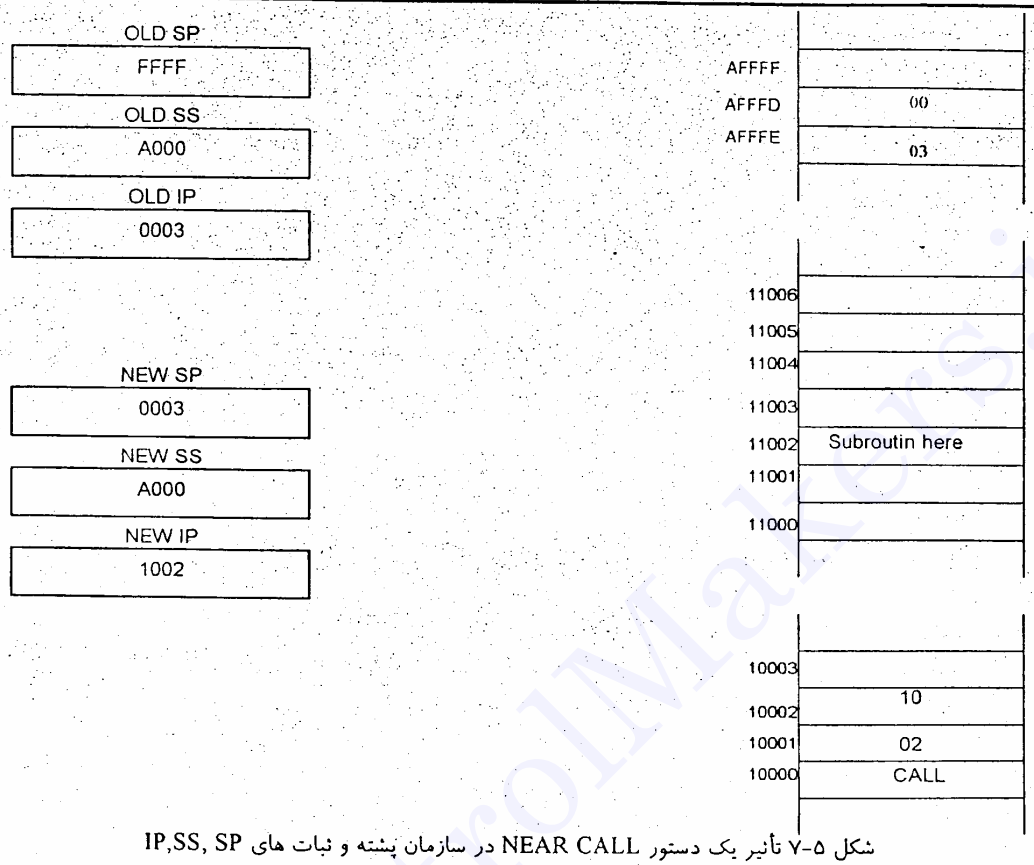
دستور CALL کنترل ماشین را از برنامه به یک زیربرنامه منتقل می کند. و این انتقال با انتقالی که در دستور JMP داشتیم متفاوت است، زیرا در اینجا باید محتوای IP را در پشته ذخیره کند (اگر CALL از نوع دور باشد باید IP و CS را در پشته ذخیره کند) آنوقت عمل انتقال صورت بگیرد. هیچگونه تأثیری روی پرچمها ندارد.

۱-۱-۲-۷-دستور NearCall

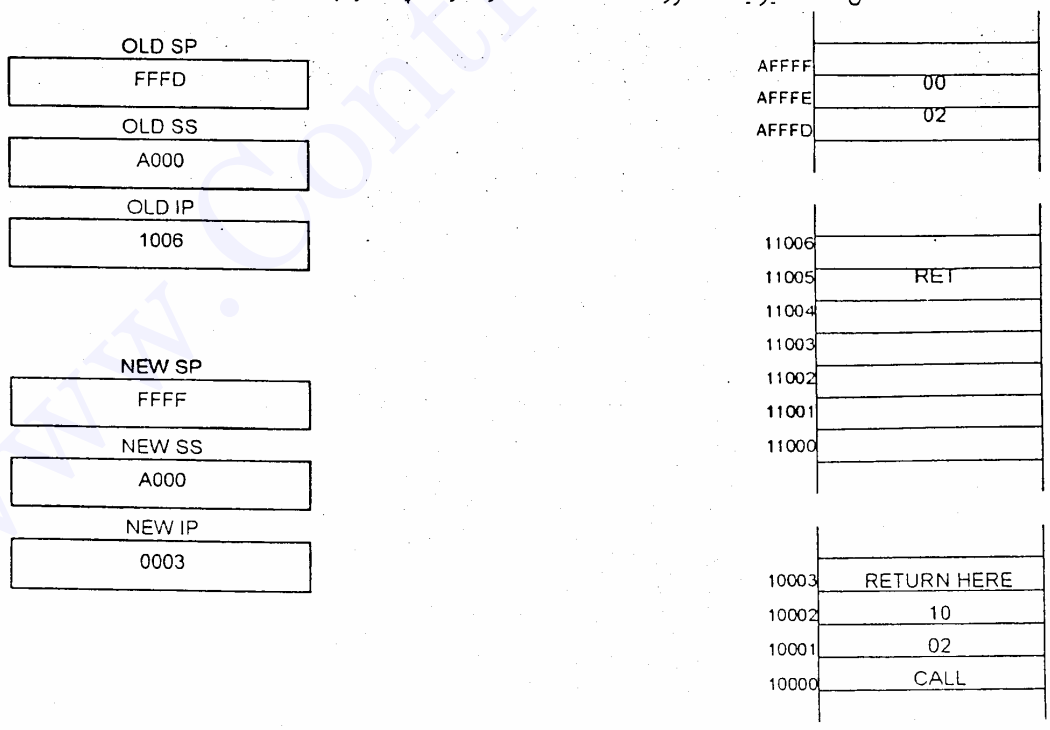
دستور CALL نزدیک ۳ بایت است، بایت اول طبق معمول OP-CODE و بایت دوم و سوم حاوی آفست آدرس محل زیربرنامه می باشد لذا پس از ذخیره کردن آدرس دستور بلافاصله بعد از CALL در سازمان پشته بایت دوم و سوم بعد از OP-CODE، درون IP ریخته خواهد شد. ذخیره کردن آدرس برگشت توسط عمل PUSH در سازمان پشته صورت می گیرد. شکل ۵-۷ نشان می دهد که چگونه آدرس برگشت ذخیره شده و عمل پرش به زیربرنامه صورت گرفته است.

۲-۱-۲-۷- دستور FAR CALL:

دستور FAR CALLشبیه FAR JUMP است زیرا امکان صدا زدن زیربرنامه در هر محلی از حافظه را فراهم می کند. این دستور ۵ بایت طول دارد که بایت اول OP-CODE و بایت دوم و سوم دارای محتوای IP هستند و بایت چهارم و پنجم کد سگمنت جدید را در بر دارند که باید به ثبات CS ریخته شوند. هنگام اجرای دستور FAR CALL باید آدرس دستور برگشت هم IP و هم CS در پشته ذخیره می شوند.



