فصل اول

اصول زبان ماشین و اسمبلی

معرفی اصول زبان ماشین و اسمبلی

تمام کامپیوترها، برنامه را به زبان ماشین اجرا می‌کنند. در حقیقت زبان ماشین یک سری بیت‌های 0 و یا 1 است که کامپیوتر به وسیله آن می‌داند چه باید انجام دهد. به عنوان مثال در جدول زیر یک سری دستورات به زبان اسمبلی و معادل آن به زبان ماشین در کد باینری 0 و 1 یا معادل آنها در کد هگزادسیمال را نشان می‌دهد. همان‌طور که در جدول مذکور مشاهده می‌شود، زبان ماشین از نظر ما بسیار نامفهوم و پیچیده است، لذا به جای زبان ماشین، از زبان اسمبلی یا زبان سمبولیک استفاده می‌کنیم و برنامه و دستورات را به زبان اسمبلی می‌نویسیم. سپس برنامه مذکور را به زبان ماشین تبدیل کرده و برنامه به زبان ماشین را در کامپیوتر اجرا می‌نمائیم.

دستورات زبان اسمبلی (برنامه اسمبلی)	دستورات زبان ماشین	
	کد باینری	کد هگزادسیمال
MOV AL, ADDRES-2	10100000 0000 0001 0000 0100	A0 01 04
SUB AL, 30H	0010 1100 0011 0000	2C 30
ADD AL, BL	0000 0010 1100 0011	02 C3
HLT	1111 0101	F5

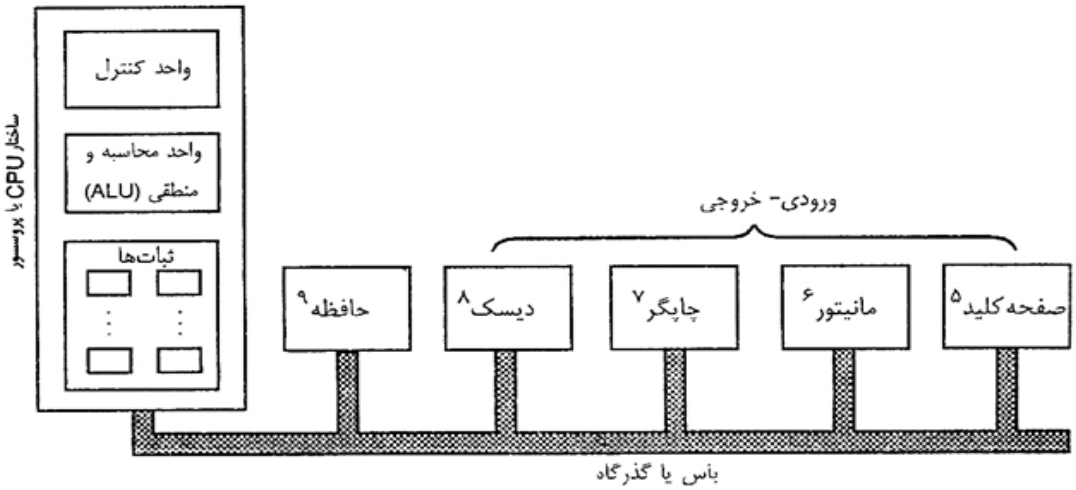
هر دستور در زبان ماشین دارای دو قسمت است یکی آدرس اطلاعات و یا داده‌ها که اصطلاحاً عملوند یا اپراند گویند و قسمت دیگر کد اجرا می‌باشد. زبان ماشین به صورت کد باینری یا کد هگزادسیمال می‌باشد که به هر صورت کار کردن با آنها بسیار دشوار است. بنابراین در عمل از زبان اسمبلی استفاده می‌شود که خیلی نزدیک به زبان ماشین است و در ضمن در زبان اسمبلی از حروف انگلیسی استفاده می‌شود که کار با آن برای ما آسان می‌باشد. برای ترجمه دستورات برنامه به زبان اسمبلی می‌توان از جدول دستورات در کاتالوگ کامپیوتر استفاده نمود و کد باینری معادل آن را بدست آورد. ولی راه آسان‌تر استفاده از برنامه مترجم اسمبلر است که دستورات زبان اسمبلی را یکی یکی ترجمه کرده و به کد باینری معادل آن در زبان ماشین تبدیل می‌نماید. در ضمن لیست برنامه اسمبلی و ترجمه آن را نیز در اختیار کاربر قرار می‌دهد.

ساختار کامپیوتر

کامپیوتر دارای دو قسمت نرم‌افزار و سخت‌افزار می‌باشد. سخت‌افزار کامپیوتر شامل مدارهای الکترونیکی و الکترومکانیکی آن است و نرم‌افزار از برنامه‌های سیستم عامل، کامپایلرها، اسمبلرها و برنامه‌های کاربران تشکیل می‌شود. برای یک کاربر که برنامه را به زبان ماشین و یا زبان اسمبلی می‌نویسد، اطلاع از طرز کار قسمت‌های مختلف و سخت‌افزار کامپیوتر بسیار ضروری می‌باشد.

اصولا کامپیوتر دستگاهی است که اطلاعات دیجیتالی به صورت 0 و 1 را به عنوان ورودی می‌گیرد و بر طبق دستوراتی که در حافظه آن قرار دارند، یک سری عملیات بر روی اطلاعات مذکور انجام می‌دهد و خروجی مورد نظر را تولید می‌کند. لیست دستورات را برنامه کامپیوتر نامند و محلی که این دستورات ذخیره شده‌اند، حافظه‌ی کامپیوتر نام دارد. هر کامپیوتر از

واحدهای باس یا گذرگاه، واحد کنترل CU، ورودی، خروجی، حافظه و واحد محاسبه و منطقی تشکیل شده است. مجموعه واحد محاسبه و منطقی ALU و واحد کنترل و ثبات ها را، پروسسور یا CPU می‌نامند که وظیفه‌ی اجرای دستورات را برعهده دارد. برای درک بهتر به شکل زیر توجه کنید.



تشکیلات یک کامپیوتر، با CPU، حافظه و دستگاه‌های ورودی خروجی

واحد پروسسور یا CPU

همان‌طور که قبلاً ملاحظه شد وظیفه پروسسور یا CPU، اجرای دستورات برنامه کامپیوتر می‌باشد. پروسسور از واحدهای محاسبه و منطق، کنترل و یک سری ثبات جهت درج اطلاعات یا اعداد در آن‌ها تشکیل شده است. البته عملیات کامپیوتر در واحد محاسبه و منطقی پروسسور انجام می‌شود. به عنوان مثال فرض کنید می‌خواهیم دو عدد که در دو ثبات قرار دارند را با هم جمع کنیم. برای این کار می‌بایست این دو عدد به واحد محاسبه و منطقی ALU آورده شوند، که در آنجا با هم جمع گردند و نتیجه جمع ممکن است به حافظه برگردد و یا برای استفاده بعدی در ثبات‌های پروسسور ذخیره گردد. بدیهی است عملیات ریاضی و منطقی دیگر مانند ضرب، تقسیم، مقایسه و.... نیز در واحد محاسبه و منطقی پروسسور انجام می‌شود.

ارتباط مداومی بین پروسسور و هر یک از دستگاه‌های ورودی خروجی و حافظه برقرار می‌باشد. برای این ارتباط راحت‌ترین وسیله ارتباط مستقیم از طریق سیم‌هایی بین آن‌ها است. ولی اگر برای ارتباط دو دستگاه با هم حدود یکصد سیم نیاز باشد و اگر قرار باشد یک دستگاه مستقیماً با همه دستگاه‌های دیگر ارتباط داشته باشد، تعداد سیم‌های رابط، بسیار زیاد و غیر اقتصادی می‌شود. لذا راه عملی‌تر این است که سیم‌های ارتباطی بین عده‌ای از وسائل مشترک باشند. یعنی تعدادی از دستگاه-ها، از یک مقدار سیم ارتباطی، بین عده‌ای از وسائل مشترک باشند. یعنی تعدادی از دستگاه‌ها از یک مقدار سیم، به طور مشترک استفاده نمایند، که این سیم‌های مشترک را گذرگاه یا باس نامند.

تعداد سیم‌های گذرگاه تابع نوع دستگاه‌های متصل در کامپیوترهای معمولی بین 40 تا 200 سیم می‌باشند. به این ترتیب پروسسور با هر واحد کامپیوتر، از طریق یک گذرگاه ارتباط برقرار می‌کند و برای این کار لازم است پروسسور آدرس آن‌ها را روی گذرگاه قرار دهد. لذا قسمتی از گذرگاه، به نام گذرگاه آدرس معروف است که پروسسور آدرس دستگاه‌های ورودی خروجی و یا حافظه را روی آن قرار می‌دهد. بعد از اینکه پروسسور آدرس حافظه، یا یک دستگاه ورودی خروجی را داد، باید اطلاعاتی را در آنها بنویسد و اطلاعاتی را از آنها بخواند. به این ترتیب قسمتی از گذرگاه نیز، به نام گذرگاه داده، یا دیتا معروف می‌باشد، که اطلاعات از طریق آن بین حافظه و دستگاه‌های ورودی خروجی و CPU انتقال می‌یابد.

طرز اجرای دستورات برنامه در پروسسور و سیکل دستور (زمان اجرای دستور)

اصولاً کاربر با نوشتن تعدادی دستور طبق ترتیب خاص، برنامه مورد نظر را می‌نویسد و آن را در حافظه بار می‌کند و سپس پروسسور دستورات را یکی یکی از حافظه می‌خواند و آن را یررسی می‌نماید. یعنی از کد اجرایی آن متوجه می‌شود که چه باید بکند و بالاخره آن دستور را اجرا می‌نماید. سپس دستور بعدی را از حافظه می‌خواند و اجرا می‌نماید و به ترتیب ادامه می‌دهد تا تمام دستورات برنامه اجرا شوند. در حقیقت زمانی برنامه شروع به اجرا می‌شود که آدرس اولین دستور برنامه توسط اشاره‌گر دستور IP، به حافظه داده شود و فرمان خواندن نیز از اطراف واحد کنترل به حافظه ارسال گردد. بعد از مدتی که برابر زمان دستیابی حافظه است محتوای خانه حافظه‌ای که آدرس آن داده شده (در این حالت اولین دستور برنامه می‌باشد) از حافظه خوانده می‌شود و وارد پروسسور می‌گردد. این عملیات را سیکل واکشی دستور نامند. اکنون که دستور داخل پروسسور قرار دارد در واحد محاسبه و منطق ALU اجرا می‌گردد که عملیات اجرای دستور را سیکل اجرا نامند. در حقیقت زمان اجرای دستور مدتی است که یک دستور اجرا می‌شود.

فصل دوم

نمایش اطلاعات در کامپیوتر

مقدمه

سیستم اعداد باینری که از رقم‌های صفر و یک تشکیل می‌شود در کامپیوتر استفاده می‌گردد که هر یک از اعداد 0 یا 1 را یک بیت (یا یک رقم باینری) نامند و مبنای این سیستم اعداد، عدد 2 می‌باشد. البته معمول‌ترین روش نمایش اعداد در کامپیوتر، سیستم باینری است ولی چون مردم با سیستم دهدهی کار می‌کنند، بعضی اوقات مناسب‌تر است که در کامپیوتر از این سیستم اعداد نیز استفاده شود. اصولاً یک سیستم اعداد در مبنای r دارای r رقم می‌باشد که اعداد در این سیستم با مجموعه-ای از این r رقم نمایش داده می‌شوند. برای تبدیل عدد دهدهی به معادل باینری آن، می‌توان عدد دهدهی را تقسیم بر 2 نمود و باقیمانده‌ها را تا موقعی که خارج قسمت صفر شود، در نظر گرفت. به عنوان مثال عدد 25 را به عدد باینری معادل تبدیل می‌کنیم:

25÷ 2=12

12÷ 2=6

6÷ 2=3

3÷ 2=1

1÷ 2=0

پس عدد 25 برابر 11001 در سیستم باینری است.

برای بدست آوردن معادل دودویی قسمت کسری نیز با ضرب در 2، دارای یک عدد صحیح می‌شود که این عدد بزرگ‌ترین یا با ارزش‌ترین بیت قسمت کسری عدد باینری است. قسمت کسری حاصلضرب را دوباره ضرب در 2 می‌کنیم که قسمت صحیح حاصلضرب، دومین رقم باینری است. به همین ترتیب قسمت کسری حاصلضرب جدید را دوباره در2 ضرب می‌کنیم و روش را ادامه می‌دهیم تا اینکه قسمت کسری برابر صفر شود یا اینکه تعداد رقم‌ها دقت مورد نظر را حاصل کند. در انتها قسمت صحیح عدد باینری و قسمت کسری عدد باینری پهلوی هم قرار داده می‌شود که عدد معادل باینری مذکور بدست می‌آید. مثلاً برای عدد 41/6875 خواهیم داشت:

برای قسمت کسری ۰/۶۸۷۵:

۰/۶۸۷۵ × ۲ = ۱/۳۷۵۰ کوچکترین بیت

۰/۳۷۵ × ۲ = ۰/۷۵۰

۰/۷۵۰ × ۲ = ۱/۵۰۰

۰/۵۰۰ × ۲ = ۱/۰۰۰

پس

$(۰/۶۸۷۵)_{۲}=(۰.۱۰۱۱)_{۲}$ بزرگترین بیت

پس

برای عدد صحیح ۴۱:

باقیمانده	خارج قسمت
۱	۴۱ ÷ ۲ = ۲۰
۰	۲۰ ÷ ۲ = ۱۰
۰	۱۰ ÷ ۲ = ۵
۱	۵ ÷ ۲ = ۲
۰	۲ ÷ ۲ = ۱
۱	۱ ÷ ۲ = ۰

$(۴۱)_{۱۰} = (۱۰۱۰۰۱)_{۲}$

در نتیجه عدد ۴۱/۶۸۷۵ برابر عدد باینری $(۱۰۱۰۰۱ . ۱۰۱۱)^{۲}$ می‌باشد یا:

$(۴۱/۶۸۷۵)_{۱۰} = (۱۰۱۰۰۱ . ۱۰۱۱)_{۲}$ است.

اعداد هگزادسیمال

برای راحتی کار در کامپیوترها، از سیستم اعداد هگزادسیمال یعنی اعداد در مبنای 16 استفاده می‌شود. به عنوان مثال عدد باینری 100010010110 که از یک سری صفر و یک‌ها تشکیل شده را می‌توان به صورت 896H هگزادسیمال که بسیار آشناتر است و آسانتر خوانده می‌شود، را نشان داد. سیستم اعداد باینری دارای دو رقم صفر و یک می‌باشند ولی سیستم اعداد دهدهی از رقم صفر تا نه تشکیل می‌شوند و سیستم اعداد هگزادسیمال دارای رقم‌های صفر تا نه و رقم‌های A، B، C، D،E و F می‌باشد.

تبدیل عدد باینری به هگزادسیمال

برای تبدیل اعداد باینری به هگزادسیمال کافیست عدد باینری را از سمت راست، چهار بیت چهار بیت جدا کنیم و به جای هر چهار بیت کد معادل نظیر هگزادسیمال آن را قرار دهیم. به عنوان مثال عدد باینری (1010111101100011) را به صورت چهار بیت از سمت راست جدا می کنیم که خواهیم داشت:

عدد باینری	1010	1111	0110	0011
عدد هگزادسیمال	A	F	6	3

لذا عدد باینری فوق، معادل عدد AF63H در سیستم هگزادسیمال می‌باشد (علامت H در انتهای عدد یعنی سیستم اعداد هگزادسیمال است). در صورتی که در انتهای سمت چپ عدد، کمتر از 4 بیت باقی مانده، به انتهای سمت چپ عدد آنقدر صفر اضافه می‌کنیم تا گروه آخری هم 4 بیتی شود که بتوان معادل هگزادسیمال آن را نوشت.

تبدیل عدد هگزادسیمال به باینری

برای تبدیل عدد هگزادسیمال به مقدار باینری نیز کافیست به جای هر رقم هگزادسیمال، عدد معادل باینری آن را قرار داد. به عنوان مثال عدد گزا دسیمال 8E6H برابر است با:

عدد هگزادسیمال	6	E	A
عدد باینری	0110	1110	1000

یعنی عدد هگزادسیمال 8E6H برابر 0110 1110 1000 باینری است.

تبدیل عدد دهدهی به هگزادسیمال

برای تبدیل اعداد دهدهی به هگزادسیمال دو راه وجود دارد:

الف: تبدیل عدد دهدهی به باینری و سپس تبدیل عدد باینری به هگزادسیمال

ب: تبدیل عدد دهدهی مستقیماً به هگزادسیمال که برای این کار، کافیست عدد دهدهی را بر 16 تقسیم کنیم که باقی‌مانده، اولین یا کم ارزش‌ترین رقم هگزادسیمال است. سپس خارج قسمت را بر16 تقسیم کنیم و باقیمانده را در نظر بگیریم، آنقدر این کار را ادامه می‌دهیم تا خارج قسمت صفر شود.

تبدیل عدد هگزادسیمال به عدد دهدهی

برای تبدیل عدد هگزادسیمال به عدد دهدهی معادل آن می‌توان:

الف: ابتدا عدد را به باینری تبدیل می کنیم و سپس عدد باینری را به عدد دهدهی تبدیل نمائیم.

ب: مستقیماً عدد هگزادسیمال را به دهدهی تبدیل کنیم. به عنوان مثال عدد هگزادسیمال 6B2H را مطابق زیر به دهدهی تبدیل می‌کنیم

$$6B2=6*16^{2}+B*16^{1}+2*16^{0}$$

پس خواهیم داشت:

6B2 هگزا دسیمال =	$2*16^{0}$	2+
	$11*16^{1}$	176
	$6*16^{2}$	1536
		1714

یعنی عدد هگزادسیمال 6B2H برابر عدد 1714 در سیستم اعداد دهدهی است.

جمع و تفریق اعداد باینری

جمع و تفریق اعداد باینری در مدارهای جمع کننده کامپیوتر، انجام می‌شوند. در مورد اعداد علامت‌دار، آخرین بیت سمت چپ به عنوان بیت علامت (S) در نظر گرفته می‌شود که اگر S=0 باشد یعنی عدد مثبت است و در صورتی که S=1 باشد عدد منفی است. در مورد جمع و تفریق اعداد باینری، بیت علامت نیز مانند سایر بیت‌ها در نظر گرفته می‌شود.

جمع باینری

جمع اعداد باینری بسیار ساده است. (مطابق جدول زیر) در این جا مجموع دو بیت، S و بیت نقلی C می‌باشند. همان طور که ملاحظه می‌شود اگر دو بیت صفر باشند، مجموع آنها نیز صفر است و اگر یکی از آنها یک باشد، مجموع یک می‌باشد و در صورتی که هر دو بیت یک باشند، مجموع S برابر صفر و بیت نقلی C مساوی یک است، که برای جمع با بیت بعدی آن را نیاز خواهیم داشت. جهت جمع هر بیت دو عدد، از مداری به نام .F.A (جمع کننده کامل) استفاده می‌شود که اگر اعداد ما چهار بیتی باشند، چهار مدار جمع کننده کامل .F.A استفاده می‌شود.

مثال: اگر عدد باینری A=00000110 و عدد باینری B=00001101 باشند، در این صورت مجموع این دو عدد، با جمع بیت با بیت این دو عدد از سمت راست انجام می‌پذیرد. یعنی با توجه به جدول زیر خواهیم داشت:

مجموع دو بیت باینری

a + b	مجموع s	بیت نقلی C
۰ + ۰	۰	۰
۰ + ۱	۱	۰
۱ + ۰	۱	۰
۱ + ۱	۰	۱

	7	6	5	4	3	2	1	0	شماره بیت
A=0	0	0	0	0	0	1	1	0	
B=0	0	0	0	0	1	1	0	1	
	0	0	0	1	0	0	1	1	
مجموع S	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0	
بیت نقلی C	0	0	0	0	1	1			

مجموع دو بیت صفرم عدد A و B برابر $S_0 = 1$ و مجموع دو بیت یکم A و B برابر $S_1 = 1$ می باشد. در مورد بیت دوم A و B، چون هر دو عدد یک هستند، پس طبق جدول (مجموع دو بیت باینری)، $S_2 = 0$ و یک بیت نقلی C=1 برای جمع با بیت بعدی سمت چپ خواهد بود بنابراین برای جمع بیت سوم، بیت نقلی C که برابر یک است، با بیت سوم دو عدد جمع می‌شوند که مجموع $S_3 = 0$ و بیت نقلی آخری C=1 می‌گردد. با جمع این بیت نقلی با بیت چهارم عدد، مجموع دو بیت چهارم $S_4 = 1$ و C=0 می‌شود و به همین ترتیب مجموع بقیه بیت‌ها بدست می‌آید. با روش مذکور جمع دو رقم باینری A و B در جمع کننده‌های کامپیوتر انجام می‌گردد.