شکل ۵-۷ تأثیر یک دستور NEAR CALL در سازمان پشته و ثبات های IP,SS, SP



شکل شماره ۶-۷- چگونگی اجرای دستور RET و بازگشت به برنامه اصلی

۳-۱-۲-۷- دستور CALL با مدهای آدرسی دهی مختلف

سه روش برای نوشتن آدرس رویه NEAR و دو روش مختلف برای آدرس دهی FAR وجود دارد که ذیلاً به شرح آنها پرداخته می شود :

الف- مستقیم : مانند

```
CALL     PROC1
 ---      ---
 ---      ---
PROC1    PROC     NEAR
 ---      ---
 ---      ---
          RET
PROC1    ENDP
```

ب- غیر مستقیم ثباتی، مانند

```
         CALL   [SI]
```

که انتقال به آدرسی که در SI می باشد صورت می گیرد. مثال ۷-۷ چگونگی استفاده از ثبات را نشان می دهد، در این مثال زیر برنامه ای صدا زده می شود که شروع آن در محلی است بنام COMPUTER

ج- غیر مستقیم حافظه ای

```
CALL     WORD  PTR[DI]
```

در این روش عمل انتقال به محلی صورت می گیرد که آدرس آن محل در حافظه است و بوسیله DI محل آدرس در حافظه مشخص می شود. این روش در رویه FAR نیز وجود دارد که در معرفی باید DWORD آورد که ذکر خواهد شد.

```
Example      7-7
             MOV    SI,OFFSET COMPUTE
             CALL   SI
               :      :
               :      :
COMPUTE      PROC     NEAR
             PUSH     DX
             MOV      DX,AX
             IN       AX,DATA
             OUT      PORT,AX
             MOV      AX,DX
             POP      DX
             RET
COMPUTE      ENDP
```

مثال شماره ۸-۷ چگونگی آدرس دهی غیر مستقیم حافظه ای را نشان می دهد. در اینجا فرض شده DI دارای عدد یک یا دو یا سه است که هر کدام را به 0 ,2 ,4 تبدیل و با SI جمع

می کند. تولید آدرس می کند، در واقع از تکنیک LOOK UP TABLE استفاده می کند که TABLE این مثال دارای سه آدرس برای سه رویه مختلف است به یکی از آنها دسترسی پیدا می کند.

```
Example     7-8
  TABLE:    DW      ONE
            DW      TWO
            DW      THREE
              :       :
              :       :
```

ابتدا TABLE LOOKUP را تعریف می کند ;

عمل تبدیل یک یا دو یا سه به 0 یا 2 یا 4 انجام می شود و یکی از سه رویه صدا زده می شود

```
            DEC     DI
            MOV     SI,OFFSET TABLE
            ADD     DI,SI
            CALL    [DI+SI]
              :       :

ONE         PROC    NEAR
              :       :
            RET
ONE         ENDP
;
TWO         PROC    NEAR
              :       :
            RET
TWO         ENDP
;
THREE       PROC    NEAR
              :       :
            RET
THREE       ENDP
```

زیربرنامه اول

زیربرنامه دوم

زیربرنامه سوم

اما روش های آدرس دهی برای FAR CALL که قبلاً ذکر شد دو روش به شرح زیر می باشد

الف- مستقیم ولی خارج از قطعه، مثل

```
            CALL    PROC1
              :       :

PROC1       PROC    FAR
              :       :
            RETF
PROC1       ENDP
```

ب – غیر مستقیم حافظه ای : این روش شبیه غیر مستقیم حافظه برای NEAR CALL است با این تفاوت که در دستور باید Dword (Double Word) ذکر گردد. مثال :

```
CALL        DWORD     PTR[DI]
```

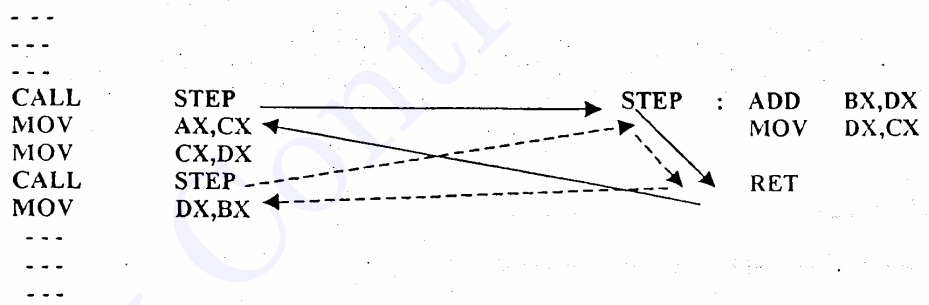
هنگام اجرای دستور کنترل ماشین به محل CS:IP منتقل می شود که مقدار IP آن از محلی از حافظه که DI و DI+1 آدرس دهی می کند پر می شود CS آن از محلی که DI+2 وDI+3 آدرس دهی می کند پر خواهد شد.

۲-۲-۷- دستورات RETS:

این دستور روی پرچمها تأثیری ندارد و مخفف کلمه RETURN است، که معنی برگشت می دهد، اگر از نوع Near باشد ۱۶ بیت تحت عنوان OFFSET آدرس از پشته برداشته و در IP می ریزد و اگر از نوع FAR باشد ۳۲ بیت برداشته ۱۶ بیت در IP و ۱۶ بیت در کد سگمنت CS می ریزد. نوع Near یا FAR بودن آن در تعریف رویه مشخص شده است. و بصورت اتوماتیک در اسمبلر انتخاب می شود.

هنگامیکه IP یا CS, IP از پشته پر شدند آدرس دستورالعمل بعدی محل جدیدی خواهد بود که این محل جدید دستور بلافاصله بعد از CALL است که به این زیربرنامه آمده بود.

شکل ۷-۷ چگونگی صدا زدن یک زیربرنامه بوسیله CALL و چگونگی برگشت از زیربرنامه توسط دستورRET را نشان می دهد.



شکل ۷-۷- یک قطعه برنامه کوتاه همراه با زیر برنامه که در برنامه اصلی صدا زده شده و پس از اجرای زیربرنامه به برنامه اصلی مراجعه می کند.

دستور برگشت دیگری داریم که به محتوای SP عددی را می افزاید، آنوقت IP را از محلی که SP نشان می دهد پر می کند تا به برنامه اصلی برگردد که بصورت RET n نمایش داده می شود. N همان عددی است که به SP اضافه می شود.

مثال ۹-۷ دستور RET 4 را نشان می دهد که چگونه عمل می کند. این دستور عدد 4 را به SP اضافه می کند. زیرا دو دستور PUSH AX و PUSH BX چهار بایت فضای پشته را پر کرده اند و لذا AX, BX را پاک می کند فقط به POP کردن IP می پردازد البته این یک RET خاص محسوب می شود و ضعیت نرمال RET چنین نیست.

```
Example     7-9
TEST-RIG    PROC    NEAR
            PUSH    AX
            PUSH    BX
            ---     - - -
            ---     - - -
            ---     - - -
            RET     4
TEST-RIG    ENDP
```

۳-۷- وقفه INTRUPT

وقفه بمعنی ایجاد توقف در روند جاری برنامه و پرداختن به یک زیربرنامه تا اتمام آن، و پس از آن به برنامه اصلی برگشتن و روند جاری را ادامه دادن است.

وقفه ممکن است توسط سخت افزار تولید شود، ممکن است بوسیله نرم افزار ایجاد شود. در هر صورت برنامه در حال اجرا را متوقف می کند و باعث صدا زدن زیربرنامه سرویس وقفه می شود و آنرا تا آخر اجرا می نماید. درباره وقفه سخت افزاری در فصل ۹ بحث خواهد شد. در این قسمت راجع به وقفه های نرم افزاری صحبت می کنیم، یکی از این وقفه ها همان دستور CALL است که قبلاً بحث شد، انواع دستور CALL دیگر در ریزپردازنده ۸۰۸۶ وجود دارد بنام دستورات وقفه که سه نوع آن را بنام INT3, INT0, INT و بردار وقفه و دستور IRET را مورد بحث قرار می دهیم.