تفریق اعداد باینری و سیستم اعداد مکمل 2

در کامپیوترها تفریق‌کننده مجزا برای تفریق دو عدد ساخته نمی‌شود و معمولاً از همان جمع‌کننده که برای جمع دو عدد به کار می‌رود، برای تفریق نیز استفاده می‌شود. در این صورت برای تفریق دو عدد، مکمل 2 مفروق را پیدا می‌کنیم، و با مفروق منه جمع می‌نمائیم. مکمل2 یک عدد یعنی مکمل یک آن عدد بعلاوه یک. برای پیدا کردن مکمل 1 هر عدد، بیت‌های آن عدد را معکوس می‌کنیم، یعنی صفر را به یک، یک را به صفر تبدیل می‌کنیم. به عنوان مثال عدد 9 = 00001001 را در نظر می‌گیریم. مکمل یک عدد 9 برابر 11110110 می‌باشد که هر بیت عدد معکوس شده‌اند. یعنی اگر صفر بوده تبدیل به یک و اگر یک بوده تبدیل به صفر شده‌اند. برای پیدا کردن مکمل 2 عدد کافیست به مکمل 1 عدد، یک واحد اضافه نمائیم، یعنی مکمل 2 عدد 9 برابر 11110110 + 1 = 111110111 می‌شود. حال اگر بخواهیم عدد 9 = 00001001 را از عدد A= 00001101 کم کنیم، کافیست مکمل 2 عدد B یعنی 11110111 را با عدد A جمع کنیم، در نتیجه خواهیم داشت:

$$\begin{array}{rl} 13 = 00001101 & + \\ \text{مکمل ۲ عدد ۹} = \underline{11110111} & \\ A\text{-}B = 00000100 & = 4 \end{array}$$

همچنین می‌بایستی از بیت نقلی آخری صرف‌نظر کنیم که نتیجه تفریق برابر 4 بدست می‌آید. با توجه به مطالب فوق ملاحظه می‌شود برای عمل تفریق می‌توان از همان جمع‌کننده‌های کامپیوتر استفاده نمود، بشرط اینکه بجای مفروق، مکمل 2 آن به ورودی جمع‌کننده داده شود. به این ترتیب نیازی به تفریق‌کننده مجزا نیست. جمع و تفریق اعداد علامت‌دار نیز عیناً مانند فوق انجام می‌شود و آخرین بیت سمت چپ هر عدد که به عنوان بیت علامت است، مانند سایر بیت‌ها در نظر گرفته می‌شود و در سیستم اعداد مکمل2، نتیجه صحیح بدست می‌آید.

اعداد علامت‌دار مثبت و منفی

بزرگترین عدد هفت بیتی موقعی است که همه بیت‌ها برابر یک باشند، یعنی 1111 111 = 127 که برابر $2^7 - 1$. اگر عدد هشت بیتی باشد، بزرگترین عدد برابر $2^8 - 1 = 255$ می‌باشد و بالاخره اگر عدد باینری با n بیت داشته باشیم، بزرگترین عدد باینری n بیتی مساوی $2^n - 1$ است. کوچکترین عدد باینری نیز برابر صفر می‌باشد. برای اعداد علامت‌دار، بزرگترین یا باارزش‌ترین بیت را، بیت علامت (S) در نظر می‌گیرند. مثلاً اگر عدد 8 بیتی را در نظر بگیریم بیت آخر، بیت علامت و هفت بیت دیگر، قدر مطلق عدد است. به این ترتیب بازه‌ی اعداد مثبت 0 , 1 , 2 , 3 , 4 ... 127

بیت علامت
↓

```
0000  0000  =  0
0000  0001  =  1
0000  0010  =  2
0000  0011  =  3
....  ....  =  .
....  ....  =  .
0111  1111  = 127
```

می باشند. اعداد منفی معمولاً به صورت مکمل 2 نشان داده می‌شوند. مثلاً عدد 3- برابر مکمل2 عدد 3 یعنی:

علامت عدد
↓

1 1111 1101

است و عدد 124- برابر مکمل2 عدد 127+ و مساوی:

علامت
↓

1000 0001

می‌باشد. به این ترتیب با 8بیت می‌توان اعداد مثبت بین 0 تا 127+ و اعداد منفی بین 0 تا 128- را نشان داد و یا اعداد بدون علامت بین 0 تا 255 را نمایش داد.

نمایش اعداد دهدهی BCD

گر چه در کامپیوتر انجام محاسبات در سیستم اعداد باینری آسان‌تر است ولی مردم به سیستم اعداد دهدهی عادت کرده‌اند. یک راه برای حل این مساله این است که تمام اعداد دهدهی را در ورود به کامپیوتر، به اعداد باینری تبدیل کنیم و محاسبات را در سیستم اعداد باینری در کامپیوتر انجام دهیم و برای خارج کردن نتایج، دوباره اعداد باینری را به اعداد دهدهی تبدیل

نمائیم. راه دیگر این است که ما محاسبات را مستقیماً در سیستم اعداد دهدهی BCD در کامپیوتر انجام دهیم. در نمایش اعداد دهدهی BCD چون ما 10 رقم داریم، بنابراین 4 بیت برای نمایش آن نیاز است ولی با چهار بیت $2^4 = 16$ ترکیب داریم که از ده ترکیب آن برای نمایش رقم‌های صفر تا نه استفاده می‌شود و از بقیه شش ترکیب استفاده نمی‌گردد.

نمایش اعداد و حروف در کد ASCII

اکثر کاربردهای کامپیوتر نیاز به حروف بزرگ و کوچک انگلیسی و رقم‌های صفر تا نه و حروف خاص مانند +، $، =، ... دارند که تعداد آنها بیش از 70 نویسه می‌باشند لذا برای نمایش آنها حداقل 7 بیت لازم است.

فصل سوم

اصول برنامه‌نویسی به زبان اسمبلی

مقدمه

همان‌طور که قبلاً گفته شد، نوشتن و درک برنامه به زبان ماشین کار مشکل و خسته‌کننده‌ی است لذا امروزه ترجیح می‌دهند کمتر با زبان ماشین کار کرده و برنامه‌های کامپیوتر را به زبان اسمبلی بنویسند. زبان اسمبلی به جای استفاده از اعداد، معمولاً از حروف الفبای انگلیسی استفاده می‌کند و برنامه‌ی مترجم یا اسمبلر، برنامه به زبان اسمبلی را به زبان ماشین ترجمه می-نماید. گرچه اصول زبان اسمبلی برای همه کامپیوترها یکسان است ولی هر ماشین دارای زبان اسمبلی ویژه خود می‌باشد بنابراین زبان اسمبلی همانند زبان ماشین وابسته به ساختمان ماشین است و دستورهای زبان اسمبلی در انواع ماشین‌ها متفاوت می‌باشد.

هر دستور زبان اسمبلی فقط به یک دستور زبان ماشین ترجمه می‌شود. بنابراین اگر برنامه‌ایی در زبان اسمبلی N دستور داشته باشد، این برنامه توسط مترجم اسمبلر تبدیل به N دستور زبان ماشین خواهد شد. در صورتی که این موضوع در برنامه‌های به زبان دیگر که دارای کامپایلر هستند صادق نبوده و ترجمه هر دستور برنامه‌ها ممکن است تولید چند دستور به زبان ماشین بنماید. استفاده از زبان اسمبلی (به جای زبان ماشین) کار برنامه‌نویسی را ساده می‌نماید، چون یادگیری سمبول‌های اسمبلی بسیار آسانتر از کدهای باینری، یا هگزادسیمال زبان ماشین است.

به کمک زبان اسمبلی می‌توان به تمام عملیات ماشین تسلط داشت و کلیه موقعیت‌های پیش آمده را بررسی نمود و به محتوای ثبات‌های پروسسور دسترسی پیدا کرد در حالی که در زبان‌های برنامه‌نویسی سطح بالا امکان استفاده از این تسهیلات نیست. لذا در کاربردهایی که نیاز به دسترسی مستقیم به سخت‌افزار می‌باشد، تهیه و اجرای برنامه‌ها به زبان اسمبلی، نسبت به برنامه‌های سطح بالا بسیار سریع‌تر می‌باشد به همین دلیل است که، اکثر برنامه‌های سیستم عامل و همچنین برنامه‌های کامپیوتری دستگاه‌های صنعتی و میکروکنترلرها به زبان اسمبلی می‌باشند.

نمونه‌ای از برنامه اسمبلی

همان‌طور که ملاحظه گردید، برنامه‌نویسی به زبان اسمبلی بسیار آسان‌تر از برنامه‌نویسی به زبان ماشین می‌باشد. اصولاً هر برنامه به زبان اسمبلی از تعدادی دستورات زبان اسمبلی تشکیل می‌گردد. به عنوان مثال برنامه زیر نمونه‌ایی از یک برنامه اسمبلی می‌باشد که از یک سری دستورات زبان اسمبلی تشکیل شده است. این برنامه محتویات دو خانه حافظه به آدرس‌های DATA1 و DATA2 که در کد اسکی هستند را با هم جمع می‌نماید و نتیجه جمع در (کد باینری) خانه DATA3 قرار می‌دهد و در پایان به سیستم عامل برمی‌گردد.

```
page 110,100
title 'Exam1.asm'
stacksg segment stack  'stack'
        dw 32h DUP(0)
stacksg ends
datasg segment 'data'
        Data1 db 10
        Data2 db 20
        Data3 db ?
datasg ends
```

```
codesg segment 'code'
        assume ss:stacksg,ds:datasg,cs:codesg
main proc far
        mov ax,datasg
        mov ds,ax
        mov al,Data1
        mov bl,Data2
        add al,bl
        mov Data3,al
exit:
        mov ax,4c00h
        int 21h
main    endp
codesg  ends
end main
```

قالب یا فرمت دستورات زبان اسمبلی

قالب دستورات زبان اسمبلی به صورت کلی دارای قسمت یا فیلدهای برچسب دستور، کد اجرا، عملوند یا اپراندها و توضیحات می‌باشد.

Label	:	Opcode	Operand1	,	Operand2	, . . . ;	Comment
برچسب	:	کد اجرا	اُپراند۱ یا عملوند۱	,	اُپراند ۲ یا عملوند۲	, . . . ;	توضیحات

به عنوان مثال: MAIN: MOV AH,0 ; Move zero to

در دستور فوق، بر چسب دستور، MAIN است. کد اجرا سمبول MOV می‌باشد که معنی آن این است که صفر به عملوند یا اپراند AH انتقال می‌یابد و بعد از نقطه‌ویرگول توضیح دستور به زبان انگلیسی نوشته شده است.

برچسب

برچسب نام اختیاری است که به دستور یا اطلاعات داده می‌شود. قسمت یا فیلد برچسب، به هشت حرف محدود می‌شود و در اولین قسمت دستور اسمبلی قرار می‌گیرد و بعد از آن دو نقطه می‌باشد که علامت برچسب دستور است. اصولاً برچسب دستور اختیاری است و برای مراجعه به آن دستور در دستورالعمل‌های پرش به کار برده می‌شود.

کد اجرا یا عملگر

کد اجرای دستور، نوع کاری را که دستور باید اجرا نماید، مشخص می‌کند و معمولاً به صورت سمبولیک در زبان انگلیسی نوشته می‌شود مانند ADD، MOV، SUB.... که به ترتیب نمایش جمع، انتقال و تفریق و .... می‌باشند و اصولاً کد اجرا سه یا چهار حرفی می‌باشد.

عملوند یا اپراند

اپراند می‌تواند آدرس حافظه یا ثبات و یا عدد ثابت باشد. تعدادی از دستورات ممکن است یک یا دو یا چند اپرند داشته باشند. در بعضی دیگر مانند MOVSD یا NOP ممکن است اصلاً اپراند نداشته باشند.

توضیحات برنامه

برای توضیح و شرح کار دستورها و برنامه می‌توان با استفاده از علامت (نقطه ویرگول) عبارتی را به عنوان توضیح در هر دستور یا خط برنامه نوشت البته این قسمت در برنامه اسمبلی اختیاری است. به عنوان مثال:

STRAT: JMP MAIN ; Jump over data

که در دستور فوق START برچسب دستور، JMP کد اجرا و MAIN اپراند یا آدرس حافظه و بالاخره Jump over data توضیح این دستور است که می‌گوید از قسمت داده‌ها پرش کند.

روش نام‌گذاری آدرس اطلاعات و برچسب دستور

همان‌طور که در فصل اول ملاحظه شد خانه‌های حافظه با آدرس مشخص می‌شوند. به عنوان مثال 1500 آدرس یک خانه حافظه است و اگر بخواهیم محتوای این خانه از حافظه را به ثباتی مانند R0 منتقل کنیم از دستور:

MOV R0,[1500]

استفاده می‌کنیم. در حقیقت [1500] یعنی محتوای خانه حافظه‌ای که آدرس آن 1500 است. ولی اگر در خانه حافظه 1500 نتیجه‌ی جمع چند عدد قرار گرفته باشد، بهتر است به جای به کار بردن آدرس 1500، نام سمبولیک اختیاری مثلاً SUM را به این آدرس اختصاص دهیم. در این صورت دستور بالا به صورت:

MOV R0,SUM

نوشته می‌شود. لذا کسی که این دستور را می‌خواند، متوجه می‌شود که مجموع چند عدد در آدرس SUM به ثبات R0 منتقل شده است. بنابراین در زبان اسمبلی معمولاً به جای استفاده از آدرس عددی، نام سمبولیک اختیاری به آدرس اطلاعات حافظه می‌دهیم که در ارتباط با مفهوم مسئله باشد. البته نام آدرس اطلاعات کاملاً اختیاری است. علاوه بر این، برای مراجعه به دستورات، به آدرس دستور در حافظه نیز، نام سمبولیکی اختصاص می‌دهیم که برچسب دستور نامیده می‌شود، مانند دستور:

LOOP1: MOV R0,10

که LOOP بر چسب یا افست آدرس دستور در حافظه است و نام سمبولیکی است که ما به آدرس آن دستور داده‌ایم. البته هر نامی برای آدرس اطلاعات یا آدرس دستور که با حروف A تا Z (حروف کوچک یا بزرگ) و رقم‌های 0 تا 9 باشد، می‌توان انتخاب نمود بشرطی که اولین حرف یک رقم نباشد و جزء کلمات رزرو شده مانند نام ثبات AX، BXو CX ... نباشد. معمولاً نام‌های مذکور را حداکثر 7 حرفی مانند COUNT، TOTAL1، JAMEH2، MAIN، LOOP2، ADDRES1،.... انتخاب می‌کنند.

فصل چهارم

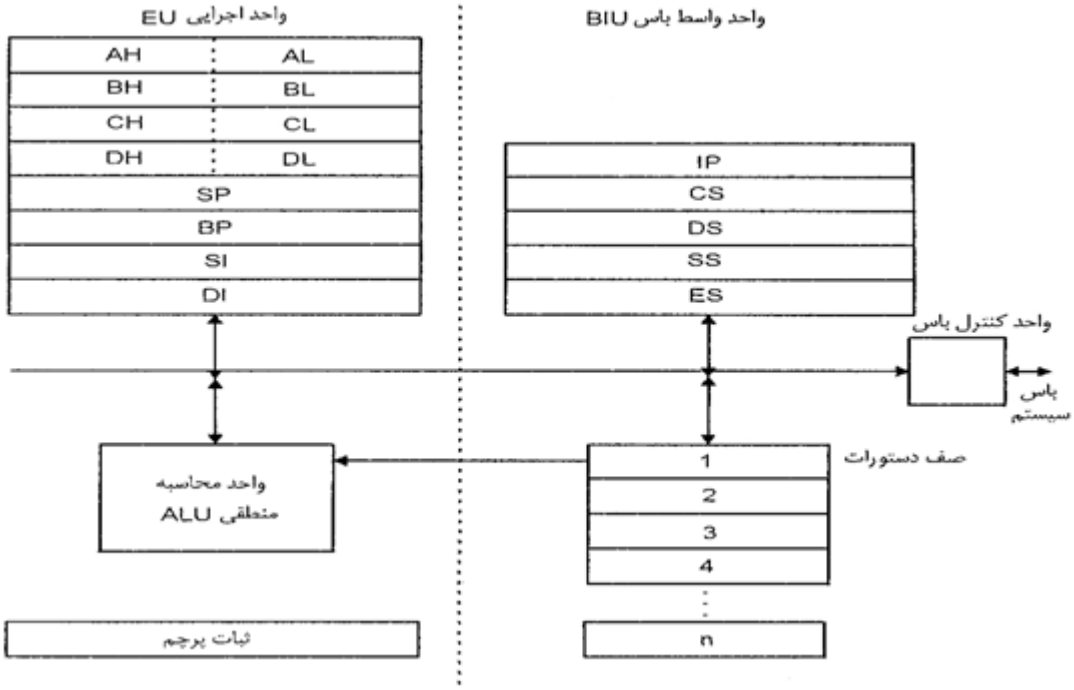
ساختار اصولی کامپیوترهای شخصی

مقدمه

کامپیوترهای شخصی دارای پروسسور از خانواده 8086 شرکت اینتل هستند که پروسسورهای اولیه دارای ثبات‌ها و آدرس‌دهی 16 بیتی و سرعت حدود 10 مگاهرتز بودند. ولی پروسسورهای جدیدتر پنتیوم 80586 و پنتیوم پرو دارای ثبات‌های 32 بیتی هستند که می‌توان از آن‌ها به صورت 8 بیتی، 16بیتی یا 32 بیتی متناسب با نیاز استفاده نمود. علاوه بر این پروسسورهای مذکور سرعت بالای 400 مگاهرتز دارند. همانطوری که ملاحظه می‌شود، نیاز روز به روز به سرعت، ظرفیت حافظه، ظرفیت ثبات‌ها و امکانات سخت‌افزای و نرم‌افزاری پروسسورهای کامپیوترها اضافه شده است. به طوری که پروسسور کامپیوترهای امروزی بسیار سریع، دارای ظرفیت حافظه بالا و دستورات بیشتر و در نتیجه کارآیی بالاتری هستند لذا نرم‌افزارهای بسیار پیشرفته (مانند نرم‌افزارهای سه‌بعدی، گرافیکی، شبیه سازی...) را می‌توان با آنها اجرا نمود.

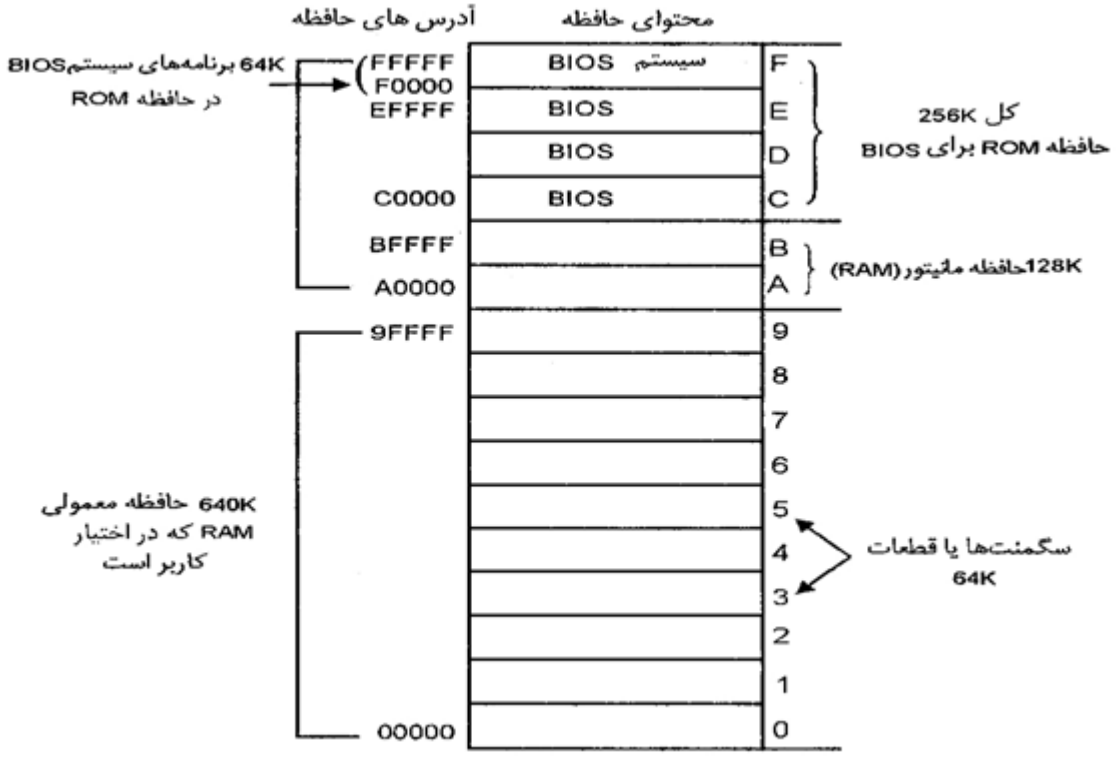
ساختار داخلی پروسسور

پروسسور یا CPU کامپیوترهای شخصی از دو قسمت واحد اجرایی EU و واحد واسط باس BIU تشکیل شده است. واحد اجراییEU مسئول اجرای دستورات است که از یک واحد محاسبه و منطقی ALU و تعدادی ثبات تشکیل شده است. واحد واسط باس BIU، شامل واحد مدیریت کنترل باس، ثبات‌های سگمنت و صف دستورات می‌باشد. یکی از کارهای مهم واحد واسط باس BIU، پیش‌خوانی دستورات از حافظه و قرار دادن آنها در صف دستورات است. لذا همیشه یک سری دستورات از قبل، از حافظه خوانده می‌شوند و در صف دستورات قرار می‌گیرند که هر لحظه واحد اجرایی بخواهد دستور را اجرا کند بلافاصله از صف دستورات، دستور را می گیرد و منتظر خواندن دستور نمی‌شود و در زمانی که واحد اجرایی EU دستور مذکور را اجرا می‌کند واحد واسط باس BUI، دستور دیگری را از حافظه می‌خواند و در صف دستورات قرار می‌دهد. به این ترتیب این دو واحد به صورت موازی با هم کار می‌کنند، در نتیجه سرعت CPU بالا می‌رود. با توجه به مطالب فوق، پروسسورهای از سری پنتیوم به بعد بسیاری از عملیات را به طور موازی انجام می‌دهند. در نتیجه سرعت آنها روز به روز افزایش می‌یابد.



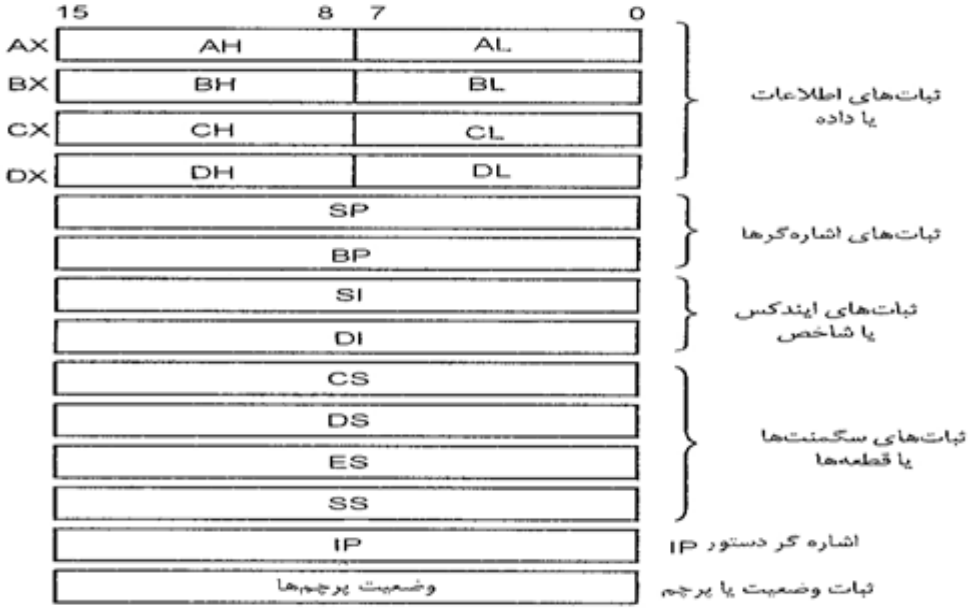
حافظه

حافظه کامپیوترهای شخصی در یک مگابایت اول مطابق شکل (2-4)می‌باشد. همان‌طوری که ملاحظه می‌شود از آدرس صفر 00000 تا آدرس9FFFFH (مقدار 640K)، حافظه رم می‌باشد که در اختیار کاربر است و از آدرس A0000H تا BFFFFH یعنی از640K تا 768K (128K از حافظه RAM است) برای عملیات مانیتور می‌باشد. از آدرس C0000H تا EFFFFH یعنی از 768 K تا 960K به حافظه ROM اختصاص دارد و برای روتین‌های ورودی خروجی پایه‌ی BIOS، که جهت سرویس دستگاه‌های ورودی خروجی مانند دیسک، چاپگر و... بکار می‌روند و بلاخره از آدرس F0000H تا FFFFH مقدار64K حافظه است که برای برنامه‌های سیستم می‌باشند و جهت تست کردن قسمت‌های مختلف کامپیوتر، روتین‌های BIOS و برنامه بوتینگ یا برنامه راه‌اندازی از آن استفاده می‌کنند.



ثبات‌های پروسسور

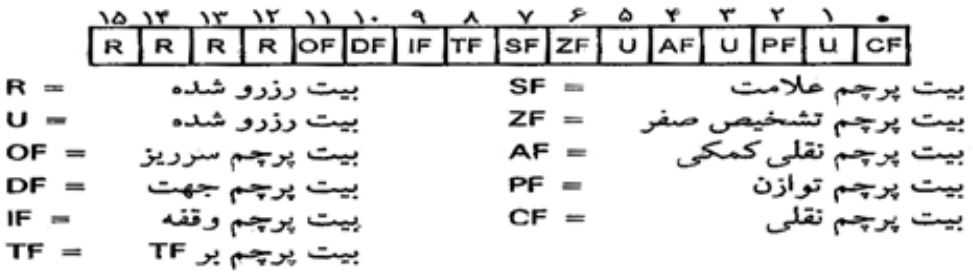
کامپیوترهای شخصی دارای تعدادی ثبات می‌باشند که کاربر می‌تواند روی آنها عملیات انجام دهد. اصولاً در این کامپیوتر 16 بیت اطلاعات را، یک کلمه گویند که از بیت0 تا 15 شماره‌گذاری شده‌اند، به این ترتیب هر کلمه از دو بایت تشکیل می‌شود. این کامپیوترها دارای چهار نوع ثبات 16بیتی از جمله AX، BX، CX، DX، SI، DI، BP، SP هستند.



نکته: عملیات محاسباتی معمولاً بر روی ثبات‌های AX، BX و DX انجام می‌شوند و ثبات AX را بعضی اوقات آکومولاتور نیز می‌نامند. علاوه بر این، در کامپیوترهای شخصی، ثبات‌های اشاره‌گری مانند ثبات اشاره‌گر پشته SP و ثبات اشاره‌گر پایه BP، ثبات اشاره گر دستور IP، ثبات ایندکس SI، ثبات ایندکس DI نیز وجود دارند، که کاربردهای آنها را در فصل‌های بعدی خواهیم دید. ثبات‌های سگمنت داده DS، سگمنت کدCS، سگمنت پشته SS و سگمنت داده اضافی ES نیز جهت سگمنت-بندی برنامه به کار می‌روند.

ثبات وضعیت یا پرچم

کامپیوترهای شخصی دارای ثبات وضعیت PSW یا ثبات پرچم FR شانزده‌بیتی هستند که وضعیت فعلی پروسسور را مشخص می‌کند. شش عدد از بیت‌های پرچم این ثبات عبارتند از CF، ZF، SF ، AF ، PF، OF که پرچم‌های شرطی نامیده می‌شوند، چون در نتیجه اجرای دستورات محاسباتی یک یا صفر می‌شوند. و سه بیت پرچم دیگر بیت‌های IF، TF، DF هستند که پرچم‌های کنترل می‌باشند. زیراکه برای کنترل عملیات دستورات استفاده می‌گردند و بقیه بیت‌های پرچم رزرو شده هستند و کاربردی ندارند. اصولاً هر یک از بیت‌های پرچم جهت بررسی وضعیت بخصوصی از نتیجه عملیات CPU می‌باشند که در ذیل کار هریک از آن ها شرح داده می‌شود.



1- بیت پرچم نقلی CF: این بیت پرچم موقعی یک می‌شود. (CF = 1) که:

الف: در دستورات 8 بیتی از بیت 7 (هشتمین بیت) یک بیت نقلی ایجاد می‌شود.

ب: در دستورات 16 بیتی از بیت 15 (شانزدهمین بیت) یک بیت نقلی ایجاد می‌شود.

2- بیت تشخیص صفر ZF: این بیت پرچم موقعی یک می‌شود (ZF = 1) که نتیجه عملیات محاسباتی برابر صفر شود، در غیر این صورت (ZF = 0) می‌باشد.

3- بیت پرچم علامتSF: در نمایش اعداد علامت‌دار باینری، بزرگترین یا پرارزش‌ترین بیت، بیت علامت است. بعد از اتمام عملیات محاسباتی و منطقی، مقدار این بیت علامت، بر بیت پرچم علامت SF کپی می‌شود، لذا بیت پرچم SF، علامت نتیجه آخرین محاسبات را نشان می‌دهد.

4- بیت پرچم نقلی کمکی AF: در محاسبات با کدهای BCD، اگر از بیت شماره سه (چهارمین بیت) بیت نقلی به بیت بعدی ایجاد شود، بیت پرجم نقلی کمکی AF یک می‌گردد، در غیر این صورت صفر می‌شود.

5- بیت پرچم توازن PF: بعد از عملیات محاسباتی یا منطقی، بایت کوچکتر بررسی می‌شود، اگر تعداد بیت‌های یک این بایت، زوج باشد، بیت توازن PF یک می‌شود، در غیر این صورت بیت توازن PF صفر می‌گردد.

6- بیت پرچم سرریز OF: این بیت پرچم موقعی یک می‌شود که، نتیجه عملیات اعداد جبری بزرگتر از مقدار مجاز باشد، که سرریز ایجاد گردد.

7- بیت فعال کردن وقفه IF: این بیت برای فعال کردن، یا غیر فعال نمودن وقفه‌های خارجی است.

8- بیت پرچم TF: موقعی که بیت پرچم TF برابر یک می‌شود، یک دستور اجرا می‌شود، که بعد از بررسی نتیجه محاسبات توسط کاربر، دستور بعدی اجرا می‌گردد، این عمل برای پیدا کردن اشتباه در برنامه بسیار مناسبی می‌باشد.

9- بیت پرچم DF: این بیت برای کنترل جهت عملیات دستورات رشته به کار می‌رود.

البته تمام دستورات روی بیت‌های پرچم اثر نمی‌گذارند. به عنوان مثال دستور MOV، فقط اطلاعات را منتقل می‌کند و روی بیت‌های پرچم اثر نمی‌گذارند ولی دستورات محاسباتی و منطقی مانند جمع ADD تفریق SUB و.... روی بیت‌های پرچم اثر می‌گذارند.

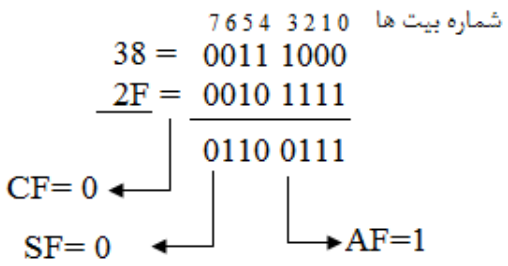
برای روشن شدن مطلب، اثر دستور جمع ADD را روی بیت‌های پرچم بررسی می‌نمائیم. به عنوان مثال می‌خواهیم ببینیم که در جمع عدد 38H با2FH وضع بیت‌های پرچم چگونه است؟

حل: دستورات زیر را می‌نویسیم:

MOV BH, 38H (1)

ADD BH, 2FH (2)

دستور (1) عدد 38H را وارد ثبات می ماید و دستور (2) مقدار 2FH را به طریق زیر با BH (BH=38H) جمع می‌کند.



در عملیات جمع فوق چون:

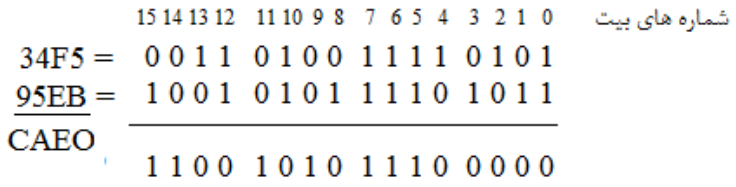
- از بیت 7 بیت نقلی ایجاد نمی‌شود، پس CF =0 است.
- از بیت 3 بیت نقلی ایجاد می‌شود، لذا AF=1 می‌باشد.
- نتیجه محاسبات صفر نیست پسZF=0 است.
- مجموعه بیت‌های نتیجه فرد است، لذا PF=0 می‌باشد.
- بزرگترین بیت نتیجه (بیت علامت) صفر است، پس SF=0 یعنی نتیجه محاسبات مثبت است.

مثال: اثر دستورات 16 بیتی زیر را بر روی بیت‌های پرچم بررسی نمائید.

MOV AX, 34F5H (3)

MOV AX, 95EBH (4)

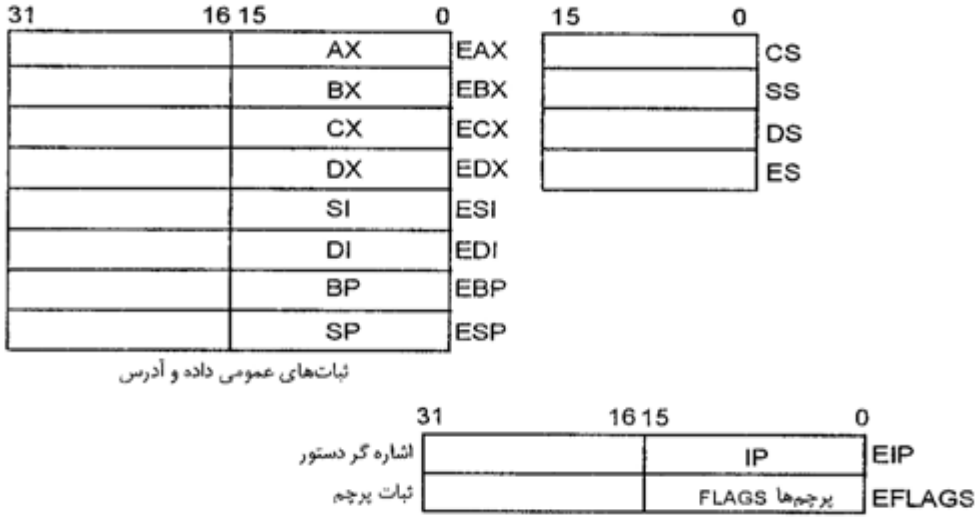
دستور 3 عدد 34F5H را وارد ثبات 16 بیتی AX می‌نماید و دستور 4 عدد 95EBH را با ثبات 16 بیتی AX طبق زیر جمع می‌نماید.



- از بیت 15 بیت نقلی ایجاد نمی‌شود، پس CF=0 است.
- از بیت 3 بیت نقلی ایجاد می‌شود لذا AF=1 می‌باشد.
- مجموع بیت‌های 8 بیت کوچک‌تر نتیجه فرد است لذا PF=0 می‌باشد.
- بزرگتر بیت نتیجه (بیت علامت) یک است، پس SF=1 می‌باشد.

باتوجه به مطالب فوق ملاحظه می‌شود که بعضی از دستورات روی بیت‌های پرچم اثر می‌گذارند و عده‌ای دیگری بر آن‌ها بی-اثرند.

در کامپیوترهای امروزی ثبات‌ها 32 بیتی می‌باشند که می‌توان به صورت ثبات‌های 8 بیتی AL، AH، BL، BH، CL، DL، DH و یا به صورت 16 بیتی AX، BX، CX، DX و یا بالاخره بصورت 32 بیتی EAX، EBX، ECX و EDX استفاده نمود.



همچنین ثبات‌های SI، DI، BP، SP 16بیتی یا 32بیتی به صورت ESI، EDI، EBP، ESP قابل استفاده‌اند و ثبات وضعیت FLAGS، 16بیتی و یا EFLAG 32بیتی می‌باشد.

کامپیوترهای امروزی دارای 32بیت آدرس هستند که تا 4 گیگابایت (2^{32} بایت) می‌توانند آدرس‌دهی کنند یا حافظه داشته باشند. کامپیوترهای مذکور در حالت عادی با سرعت بالایی کار می‌کنند و از ثبات‌های 16 بیتی استفاده می‌نمایند. ولی اگر بخواهیم با ثبات‌های 32بیتی کار کنیم، کافیست در ابتدای برنامه اسمبلی شبه دستور 486. یا 586 بکار برده شود که در این صورت کامپیوتر با ثبات‌های 32بیتی کار خواهد کرد.

سگمنت یا قطعه

برنامه اسمبلی کامپیوترهای شخصی از قطعه‌ها یا سگمنت‌های پشته، داده و کد تشکیل می‌شود. در سگمنت پشته چند خانه حافظه برای ذخیره اطلاعات و آدرس‌هایی که در اجرای برنامه اسمبلی نیاز است رزرو می‌گردد و آدرس ابتدای سگمنت پشته توسط ثبات سگمنت پشته SS، مشخص می‌شود. در سگمنت داده نیز متغیرهای برنامه قرار می‌گیرند و آدرس ابتدای سگمنت داده، توسط ثبات سگمنت داده DS، تعیین می‌شود. در سگمنت کد نیز، دستورات برنامه اسمبلی نوشته می‌شوند و آدرس ابتدای سگمنت کد، توسط ثبات سگمنت کد CS، مشخص می‌شود. البته بعضی برنامه‌ها سگمنت داده اضافی نیز دارند، که آدرس ابتدای سگمنت داده اضافی، توسط ثبات سگمنت داده اضافی ES تعیین می‌گردد.

هر سگمنت برنامه اسمبلی، در یک قطعه یا یک سگمنت حافظه قرار می‌گیرد. یک سگمنت حافظه، متناسب با نیاز تا 2^{16} بایت (یا 64 کیلو بایت) حافظه دارد. البته قطعه یا سگمنت می‌تواند در محل حافظه قرار داشته باشد ولی معمولاً از محل‌هایی از حافظه که سمت راست آدرس آن‌ها صفر هستند، شروع می‌شوند. مانند آدرس‌های 00000H، 00010H، 00020H، 00030H... یعنی آدرس شروع هر سگمنت بعدی به اندازه 10H یا 16 بایت فاصله دارد.

فصل پنجم

اجرای دستورات در محیط Debug

نرم‌افزار Debug

نرم‌افزار Debug ابزاری کمکی جهت اجرای دستورات اسمبلی و برای ایجاد، آزمایش، اجرا، تغییر و اشکال‌یابی برنامه‌ی به زبان اسمبلی است. یکی از مزایای نوشتن برنامه در محیط نرم افزار Debug این است که برنامه مستقیماً قابل اجرا بوده و نیازی به ترجمه ندارد. به این ترتیب با نرم افزار Debug می‌توان:

1- مستقیماً به زبان اسمبلی برنامه نوشت و آن را اجرا نمود.
2- برنامه آماده به زبان اسمبلی را می‌توان به محیط Debug انتقال داد و آن را اجرا نمود.
3- برنامه اسمبلی را می‌توان دستور به دستور اجرا کرد و محتویات ثبات‌ها و خانه‌ی حافظه را مشاهده و تغییرات لازم را انجام داد.
4- محتویات حلفظه سگمنت را می‌توان دید.
5- یکی از محاسن برنامه‌ی Debug این است که دستورات را به زبان اسمبلی و معادل آن‌ها، به زبان ماشین (در کد هگزادسیمال)، نشان می‌دهد.

وارد شدن و خارج شدن از Debug

برای وارد شدن یا بارکردن برنامه Debug به حافظه کافیست در محیط ویندوز گزینه Command prompt را اجرا نمائیم سپس در محیط command prompt، فایل اجرایی این نرم‌افزار، یعنی Debug را اجرا کنیم، در این صورت وارد محیط نرم افزار Debug می‌شویم که یک علامت تیره"-" ظاهر می‌شود. بعد از علامت خط‌تیره"-" می‌توان فرمان‌های نرم‌افزار Debug را برای کارهای مختلف به کار برد. جهت خارج شدن از محیط Debug از فرمان Q استفاده می‌کنیم که این فرمان باعث می-شود، به محیط Command prompt سیستم عامل برگردیم.

خلاصه‌ای از فرمان ها

فرمان‌های برنامه Debug به ما امکان عملیات متنوعی می‌دهد که عده‌ای از آن‌ها به شرح زیر می‌باشد.

1- فرمان A(Assemble): با این فرمان می‌توان دستورات زبان اسمبلی را تایپ نمود، که این کار باعث می‌شود ترجمه برنامه به زبان ماشین نیز انجام شود و در حافظه قرار گیرد.
2- فرمان C(compare) : دو بلوک اطلاعات مقایسه می‌گردد.
3- فرمان E(Enter): اطلاعاتی را در محلی از حافظه قرار می‌دهد.
4- فرمان F(Fill): برای ذخیره کردن داده‌ای، در ناحیه‌ی خاص از حافظه به کار برده می‌شود.
5- فرمان G(GO): برنامه داخل حافظه را اجرا می‌نماید.
6- فرمان H(Hexarithmetic) : جمع و تفریق عدد هگزادسیمال را انجام می‌دهد.
7- فرمان I(Input): اطلاعلت را از ورودی می‌خواند.
8- فرمان L(Load): بار کردن برنامه از دیسک به حافظه.
9- فرمان M(Move): اطلاعات را از محلی به محل دیگر حافظه انتقال می‌دهد.
10- فرمان N(Name): نامی به برنامه اسمبلی می‌دهد.
11- فرمان O(Out): اطلاعات را به پورت خروجی منتقل می‌کند.
12- فرمان P(Proceed): یک سری دستورات، بخصوص دستور وقفه را اجرا می‌نماید.