۱-۳-۷- بردار وقفه چیست؟ (INTRUPT VECTOR)

بردار وقفه چهار بایت حافظه ای است که در قسمت ۱۰۲۴ محل اول حافظه (یعنی از آدرس 00000H تا 003FFH) بمنظور نگهداری آدرس برنامه سرویس یک وقفه خاص انتخاب شده است. ریزپردازنده ۸۰۸۶ دارای ۲۵۶ نوع وقفه است و لذا ۲۵۶ بردار وقفه باید داشته باشد که آدرس اولین دستور برنامه سرویس وقفه را نگهداری کنند. هم وقفه های سخت افزاری و هم وقفه های نرم افزاری دارای چنین برداری هستند. بردارهای وقفه و توابع آنها و آدرس برنامه هر کدام در جدول ۴-۷ آورده شده اند. هر بردار ۴ بایت طول دارد که دو بایت حاوی مقداری که باید درون IP و دو بایت مقداری که باید درون ثبات CS ریخته شود

۲-۳-۷- دستورات وقفه :

ریزپردازنده ۸۰۸۶ دارای سه نوع مختلف وقفه است. INT3, INT0, INT هر کدام از این سه نوع، یک بردار را از جدول بردارهای وقفه می خوانند و پس از آن به زیربرنامه مورد نظر که

Intrupts and interrupt service pro Codures

اینترایپت ها و روتین های سرویس اینترایپت در ۸۰۸۶

اغلب میکروپروسسورها اجازه می دهند برنامه عادی که در حال اجرای آن هستند توسط بعضی سیگنالهای خارجی متوقف یا بعضی از دستورات خاص برنامه این متوقف گونه

به این معنی که اجرای آن روند جاری برنامه متوقف شود و زیربرنامه یا قطعه برنامه دیگری صدا زده شود و به اجرای آن بپردازد که به آن زیربرنامه یا برنامه اینترایپت وقفه (intrupt) می گویند. پس از اتمام اجرای آن برنامه به برنامه اولی که مشغول اجرای آن بود برمیگردد. در این قسمت می خواهیم چگونگی پاسخ به اینترایپت و چگونگی اولین یک برنامه اینترایپت و چگونگی تقاضای اینترایپت توسط دستگاه متقاضی (اینترایپت دهنده) را نشان دهیم.

اینترایپت در ۸۰۸۶ به یکی از سه صورت زیر به سیستم وارد میشود.

۱- یک سیگنال از بیرون به پایه NMI (non maskable intrrupt) یا پایه INTR اعمال میشود که به آن ها اینترایپت سخت افزاری میگویند

۲- دومین راه اجرای یک برنامه اینترایپت اجرای یک دستور اینترایپت است (INT) که به این نوع اینترایپت اینترایپت نرم افزاری گویند

۳- سومین نوع اینترایپتی که به ۸۰۸۶ اعمال میشود توسط بعضی از شرایط بوجود آمده در زمان میکروپروسسور است که بعنوان مثال اینترایپت صفر را میتوان نام برد. اگر دستوری در برنامه داشته باشید که اپراندی را بر صفر تقسیم کند بطور اتوماتیک اینترایپت صفر (zero = intrrupt) فعال میشود.

البته به این نوع اینترایپت ها نیز اینترایپت نرم افزاری میگویند

در پایان هر سیکل دستورالعمل ۸۰۸۶ چک می کند که آیا تقاضای اینترایپتی داده شده یا خیر؟ اگر پاسخ مثبت باشد ۸۰۸۶ طبق قدمهای زیر عمل می کند و به اینترایپت دهنده پاسخ میدهد

۱- در دو واحد از SP کم می کند و ثبات پرچم را در حافظه پشته push می کند.

۲- پرچم اینترایپت (IF) را صفر می کند تا اینترایپت دادن غیرفعال شود

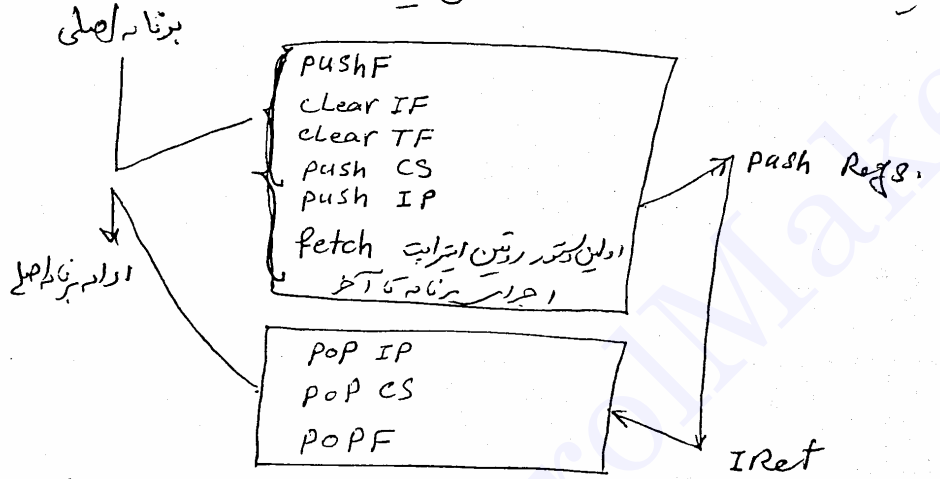
۳- پرچم تله (TF) را صفر می کند

۴- دو واحد از SP کم می کند و CS را در حافظه پشته push می کند

۵- دو واحد از SP کم می‌کند و IP را در حافظه پشته push می‌کند.

۶- یک عمل far JMP به اولین دستور برنامه‌ای که شما برای سرویس دادن به اینترایت نوشته‌اید انجام می‌دهد.

بطور خلاصه قدمهای بالا را میتوان گفت: ثبات پرچم را در پشته قرار میدهد، TF و IF را Reset می‌کند. یک دستور far call اجرا می‌کند، شکل زیر کلیه مطالب بالا را نشان میدهد.



لازم به ذکر است که در انتهای برنامه اینترایت از دستور IRet برای مراجعه به برنامه اصلی استفاده میشود.

در ۸۰۸۶ اولین یک کیلوبایت فضای حافظه (00000 ~ 003FFH) اختصاص داده شده برای ذخیره کردن آدرسهای شروع برنامه‌های سرویس‌های اینترایت ها. برای هر آدرس ۴ بایت لازم است دو بایت برای CS و دو بایت برای IP بنابراین در این فضا آدرس شروع ۲۵۶ برنامه اینترایت را ذخیره میکنند. به این جدول آدرسها اصطلاحاً بردار اینترایت (intrrupt vector) میگویند.

۵ اینترایت اولی را اینترایت‌های خاص میگویند مثل Div by zero و NMI

۲۷ ، بعد از ۵ تا ۲۱ را مشترک اینتل برای میکروهای آینده رزرو کرده است

۲۲۴ ، آخر از ۳۲ تا ۲۵۵ قابل دسترسی برای کاربر است که بصورت نرم‌افزاری

آدرسش را خواند دسترسی پیدا می کند. بعبارت دقیقتر برای اینکه وقفه اجرا شود اتفاقات زیر صورت می پذیرد :

۱-SP دو واحد کسر می شود و پرچم ها در پشته ذخیره می گردند.

۲-SP دو واحد کسر می شود و CS در پشته ذخیره می گردد

۳-SP دو واحد کم شده و IP که آدرس دستور بعد از دستور وقفه را دارد در پشته ذخیره می شود.

۴-شماره نوع وقفه را در چهار ضرب می کند تا آدرس جدول بردار بدست آید.

با شروع از این آدرس دو بایت اول را در IP و دو بایت بعد را در CS می ریزد، تا از اولین دستور برنامه وقفه شروع به واکشی دستور (Fetch) و اجرا نماید.

۵-IF و TF را پاک می کند.

جدول شماره ۴-۷- جدول بردارهای وقفه

حالت و منظور از استفاده	آدرس	شماره بردار
خطای تقسیم	0~3H	0
اجرای قدم به قدم برنامه STEP BY STEP	4H~7H	1
وقفه سخت افزاری غیر قابل پوشش (NMI)	8H~BH	2
ایجاد نقطه شکست در برنامه BREAK POINT	CH~FH	3
ایجاد وقفه سررفتگی OVER FLOW	10H~13H	4
رزرو برای آینده	14H~7FH	5~31
وقفه های استفاده کنند.	80H~3FFH	32~255

ملاحظه می کنید که دستور INT نرم افزاری شبیه دستور CALL است با این فرق که در دستور وقفه پرچمها هم در پشته ذخیره می شوند..

گفته شد که ۲۵۶ نوع وقفه داریم و هر وقفه دارای عددی است که ممکن از ۰ تا ۲۵۵ (از 00H تا FFH) باشد بعنوان مثال وقفه شماره ۱۰۰، بردار وقفه شماره ۱۰۰ را در اختیار می گیرد که هر کدام از دستورات دارای ۲ بایت طول هستند، بغیر از دستور وقفه (INT3) که استثناً دارای یک بایت می باشد، نکته دیگری که قابل ذکر است وقتی که وقفه پذیرفته شد فیلیپ فلاپ I صفر می شود این بیت کنترل کننده وقفه خارجی است (یعنی پایه INTR) یعنی پایه درخواست کننده وقفه را غیرفعال می کند. وقتی که I دارای مقدار یک منطقی است پایه وقفه فعال می شود. راجع به پرچم T بعداً در مبحث وقفه ها صحبت می کنیم.

وقفه های نرم افزاری معمولاً در سیستم زیربرنامه ای بکار می روند تا وسایل مثل CRT، چاپگر و صفحه نمایش را کنترل کنند.

دستور far CALL که معمولاً دارای ۵ بایت طول می باشد برای اینگونه توابع کاربرد دارد و دستورات وقفه دارای دو بایت طول هستند لذا از دستورات وقفه دو بایتی بجای دستورات CALL

۵ بایتی استفاده می کنند تا ۳ بایت ذخیره شود. بدیهی است که اینگونه ذخیره سازی محل های حافظه در طول یک برنامه قابل ملاحظه خواهد شد.

۳-۳-۷- دستور IRET

دستور IRET بمعنی برگشت از برنامه وقفه است. کلیه پرچم ها را متأثر می سازد (CF, PF, AF, ZF, SF, TF, IF, DF, OF) یعنی مقدار همه پرچمها همان مقداری که قبل از وقفه داشتند بار می شوند.

دستور IRET هم برای وقفه های نرم افزاری و هم برای وقفه های سخت افزاری کاربرد دارد.

هنگام اجرای این دستور :

۱-مقدار IP که در پشته PUSH شده بود POP شده و در IP ریخته می شود.

۲-مقدار CS که در پشته PUSH شده بود POP شده و در CS ریخته می شود.

۳-مقدار ثبات پرچم ها که در پشته PUSH شده بود POP شده و در ثبات پرچم ریخته می شود

در واقع می توان گفت IRET ترکیبی است از دستور RET و POPF

گفته شد با اجرای دستور وقفه پرچمهای I, T صفر می شوند، حال با برگرداندن مقدار پرچمها به ثبات پرچمها مجدداً مقدار قبلی I, T به حالت قبل از دستور INT برمی گردند یعنی اگر قبلاً مجاز بود که به سیستم وقفه وارد شود فقط در طول اجرای برنامه وقفه است که I غیر فعال می شود.

۴-۳-۷- دستور INT3

برخلاف سایر دستورات وقفه که دو بایتی هستند این دستور یک بایتی است و نام آن ایجاد نقطه توقف در برنامه است. یعنی برای رفع عیب برنامه نوشته شده در نقاط معینی دستور وقفه 3 قرار داده می شود برنامه تا آنجا اجرا می شود تا نتیجه وارسی گردد. مجدداً می توان از یک نقطه توقف تا نقطه توقف بعدی پیشروی کرد. هنگام اجرا پس از وارسی می توان یک بایت گذاشته شده را حذف کرد.

۵-۳-۷- دستور INT0

این دستور وقفه سررفتگی است (OverFlow) پرچمهای IF, TF را متأثر می کند، این دستور در صورتی اجرا می شود که فیلیپ فلاپی که برای سررفتگی در نظر گرفته ایم نیز یک شود. حال اگر هنگام اجرای برنامه OF=1 شود، CPU به مکان حافظه به آدرس 4×4=10H می رود و محتوای مکانهای 10H, 11H را درون IP و 12H, 13H را درون CS ریخته و سراغ قطعه برنامه ای

که آدرس اولین دستور آن را برداشته است می رود. پس اگر سررفتگی اتفاق نیافتد این دستور اجرا نمی شود شبیه JO می باشد (یعنی اگر OF=1 آنگاه JMP کن)

۶-۳-۷- دستور وقفه 50 (INT 50)

فرض کنید در یک بخشی از سیستم اغلب لازم باشد که محتوای BX, BP, SI, DI را جمع کنیم و حاصل جمع را رد AX قرار دهیم. چون می خواهیم این برنامه را اغلب داشته باشیم ترجیحاً آنرا بصورت وقفه بنویسیم، البته می توان بصورت زیربرنامه هم نوشت اما بصورت وقفه بهتر است مثال ۱۰-۷ این برنامه وقفه را نشان می دهد و مشخص شده که چگونه بردار وقفه آدرس برنامه را ارائه می نماید. هر وقت که نیاز به این قسمت برنامه باشد بوسیله INT50 این برنامه صدا زده می شود

```
Example      7-10
  ADDM   :   ADD    AX,BX
  ADDEM  :   ADD    AX,SI
             ADD    AX,DI
             ADD    AX,BP
             IRET
```

برقراری آدرس بردار به شرح زیر است :

```
             ORG    00088H
             DD     ADDEM          ;  شماره بردار وقفه = ۵۰
```

۷-۳-۷- کنترل وقفه

این بخش نیز شامل وقفه های سخت افزاری نمی شود. حال ضروری است که دو دستوری که کنترل کننده وقفه های سخت افزاری هستند را معرفی کنیم، اولی دستور نشاندن پرچم وقفه STI (SET INTRUPT FLAG) که پرچم I را یک می کند که اجازه به ورود پایه سخت افزاری وقفه بدهد (INTR). دوم دستور CLI است پاک کننده پرچم وقفه (CLEAR INTRUPT FLAG) که صفر در پرچم I قرار می دهد و پایه ورودی INTR سخت افزاری را غیر فعال می کند.