13- فرمان Q(Quit): از برنامه Debug خارج می‌شود.
14- فرمان R(Register): محتویات ثبات‌ها را نشان می‌دهد.
15- فرمان S(Search): اطلاعاتی جستجو می‌شود.
16- فرمان T(Trace): اجرای دستورات، یک دستور، یک دستور انجام می‌شود.
17- فرمان U(Unassemble): برنامه به زبان ماشین را به زبان اسمبلی تبدیل می‌کند و برنامه به زبان اسمبلی، همراه با برنامه به زبان ماشین را، نشان می‌دهد.
18- فرمان W(Write): برنامه‌ای را بر روی دیسک ذخیره می‌نماید.

خصوصیات برنامه Debug

1- برنامه Debug فرقی بین حروف کوچک و بزرگ انگلیسی نمی‌گذارد، بنابراین کاربر می‌تواند هر طور که میل دارد استفاده نماید.
2- قسمت سگمنت و افست را با علامت: جدا می‌کنیم. به عنوان مثال DS:100
3- تمام اعداد را به صورت هگزادسیمال فرض می‌نماید.
4- وضعیت بیت‌های ثبات پرچم، دارای مقادیر پرچم ذیل می‌باشند.

بیت‌های پرچم		در حالت یک	در حالت صفر
سرریز	OF(Overflow Flag)	سرریز وجود دارد OV	سرریز وجود ندارد NV
جهت آدرس	DF (Direction Flag)	آدرس کاهنده DN	آدرس افزاینده UP
وقفه	IF(Interrupt Flag)	وقفه فعال EI(Enable Interrupt)	وقفه غیرفعال DI (Disable Interrupt)
علامت	SF (Sign Flag)	نتیجه منفی NG	نتیجه مثبت PL
صفر بودن نتیجه	ZF(Zero Flag)	نتیجه صفر ZR	نتیجه غیر صفر NZ
بیت‌نقلی‌کمکی	AF (Auxiliary Carry)	وجود بیت نقلی کمکی AC	عدم وجود بیت نقلی کمکی NA
بیت توازن	PF(Parity Flag)	زوج PE	فرد PO
بیت نقلی	CF (Carry Flag)	وجود بیت نقلی CY	عدم وجود بیت نقلی NC

فصل ششم

تشکیلات و ساختار برنامه اسمبلی

مقدمه

در فصل قبل چگونگی نوشتن و اجرای دستورات برنامه اسمبلی را در محیط نرم‌افزاری Debug بررسی کردیم. ولی این عملیات در محیط نرم‌افزاری Debug محدودیت زیادی دارد و فقط برای برنامه‌های کوچک عملی است و بسیار کند می‌باشد و اصولاً این نرم‌افزار جهت اشکال‌یابی برنامه‌ها طراحی شده است. راه عملی نوشتن برنامه‌ی اسمبلی، تایپ برنامه در یک محیط ویرایشگر (مانند EDIT در سیستم عامل، یا در محیط NC و ...) و اسمبل کردن آن توسط برنامه‌ی مترجم MASM یا TASM و سپس پیوند دادن آن توسط پیوند دهنده LINK یا TLINK است، که در این مرحله برنامه‌ی نهایی به صورت EXE یا COM تولید می‌شود و قابلیت اجرا توسط سیستم عامل را پیدا می‌کند. ولی برای اجرای عملیات مذکور، یک سری مقرراتی باید رعایت شود تا اینکه برنامه مترجم اسمبلر بتواند برنامه اسمبلی را ترجمه نماید و سپس برنامه پیوند دهنده بتواند آن‌ها را پیوند دهد که قابل اجرا گردد.

قواعد بر چسب دستور و نام متغیرها

اصولاً برچسب، آدرس دستور در حافظه است و نام متغیر نیز، آدرس اطلاعات در حافظه می‌باشد. به عنوان مثال در دستور:

BACK1: MOV AH, OEH

BACK1: که برچسب دستور است، در حقیقت نام سمبولیک آدرس حافظه‌ای است که دستور MOV AHوOEH در آن قرار دارد و البته آدرس حافظه یک عدد است، که ما برای درک بهتر برنامه، به آن یک نام سمبولیک اختیاری مانند BACK1 داده‌ایم.

نکته1: برچسب دستور حتماً باید دو نقطه داشته باشد مانند :BACK

نکته 2: نام متغیرها دو نقطه ندارند ولی برچسب دستورها دو نقطه دارند.

برنامه اسمبلی چهار نوع برچسب و یا نام دارد که عبارتند از:

1- برچسب دستور: نام سمبولیک به آن دستور می‌دهد که با استفاده از آن نام می‌توان به آن دستور رجوع نمود.
2- برچسب یا نام روال: نام سمبولیک به یک روال یا یک سابروتین می‌دهد که این نام باید در شبه دستور PROC و ENDP یکسان باشد.
3- نام سگمنت: نامی به یک سگمنت نسبت می‌دهند و این نام در دستورات، برای مراجعه به این متغیرها استفاده می‌شوند.
4- نام متغیرها: نامی است که به داده‌ها نسبت می‌دهند و این نام در دستورات، برای مراجعه به این متغیرها مورد استفاده قرار می‌گیرد. البته شبه دستورات DB، DW، DD، DQ و DT برای تعریف این متغیرها استفاده می‌شوند.

راهنما یا شبه دستور

برنامه‌ی مترجم، یا اسمبلر دارای فرمان‌هایی است که ما را در کنترل اسمبل کردن یا ترجمه نمودن و تهیه لیست برنامه یاری می‌نماید. این فرمان‌ها به شبه‌دستور، یا راهنمای اسمبلر معروفند و کد زبان ماشین ندارند و فقط در زمان اسمبل یا ترجمه کردن برنامه، عمل می‌نمایند. با استفاده از این شبه‌دستورات یا راهنماهای برنامه اسمبلر، ما می‌توانیم به برنامه یک نام بدهیم و شکل صفحات برنامه را مشخص کنیم و برنامه را به سگمنت‌های کد، داده، پشته و ... تقسیم نمائیم و ابتدا و انتهای آنها را

مشخص کنیم. شروع و پایان برنامه را مشخص نمائیم و .... که معمول‌ترین شبه‌دستورات یا راهنماهای اسمبلر به شرح ذیل می‌باشند.

شبه‌دستورات PAGE و TITLE

برنامه‌ی مترجم اسمبلر تعدادی شبه‌دستور، برای کنترل لیست برنامه‌ی اسمبلی دارد از جمله شبه‌دستورهایی که به این منظور می‌توانیم در هر برنامه استفاده کنیم PAGE و TITLE می‌باشند. با شبه‌دستور PAGE در شروع یک برنامه می‌توانیم، تعداد خطوط روی یک صفحه و تعداد ماکزیمم حرف روی خط را برای چاپگر تعیین نمائیم.

شکل کلی این شبه‌دستور به صورت: PAGE [Length],[Width] که Length تعداد خطوط در صفحه و Widthتعداد حروف در خط را معین می‌کند. مثلاً در شبه‌دستور: PAGE 40,80 تعداد ماکزیمم خطوط یک صفحه، 40 و تعداد حروف هر خط 80 تعیین شده است. برای این که یک عنوان در بالای هر صفحه، در سطر 2 از برنامه چاپ شود، از شبه دستور TITLE که شکل کلی آن بصورت زیر است استفاده می‌کنیم: TITLE Text [Comment] یک روش این است که به جای Text از نام برنامه‌ای که بر روی دیسک ذخیره کرده‌ایم استفاده کنیم.

شبه‌دستور END

این شبه‌دستور نقطه‌ی انتهای یک برنامه را به مترجم اسمبلر اعلام می‌دارد و شکل کلی آن به صورت: END [Entry-Point Label] و یا [برچسب نقطه شروع دستورات برنامه] END می‌باشد که جلوی END باید برچسب یا LABEL نقطه شروع دستورات برنامه اسمبلی باشد.

شبه‌دستورات تعریف قطعه، یا سگمنت SEGMENT و ENDS

همان‌طور که قبلاً ذکر شد، یک برنامه اسمبلی در کامپیوترهای شخصی از یک یا چند سگمنت تشکیل می‌شود که سگمنت پشته، محل حافظه پشته است و سگمنت داده و سگمنت کد به ترتیب محل‌های ذخیره داده‌ها و دستورات برنامه می‌باشد. شبه‌دستوری که ابتدای سگمنت را مشخص می کندSEGMENT و شبه‌دستوری که انتهای سگمنت را مشخص می‌نماید ENDS، نام دارد. شکل کلی شبه‌دستورتعریف سگمنت به صورت:

Name SEGMENT [Align] [Combine] [Class] شروع سگمنت

اختیاری

دستورات

یا

داده ها

Name ENDS پایان سگمنت

با این شبه‌دستور می‌توانیم یک سگمنت را تعریف و ابتدا و انتهای آن را مشخص نمائیم. لازم به ذکر است که حداکثر ظرفیت هر سگمنت 64k بایت است. هر سگمنت باید یک نام داشته باشد که قبل از شبه‌دستور SEGMENT و ENDS قرار می-گیرد. البته نام سگمنت اختیاری است. همان‌طوری که ملاحظه می‌شود نام اختیاری سگمنت قبل از شبه-دستورات SEGMENT و ENDS، که به ترتیب ابتدا و انتهای یک سگمنت داده را مشخص می‌کنند، بصورت یکسان آورده شده است. این شبه‌دستور می‌تواند سه عملوند داشته باشد که ذیلاً به شرح آن‌ها می‌پردازیم.

الف) هم‌ترازی Align

عملوند مذکور اختیاری است و جهت انطباق شروع یک سگمنت با آدرس حافظه، در مواقعی که چندین برنامه با هم پیوند داده می‌شوند به کار می‌رود. این عملوند می‌تواند یکی از کلمات BYTE، WORD، PARA و PAGE باشد که هر کدام معنای خاص خود را در ارتباط با شروع سگمنت دارد.

BYTE: سگمنت از هر آدرس می‌تواند شروع کند.

WORD: سگمنت از هر آدرس زوجی می‌تواند شروع کند.

PARA: سگمنت از هر آدرس که قابلیت تقسیم بر16 را داشته باشد می‌تواند شروع شود (این شبیه‌دستور پیش‌فرض برنامه اسمبلی می‌باشد و نیازی به تکرار آن نیست). در این صورت اگر سگمنتی به آدرس مثلاً 00024H پایان یابد، سگمنت بعدی از آدرس00030H شروع می‌شود که قابل تقسیم بر 10Hاست به عبارت دیگر کوچکترین رقم آدرس سگمنت در این نوع ترازبندی صفر می‌باشد.

PAGE: سگمنت از هر آدرس که قابل تقسیم بر 256(100H) را داشته باشد می‌تواند شروع شود. در برنامه‌های معمولی می-توان کلاً این قسمت از شبه‌دستور SEGMENT را حذف نمود.

ب) ترکیب Combine

این عملوند چگونگی ترکیب سگمنت در حال تعریف با سگمنت‌های همنام در سایر برنامه‌ها را مشخص می‌کند به عبارت دیگر مشخص می‌نماید که در موقع پیوند برنامه، آیا این سگمنت با سایر سگمنت‌ها باید پیوند داده شوند یا نه. عملوند Combine اکثراً یکی از کلمات public یا STACK می‌تواند باشد که در موقع پیوند، در حافظه به طور متوالی قرار می‌دهد و تشکیل یک سگمنت بزرگتری را می‌نماید. لذا در برنامه‌های معمولی که فقط یک سگمنت پشته و داده و کد دارند، نیازی به کلمه PUBLIC نیست.

کلمه‌ی STACK فقط در سگمنت پشته استفاده می‌شود و گذاشتن آن در برنامه اجباری است. به عنوان مثال:

Name SEGMENT STACK

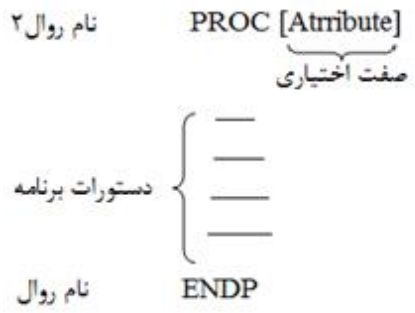
مشخص می‌نماید در موقع پیوند برنامه، سگمنت پشته برنامه کاربر با سگمنت پشته سیستم‌عامل ترکیب شود و یک سگمنت پشته واحد برای اجرای برنامه درست کند.

ج) کلاس Class

کلمه‌ی کلاس در موقع پیوند برنامه‌ها، برای ترکیب سگمنت‌های از یک نوع به کار می‌رود. به عنوان مثال سگمنت داده از یک برنامه، با سگمنت داده از برنامه دیگر و سگمنت کد از یک برنامه با سگمنت کد از برنامه دیگر باهم ترکیب می‌شوند. برای این منظور کلمات 'CODE و 'DATA' و 'STACK' در کوتیشن به ترتیب برای سگمنت کد، داده و پشته استفاده می‌شوند.

شبه‌دستورات روال شامل PROC، ENDP، FAR، NEAR

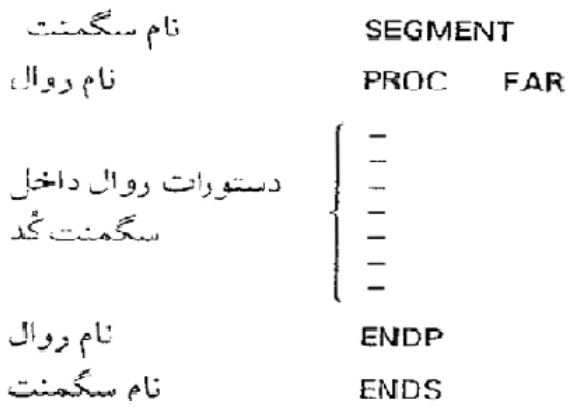
سگمنت کد که حاوی دستورات برنامه است حداقل یک یا چند روال دارد که با شبه‌دستور PROC تعریف می‌شود شکل کلی شبه‌دستور PROC به صورت زیر می‌باشد.



که صفت Attribute می‌تواند FAR و یا NEAR باشد.

شبه‌دستور PROC و ENDP ابتدا و انتهای یک روال یا سابروتین را مشخص می‌نماید و دارای نام برچسب یکسانی هستند. سابروتین یا روال به مجموعه‌ای از دستورات اطلاق می‌شود که کار معینی را انجام می‌دهد و می‌تواند از نقاط مختلف برنامه فراخوانی و اجرا شود و برنامه‌نویس را از تکرار دستورات رهایی بخشد.

اگر سگمنت کد فقط دارای یک روال باشد طبق زیر:



تعریف می‌شود. نام روال باید مشخص و منحصر به فرد باشد و از مقررات نامگذاری برنامه‌ی اسمبلی پیروی نماید. برای صفت در اینجا کلمه FAR استفاده شده است، که یک صفت برای روال می‌باشد و شبه دستور PROC FAR در یک برنامه EXE، برای سیستم‌عامل مشخص می‌کند که این نقطه ابتدای برنامه اصلی کاربر است و دستورات از اینجا به بعد باید اجرا شوند. شبه‌دستور ENDP انتهای روال را مشخص می‌کند و باید دارای همان نامی باشد که در شبه‌دستور PROC به کار برده شده است و شبه دستور ENDS پایان سگمنت را اعلام می‌کند و می‌بایستی دارای همان نامی باشد که در جلوی شبه‌دستور SEGMENT است.

سابروتین یا روال‌ها به دو دسته داخلی و خارجی تقسیم می‌شوند. اگر روال فقط از داخل سگمنتی که این روال تعریف شده قابل فراخوانی باشد، روال یا سابروتین را داخلی نامند و در شبه‌دستور PROC باید از صفت NEAR استفاده شود. درصورتی که روال تعریف شده بتواند از سگمنت‌های دیگر نیز فراخوانی شود، مثلاً از قطعه‌ای که سیستم‌عامل در آن قرار دارد، آن‌گاه روال را خارجی نامند و در شبه‌دستور PROC اگر صفت قید نشود، اسمبلر صفت NEAR را به طور پیش‌فرض در نظر می-گیرد.

شبه‌دستور ASSUME

شبه‌دستور ASSUME ارتباط بین نام هر سگمنت و ثبات‌های سگمنت را برقرار می‌کند. پروسسور، ثبات سگمنتSS را برای آدرس شروع سگمنت پشته، ثبات سگمنت DS را جهت آدرس شروع سگمنت داده و ثبات سگمنت CS را برای آدرس شروع سگمنت کد استفاده می‌کند. شبه‌دستور ASSUME در سگمنت کد بلافاصله قبل از شبه‌دستور PROC قرار می‌گیرد که ثبات‌های سگمنت را برابر نام سگمنتی که در برنامه بکار رفته قرار دهد.

برنامه‌ی اسمبلر می‌تواند دارای یکی یا چند سگمنت داده باشد. ولی چون فقط یک ثبات، برای هر سگمنت وجود دارد، لذا در هر لحظه فقط یکی از سگمنت‌ها توسط CPU می‌تواند دستیابی شود. پس ASSUME در هر لحظه به اسمبلر می‌گوید، در حال حاضر کدام سگمنت (که توسط شبه‌دستور SEGMENT تعریف شده‌اند) به کار برده می‌شود. همچنین اسمبلر را راهنمایی می‌کند که افست آدرس متغیرها از ابتدای سگمنت داده و افست آدرس دستورات از ابتدای سگمنت کد چقدر است. این شبه دستور به صورت زیر می‌باشد:

ASSUME SS:نام سگمنت پشته، DS:نام سگمنت داده، CS: نام سگمنت کد

ساختار اصولی یا ساختار استاندارد یک برنامه‌ی اسمبلی

همان‌طوری که قبلاً بحث شد، هر برنامه اسمبلی برای کامپیوترهای شخصی می‌تواند دارای سگمنت پشته، سگمنت داده و سگمنت کد باشد که دستورات کامپیوتر در سگمنت کد قرار می‌گیرد. علاوه بر این برنامه اسمبلی دارای یک عنوان یا نام است در ضمن سگمنت پشته، داده و کد، باید دارای نام، با شبه‌دستورات SEGMENT و ENDS باشد. شکل زیر ساختار اصولی یک برنامه‌ی اسمبلی با سگمنت پشته به نام STACKSG را نشان می‌دهد. سگمنت داده به نام DATASG و سگمنت کد نیز با نام CODESG تعریف شده‌اند.

```
        PAGE 110,100
TITLE   'nemoune1.asm' an EXE program
;-----------------------------------------------
;               1- Define stack segment
;               -----------------------
STACKSG SEGMENT STACK 'STACK'
        DW 32H DUP(0)     ; 32H word for stack
STACKSG ENDS              ; End of segment
;-----------------------------------------------
;               2- Define data segment
DATASG SEGMENT  'DATA'
;
;         متغیرها یا داده‌های برنامه
;         ----------------
;
DATASG  ENDS              ;End of segment
;-----------------------------------------------
;               3- Define code segment
;               ----------------------
;
CODESG  SEGMENT  'CODE'
        ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
MAIN    PROC FAR
        MOV AX,DATASG     ;Initialize DS
        MOV DS,AX         ;  register
;
;               دستورات برنامه
;         ----------------------
;
        MOV AX,4C00H      ; End of
        INT 21H           ; processing
MAIN    ENDP              ; End of procedure
CODESG  ENDS              ; End of segment
        END MAIN          ; End of program
```

تعریف متغیر و تخصیص مقدار اولیه به آن در سگمنت داده‌ی برنامه‌ی اسمبلی

برای تعریف متغیرها و تخصیص مقدار اولیه به آن‌ها، از شبه دستور DB، DW، DD، DQ، DT و DUP استفاده می‌شود که در سگمنت داده قرار می‌گیرند. فرم کلی تعریف متغیرها به صورت زیر است:

[نام] Dn عملوند

که نام داخل کروشه یک نام سمبولیک و اختیاری است. ولی اگر دستورات برنامه به آن رجوع نمایند نام مذکور لازم می‌باشد. در ذیل هر یک از شبه‌دستورات فوق و کاربرد آن‌ها تشریح می‌گردد.

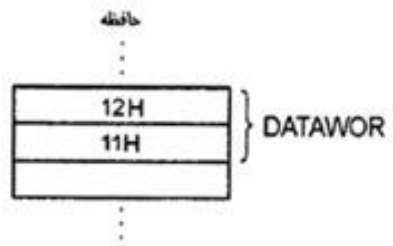
الف) شبه‌دستور DB

این شبه‌دستور در سگمنت داده تعریف می‌شود و یک بایت حافظه برای متغیرها تخصیص می‌دهد. به عنوان مثال شبه‌دستور: COUNT DB 13H باعث می‌شود که برنامه‌ی مترجم اسمبلر، خانه حافظه‌ای به نام متغیر COUNT، یا آدرس COUNT در حافظه رزرو نماید و مقدار آن را 13H قرار دهد.