۴-۷- سایر دستورات کنترل ماشین

دستورات قبلی که راجع به آنها بحث شد راجع به کنترل ماشین بودند. دستوراتی را که حال می خواهیم مطرح کنیم کنترل کننده پرچم نقلی (Carry Bit)، نمونه برداری از پایه TEST یا اجرا کننده حالات مختلف دیگر در سیستم می باشند که اغلب در کنترل سخت افزاری عمل می کند

که در اینجا ما فقط نام آنها را می بریم. در مبحثی که راجع به وقفه سخت افزاری صحبت خواهیم کرد اینها بطور دقیق و کامل مورد بحث قرار می گیرند.

۱-۴-۷- دستورات کنترلی پرچم نقلی (CONTROLLING THE CARRY FLAG BIT)

پرچم نقلی در جمع و تفریق های چند کلمه ای، نشان دهنده خطا در زیربرنامه کاربر دارد. سه دستور داریم که اجازه می دهد برنامه ریز محتوای پرچم نقلی را تغییر دهد. CMC, STC, CLC که به ترتیب در فلیپ فلاپ Carry صفر و یک می گذارد و یا متمم می کند.

۲-۴-۷- دستور WAIT

این دستور ریزپردازنده را به حالت انتظار می برد. روی پرچم ها تأثیری ندارد. به این نحو عمل می کند که پایه سخت افزاری $\overline{TEST}$ را بررسی می کند، اگر مقدار آن یک باشد در حالت WAIT می ماند این حالت تداوم دارد تا در پایه $\overline{TEST}$ یک صفر ملاحظه کند، آنوقت از WAIT خارج می شود. این پایه معمولاً توسط ریزپردازنده ۸۰۸۷ که کمکی ۸۰۸۶ محسوب می شود مورد استفاده قرار می گیرد.

۳-۴-۷- دستور HLT

مخفف کلمه HALT است. این دستور موجب می شود که ریزپردازنده اجرای دستورات را متوقف کند و به حالت کما برود. و توسط INTR یا NMI یا RESET سخت افزاری از حالت HLT خارج می شود. این دستور در حالت های خاص مثلاً در انتهای یک برنامه بکار می رود.

۴-۴-۷- دستور NOP:

روی پرچم ها تأثیری ندارد. عمل فتچ را انجام می دهد IP را بهنگام می کند که آماده رفتن به دستور بعدی باشد ولی هیچ کاری انجام نمی دهد. برای تنظیم تأخیرها در اتلاف پالس های ساعت بکار می رود سه پالس ساعت زمان می برد.

۵-۴-۷- دستور LOCK

این دستور بنام پیشوند قفل گذرگاه سیستم معروف است این دستور یک بایت است که قبل از هر دستور ۸۰۸۶ قرار می گیرد تا از دسترسی ریزپردازنده دیگر موجود در سیستم به باس(CO-PROCESSOR) ممانعت بعمل آورد.

۶-۴-۷- دستور ESC (ESCAPE INSTRUCTION)

روی پرچمهای تأثیری ندارد و باعث عبور اطلاعات به پردازنده کمکی (ریزپردازنده ۸۰۸۷) می شود. در واقع استفاده از کمک پردازنده ها که باید گذرگاه های آدرس و داده را بصورت مشترک استفاده کنند را ممکن می سازد. در واقع هر وقت دستور ESC اجرا می شود ریزپردازنده ۸۰۸۶ عمل NOP را اجرا می کند. و باس ها در اختیار ریزپردازنده کمکی قرار می گیرد تا در حافظه بنویسد یا از حافظه بخواند.

۵-۷- برنامه های نمونه

در اینجا به معرفی چند برنامه نمونه می پردازیم تا با سازمان دهی برنامه برای IBM-PC ها یا نوشتن برنامه به زبان اسمبلی آشنا شوید. همچنین بعضی از تکنیک های برنامه نویسی را که کاربرد زیادی دارند معرفی می کنیم.

۱-۵-۷- اماه کردن یک برنامه برای یک فایل اجرائی (DISK OPERATING SYSTEM)

یک فایل اجرائی عملاً یک فرمان برای سیستم عامل DOSمی باشد بعنوان مثال اگر شما یک فایل اجرائی بسازید و اسم آنرا DOG.EXE بگذارید. آنوقت باید بنویسید که اسم این برنامه DOG است و باید آنرا اجرائی کنید. راه اندازی یک برنامه برای اسمبلر سیستم های IBM-PC کار آسانی نیست اگر چه ممکن است برای بعضی سیستمهای دیگر آسان باشد. هر بخش باید توضیح داده شود. بخشها مرتب شوند. مثال ۱۱-۵ یک الگوئی را ارائه می کندکه در اغلب برنامه های به زبان اسمبلی کاربرد دارد و همه زیربرنامه ها و برنامه اصلی آورده شده است.

در این مثال وقتی که اسم صدا زده می شود وارد دیسک می شود و همراه با ادیتور برنامه را طبق مراحل زیر ترجمه می کند.

۱-اسم ماکرو اسمبلر یا اسم اسمبلر (برحسب اینکه کدام یک را دارید) را می نویسند که برنامه را ترجمه کند. این عمل فایل جدیدی می سازد که همان اسم شما را با پسوند OBJ. (name.OBJ) انتخاب می کند که این برنامه قابل اجرا نیست.

۲- یک اسمی که پیونددهنده این برنامه با قسمت های دیگرش اگر وجود داشته باشد بنام LINKnameمی نویسید که برنامه را به فایل قابل اجرا تبدیل می کند و اسمی با پسوند EXE. برای آن انتخاب می کند.

۳-اسم برنامه را می نویسید تا برنامه اجرا شود (البته اسمی که پسوند EXE. دارد) شما روی صفحه نمایش (پس از گذاشتن از ۵ خط اول) چه باید ببینید؟

GARRAEE.HASSAN
WAS HERE !

۱-۱۰-۵-۷- معادل سازی برنامه :

اولین بخش هر برنامه لیستی معرفی می شود که معادل سازی برنامه نامیده می شود. این بخش بکار می رود که با استفاده از راهنمای EQU برچسب های شما را با مقادیر مختلف بکار رفته در طول برنامه هم ارز یا معادل می سازد، راهنماهای EQU دستوراتی را توصیف می کنند که هم در برنامه اصلی و هم در زیربرنامه ها مورد استفاده قرار گرفته اند و داده های را معرفی می کنند که در برنامه استفاده شده اند. هم ارزسازها (معادل سازها) انتخابی هستند ولی برنامه را خیلی راحت قابل فهم می سازند و برنامه توأم با مستندات می شود. توجه کنید وقتی انتهای پیغام را $ معرفی می کنید چقدر راحتر می شود تا اینکه بخواهیم بنویسید : MESAGE-END