ب) شبه‌دستور DW

این شبه‌دستور در سگمت داده تعریف می‌شود و دو بایت حافظه برای متغیرها تخصیص می‌دهد.

به عنوان مثال: DATAWORD DW 1112H



برنامه‌ی مترجم اسمبلر دو بایت حافظه از آدرس DATAWOR را برای متغیر مذکور رزرو نماید که در بایت اول مقدار کوچکتر عدد یعنی 12Hو در بایت بعدی مقدار بزرگتر عدد یعنی 11Hرا قرار می‌دهد.

ج) شبه‌دستورات DD، DQ وDT برای تعریف متغیرها

این شبه‌دستورات در سگمنت داده برنامه‌ی اسمبلی برای تعریف متغیرها به کار می‌روند که شبه‌دستور DD تعداد دو کلمه (چهار بایت) و DQ تعداد تعداد چهار کلمه (هشت بایت) و DT تعداد 10بایت، برای متغیرها در نظر می‌گیرند. به عنوان مثال: DWORD5 DD 32445678H چهار بایت در حافظه رزرو می‌کند که در بایت اول78H، در بایت بعدی56 و در بایت‌های بعدی به ترتیب 44 و 32 قرار خواهند گرفت. به همین ترتیب در تعریف DQ به صورت DWORD5 DD 32445678H مقدار چهار کلمه یا 8 بایت آدرس در نظر می‌گیرد.

د) شبه‌دستور DUP

با این شبه‌دستور در سگمنت داده، می‌توان برای نوشتن یک سری اطلاعات مساوی در متغیرها استفاده نمود در غیر این صورت باید اطلاعات مذکور را چندین بار تکرار نمود. به عنوان مثال

DATA1 DB 20 DUP(80)

20 بار عدد 80 را برای متغیر DATA1 تکرار می‌کند.

اصولاً شبه‌دستور DUP به ما این امکان را می‌دهد که عبارت ساده‌تری برای متغیرها بنویسیم.

بررسی اطلاعات سگمنت داده در حافظه

برای مشاهده‌ی نحوه قرار گرفتن اطلاعات متغیرها در حافظه، نرم افزار Debug اجرا کرده و سپس محتوای سگمنت داده توسط فرمان D DS:0 بر روی مانیتور نشان داده می‌شود.

ساختار ساده شده برنامه‌ی اسمبلی

ساختار برنامه‌ی اسمبلی را می‌توان ساده‌تر نمود به شرطی که از رهنمای اسمبلی MODEL SMALL استفاده شود. راهنمای اسمبلی سگمنت‌های پشته، داده و کد را به صورت زیر تعریف می‌کند:

.STACK

.DATA

.CODE

که هر یک از راهنماهای اسمبلر فوق به طور خودکار، تولید راهنماهای اسمبلر SEGMENT و ENDS را می‌کنند و دیگر نیازی به نوشتن آن‌ها نیست. البته در این ساختار، دستوراتی که ثبات سگمنت داده DS را مقدار اولیه می‌دهند به صورت زیر می‌باشند:

MOV AX,@data ;Set the address of data

MOV DS, AX ;segment to DS

```
        PAGE 100,110
TITLE   'nemoune3.asm' a simplified program
;-----------------------------------------------
        .MODEL SMALL
        .STACK  64          ;Define stack
        .DATA               ;Define data segment
;
;               تعریف متغیرها یا داده‌های برنامه
;               ----------------
;
;-----------------------------------------------
        .CODE               ;Define code segment
MAIN    PROC FAR
        MOV AX,@data        ;1- Set data segment
        MOV DS,AX           ;2- address
;
;
;                   دستورات برنامه
;               ----------------------
;
;
        MOV AX,4C00H        ; End of
        INT 21H             ; processing
MAIN    ENDP                ; End of procedure
        END MAIN            ; End of program
```

فصل هفتم

مراحل ایجاد، اسمبل، پیوند و اجرای برنامه‌های اسمبلی

مقدمه

برنامه‌ای که با دستورات برنامه اسمبلی نوشته می‌شود، برنامه اسمبلی اصلی نامیده می‌شود و با پسوند ASM تعریف می‌گردد. از برنامه مترجم (اسمبلر) برای ترجمه برنامه‌های اسمبلی به زبان ماشین استفاده می‌کنیم که این برنامه با پسوند OBJ مشخص می‌گردد. بالاخره برنامه پیوند دهنده، برنامه‌های با پسوند OBJ را با تشخیص آدرس‌های حافظه‌ی مناسب برای آنها، به یک برنامه قابل اجرای EXE تبدیل می‌نماید.

اسمبل یا ترجمه برنامه اسمبلی

برنامه‌های اسمبلی تا زمانی که به زمان ماشین ترجمه نشوند قابل اجرا نیستند. برای ترجمه می‌توان از برنامه ماکرو اسمبلر MASM یا توربو اسمبلر TASM استفاده نمود.

اسمبل یا ترجمه‌ی برنامه با توربو اسمبلر TASM

برای این کار کافی است در محیط ویندوز، گزینه Command Prompt را اجرا نمائیم سپس در محیط Command Prompt برنامه مترجم مذکور را اجرا نماییم. برنامه توربو اسمبلر (TASM) با دادن پیغام زیر نام برنامه اسمبلی کاربر را با پسوند ASM سئوال می‌کند. به عنوان مثال اگر برنامه قبلی ما به زبان اسمبلی به نام EXAMPLE2.ASM باشد، دستور زیر را می‌نویسیم:

Source Filename[EXAMPLE2.ASM]:

سپس نام برنامه OBJ یا ترجمه شده بصورت زیر سئوال می کند.

Object Filename [EXAMPLE2.OBJ]:

که ما نام EXAMPLE2.OBJ را برای آن انتخاب کردیم.

بعد مطابق ذیل سئوال می‌کند، آیا برنامه‌ی لیست که شامل اصل برنامه و ترجمه آن و آدرس‌های متناظر هر دستور است لازم می‌باشد یا نه؟.

Source listing [NUL.LST]:

در صورتی که لیست برنامه لازم باشد، نام برنامه لیست را جلوی آن قرار می‌دهیم مانند:

Source listing [NUL.LST]: EXAMPLE2:LST

پیوند برنامه با دستور Link

برای این کار کافی است که بعد از تولید فایل ترجمه شده با پسوند OBJ را با دستور Link به زبان ماشین معادل تبدیل کنیم. شکل کلی دستور Link به صورت زیر می‌باشد:

Link *.OBJ

با اجرای این دستور فایل معادل زبان ماشین با پسوند اجرایی (Exe) تولید خواهد شد.

فصل هشتم

دستورات زبان اسمبلی در کامپیوترهای شخصی

اپراندها یا عملوندهای دستور و روش‌های آدرس‌دهی اطلاعات

اپراند یا عملوند، اطلاعاتی است که دستور بر روی آنها عملیات انجام می‌دهد. بعضی از دستورات مانند CLC (صفرکردن بیت نقلی پرچم CF) و RET (برگشت از سابروتین) اصلاً دارای اپراند یا عملوند نیستند در صورتی که عده‌ایی از دستورات کامپیوتر نیاز به یک منبع اطلاعات، به نام عملوند یا اپراند دارند که بر روی آنها عملیات انجام شود. مانند دستور: Not Al که یک عملوند AL دارد و محتوای ثبات AL را بیت به بیت معکوس می‌کند. اگر در دستور دو عملوند وجود باشد عملوند دوم را منبع و عملوند اول را مقصد نامند، مانند: Mov Al,23 که23 عملوند منبع و AL عملوند مقصد است، یعنی عدد 23 به ثبات AL منتقل می‌شود. منبع اطلاعات می‌تواند یک عدد ثابت باشد و مستقیما در دستور قرار داده می‌شود که آن‌را عملوند بلافاصله نامند و یا منبع اطلاعات ممکن است آدرس یک ثبات یا آدرس خانه‌ی حافظه باشد. عملوند اول که مقصد نامیده می-شود ممکن است شامل اطلاعات داخل یک ثبات، یا خانه حافظه باشد. به این ترتیب شکل کلی یک دستور اسمبلی به صورت زیر است:

عملوند ۲ ، عملوند۱	کد اجرای دستور : برچسب
یا	
منبع ، مقصد	کد اجرای دستور : برچسب

به عنوان مثال در دستور:

STRAT: MOV AX , BX

محتوای ثبات BX به ثبات AX انتقال می‌یابد. در دستور مذکور Mov کد اجرا، START برچسب دستور، ثبات BX عملوند2 که منبع اطلاعات است و ثبات AX عملوند است که مقصد اطلاعات می‌باشد (در اینجا نام ثبات‌ها در حقیقت آدرس ثبات‌ها می‌باشد که CPU آنها را می‌شناسد).

روش‌های مختلفی که پروسسور می‌تواند به اطلاعات یا اپراند دسترسی داشته باشد، روش‌ها یا حالت‌های آدرس‌دهی نامیده می‌شوند. تعداد حالت‌های آدرس‌دهی در زمان طراحی CPU تعیین می‌گردد که در کامپیوترهای شخصی روش‌های آدرس-دهی اپراندها عبارتند از:

1. حالت آدرس‌دهی بلافاصله
2. حالت آدرس‌دهی ثبات‌ها
3. حالت آدرس‌دهی مستقیم
4. حالت آدرس‌دهی غیرمستقیم از طریق ثبات
5. حالت آدرس‌دهی غیرمستقیم نسبی با ثبات پایه
6. حالت آدرس‌دهی غیرمستقیم نسبی با ثبات اندیس
7. حالت آدرس‌دهی غیرمستقیم با ثبات‌های پایه واندیس

دستور انتقال اطلاعات

این دستورات وظیفه‌ی جابجایی اطلاعات بین ثبات‌ها و خانه‌های حافظه را بر عهده دارند که در زیر هر یک از دستورات با ذکر مثال‌هایی شرح داده می‌شوند.

دستور MOV:

شکل کلی آن به صورت زیر است:

MOV Destination, Source

به وسیله این دستور اطلاعات منبع به مقصد منتقل می‌شود. به عبارت دیگر محتوای منبع در محل مقصد کپی می‌شود. به عنوان مثال دستور:

MOV AX,BX ; AX ← BX

محتوای ثبات BX را که یک کلمه می‌باشد، به ثبات AX منتقل می‌کند که در اینجا ثبات BX منبع و ثبات AX مقصد می-باشد.

نکته 1: هر دو اپراند دستور MOV همزمان نمی‌توانند عدد باشند. علاوه بر این اپراند اول نیز نمی‌تواند عدد باشد.

نکته 2: اگر عدد 8 بیتی کمتر ازFFH به ثبات 16 بیتی منتقل شود، در بقیه 8 بیت بزرگتر ثبات مذکور صفر قرار می‌گیرد مانند دستور MOV AX,5 که در نتیجه AX=0005 می‌شود یعنی AH=00 و AL=05 می‌گردد.

نکته 3: با این دستور نمی‌توان اطلاعات را مستقیماً بین دو خانه از حافظه انتقال داد. برای حل این مساله باید ابتدا اطلاعات یک خانه حافظه به یک ثبات منتقل شود سپس اطلاعات آن ثبات به خانه حافظه دیگر انتقال یابد. به عنوان مثال برای انتقال اطلاعات خانه حافظه DATA1 به خانه حافظه DATA2 دستورات زیر را می‌نویسیم.

MOV AL,DATA1 (1)
MOV DATA2,AL (2)

نکته 4: با دستورMOV نمی‌توان عددی را مستقیماً وارد ثبات سگمنت SS,ES,DS نمود. برای حل این مساله، باید ابتدا عدد را با دستور MOV وارد یکی از ثبات‌های Ax، Bx، Cx و یا Dx نمود و سپس توسط دستور MOV دیگر، عدد مذکور را از ثبات مربوطه به ثبات سگمنت منتقل نمود.

MOV AX,2540 (3)
MOV DS,AX (4)

نکته 5: تعداد بیت عملوند اول نمی‌تواند از تعداد بیت عملوند دوم کمتر باشد، به عنوان مثال:

MOV CL,AX اشتباه است، چون AX شانزده بیتی است و در CL که هشت بیتی است جا نمی‌گیرد.

دستور انتقال افست آدرس LEA

شکل کلی آن بصورت زیر است

LEA Register ,Source Offset Address

این دستور باعث می‌شود که افست آدرس منبع اطلاعات به یک ثبات منتقل شود. این دستور بر بیت‌های پرچم اثر ندارد. دستور مذکور معمولاً برای بار کردن یک افست آدرس در ثبات‌های اشاره‌گر BX، DI، BP و یا SI استفاده می‌شود.

به عنوان مثال:

LEA SI,Source

دستور XCHG

باعث می‌شود که محتویات یک ثبات یا یک خانه حافظه با محتویات یک ثبات مشخص دیگر تعویض شود. شکل کلی این دستور به صورت زیر است:

XCHG	Operand1 , Operand2	Operand1 ↔ Operand2
	ویا	
XCHG	اُپراند۱ ، اُپراند۲	

یعنی محتویات اپراند2 با اپراند1 جابجا می‌شوند.

به عنوان مثال: XCHG AL,BL

دستور ورودی IN

اطلاعات یک بایت یا یک کلمه را از یک دستگاه ورودی به ثبات AL یا AX منتقل می‌کند

شکل کلی آن بصورت زیر است:

IN AL,PORT

IN AX,PORT

در اینجا PORT آدرس دستگاه ورودی به صورت عدد است که از 00 تا FF یعنی حداکثر تا 256 می‌تواند باشد و AL یا AX ثبات‌های پروسسور هستند. (این دستور بر بیت‌های پرچم اثر ندارند)

به عنوان مثال: IN AX,12

دستور خروجی OUT

باعث می‌شود که یک بایت محتوای ثبات AL و یا دو بایت محتوای ثبات AX به دستگاه خروجی که آدرس آن به صورت عدد از 00 تا FFH است منتقل شود. (این دستور بر بیت پرچم اثر ندارد.)

به عنوان مثال: OUT 32,AX

فصل نهم

دستورات محاسباتی و کاربرد آن‌ها

دستورات محاسباتی

دستورات محاسباتی اصلی شامل جمع ADD، تفریق SUB، ضریب MUL و تقسیم DIV بر روی اطلاعات باینری می‌باشند که در اثر انجام این محاسبات مقدار بیت‌های پرچم ممکن است تغییر نماید. در این دستورات می‌توان هریک از روش‌های آدرس‌دهی را به کار برد.

دستور جمع ADD

شکل کلی دستور جمع ADD به صورت زیر است:

```
ADD Destination, Source   ; Destination = Destination + Source
                          یا
ADD   اپراند منبع،اپراند مقصد   ; اپراند مقصد = اپراند مقصد + اپراند منبع
```

که با این دستور اطلاعات منبع با مقصد جمع و نتیجه در محل مقصد قرار می‌گیرد. این دستور بر بیت‌های پرچم PF، OF، CF، SF، AF اثر می‌گذارد. به عنوان مثال: ADD AL: BL

دستور تفریق SUB

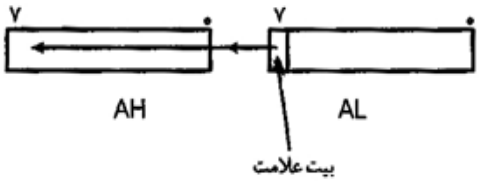
شکل کلی دستور به صورت زیر می باشد:

```
ADD Destination, Source   ; Destination = Destination - Source
                          یا
ADD   اپراند منبع،اپراند مقصد   ; اپراند مقصد = اپراند مقصد - اپراند منبع
```

در این صورت اپراند منبع از مقصد کسر و نتیجه در مقصد قرار می‌گیرد. این دستور بر بیت‌های پرچم SF،OF ، ZF، AF، PF، CF اثر می‌گذارد. به عنوان مثال: SUB AX,CX

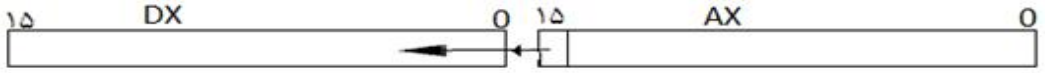
دستورات CBW و CWD

این دستورات را می‌توان برای تبدیل یک بایت به یک کلمه و یا یک کلمه به کلمه مضاعف به کار برد. برای این منظور دستور CBW، بیت علامت ثبات AL را در ثبات AH کپی می‌نماید.




```
MOV   AL, +96    (۱)  ; AL=01100000
CBW              (۲)  ; AH=00000000 و AL=01100000
```

دستور CWD باعث می‌شود که بیت علامت ثبات AX در ثبات DX کپی‌گردد.



به عنوان مثال در دستورات :

```
MOV  AX,0104H  (5) ;  AX=0000 0001 0000 0100
CWD            (6) ;  DX = 0000H , AX = 0104H
                                  ویا
MOV  AX,-32766 (7) ; AX = 1000 0000 0000 0010 یا AX = 8002H
CWD            (8) ; DX = FFFF  , AX = 8002H
```

دستور(6) باعث می‌شودکه بیت علامت AX که برابر صفر است در تمام بیت‌های DX کپی گردد یعنی DX=0000H گردد.
دستور(8) نیز باعث می‌شود که بیت علامت AX که برابر یک است در تمام بیت‌های DX کپی گردد، یعنی DX = FFFFH شود.

دستور ضرب MUL

در دستور ضرب دو عدد، متناسب با نیاز، ثبات‌های AL، AH، AX، DX به کار برده می‌شوند. عملیات ضرب چندین حالت ممکن است داشته باشد.

1- ضرب یک بایت در یک بایت
2- ضرب یک کلمه در یک کلمه
3- ضرب یک بایت در یک کلمه

الف) ضرب یک بایت در یک بایت:

اگر بخواهیم یک بایت را در یک بایت ضرب کنیم یکی از اپراندها باید AL باشد که این ثبات در دستور ضرب نوشته نمی‌شود و اپراند دیگر می‌تواند یک ثبات و یا یک خانه حافظه باشد که در دستور قرار می‌گیرد. بعد از عمل ضرب، نتیجه در دو بایت و در ثبات AX قرار داده می‌شود.

شکل کلی دستور ضرب به صورت زیر است:

```
MUL     Operand
      و یا
MUL     عملوند
```

که عملوند مقداری است که قرار است در AL یا AX ضرب شود. این دستور بر بیت‌های پرچم CF و OF اثر می‌گذارد.

به عنوان مثال دستور: MUL BL

ب) ضرب یک کلمه در یک کلمه:

در این حالت یکی از اپراندها باید در ثبات AX و اپراند دیگر می‌تواند در ثبات یا در حافظه باشد. 16 بیت پرارزش نتیجه ضرب در DX و 16 بیت کم ارزش یا کلمه کم ارزش‌تر در ثبات AX قرار می‌گیرد. به عنوان مثال در دستور MUL CX محتوای ثبات CX در ثبات AX ضرب و نتیجه در ثبات‌های AX و DX قرار می‌گیرد.

ب: ضرب یک بایت در یک کلمه:

در صورتی که بخواهیم یک بایت را در یک کلمه ضرب کنیم ابتدا از دستور CBW برای تبدیل یک بایت به کلمه سپس از ضرب کلمه در کلمه استفاده می‌کنیم. به عنوان مثال اگر بخواهیم یک بایت در خانه‌ی حافظه به افست آدرس BYTE1 را در یک کلمه‌ی حافظه به افست آدرس WORD1 ضرب کنیم از دستورات زیر استفاده می کنیم:

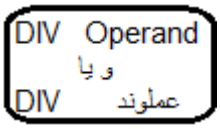
MOV AL,BYTE (1)

CBW (2)

MUL WORD1 (3)

دستور تقسیم DIV

با دستور تقسیم DIV، عددی که داخل ثبات AX است بر روی اپراند یا عملوند یک بایتی که در دستور نوشته شده، تقسیم می‌شود و خارج قسمت در ثبات AL و باقیمانده در ثبات AH قرار می‌گیرد. شکل کلی این دستور به صورت زیر است:



و اگر عملوند دو بایتی (کلمه) باشد، محتوای DX:AX بر آن تقسیم و خارج قسمت در AX و باقیمانده در DX قرار می‌گیرد. اثر این دستور بر بیت‌های پرچم غیر قابل پیش‌بینی است. به عنوان مثال دستور DIV CL باعث می‌شود که محتوای ثبات AX، بر محتوای ثبات CL تقسیم و خارج قسمت در ثبات AL و باقیمانده در ثبات AH قرار گیرد.

همانطور که گفته شد در صورتیکه یک عدد 32 بیتی که در ثبات‌های DX AX قرار دارد را بخواهیم بر یک عدد 16 بیتی که در دستور مشخص شده تقسیم کنیم در این صورت بعد از اجرای دستور تقسیم خارج قسمت در ثبات AX و باقیمانده در ثبات DX قرار خواهد گرفت. به عنوان مثال دستور DIV BX باعث می‌شود، محتوای ثبات AX وDX بر ثبات BX تقسیم شود و خارج قسمت در ثبات AX و باقیمانده در ثبات DX قرار گیرد.

دستور شمارنده صعودی INC

با این دستور به محتوای یکی از ثبات‌ها، یا یک خانه حافظه یک واحد اضافه می‌شود. شکل کلی این دستور به صورت زیر است:

Inc Operand

این دستورات بر بیت های پرچم OF، SF،ZF، AF، PF اثر می‌گذارد و CF تغییر نمی‌نماید. به عنوان مثل دستور INC CL یک واحد بر محتوای ثبات CL می افزاید.

دستور شمارنده نزولی DEC:

با این دستور محتوای یکی از ثبات‌ها یا خانه حافظه، یک واحد کم می‌گردد. شکل کلی این دستور بصورت زیر است.

Dec Operand

این دستورات بر بیت های پرچم OF، ZF،SF، AF، PF اثر می گذارد و CF تغییر نمی نماید. به عنوان مثال DEC AX یک واحد از محتوای ثبات AX را کاهش می‌دهد.

دستور تغییر علامت NEG:

این دستور مکمل2 یک عملوند را پیدا می‌کند. شکل کلی آن بصورت زیر است.

NEG Operand

این دستور بر بیت‌های OF، ZF،SF، ZF، PF،AF اثر می‌گذارد. به عنوان مثل NEG AL

فصل دهم

ساختار و اجرای برنامه‌های اسمبلی بصورت COM

مقدمه

شکل دیگر برنامه‌ی اسمبلی به نام فایل COM معروف است که برای برنامه‌هایی استفاده می‌شود که کمتر از 64K حافظه (در کامپیوترهای اولیه و همچنین میکرو پروسسورهای امروزی به کار می‌روند) نیاز دارند. شکل زیر ساختار اصولی برنامه‌های به صورت COM می‌باشند.

```
        PAGE 110,100
TITLE   'EXAMPLE1.ASM'   add two ASCII
;-----------------------------------------------
;               Defining segment of program
;               ---------------------------
CODESG SEGMENT
        ASSUME SS:CODESG,DS:CODESG,CS:CODESG
        ORG 100H
START:  JMP MAIN        ;1-Jump over data
;
;                1- Define data
;                --------------
DATA1 DB 35H
DATA2 DB 32H
DATA3 DB ?
;
;               2- The rest of instructions
;               ---------------------------
MAIN    PROC NEAR
        MOV BL,DATA1    ;2- Move 35h to BL
        SUB BL,30H      ;3- Change to binary
        MOV AL,DATA2    ;4- Move 32 to AL
        SUB AL,30H      ;5- Change to binary
        ADD AL,BL       ;6- Add BL to AL
        MOV DATA3,AL    ;7- Store sum in DATA3
;
        MOV AX,4C00H    ;8- End of
        INT 21H         ;9- processing
MAIN    ENDP            ; End of procedure
CODESG  ENDS            ; End of segment
        END START       ; End of program
```

در برنامه‌های COM باید مقررات ذیل رعایت شوند:

1- فقط یک سگمنت کد برای متغیرها و دستورات تعریف می‌شود. به عبارت دیگر سگمنت کد و سگمنت داده یکسان می‌باشند، و سگمنت داده‌ی مجزا تعریف نمی‌شود.

2- سگمنت پشته‌ی مجزا تعریف نمی‌شود (سیستم عامل به طور خودکار از انتهای سگمنت کد برای حافظه پشته استفاده می‌نماید)

3- در شبه دستور ASSUME تمام ثبات‌های سگمنت را به سگمنت کد نسبت می‌دهیم مانند:

ASSUME SS:CODESG,DS:CODESG,CS:CODSEG

4- دستورات و اطلاعات از افست آدرس 0100H سگمنت کد و با شبه دستور 100H ORG شروع می‌شوند. (چون سیستم عامل ابتدای سگمنت، یعنی از آدرس 00 تا FFH را برای کارهای خود نیاز دارد).

5- با دستور JMP برنامه را شروع می‌کنیم که به اولین دستور اصلی برنامه برود مانند:

START JMP MAIN

تذکر: چون برنامه‌های COM حتماً باید با یک دستور شروع شوند از طرفی نیاز به تعریف متغیرها هم داریم با گذاشتن یک دستور JMP در ابتدای برنامه و پریدن از روی داده‌ها به این مشکل فایق می‌آییم.

6- تمام متغیرها را بعد از دستور JMP تعریف می‌کنیم.

7- تمام روال‌ها با صفت NEAR تعریف می‌شوند مانند دستور:

MAIN PROC NEAR

8- مقابل دستور END برچسب اولین دستور برنامه یعنی END START باید باشد.

فصل یازدهم

دستورات کنترل برنامه و دستورات منطقی

مقدمه

تا به حال برنامه‌های نوشته شده، دستورات را پشت سر هم اجرا می‌نمودند ولی بعضی از اوقات نیاز است که اجرای دستورات از محل دیگری در برنامه ادامه یابد که در این موقع می‌توان از دستورات کنترل برنامه شامل دستورات پرش استفاده نمود. علاوه بر این اکثر برنامه‌ها شامل تعدادی عملیات مقایسه اطلاعات هستند که متناسب با نتیجه مقایسه می‌بایستی به محل بخصوصی از برنامه مراجعه شود و دستورات مورد نظر اجرا گردند. در این مواقع نیز از دستورات پرش شرطی استفاده می‌گردد. بسیاری از مواقع نیر نیاز داریم قسمتی از برنامه چندین بار پشت سر هم اجرا گردد. در این مواقع از دستور حلقه بهره می‌گیریم. علاوه بر این با دستورات منطقی AND، OR، XOR، TEST، NOT، شیفت، چرخش و غیره می‌توان یک یا تعدادی بیت را یک یا صفر نمود و یا بیت بخصوصی را تست کرد.

دستورات کنترل برنامه

دستورات کنترل برنامه عبارتند از دستورات پرشJMP و دستورات LOOP که هر یک را ذیلاً مورد بررسی قرار می‌دهیم.

دستور پرش JMP

دستور پرش بصورت زیر تعریف می‌شود:

JMP برچسب یا آدرس دستور هدف

این دستور برای کنترل برنامه استفاده می‌شود، یعنی کنترل از نقطه‌ای از برنامه به نقطه دلخواه دیگر از برنامه منتقل می‌شود (این دستور بر بیت‌های پرچم اثر ندارند). در حقیقت عملوند یا اپراند دستورJMP، برچسب یا آدرس دستور دیگری می‌باشد که برنامه از آن دستور می‌بایستی شروع و ادامه یابد.

به عنوان مثال در دستورات:

```
JMP BEGIN
------
BEGIN: ADD AL, BL
-------
```

دستور LOOP

دستورLOOP برای کنترل برنامه و تکرار دستورات به تعداد معین بکار می‌رود. برای این کار می توان از دستورLOOP استفاده نمود، به شرطی که تعداد تکرار حلقه را، در ثبات CX قرار دهیم. در این صورت هر بار که حلقه اجرا می‌شود، دستورLOOP به طور خودکار از ثبات CX یک واحد کم می‌کند، به محض اینکه CX صفر شد، اجرای حلقه پایان می‌یابد و دستور بعد از LOOP اجرا می‌شود. شکل کلی این دستور به صورت زیر است:

Mov Cx , مقدار

آدرس هدف یا ابتدای حلقه LOOP

که آدرس جلوی دستور یک بایتی است و حداکثر بین 127+ و 128- می‌باشد. در ضمن این دستور بر روی بیت‌های پرچم تاثیر نمی‌گذارد.

دستور مقایسه CMP

این دستور محتویات دو عملوند را با هم مقایسه می‌کند و روی بیت‌های پرچم اثر می‌گذارد. شکل کلی آن بصورت زیر است:

CMP Operand1,Operand2

CMP 1عملوند، عملوند2

اجرای این دستور باعث می‌شود که عملوند2 از عملوند1 کسر گردد و بر روی بیت‌های پرچم OF، SF، ZF، AF، DF و CF اثر بگذارد که از روی وضعیت بیت‌های پرچم، پردازنده تشخیص می‌دهد که آیا دو عملوند مساوی یا عملوند1 بزرگتر از عملوند2 می‌باشد و بالعکس هستند.

طبق شکل زیر داریم:

CF	ZF	
0	0	عملوند - 1>عملوند - 2
0	1	عملوند - 1=عملوند - 2
1	0	عملوند - 1< عملوند - 2

عملوند1 می‌تواند یک ثبات یا یک خانه حافظه باشد و عملوند2 نیز می‌تواند هر یک از ثبات‌ها، یک خانه حافظه و یا یک عدد ثابت باشد.

ایجاد حلقه تاخیر با دستور LOOP

با دستورLOOP می‌توان یک تاخیر زمانی نرم‌افزاری ایجاد کرد. برای این کار دستورات زیر را می‌نویسیم

```
MOV CX, N
AGAIN: LOOP AGAIN
```

دستورات پرش شرطی

کامپیوترها دارای تعداد زیادی دستورات پرش شرطی برای کاربردهای متفاوت می‌باشند. اصولاً دستورات پرش شرطی بعد از دستوراتی که بر روی بیت‌های پرچم اثر می‌گذارند، قرار داده می‌شوند. معمول‌ترین این دستورات دستور مقایسه CMP و دستورات منطقی یا محاسباتی می‌باشند. انواع دستورات پرش شرطی مبتنی بر بیت‌های پرچم در سه جدول زیر اشاره شده‌اند.

بیت‌های پرچمی که بررسی می‌شود	توضیح	دستور پرش
بیت علامت SF	Jump Sign (Negative)	JS Tarjet اگر نتیجه محاسبات منفی (SF = 1) است عمل پرش انجام شود.
بیت علامت SF	Jump Not Sign (Positive)	JNS Tarjet اگر نتیجه محاسبات مثبت (SF = 0) است عمل پرش انجام شود.
بیت نقلی CF	Jump Carry	JC Tarjet اگر بیت نقلی CF برابر یک است پرش انجام شود.
بیت نقلی CF	Jump Not Carry	JNC Tarjet اگر بیت نقلی CF برابر صفر است پرش انجام شود.
بیت سرریز OF	Jump Overflow	JO Tarjet اگر بیت سرریز OF یک است عمل پرش انجام شود
بیت سرریز OF	Jump Not Overflow	JNO Tarjet اگر بیت سرریز OF صفر است عمل پرش انجام شود.
بیت توازن PF	Jump Parity یا Jump Parity Even	JP/JPE Tarjet* اگر بیت توازن P زوج** است عمل پرش انجام شود.
بیت توازن PF	Jump Not Parity یا Jump Parity Odd	JNP/JPO Tarjet اگر بیت توازن P فرد** است. عمل پرش انجام شود.

مقدار بیت‌های پرچم	توضیح	دستور
ZF = 1	(Jump Equal/or Jump Zero)	JE/JZ Tarjet اگر اُپراندها* مساوی است، یا بیت تشخیص صفر ZF = 1 است عمل پرش انجام شود.
ZF = 0	(Jump Not Equal/ or Jump Not Zero)	JNE/JNZ Tarjet اگر اُپراندها* مساوی نیستند، یا اگر بیت تشخیص صفر ZF = 0 است عمل پرش انجام شود.
CF = 0 , ZF = 0	(Jump Above or Jump Not Below or Equal)	JA/JNBE Tarjet اگر اُپراند اوّل بالاتر است (اگر بزرگتر است) یا، اگر کمتر، یا مساوی نیست عمل پرش انجام شود.
CF = 0	(Jump Above or Equal/or Jump Not Below)	JAE/JNB Tarjet اگر اپراند* اول بالاتر (اگر بزرگتر است) یا مساوی است، یا اگر کمتر نیست عمل پرش انجام شود.
CF = 1	(Jump Below/or Jump Not Above or Equal)	JB/JNAE Tarjet اگر اُپراند* اوّل کمتر است، یا اگر بالاتر یا مساوی نیست، عمل پرش انجام شود.
CF = 1 یا ZF = 1	(Jump Below or Equal/or Jump Not Above	JBE/JNA Tarjet اگر اُپراند* اوّل کمتر یا مساوی است، یا اگر بالاتر نیست، عمل پرش انجام شود.

دستور	توضیح	مقدار بیت‌های پرچم
JE/JZ Tarjet اگر دو عدد مساوی است، یا نتیجه محاسبات صفر است، عمل پرش انجام شود.	(Jump Equal/or Jump If Zero)	ZF = 1
JNE/JNZ Tarjet اگر دو عدد مساوی نیستند، یا نتیجه محاسبات صفر نیست، عمل پرشی انجام شود.	(Jump Not Equal/or Jump If Not Zero)	ZF = 0
JG/JNLE Tarjet اگر اُپراند اوّل* بزرگتر است، یا کوچکتر یا مساوی نیست، عمل پرش انجام شود.	(Jump Greater/or Jump If Not Less or Equal	SF ⊕ OF = 0 ZF = 0
JGE/JNL Tarjet اگر اُپراند* اول بزرگتر یا مساوی است، یا کوچکتر نیست، عمل پرش انجام شود.	Jump Greater or Equal/or Jump Not Less	SF ⊕ OF = 0
JL/JNGE Tarjet اگر اُپراند* اوّل کوچکتر است یا بزرگتر و مساوی نیست عمل پرش انجام شود.	(Jump Less/or Jump Not Greater or Equal)	SF ⊕ OF = 1
JLE/JNG Tarjet اگر اُپراند اوّل* کوچکتر یا مساوی است و یا بزرگتر	(Jump Less or Equal /or Jump Not Greater)	SF ⊕ OF = 1 ZF = 1

دستورات منطقی

این دستورات عبارتند از AND، OR، XOR، TEST که توسط این دستورات عملیات منطقی بر روی هر یک از بیت‌های ثبات‌ها، یا خانه‌های حافظه انجام می‌شود. با این عملیات می‌توان بر روی یک بیت یا گروهی از بیت‌های ثبات‌ها یا خانه‌های حافظه عملیاتی انجام داد و آنها را متناسب با کاربرد، یک یا صفر نمود. به این دلیل در بعضی از کامپیوترها، دستورات منطقی را دستورات عملیات روی بیت نیز می‌نامند. شکل کلی این دستور به صورت زیر است:

Operation Destination, Source

که کد اپراند مقصد می‌تواند هر یک از ثبات‌ها یا خانه‌های حافظه باشد و منبع نیز می‌تواند هر یک از ثبات‌ها، خانه‌های حافظه یا یک عدد ثابت باشد. این عملیات می‌توانند روی اطلاعات 8بیتی، 16بیتی و یا 32بیتی اجرا شوند و بر روی بیت‌های پرچم نقلیCF، سرریزOF، توازنPF و تشخیص صفرZF اثر می‌گذارند.

الف) دستور AND

در دستور AND اگر هر دو بیت نظیر به نظیر دو اپراند یک باشند، بیت نظیر نتیجه نیز یک می‌باشد ولی اگر هر یک از دو بیت دو اپراند صفر باشند بیت نظیر نتیجه، صفر خواهد بود. به عنوان مثال داریم:

0101= اپراند1 یا اطلاعات 1

0011= اپراند2 یا اطلاعات 2

0001= نتیجه AND دو اپراند یا دو اطلاعات

ب) دستور OR

در دستور OR اگر هر دو بیت نظیر به نظیر اطلاعات صفر باشند، بیت نظیر نتیجه صفر می‌شود در غیر این‌صورت بیت نظیر نتیجه یک خواهد بود. به عنوان مثال داریم:

0101= عملوند 1 یا اطلاعات 1

0011= عملوند 2 یا اطلاعات 2

0111= نتیجه OR دو عملوند یا دو اطلاعات

ج) دستور XOR

در دستور XOR اگر فقط یکی از دو بیت اطلاعات یک باشد، خروجی یک است ولی اگر هر دو بیت اطلاعات یک یا هر دو بیت صفر باشند، خروجی صفر خواهد بود. به عنوان مثال داریم:

0101= عملوند 1 یا اطلاعات 1

0011= عملوند 2 یا اطلاعات 2

0110= نتیجه XOR دو عملوند یا دو اطلاعات

د) دستور NOT

این دستور باعث می‌شود عملوند یا محتوای ثبات یا خانه حافظه، بیت به بیت معکوس می‌گردد، به عبارت دیگر مکمل یک آن بدست می‌آید.

ح) دستور TEST

این دستور دارای دو عملوند است که باعث می‌شود این دو عملوند بیت به بیت باهم AND شوند و روی بیت‌های پرچم اثر بگذارند ولی محتویات هیچ‌کدام از عملوندها عوض نمی‌شوند. شکل کلی این دستور به صورت زیر است:

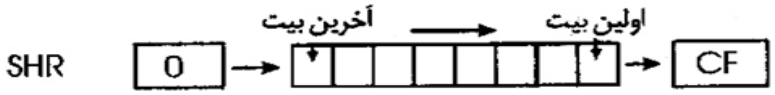
TEST 1 عملوند، 2 عملوند

دستورات شیفت و چرخش

این دستورات عبارتند از SHL، SAL، SHR، SAR، ROR، RCR، ROL و RCL که در ذیل هر یک از آنها بررسی می‌گردد. توسط دستور شیفت، محتوای ثبات‌های پروسسور یا خانه حافظه، یک یا چند بیت به طرف راست، یا چپ شیفت داده می‌شود و دو نوع شیفت ریاضی و منطقی وجود دارند.

شیفت به چپ SHL و SAL

در عملیات شیفت به چپ، صفر وارد اولین بیت ثبات خانه حافظه می‌شود و هر بیت به خانه‌ی سمت چپ خود منتقل می-گردد، این نوع شیفت برای شیفت ریاضی و منطقی یکسان است و این دستور بر بیت‌های OF، SF، ZF، PF و CF اثر می-گذارد.



شکل (۱۱-۲۱) نمایش دستور شیفت منطقی به راست

شکل کلی این دستور برای شیفت منطقی به صورت ذیل است:

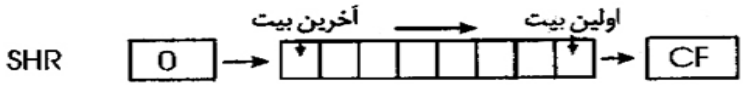
SHR	۱ ، عملوند
	و
SHR	CL ، عملوند

شکل کلی شیفت ریاضی به چپ به صورت ذیل است:

SAR	۱ ، عملوند
	و
SAR	CL ، عملوند

شیفت به راست SHR و SAR

شیفت به راست دو نوع می‌باشد. شیفت منطقی برای اعداد بدون علامت و دیگری شیفت ریاضی جهت اعداد علامت‌دار. این دستورات بر بیت‌های پرچم OF، SF، ZF، PF و CF اثر می‌گذارند. در شیفت منطقی به راست (SHR) محتویات ثبات یا خانه حافظه یک بیت یا به اندازه عددی که داخل ثبات CL است به راست شیفت می‌کند یعنی هر بیت، به خانه‌ی سمت راست خود منتقل می‌شود.

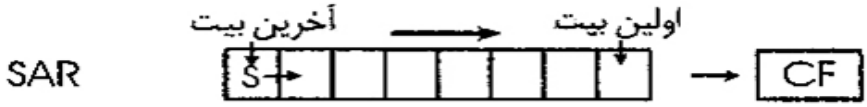


در این دستور، یک بار و یا به اندازه محتوای ثبات CL، به آخرین بیت صفر وارد می‌شود و اولین بیت نیز وارد بیت پرچم CF می‌گردد

شکل کلی این دستور به صورت زیر است:

SHR	۱ ، عملوند
	و
SHR	CL ، عملوند

شیفت ریاضی به راست SAR، فرق مهمی که با شیفت منطقی به راست دارد در این است که بیت علامت تغییر نمی‌کند و از چپ وارد بیت‌های دیگر می‌شود. در این صورت اعداد مثبت و منفی، علامت خود را حفظ می‌کنند. در شیفت ریاضی به راست SAR، محتویات ثبات یا خانه‌ی حافظه یک بیت، یا به اندازه عددی که داخل ثبات CL است، به راست شیفت می‌کند (یعنی هر بیت به خانه‌ی سمت راست خود منتقل می‌شود.) ولی علامت آن عوض نمی‌شود.