یک نمونه برنامه که چند پیام را روی صفحه نمایش ارسال خواهد کرد.

```
Example          7-11
;-----------------------------------------------
;     PROGRAM    EQUATES
;-----------------------------------------------

MESAGE-END       EQU     '$'      ;     پایان پیغام هم ارز$است
        CR       EQU     0DH      ;     ENTER هم ارز 0DHست
        LF       EQU     0AH      ;     LineFeed هم ارز 0AHست
        SP       EQU     ' '      ;     SPACEمعادل یک فاصله
      ZERO       EQU       0      ;
      NINE       EQU       0      ;

;-----------------------------------------------
;        STACK SEGMENT
;-----------------------------------------------

STACK-SEG        SEGMENT  PARA     PUBLIC
                 DB       256  DUP(?) ;     ۲۵۶بایت برای پشته ذخیره می کند
STACK-SEG        ENDS

;-----------------------------------------------
;        DATA    SEGMENT
;-----------------------------------------------
DATA-SEG         SEGMENT     PARA     PUBLIC
MESAGE-ONE       DB       CR
                 DB       5  DUP(0AH)
                 DB       'GHARAEE      'HASSAN'
                 DB       CR
                 DB       LF
                 DB       MESAGE-END
MESAGE-TWO       DB       'WAS HERE?!!'
                 DB       MESAGE-END
DATA-SEG         ENDS
```

```
;-----------------------------------------------
;        CODE    SEGMENT
;-----------------------------------------------

CODE-SEG        SEMENT      PARA    PUBLIC
                ASUME       CS:CODE-SEG     DS:DATA-SEG

;-----------------------------------------------------------
;        MAIN   PROGRAM  PROCEDURE
;-----------------------------------------------------------

MAIN            PROC        FAR
                PUSH        SS
                MOV         AX,ZERO
                PUSH        AX
                MOV         AX,DATA-SEG
                MOV         DS,MESAGE-ONE
                CALL        DISPLAY-MESAGE
                MOV         DX,MESAGE-TWO
                CALL        DISPLAY-MESAGE
                RET
MAIN            ENDP

DISPLAY-MESAGE  PROC        NEAR
                MOV         AH,NINE
                INT         21H
                RET
DISPLAY-MESAGE  ENDP
CODE-SEG        ENDS
                END         MAIN
```

۲-۱-۵-۷- بخش پشته (STAK-SEGMENT)

بخش بعدی برنامه قسمت پشته است. در واقع اینجا پشته برقرار و تعریف می شود و اجازه می دهد که شما از دستورات RET, CALL و یا هر دستور دیگری در رابطه با POP, PUSH استفاده کنید.

اولین بخش در اینجا شبه عمل SEGMENT که معرفی کننده برای زبان اسمبلی است نه کد ماشین که عمل انجام می دهد. یعنی شبه عمل SEGMENT شروع را برای زبان اسمبلی تعریف می کند یعنی می گوید شروع بخش است (Paragraph Boundary=PARA)

بعلاوه کلمه SEGMENT به بخش متصل کننده اسمبلی (STACK-SEG IS PUBLIC) و لذا بخش متصل کننده می تواند با سایر بخش ها بهمان نام آنرا متصل کند دستورالعمل بعدی DB 256 DUP(?) است، که ۲۵۶ بایت برای پشته جا در نظر می گیرد.

وقتی که متصل کننده (LINKER) فایل را به یک فایل EXE. تبدیل کرد بطور اتوماتیک (SETUP)برقرار می شود. هم SP, SS مقادیر خود را دریافت می کنند و آماده دریافت اطلاعات و

تحویل آن می شوند، بخش دیگر پشته دستور ENDS می باشد که پایان پشته را اعلام می کند. البته مشاهده می کنید که برای اعلام ENDS از برچسب STACK-SEG نیز استفاده می شود.

۳-۱-۵-۷- بخش دیتا (DATA-SEGMENT)

این قسمت با نام DATA-SEG شروع می شود. هر اسمی که برنامه زیر بخواهد می تواند برای آن انتخاب کند و صدا بزند. همانند بخش پشته با اسم DATA-SEG شروع می شود و با DATA-SEG و ENDS پایان می پذیرد. برنامه زیر همچنین آزاد است که نام اکسترا سگمنت را برای آن انتخاب کند.

۴-۱-۵-۷- بخش کد سگمنت :

این بخش شامل برنامه است و امکان تغییر اسن آن وجود ندارد انتخاب شده است. کد سگمنت هم با عبارت سگمنت (SEGMENT) شروع می شود و همچنین به ENDS ختم می شود.
جمله دوم در کد سگمنت جمله ASSUME است که به اسمبلر می گوید که نامی که برای کد سگمنت انتخاب کرده اید CODE-SEGMENT است و بخش دیتا DATA-SEG است و همچنین اگر از اگسترا سگمنت هم استفاده می کنید باید آورده شود. بخش بعد از نظر نرم افزاری زیربرنامه اصلی است (MAIN PROCEDURE) معمولاً این زیربرنامه اصلی بصورت FAR انتخاب می شود و چند دستور اول این زیربرنامه برقراری شرایط برای برگشت به DOS است مثل PUSH DS زیرا که کد سگمنت DOS را دارد. و پس از آن انتقال پیغام ها به صفحه نمایش است.

۲-۵-۷- برنامه نمونه ورودی و خروجی:

این مسئله، کامپیوتر شما را مثل یک ماشین تحریر در می آورد، برنامه فوق العاده ساده ای است. به شما نشان می دهد که چگونه بخش اول برنامه ۱۱-۷ را با اندکی تغییر می توانید برای هر برنامه ای بکار بگیرید.
در این برنامه بخش هم ارزسازی فقط سه جمله است در یک برنامه بزرگ ممکن است دارای ۱۰۰ جمله هم ارز باشد. جالب است بدانید این برنامه بخش دیتا سگمنت هم ندارد. زیرا قرار نیست دیتائی در حافظه ذخیره کند یا قرار نیست دیتائی از حافظه خوانده شود. فقط قرار است یک کلید ساده خوانده شود و این عمل تکرار شود تا کارکتر $ تایپ شود. آنوقت کامپیوتر را به محیط DOS برگرداند. بطوریکه بتوانید برنامه اجرائی دیگری را، اجرا نمائید. در زبان اسمبلی پایه این برنامه را بصورت زیربرنامه ای که خواندن صفحه کلید نام دارد صدا می زند.

این برنامه از صفحه کلید اطلاعات دریافت می کند و در مونیتور نشان می دهد تا یک '$' تایپ شود.

```
;-----------------------------------------------
;        PROGRAM    EQUADES
;-----------------------------------------------

ZERO            EQU     0
TWO             EQU     2
DOLAR-SIGN      EQU     '$'

;-----------------------------------------------
;            STACK-SEGMENT
;-----------------------------------------------

STACK-SEG       SEGMENT PARA PUBLIC
                DB      256   DUP(?)
STACK-SEG       ENDS

;-----------------------------------------------
;     CODE-SEGMENT
;-----------------------------------------------

CODE-SEG        SEGMENT PARA PUBLIC
                ASSUME : CS : CODE-SEG

;-----------------------------------------------
;     MANIN PROGRAM PROCEDURE
;-----------------------------------------------

MAIN            PROC    FAR
                PUSH    DS
                MOV     AX,ZERO
                PUSH    AX

AGAIN :         CALL    READ-KEY
                CMP     AL,DOLLAR-SIGN
                JNE     AGAIN
                RET
MAIN            ENDP

READ-KEY        PROC    NEAR
                MOV     AH,TWO
                INT     21H
                RET
READ-KEY        ENDP

CODE-SEG        ENDS
                END     MAIN
```

تابع شماره ۲ در انتظار زدن دکمه کلید می ماند. کلید زده شده را می خواند و با قرار دادن آن در AL برمی گردد و آنرا نمایش می دهد.