شکل کلی این دستور به صورت زیر است:

SAR عملوند ، ۱
و
SAR عملوند ، CL

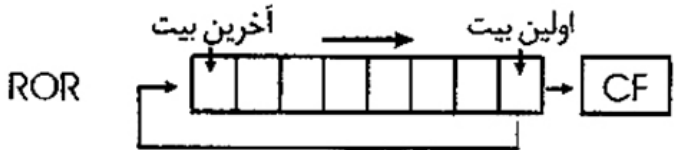
دستورات چرخش

دستورات چرخش شبیه دستورات شیفت هستند، یعنی محتویات یک ثبات یا یک خانه حافظه 8بیتی یا 16بیتی یا 32 بیتی، یک بیت، یا به تعداد عددی که در ثبات CL است، به راست یا به چپ شیفت داده می‌شوند، در دستورات چرخش، اولین بیت وارد آخرین بیت و بالعکس می‌شود. این دستورات بر بیت‌های پرچم‌های CF وOF اثر می‌گذارند

دستور چرخش راست ROR

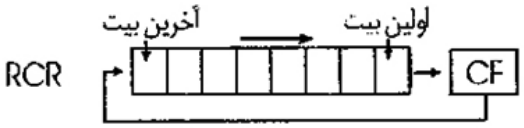
شکل کلی دستور چرخش به راست به صورت زیر است:

ROR عملوند ، 1	(۱)
و	
ROR عملوند ,CL	(۲)



دستور چرخش به راست از طریق بیت پرچم نقلی RCR

دستور چرخش به راست می‌تواند از طریق بیت پرچم نقلی CF نیز انجام پذیرد. یعنی محتوای بیت صفر عملوند (اولین بیت سمت راست) وارد بیت نقلی CF می‌گردد و هر بیت به خانه‌ی سمت راست خود منتقل می‌شود و بیت نقلی CF وارد آخرین بیت، یعنی بزرگترین بیت می‌گردد



دستور چرخش به راست از طریق بیت نقلی CF

شکل کلی این دستور به صورت زیر است:

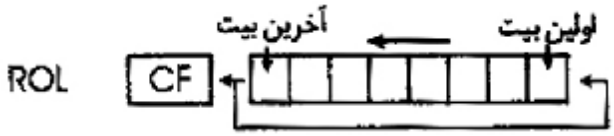
RCR عملوند ، 1	(۱)
و	
RCR عملوند ، CL	(۲)

دستور چرخش به چپ ROL

شکل کلی این دستور به صورت زیر است:

RCL عملوند ، 1	(۱)
و	
RCL عملوند ،CL	(۲)

این دستور بر بیت های پرچم OF، CF، اثر می گذارد. به این ترتیب در دستور (1) عملوند که یک ثبات یا یک خانه حافظه است، یک بیت به طرف چپ شیفت می‌کند یعنی هر بیت به خانه‌ی سمت چپ خود منتقل می‌شود و آخرین بیت وارد اولین بیت و همچنین بیت نقلی CF می‌گردد .



ساختار روال یا سابروتین

برنامه‌ی اسمبلی می‌تواند دارای تعدادی روال یا سابروتین باشد که هر روال با شبه‌دستور PROC شروع و با شبه دستور ENDP پایان می‌یابد و وظیفه‌ی خاصی را انجام می‌دهد. (مانند گرفتن اطلاعات از صفحه کلید و ...) از لحاظ اصول برنامه-نویسی بهتر است برنامه‌ی اسمبلی را به چندین روال کوچکتر تقسیم نمائیم که در این صورت برنامه‌ی اسمبلی دارای مزایای ذیل خواهد بود.

1- تعداد دستورات برنامه حداقل می‌شود چون یک روال می‌تواند از محل‌های مختلف در سگمنت کد فراخوانی شود.

2- تشکیلات برنامه بهتر می‌گردد و آسان‌تر می‌توان محل اشکالات برنامه را پیدا نمود.

3- نگهداری و به روز درآوردن و اصلاحات آتی برنامه نیز راحت می‌شود.

لذا در نوشتن برنامه‌های اسمبلی سعی می‌شود که حتی الامکان از روال یا سابروتین ساده و کوچک استفاده شود.

دستور فراخوانی روال CALL و برگشت از روال RET

برنامه‌ی فرعی یا سابروتین در کامپیوترهای شخصی روال یا Procedure نیز نامیده می‌شود. هر روال دارای یک نام است و ساختار کلی آن به طریق زیر مشخص می‌شود.

```
نام روال[3]   PROC
              —
              —
              —
دستورات       —
              —
              —
              —
              RET
نام روال      ENDP
```

برای فراخوانی یک روال از پیش نوشته شده کافی است نام روال را بعد از کلمه رزرو شده Call بنویسید. در این حالت اجرای برنامه اصلی به صورت موقت قطع شده و اجرای برنامه از ابتدای روال آغاز خواهد شد. بعد از اتمام برنامه روال، با اجرای دستور RET دوباره برنامه از محل قطع برنامه اصلی از سر گرفته می‌شود.

فصل دواز دهم

نمایش اطلاعات بر روی مانیتور در حالت متن و گرافیک

مقدمه

برای انجام عملیات روی مانیتور، صفحه کلید و ... در کامپیوتر تعداد زیادی برنامه یا روتین‌های سرویس وقفه به نام BIOS از طریق دستور INT N، (N بین 0 تا FFH می‌باشد) در اختیار کاربر قرار داده شده است. دستور INT N شبیه دستور فراخوانی سابروتین CALL است و هر وقت که استفاده شود عملیات زیر انجام می‌گیرد.

1- برنامه‌ی اصلی کاربر متوقف شده یا آدرس در آن وقفه قرار می‌گیرد.
2- محتویات ثبات‌های CS:IP و همچنین محتوای ثبات پرچم مربوط به برنامه‌ی کاربر در حافظه پشته ذخیره می‌گردد.
3- روتین سرویس وقفه مربوطه به دستور INT N اجرا می‌شود و عملیات خواسته شده را انجام می‌دهد.
4- بعد از اتمام برنامه‌ی وقفه بازیابی مقدار اولیه ثبات‌های CS:IP و ثبات پرچم از حافظه پشته و برگشت به برنامه‌ی اصلی و ادامه اجرای برنامه‌ی کاربر می باشد.

برخی از سرویس‌های دستور وقفه INT 10H و همچنین INT 21H برای نمایش اطلاعات و عملیات بر روی مانیتور در نظر گرفته شده‌اند. سرویس‌های دستورات INT 10H و INT 21H برای عملیات بر روی مانیتور بشرح زیر می‌باشند.

سرویس های دستور INT 10H (دستورات BIOS)

- سرویس 00H: تغییر حالات یا مد مانیتور
- سرویس 01H: تعیین اندازه‌ی مکان‌نما
- سرویس 02H: تغییر محل مکان‌نما
- سرویس 03H: بررسی موقعیت و اندازه‌ی مکان‌نما
- سرویس 05H: انتخاب صفحه فعال برای نمایش اطلاعات
- سرویس 06H: پاک کردن و چرخش اطلاعات مانیتور به بالا
- سرویس 07H: پاک کردن و چرخش اطلاعات مانیتور به پایین
- سرویس 08H: خواندن حرف و رنگ مربوط به آن در محل فعلی مکان‌نما
- سرویس 09H: نوشتن یک یا تعدادی حرف و رنگ آن در محل مکان‌نما بدون تغییر محل مکان‌نما
- سرویس 0AH: نوشتن یک یا تعدادی حرف در محل مکان‌نما بدون تغییر رنگ و بدون تغییر محل مکان‌نما
- سرویس 0CH: روشن کردن یا نوشتن یک نقطه یا یک پیکسل مانیتور در حالت گرافیک
- سرویس 0DH: خواندن مشخصات یک نقطه یا یک پیکسل مانیتور در حالت گرافیک
- سرویس 0EH: نوشتن یک حرف روی مانیتور و تغییر محل خودکار مکان‌نما
- سرویس 0FH: تعیین حالت یا مد مانیتور

سرویس های دستور INT 21H (ازDOS)

- سرویس02H: نمایش یک حرف روی مانیتور و تغییر محل خودکار مکان‌نما
- سرویس09H: نمایش یک رشته اطلاعات یا یک پیغام روی مانیتور

در زیر برخی از آنها را به همراه مثال‌های کاربردی به تفضیل مورد بحث قرار خواهیم داد.

کاربرد سرویس‌های دستور INT 10H (سرویسهای BIOS) برای حالت متنی مانیتور

دستور INT 10H سرویس‌های مختلفی برای حالت مانیتور دارد که مهم‌ترین و پرکاربردترین آنها به شرح زیر می‌باشند:

- **سرویس 00 دستور INT 10H: تغییر حالت یا مد مانیتور**

شماره سرویس در AH قرار می‌گیرد.

حالت مانیتور در AL قرار می‌گیرد.

فراخوانی سرویس INT 10H

```
MOV AH,00
MOV AL,03
INT 10H
```

در زیر جدول حالات مانیتور برای حالت متنی نوشته شده است

حالت مانیتور	دقت(پیکسل)	تعداد حروف	رنگ
0	400 ×360	25 × 40	تک رنگ
1	400 ×360	25 × 40	16 رنگ
2	400 ×720	25 × 80	تک رنگ
3	400 ×720	25 × 80	16 رنگ
7	400 ×720	25 × 80	تک رنگ

- **سرویس 02H دستور INT 10H: تغییر محل مکان‌نما**

شماره سرویس در AH قرار می‌گیرد.

شماره سطر (مقدار مختصات y) در DH قرار می‌گیرد.

شماره ستون (مقدار مختصات x) در DL قرار می‌گیرد.

شماره صفحه فعال مانیتور در BH قرار می‌گیرد.

فراخوانی سرویس وقفه INT 10H

مثال: نحوه استفاده از سرویس 02 برای تغییر مکان مکان‌نما

```
MOV AH,02H
MOV BH,00 صفحه صفر استفاده می‌شود
MOV DH,10 شماره سطر
MOV DL,8 شماره ستون
INT 10H اجرای دستور وقفه
```

- **سرویس 06H دستور INT 10H: پاک کردن وچرخش اطلاعات مانیتور به بالا**

رنگ زمینه و حروف = BH

شماره ستون گوشه سمت چپ بالای پنجره = CL

شماره سطر گوشه سمت چپ بالای پنجره = CH

شماره ستون گوشه سمت راست بالای پنجره = DL

شماره سطر گوشه سمت چپ پایین پنجره = DH

در زیر جداول رنگ حروف و زمینه مانیتور برای حالت متنی نوشته شده است

جدول مشخصات بایت صفت هر حرف

		رنگ حروف	رنگ زمینه
صفت	BL	IRGB	RGB
شماره بیت	7	654	3210

جدول ترکیبات رنگ برای حروف

RGB	I	رنگ حروف	RGB	BL	رنگ زمینه
000	1	خاکستری	000	0	سیاه
001	1	آبی کمرنگ	001	0	آبی
010	1	سبز روشن	010	0	سبز
011	1	آبی آسمانی کمرتگ	011	0	آبی آسمانی
100	1	قرمز روشن	100	0	قرمز
101	1	زرشکی روشن	101	0	زرشکی
110	1	زرد	110	0	قهوه‌ای
111	1	سفید براق	111	0	سفید

- **سرویس 09H دستور INT 10H: نوشتن یک یا تعدادی حرف و رنگ آن در محل مکان‌نما بدون تغییر محل مکان‌نما**

شماره سرویس در AH قرار می‌گیرد AH = 09

کد اسکی حرف مورد نظر برای نمایش = AL

کد رنگ = BL

شماره‌ی صفحه‌ی فعال مانیتور = BH

تعداد دفعاتی که حرف می‌بایستی تکرار شود = CX

- **سرویس OEH دستور INT 10H: نوشتن یک حرف روی مانیتور و تغییر محل خودکار مکان‌نما**

شماره صفحه 00 در BH قرار می‌گیرد.

شماره سرویس در AH قرار می‌گیرد.

کد اسکی حرف در AL قرار می‌گیرد.

شماره رنگ حرف در BL قرار می‌گیرد.

وقفه با INT 10H فعال می‌گردد.

مثال: نوشتن حروف سفید بر روی صفحه سیاه رنگ

```
MOV BH,00
MOV AH,OEH
MOV AL,41H
MOV BL,07
INT 10H
```

کاربرد سرویس‌های دستور INT 21H (سرویسهای DOS) برای حالت متنی مانیتور

- **سرویس 02H دستور INT 21H: نمایش یک حرف روی مانیتور و تغییر محل خودکار مکان‌نما**

کد اسکی حرف به DL منتقل می‌شود.

شماره سرویس نمایش متن در ثبات AH قرار می‌گیرد.

وقفه با INT 21H فعال می‌گردد.

```
MOV DL,'A'
MOV AH,02H
INT 21H
```

- **سرویس 09H دستور INT 21H: نمایش یک رشته اطلاعات یا یک پیغام روی مانیتور**

این سرویس باعث می‌شود که یک رشته اطلاعات که در سگمنت داده برنامه تعریف شده است روی مانیتور نوشته شود. البته در پایان رشته اطلاعات می بایستی علامت $ باشد. برای این کار مقدار ثبات AH را با دستور MOV AH,09H برابر 09H قرار می‌دهیم و افست آدرس شروع رشته اطلاعات در سگمنت داده را با دستور LEA در ثبات DX قرار می‌دهیم و بالاخره دستور INT 21H را اجرا می‌نمائیم در این صورت رشته حروف از چپ به راست روی مانیتور نوشته می‌شوند و نمایش حروف به علامت $ که رسید متوقف می‌گردد.

```
Mov Ah ,09H
Lea Dx ,Data1
Int 21H
```

کاربرد سرویس‌های دستور INT 10H (سرویسهای BIOS) برای حالت گرافیکی مانیتور

- **سرویس 00 دستور INT 10H: قرار دادن مانیتور در حالت گرافیک:**

قرار دادن شماره سرویس برای مانیتور در حالت گرافیکی در AH

قرار دادن مانیتور در حالت گرافیکی در AL

اجرای دستورات با دستور INT 10H

```
MOV AH,00
MOV AL, 12H
INT 10H
```

در جدول زیر حالات مختلف مانیتور در حالت گرافیکی نوشته شده است.

رنگ	دقت (پیکسل)	حالت
4 رنگ	200 × 320	04H ,05H
تک رنگ	200 × 320	06H
16 رنگ	200 × 320	0DH
16 رنگ	200 × 320	0EH
تک رنگ	200 × 320	0FH
16 رنگ	200 × 320	10H
2 رنگ	200 × 320	11H
16 رنگ	200 × 320	12H
256 رنگ	200 × 320	13H

شانزده رنگ مختلف پیسکل در حالت گرافیکی

رنگ پیکسل	شماره رنگ		رنگ پیکسل	شماره رنگ	
	باینری	شانزدهی		باینری	شانزدهی
سیاه	0000	00	خاکستری	1000	08
آبی	0001	01	آبی کمرنگ	1001	09
سبز	0010	02	سبز روشن	1010	0A
آبی آسمانی	0011	03	آبی روشن	1011	0B
قرمز	0100	04	قرمز روشن	1100	0C
زرشکی	0101	05	زرشکی روشن	1101	0D
قهوه‌ای	0110	06	زرد	1110	0E
سفید	0111	07	سفید براق	1111	0F

- **سرویس OCH دستور INT 10H: روشن کردن یا نوشتن یک پیکسل در حالت گرافیک**

شماره سرویس = AH

شماره رنگ پیکسل = AL

ستون یا مختصات x پیکسل = CX

سطر یا مختصات y پیکسل = DX

شماره صفحه فعال مانیتور (در حالت عادی BH =00)

اجرای دستورات INT 10H

- **سرویس ODH دستور INT 10H: خواندن رنگ یک پیکسل در حالت گرافیک**

شماره سرویس AH =0DH

ستون یا مختصات x پیکسل = CX

سطر یا مختصات y پیکسل = DX

شماره صفحه فعال مانیتور = BH

اجرای دستورات INT 10H

تذکر: پس از اجرای دستورات، رنگ پیکسل با مختصات مذکور در داخل ثبات AL ذخیره می‌شود.

فصل سیز دهم

آشنایی با طرز گرفتن اطلاعات از صفحه کلید

سرویس‌های دستور INT 21 برای خواندن از صفحه کلید

- سرویس 01H: خواندن یک حرف از صفحه کلید و نمایش آن بر روی مانیتور (حساس به کلید Ctrl + Break)
- سرویس 06H: تشخیص فشار دادن کلید صفحه کلید
- سرویس 07H: خواندن یک حرف از صفحه کلید بدون نمایش آن بر روی مانیتور
- سرویس 08H: خواندن یک حرف از صفحه کلید بدون نمایش آن بر روی مانیتور (حساس به کلید Ctrl + Break)
- سرویس OAH: خواندن یک رشته اطلاعات از صفحه کلید و قرار دادن آن در محلی از بافر حافظه و نشان دادن آن بر روی مانیتور
- سرویس OBH: تست بافر صفحه کلید
- سرویس OCH: پاک کردن بافر صفحه کلید و فراخوانی یک سرویس صفحه کلید

سرویس‌های دستور INT 16H برای خواندن از صفحه کلید

- سرویس 00H: خواندن یک حرف از صفحه کلید بدون نمایش آن بر روی مانیتور
- سرویس 01H: بررسی فشار دادن کلید صفحه کلید
- سرویس 02H: برای گزارش وضعیت بعضی کلیدهای صفحه کلید
- سرویس10H: مانند سرویس 00 است و برای صفحه کلیدهای جدید کاربرد دارد.
- سرویس 11H: مشخص می‌کند که آیا حرفی در بافر صفحه کلید وجود دارد یا نه ؟

کاربرد سرویس 06H دستور INT 21H برای تشخیص فشار دادن کلید صفحه کلید

این سرویس هم برای ورود اطلاعات و هم خروج اطلاعات به کار می‌رود. اگر بخواهیم از آن برای ورود اطلاعات استفاده کنیم مقدار ثبات DL را برابر FFH قرار می‌دهیم. در این صورت اگر حرفی در بافر نباشد بیت پرچم ZF را یک می‌کند، AL=0 می‌گردد و منتظر فشار کلید صفحه کلید نمی‌شود. در صورتی که حرفی در بافر صفحه کلید باشد کد اسکی آن را در ثبات AL قرار می‌دهد و بیت پرچم ZF را صفر می‌کند. این سرویس حرف مربوطه را روی مانیتور نشان نمی‌دهد. از این سرویس می‌توان برای تشخیص اینکه آیا کلیدی فشار داده شده است یا نه استفاده نمود. در صورتی که بخواهیم از این سرویس به عنوان خروجی استفاده کنیم کافیست کد اسکی حرف را در ثبات DL قرار دهیم.

کاربرد سرویس 07H برای خواندن یک حرف از صفحه کلید بدون نمایش آن روی مانیتور

برای گرفتن یک کلید از صفحه کلید بدون نمایش آن بر روی مانیتور استفاده می‌شود. دستورات زیر نحوه استفاده از این سرویس را نمایش می‌دهند.

```
MOV AH,7H
INT 21H
```

کاربرد سرویس 0BH دستور INT 21H برای تست بافر صفحه کلید

با قرار دادن AH = OBH و اجرای دستور INT 21H بررسی می‌شود که آیا حرفی در بافر صفحه کلید وجود دارد (اگرAL=OFFH باشد) و یا وجود ندارد. (اگر AL=0 باشد) البته این سرویس منتظر فشار کلیدی از صفحه کلید نمی‌شود فقط بافر را بررسی می‌کند. این سرویس به کلیدهای Ctrl + Break حساس می‌باشد.

کاربرد سرویس00H دستور INT 16H برای خواندن یک حرف از صفحه کلید بدون نمایش آن بر روی مانیتور

برای استفاده از این سرویس مقدار AH=OO قرار داده می‌شود و دستور INT 16H اجرا می‌گردد. بعد از اجرای دستور مذکور کامپیوتر بافر صفحه کلید را بازبینی می‌کند اگر بافر خالی باشد کامپیوتر آنقدر منتظر می‌ماند تا کلیدی فشار داده شود و اگر در بافر حرفی باشد در این صورت کد اسکی آن در ثبات AL و کد اسکن SCAN آن در ثبات AH قرار می‌گیرند. برای کلیدهای F1 تا F12 کد اسکی ندارند. مقدار AL=0 و در AH کد اسکن آن‌ها را قرار می‌دهد که به محض فشار هر کلیدی، کد اسکی آن در ثبات AL قرار می‌گیرد و کار سرویس پایان می‌یابد.

کاربرد سرویس 01H دستور INT 16H برای بررسی فشار دادن کلید صفحه کلید

این سرویس مشخص می‌کند که آیا کلیدی فشار داده شده است یا نه؟ برای این کار دستورات زیر را به کار می‌بریم.

```
MOV AH, 01H
INT 16H
```

کاربرد سرویس 02H دستور INT 16H برای گزارش وضعیت بعضی کلیدهای صفحه کلید

در صورتی که AH=02H قرار گیرد، بعد از اجرای دستورات INT 16H، در ثبات AL عددی قرار می‌گیرد که هر بیت آن نشانه‌ی این است که کدام کلید کنترلی فشار داده شده است.

- بیت 7 ثبات AL اگر یک باشد یعنی کلید Insert فشار داده شده است.
- بیت 6 ثبات AL اگر یک باشد یعنی کلید Caps lock فشار داده شده است.
- بیت 5 ثبات AL اگر یک باشد یعنی کلید Nom Lock فشار داده شده است.
- بیت 4 ثبات AL اگر یک باشد یعنی کلید Scroll Lock فشار داده شده است.
- بیت 3 ثبات AL اگر یک باشد یعنی کلید Alt فشار داده شده است.
- بیت 2 ثبات AL اگر یک باشد یعنی کلید Ctrl فشار داده شده است.
- بیت 1 ثبات AL اگر یک باشد یعنی کلید Left Shift فشار داده شده است.
- بیت 0 ثبات AL اگر یک باشد یعنی کلید Right Shift فشار داده شده است.