وقتی که یک کارکتر تایپ شد برنامه اصلی زیربرنامه خواندن کلید را صدا می زند و آنرا با $ چک می کند اگر $ تایپ شده بود به DOS برمی گردد والا به زیربرنامه خواندن کارکتر دیگر از صفحه کلید بر می گردد.

۱-۲-۵-۷- نمونه برنامه ای که دو عدد را جمع می کند :

این برنامه دو عدد را از صفحه کلید دریافت می کند آنها را با هم جمع می کند و در صفحه نمایش نشان می دهد، بمراتب پیچیده تر از برنامه قبلی است، مثال شماره ۱۳-۷ را ببینید.

در این برنامه هم از بخش دیتا (DS) و هم از بخش اکسترا (ES) استفاده شده است. زیرا می خواهیم از عملیات رشته ای استفاده کنیم.

برنامه دو عدد تا ۴۰ رقم را از صفحه کلید می پذیرد. طول اولین عدد ادامه پیدا می کند تا اپراتور + را تایپ کنید، پس از آن عدد دوم شروع می شود و ادامه پیدا می کند تا اپراتور = تایپ شود. وقتی که علامت = تایپ شد. برنامه دو عدد را با هم جمع می کند (با بکار گرفتن جمع در ASCII) و حاصل جمع در یک بافری بنام ANSWER قرار داده می شود. و آنرا روی مونیتور می فرستد. فرم وارد کردن دیتا بصورت زیر است :

123 + 333 = 456

برنامه ای که دو عدد تا ۴۰ رقم را دریافت می کند و حاصل جمع را روی مونیتور نمایش می دهد :

```
;-----------------------------------------------
;       PROGRAM     EQUATION
;-----------------------------------------------

ZERO            EQU     0
TWO             EQU     2
EIGHT           EQU     8
NINE            EQU     9
FPRTY           EQU     40
CR              EQU     13
LF              EQU     10
EDM             EQU     '$'
DOS-FUNCTION    EQU     21H
NO              EQU     'N'
YES             EQU     'Y'
PLUS            EQU     '+'
EQUAL           EQU     '='
NINE-ASCII      EQU     '9'
ZERO-ASCII      EQU     '0'

;-----------------------------------------------
;       STACK     SEGMENT
;-----------------------------------------------

STACK-SEG       SEGMENT PARA PUBLIC
                DB      256     DUP(?)
STACK-SEG       ENDS
```

```
;-----------------------------------------------
;       DATA SEGMENT
;-----------------------------------------------

DATA-SEG        SEGMENT PARA PUBLIC

SIGN-ON         DB      CR,LF,LF
                DB      'ADD TWO NUMBER(Y/N)?'
                DB      EDM
NUMB1           DB      40      DUP(?)
OVER            DB      (?)
NUMB2           DB      40      DUP(?)
ANSWER          DB      41      DUP(?)
END-ANS         EQU     'THIS BYTE'
                DB      EDM

;-----------------------------------------------
;       CODE SEGMENT
;-----------------------------------------------

CODE-SEG        SEGMENT PARA PUBLIC
                ASSUME CS:CODE-SEG      DS : DATA-SEG      ES : DATA-SEG

;-----------------------------------------------
;       MAIN    PROGRAM   PROCEDURE
;-----------------------------------------------

MAIN            PROC    FAR
                PUSH    DS
                MOV     AX,ZERO
                PUSH    AX
                MOV     AX,DATA-SEG
                MOV     DS,AX
                MOV     ES,AX

TOF :           MOV     DX,OFFSET SIGN-ON
                CALL    OUT-MESEGE
AGAIN :         CALL    READ-KEY
                CMP     AL,NO
                JE      MAIN-END
                CMP     AL,YES
                JNE     AGAIN
                CALL    OUT-CHAR
                MOV     BL,PLUS
                MOV     DI,OFFSET NUMB1
                CALL    READ-NUMB
                MOV     BL,EQUAL
                MOV     DI,OFFSET NUMB2
                CALL    READ-NUMB2
;-----------------------------------------------
;       READ-KEY PROCEDURE
;-----------------------------------------------

READ-KEY        PROC    NEAR
                PUSH    DI
                PUSH    BX
                MOV     AH,EIGHT
                INT     DOS-FUNCTION
                POP     BX
                POP     DI
                RET
READ-KEY        ENDP
```

```
;-----------------------------------------------------------
;       DISPLAY CHARACTER PROCEDURE
;-----------------------------------------------------------

OUT-CHAR    PROC    NEAR
            PUSH    DI
            PUSH    BX
            MOV     AH,TWO
            MOV     DL,AL
            INT     DOS-FUNCTION
            POP     BX
            POP     DI
            RET
OUT-CHAR    ENDP

;-----------------------------------------------------------
;       READ NUMBER PROCEDURE
;-----------------------------------------------------------

READ-NUMB       PROC    NEAR
                CALL    CLEAR
                MOV     BH,ZERO
READ-NUMB1 :    CALL    READ-KEY
                CALL    CHAR-NUMB
                JC      READ-NUMB2
                PUSH    AX
                CALL    OUT-CHAR
                INC     BH
                JMP     READ-NUMB1
READ-NUMB2 :    CMP     AL,BL
                JNE     READ-NUMB1
                CALL    OUT-CHAR
                MOV     CL,BH
                MOV     CH,ZERO
                CLD
READ-NUMB3 :    POP     AX
                STOSB
                LOOP    READ-NUMB3
READ-NUMB       ENDP

;-----------------------------------------------------------
;       CLEAR PROCEDURE
;-----------------------------------------------------------

CLEAR           PROC    NEAR
                PUSH    DI
                MOV     CX,FORTY
                CLD
                MOV     AL,ZERO-ASCII
                REP
                STOSB
                POP     DI
                RET
CLEAR           ENDP

;-----------------------------------------------------------
;       CHK-NUMB PROCEDURE
;-----------------------------------------------------------

CHK-NUMB        PROC    NEAR
                CMP     AL,ZERO-ASCII
                JB      CHK-ERR
```

این قسمت بافر را پاک می کند

این قسمت شماره زیربرنامه را چک می کند

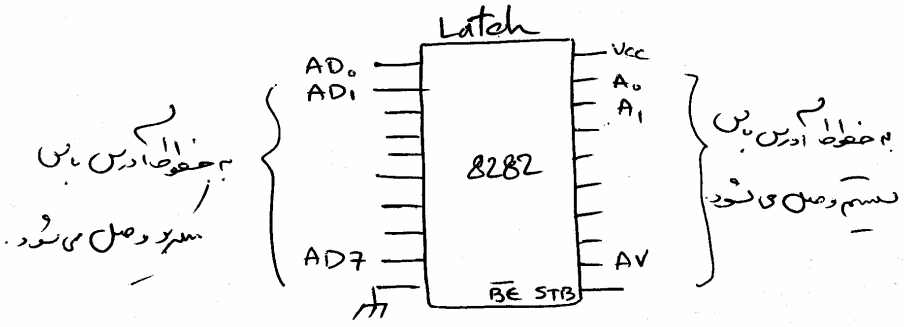
```
                CMP     AL,NINE-ASCII
                JA      CHK-ERR
                CLC
                JMP     CHK-END
CHK-ERR :       STC
CBH-END         RET
CHK-NUMB        ENDP
CODE-SEG        ENDS
                END     MAIN

                MOV     DI,OFFSET ANSWER
                CALL    CLEAR
                MOV     SI,OFFSET NUMB1
                MOV     DI,OFFSET END-ANS
                MOV     BX,OFFSET NUMB1
                MOV     CX,FORTY
                MOV     OVER-ZERO-ASCII

MAIN1 :         CLD
                LODSB
                ADD     AL,[BX]
                MOV     AH,ZERO
                AAA
                ADD     [DI],AH
                STD
                ADD     AL,ZERO-ASCII
                STOSB
                INC     BX
                LOOP    MAIN1
                MOV     AL,OVER
                STOSB
                MOV     DI,OFFSET ANSWER
                MOV     AL,ZERO-ASCII
                MOV     CX,FORTY
                REPE
                SCASB
                CALL    OUT-MESSAGE
                JMP     TOP
MAIN-END :      CALL    OUT-CHAR
                RET
                ENDP

;-----------------------------------------------------------
;  OUT MESSAGE PROCEDURE
;-----------------------------------------------------------

OUT-MESSAGE     PROC    NEAR
                MOV     AH,NINE
                INT     DOS-FUNCTION
                RET
OUT-MESSAGE     ENDP
```

پیغامی که باید به بیرون ارسال شود

①

آی سی ۸۲۸۲ :

* ۳ وضعیتی است.



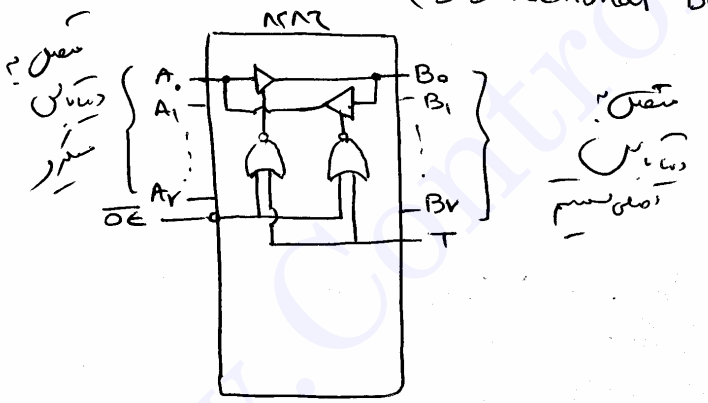
اگر تنها یک میکرو در مدار داشته باشیم، می‌توان OE را مستقیماً به GND وصل کرد.

← برای ۸۰۸۶ باید از ۳ ۸۲۸۲ استفاده شود (برای پوشش داده و آدرس).

آی سی ۸۲۸۶ (Transiver) :

Transmitter + Receiver

← یا فرد دو جهته است (BiDirectional Buffer)



T به پایه DT/R متصل می‌شود.

OE هم به DEN وصل می‌شود.

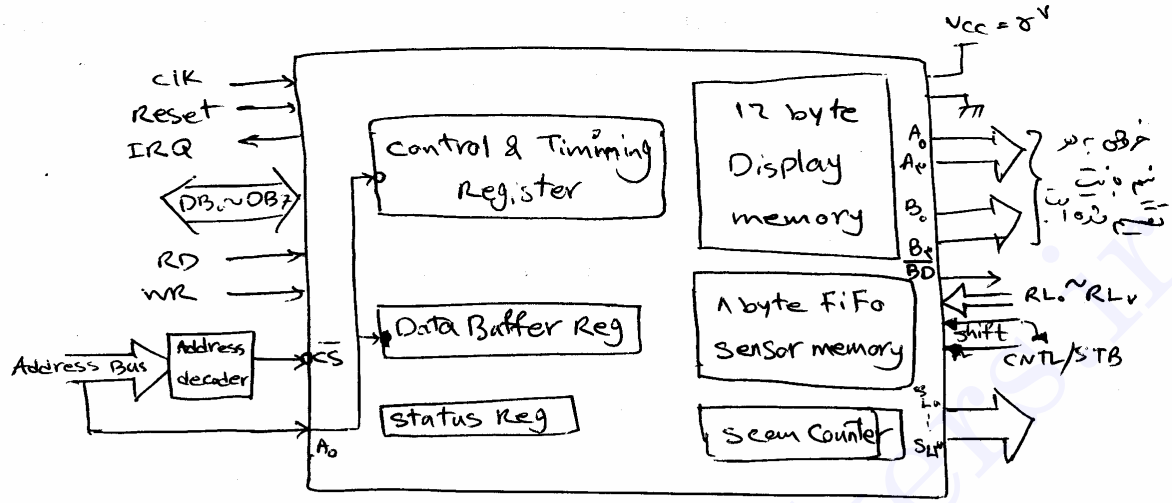
آی سی ۸۲۷۹ : KeyBoard / Display controller

← تا ۱۶ سوئیچ مشکلات را می‌توانند راه اندازی کند.

← KeyBoard از نوع 4x4 تا نهایت 8x8 است.

← این آی سی از نوع LSI است.

(۲)



انواع روش کنترل
۱- polling (چک کردن دائم)
۲- intrupt

انواع اطلاعات در ۸۲۷۹
- اطلاعات کنترلی
-

← بخش کنترل با آدرس زوج و بخش داده با آدرس فرد فعال می‌شود.

Blank Display: بعد از ارسال عدد اول به Display، $\overline{BD}=0$ و یک واحد شیفت می‌دهد و رقم بعدی نمایش داده می‌شود.

$SL_0 \sim SL_3$: برای آدرس دهی است (۰ تا ۱۵ برای تعیین سطرهای کی‌بورد).
با کمک دیکودر در خروجی

سیگنال‌های کنترلی
$\overline{CS}$ ← برای فعال شدن آی‌سی = ۰
$\overline{RD}$ ← برای خواندن = ۰
$\overline{WR}$ ← " نوشتن = ۰
A_0

$\overline{CS}$	RD	WR	A_0	
0	1	0	0	data Bus Send to Buffer Register
0	1	0	1	" " " " control "
0	0	1	0	" Buffer Reg " " Data Bus
0	0	1	1	control Reg " " " " "

(۲)

کلمات کنترلی مهم : ۸۲۷۹

control ← | CNTL | shift | R | R | R | C | C | C |

1- Key Board Display Mode Set

Display KeyBoard

DDD KKK

KKK = ۰۰۰ ← اگر دو کلید زده شد، حذف می‌شوند.

۰۰۱ ← اسکن کی برد دکد شده اگر ۲ کلید همزمان زده شود، حذف می‌شود.

۰۱۰ ← اسکن کی برد دکد شده با نگهداری ۸ کلید همزمان. در این حالت به ترتیب زمان زده شدن در ۸ بایت ذخیره می‌شوند.

بایت همین است اما دکد نشده

۰۱۱ ← برای دو کلید زده شده قابل برنامه‌ریزی است (همان ۰۱۰ با این فرق که دکد شده است نه دکد شده)

۱۰۰ ← اسکن سنسور ماتریسی اسکن شده (بدون دکد)

۱۰۱ ← دکد کردن سنسور ماتریسی اسکن شده.

۱۱۰ ← ورودی strob شده با صفحه نمایش کد شده

۱۱۱ ← ورودی strob شده با صفحه نمایش دکد شده

DDD

۰۰۰ ← نمایشگر چپ ورودی ۸ بیتی

۰۰۱ ← " " " ۱۶ بیتی

۰۱۰ ← " راست " ۸ "

۰۱۱ ← " " " ۱۶ "

امتحان شنبه ۱۴

۳۰ نمره تا اول بخش ۴ - یک بخش امتحان closed book.