کاربرد سرویس 11H دستور INT 16H برای تشخیص اینکه آیا حرفی در بافر صفحه کلید وجود دارد یا نه؟

اگر کد اسکی حرفی در بافر صفحه کلید باشد، این سرویس بیت پرچم تشخیص صفر را صفر می‌کند و کد اسکی حرف مربوط را وارد ثبات AL و کد اسکن آن را در ثبات AH قرار می‌دهد. اگر بافر صفحه کلید خالی باشد، بیت تشخیص صفر را برابر یک کرده و منتظر فشردن کلید نمی‌شود. دستورات زیر نحوه استفاده از این سرویس را نمایش می‌دهد.

```
MOV AH, 10H
INT 16H
```

فصل پانزدهم

وقفه در کامپیوترهای شخصی

مقدمه

برای ارتباط با دستگاه‌های ورودی خروجی نیاز به یک سری برنامه‌ها یا روتین‌هایی می‌باشد که این برنامه‌ها تحت نام سرویس-های وقفه BIOS و DOS معروف هستند. به این ترتیب کاربران و همچنین سیستم‌عامل می‌توانند از این سرویس‌ها یا روتین-های سرویس وقفه ISR، برای ارتباط با دستگاه‌های ورودی خروجی و یا سایر خدمات استفاده نمایند. این سرویس‌ها و روتین-ها با دستور وقفه INT N قابل دسترس می‌باشد (که N از00تا FFH می‌تواند باشد) که البته هر یک از این دستورات نیز دارای سرویس‌های متفاوت می‌باشند.

روتین‌های سرویس وقفه و طرز اجرای آنها

کار وقفه یعنی ایجاد وقفه و قطع موقت برنامه‌ای که در حال اجرا است و اجرای برنامه‌ی دیگری توسط پروسسور به نام روتین سرویس وقفه، که از اهمیت بیشتری برخوردار می‌باشد. البته بعد از اینکه اجرای برنامه سرویس وقفه تمام شد CPU به برنامه اصلی برمی‌گردد و کارش را ادامه می‌دهد. اصولاً وقفه در پروسسور ممکن است توسط هر یک از موارد زیر پیش آید:

- **وقفه‌های خارجی**: از دستگاه‌های خارجی یا دستگاه‌های جانبی به پروسسور وارد می‌شوند که برای این کار دو خط وقفه خارجی NMI و INTR پیش‌بینی شده است. وقفه NMI برای تشخیص اشتباهات دستگاههای I/O و کارهای خیلی با اهمیت و فوری در نظر گرفته شده است و خط وقفه INTR نیز از طریق یک مدار مجتمع IC به نام کنترل‌کننده وقفه توسط تایمر صفحه کلید، پورت سری، دیسک پارالل و غیره فعال می‌شود و تقاضای وقفه ازCPU می‌نمایید.
- **وقفه‌های داخلی**: این نوع وقفه‌ها در اثر استفاده از دستور وقفه INT N و یا توسط سیستم عامل یا هنگام رخ دادن تقسیم یک عدد بر صفر، ایجاد سرریز، اجرای یک دستور- یک دستور برنامه و غیره فعال می‌شوند و از پروسسور تقاضای وقفه می‌نمایند که روتین وقفه مر بوط را اجرا کند. هر وقفه‌ی داخلی یا خارجی باعث می‌گردد که CPU ابتدا محتوای ثبات‌های IP و CS و ثبات پرچم مربوط به برنامه‌ی اصلی را در حافظه پشته ذخیره نماید و سپس بیت‌های پرچم وقفه IF وTF را صفر کند که وقفه‌ی دیگری پذیرفته نشود و بالاخره روتین سرویس وقفه مربوط به آن وقفه را اجرا می‌نماید. آدرس این روتین سرویس وقفه در حافظه، در چهار بایت یک جدول بردار وقفه قرار دارد.

فصل شانزدهم

تعریف ماکرو و کاربرد آن

مقدمه

در مواقعی که چندین بار می‌بایستی تعدادی دستور در برنامه اسمبلی تکرار شود، به جای این کار می‌توان ماکرو را تعریف نمود، نامی به آن داد و دستورات را فقط یک بار در ماکرو نوشت. در این صورت هر موقع که برنامه اصلی نیاز به به این دستورات باشد فقط ذکر نام ماکرو کافیست. علاوه بر این ماکروها با گرفتن پارامتر، همانند توابع در برنامه‌های سطح بالاتر ابزاری توانمند در مدیریت بهتر برنامه‌های اسمبلی می‌باشد.

تعریف ماکرو

هر ماکرو با شبه دستور MACRO در ابتدای برنامه و قبل از تعریف سگمنت، به طریق زیر تعریف می‌شود.

```
نام ماکرو   MACRO    [نام پارامترها۲]    ;    Comment
                  ..........
                  ..........
بدنه ماکرو۳ که شامل   دستورات اسمبلی
دستورات و توضیحات است.   ..........
                  ..........
                  ..........
                  ENDM                    ;  End of MACRO
```

برای تعریف ماکرو:

1. در شبه دستور ماکرو MACRO یک نام اختیاری برای ماکرو انتخاب می‌شود و در مقابل آن نام پارامترها (اختیاری است) که در حقیقت نام متغیرهایی هستند که در دستورات بدنه ماکرو استفاده می‌شوند، قرار می‌گیرد. علاوه بر این پارامترهای ماکرو ثبات‌هایCPU نیز می‌تواند باشند.
2. در بدنه ماکرو تعدادی دستور و توضیحات مربوطه قرار داده می‌شود.
3. در انتهای دستور ENDM پایان ماکرو را اعلام می‌نماید.

فصل هفدهم

مقیم کردن برنامه در حافظه یا برنامه‌های TSR

مقدمه

معمولاً برنامه‌های کامپیوتر بر روی دیسک قرار دارند و هر برنامه‌ای که از روی دیسک می‌خواهد اجرا شود، سیستم عامل محلی از حافظه را برای آن تخصیص می‌دهد و سپس برنامه را از دیسک به حافظه بارگذاری می‌کند. بعد از اجرای برنامه، سیستم عامل محلی از حافظه که برنامه‌ی کاربر آن را اشغال کرده بود را آزاد می‌نماید که بتواند در اختیار برنامه‌ی دیگری قرار دهد. در غیر این صورت بعد از اجرای چند برنامه حافظه کاملاً پر می‌شود و کامپیوتر نمی‌تواند برنامه‌ی دیگری را اجرا نماید لذا هر برنامه‌ای که از روی دیسک می‌خواهد اجرا شود به طور موقت وارد حافظه می‌شود و پس از اجرا، حافظه مذکور در اختیار برنامه‌ی دیگری قرار می‌گیرد به عبارتی دیگر مانند این است که برنامه‌ی کاربر از حافظه خارج شده است. فلسفه برنامه‌های TSR یا مقیم کردن برنامه در حافظه، این است که پس از اجرای برنامه، برنامه‌ی مذکور در حافظه باقی بماند به عبارتی دیگر سیستم عامل برنامه‌ی مذکور را در حافظه مقیم کند که در این صورت نیازی نیست برای اجرای برنامه آن را از روی دیسک فراخوانی کرد، چون برنامه در حافظه قرار دارد، لذا بسیار سریع‌تر اجرا می‌گردد. البته اشکال آن این است که برنامه‌ی مقیم در حافظه، قسمتی از حافظه را اشغال می‌نماید و جای کمتری در حافظه برای برنامه‌های دیگر خواهد ماند. لذا هر چه تعداد بیشتری برنامه در حافظه مقیم شوند، جای کمتری برای بقیه برنامه‌های کاربر باقی خواهند ماند. به عنوان نمونه برنامه‌های ساعت گوشه مانیتور یا ماشین حساب کامپیوتر و غیره در حافظه مقیم می‌باشند و بقیه برنامه‌ها، بر روی دیسک قرار دارند. برنامه‌های مقیم یا TSR تا موقعی که کامپیوتر روشن است در حافظه قرار دارند.

روش مقیم کردن برنامه در حافظه

برای مقیم کردن برنامه‌ی کاربر در حافظه کافیست در انتهای برنامه به جای برگشت به سیستم عامل (با سرویس 4CH دستور INT 21H) از سرویس 31H دستور INT 21H استفاده نمود به شرطی که اندازه برنامه کاربر بر حسب تعداد پاراگراف (هر پاراگرف 16 بایت است) قبلاً در ثبات DX قرار داده شود. به عنوان مثال اگر برنامه‌ای دارای 12 پاراگراف باشد، دستورات زیر جهت مقیم کردن آن در حافظه در انتهای برنامه کاربر باید گذارده شود:

```
MOV AH, 31H ;
MOV DX, 12 ;
INT 21H ;
```

فراخوانی یا فعال کردن برنامه‌ی مقیم

روش کلی برای فعال کردن برنامه‌ی مقیم در حافظه این است که آدرس برنامه‌ی مقیم در حافظه را جایگزین آدرس یا بردار روتین وقفه سخت‌افزاری نمود. لذا هر موقع که وقفه مذکور فعال شود، به جای روتین وقفه سخت افزاری، برنامه‌ی مقیم در حافظه اجرا می‌گردد. برای اینکه وقفه‌ی سخت‌افزاری نیز دچار اشکال نشود، آدرس روتین وقفه‌ی سخت‌افزاری در محل دیگری در حافظه ذخیره می‌گردد که در انتهای برنامه‌ی مقیم، به آن مراجعه می‌شود. یکی از وقفه‌های سخت‌افزاری که به کار می‌رود عبارتند از دستور INT 09 که با فشار دادن کلید صفحه کلید فعال می‌شود و یا وقفه تایمر که با دستور INT 08H فعال می-گردد. حال بررسی می‌نمائیم که چطور می‌توان آدرس برنامه‌ی مقیم در حافظه را، جایگزین آدرس روتین وقفه سخت‌افزاری کرد.

جایگزینی آدرس برنامه‌ی مقیم در حافظه با آدرس یا بردار روتین وقفه

برای اینکار از سرویس 35H و 25H استفاده می‌کنیم که در ذیل هر یک از آنها شرح داده می‌شوند

الف: استخراج مقادیرCS:IP یک روتین وقفه از جدول بردار وقفه توسط سرویس 35H دستور INT 21H:

در این حالت با قرار دادن عدد 35H در ثبات AH و شماره وقفه در ثباتAL و اجرای دستور INT 21H، آدرس سگمنت کد CS روتین وقفه در ثبات ES قرار می‌گیرد و مقدار افست آدرس IP نیز در ثبات BX قرار می‌گیرد. یعنی آدرس CS:IP روتین وقفه مورد نظر، در ثبات ES:BX گذارده می شوند. به عنوان مثال اگر بخواهیم مقدار CS:IP روتین وقفه INT 08H را پیدا کنیم و محل‌های حافظه OLDVECT و OLDVECT +2 قرار دهیم دستورات زیر را می‌نویسیم.

```
MOV AH, 35H
MOV AL, 08H
INT 21H
MOV OLDVECT, BX
MOV OLDVECT +2, EX
```

ب: جایگزین کردن آدرس برنامه‌ی مقیم در جدول بردار وقفه به جای آدرس روتین وقفه

برای این منظور از سرویس 25H دستور INT 21H را استفاده می‌کنیم. در این حالت با قرار دادن عدد 25H در ثبات AH و شماره‌ی وقفه در ثبات AL و همچنین گذاردن افست آدرس روتین جدید وقفه در ثبات DX، بالاخره اجرای دستورات INT 21H آدرس روتین وقفه‌ی جدید در محل بردار وقفه مربوطه در جدول بردار وقفه قرار می‌گیرند.

مثال: می‌خواهیم آدرس برنامه یا روتین خودمان به نام NEWISR را در محل آدرس دستور وقفه INT 08H، در جدول بردار وقفه قرار دهیم.

```
MOV   AH,25H              ;   شماره سرویس 25H را در AH قرار بده
MOV   AL,08H              ;   آدرس بُردار وقفه 08H را انتخاب کن
MOV   DX,OFFSET NEWISR    ;   افست آدرس روتین جدید را در ثبات DX قرار بده
INT 21H                   ;   وقفه 21H را فعال کُن
```

```
        PAGE 110,100
TITLE   'tsr_exa.asm'  structure of TSR program
;-----------------------------------------------
;
;               Defining Segment of Program
;               ---------------------------
CODESG  SEGMENT 'CODE'
        ASSUME CS:CODESG
        ORG 100H
MAIN:  JMP LOAD1          ; Jump to instuctions
;-----------------------------------------------
OLDVECT DD ?     ;Four byte to save CS:IP of
;       origin ISR
;
```

```
;                          تعریف بقیه داده‌ها
;
;-----------------------------------------------
;                  PART1:                    قسمت اول
;This part of program reside in memory as new ISR
;
NEWISR   PROC NEAR
         PUSH AX
;
;              بقیه دستورات برنامه مقیم در حافظه
;
         POP AX
         JMP CS:OLDVECT   ;Perform original ISR
NEWISR   ENDP
;-----------------------------------------------
;                  PART2:                    قسمت دوم
;This part run once only, during initialization
LOAD1    PROC NEAR
;
;    1-Get vector of ISR & save on OLDVECT
;     -------------------------------------
         MOV AH,35H       ;Get the
         MOV AL,- -       ; vector
         INT 21H          ; of OLDISR
         MOV WORD PTR OLDVECT,BX     ;Save them
         MOV WORD PTR OLDVECT+2,ES   ; on OLDVECT
;
;    2-Set the offset of new ISR in vector table
;     -------------------------------------------
         MOV AH,25H       ;Set vector of
         MOV AL,- -       ; new ISR in vector table
         MOV DX,OFFSET NEWISR  ;DX=IP,DS=CS (Set by
         INT 21H          ; COM program)
;
;    3-Make resident of PART1 of program (NEWISR)
;     --------------------------------------------
         MOV DX,(OFFSET LOAD1-OFFSET CODESG) ;Find
;                how many byte resident
         ADD DX,15        ;Make it
         MOV CL,4         ; multiple of
         SHR DX,CL        ; 16 byte
;
         MOV AH,31H       ;Make it
         INT 21H          ; resident
LOAD1    ENDP
CODESG   ENDS
         END MAIN
```

شکل (۱۸-۱) ساختار اصولی برنامه اسمبلی مقیم در حافظه

برنامه‌ی نوشته شده شامل قسمت‌های زیر می‌باشد

1. بردار یا آدس سرویس روتین قبلی مربوط به وقفه 08H یا 09H را می‌گیرد و در محل OLDVECT قرار می‌دهد.
2. آدرس روتین وقفه جدید NEWVECT را در محل بردار روتین قبلی قرار می‌دهد.
3. روتین جدید وقفه NEWISR را در حافظه مقیم می‌کند

همان طور که قبلاً اشاره شد برای اینکار از سرویس 31H استفاده می‌شود و اندازه سرویس روتین وقفه جدید، باید برحسب پاراگراف در ثبات DX قرار گیرد. برای این کار آدرس CODSEG و LOAD1 اندازه سرویس روتین وقفه‌ی جدید NEWISR را برحسب بایت تعیین می‌نمایند که این مقدار توسط دستور:

MOV DX,(OFFSET LOAD1-OFFSET CODSEG)

در ثبات DX قرار می‌گیرد ولی باید مقدار را تقسیم بر 16 نمود که بر حسب پاراگراف گردد، لذا توسط دستورات:

MOV CL, 4

SHR DX, CL

محتوای ثبات DX را چهار بار به طرف راست شیفت می‌دهیم که تقسیم بر 16 گردد. اما ممکن است اندازه روتین وقفه جدید، دقیقاً مضارب 16 نباشد و تعداد پاراگراف در DX کمتر از مقدار واقعی گردد لذا برای اطمینان بیشتر قبل از دستورات فوق توسط دستور:

ADD DX, 15

15 بایت به DX اضافه می‌شود که پس از تقسیم DX بر 16 تعداد پاراگراف کمتر از مقدار واقعی مورد نیاز نگردد.

برنامه نمونه:

برنامه‌ی زیر ساختار کلی برنامه‌های مقیم در حافظه می‌باشد که با این روش می‌توان هر برنامه را در حافظه مقیم نمود. مثال زیر برنامه‌ای است که در حافظه مقیم می‌شود و هر 30 ثانیه از بلندگوی داخلی کامپیوتر صدای بوق به صدا درمی‌آید.

توضیح: برای این منظور از وقفه‌ی زمانی INT 08H استفاده می‌نمائیم. همان طوری که قبلا بحث شد وقفه INT 08H در هر 55 میلی ثانیه فعال می‌شود (دقیقاً 54/94 میلی‌ثانیه یا در هر ثانیه 18/2 بار). بنابراین اگر بخواهیم کامپیوتر هر 30 ثانیه بوق بزند باید 550 بار وقفه فعال شود. (میلی‌ثانیه 30000 = 54/94 × 550) برای این کار در برنامه متغیر COUNT را با مقدار اولیه 550 بصورت زیر معرفی می‌کنیم.

```
COUNT DW 550
```

```
        PAGE 110,100
TITLE   'timer_8.asm'  a TSR program with timer 8
;-----------------------------------------------
;
;               Defining Segment of Program
;               ---------------------------
CODESG  SEGMENT 'CODE'
        ASSUME CS:CODESG
        ORG 100H
MAIN:  JMP LOAD1        ;1-Jump to instuctions
;-----------------------------------------------
OLDINT8 DD ?    ;Four byte to save CS:IP of INT 8H
```

```
COUNT   DW 550  ;550 *55 ms=30 seconds
;-----------------------------------------------
;                PART1:
;This part of program reside in memory as new ISR
;
NEWISR  PROC NEAR
        DEC CS:COUNT     ;2-Is time
        JNZ EXIT         ;3- over?
        MOV CS:COUNT,550 ;4-If yes intialize COUNT
        MOV CX,00FFH     ;5- make delay
AGAIN:  MOV AH,0EH       ;6- and beep
        MOV AL,07        ;7- the speaker
        INT 10H          ;8- for
        LOOP AGAIN       ;9- a delay
EXIT:   JMP CS:OLDINT8   ;10- take care INT 08
NEWISR  ENDP
;-------------------------------------------
;                PART2:
;This part run once only, during initialization
LOAD1   PROC NEAR
;
;   1-Get vector of ISR & save on OLDVECT
;    ------------------------------------
        MOV AH,35H       ;11-Get the
        MOV AL,08H       ;12- vector
        INT 21H          ;13- of OLDISR INT 8H
        MOV WORD PTR OLDINT8,BX     ;14-Save them
        MOV WORD PTR OLDINT8+2,ES   ;15- on OLDINT8
;
;   2-Set the offset of new ISR in vector table
;    ---------------------------------------------
        MOV AH,25H       ;16-Set vector of
        MOV AL,08H       ;17- new ISR in vector table
        MOV DX,OFFSET NEWISR  ;18-DX=IP,DS=CS (Set by
        INT 21H          ;19- COM program)
;
;   3-Make resident of PART1 of program (NEWISR)
;    ---------------------------------------------
        MOV DX,(OFFSET LOAD1-OFFSET CODESG) ;20-
;       Find how many byte resident
        ADD DX,15        ;21-Make it
        MOV CL,4         ;22- multiple of
        SHR DX,CL        ;23-  16 byte
;
        MOV AH,31H       ;24-Make it
        INT 21H          ;25- resident
LOAD1   ENDP             ;End of procedure LOAD1
CODESG  ENDS             ;End of CODESG
        END MAIN         ;End of program
```

هر بار که وقفه 08 فعال شود، یک واحد از مقدار متغیر COUNT کم می‌گردد، تا COUNT صفر شود که در این دستور مدت 30 ثانیه گذشته است. برای فعال کردن برنامه‌ی مقیم در حافظه از امکانات وقفه 08H استفاده شده است یعنی آدرس برنامه‌ی مقیم، جانشین آدرس وقفه می‌شود. لذا بعد از دستور 1، متغیر ? OLDINT8 DD برای ذخیره آدرس وقفه 08H تعریف می‌شود.

قسمت اول برنامه

روتین جدید وقفه NEWISR: از دستورات 2 تا 10 می‌باشد. دستورات 2 و 3 یک واحد از مقدارCOUNT کم می‌کند. اگر COUNT صفر نشده بود به دستور 10 می‌رود و کار وقفه 08H ادامه می‌یابد. و با فعال شدن وقفه 08H، دوباره دستورات 2 و 3 و 10 اجرا می‌شوند و این حلقه آنقدر اجرا می‌گردد تا COUNT صفر شود، یعنی وقتی که 550 بار وقفه 08H اجرا شد که معادل 30 ثانیه است، دستور4 اجرا می‌گردد. دستور4 مقدار اولیه به متغییر COUNT می‌دهد و دستورات 5 تا 9 دستورات تولید بوق هستند. در دستور10 دوباره کنترل به وقفه 08H برمی‌گردد و کار عادی وقفه مذکور ادامه می‌یابد.

قسمت دوم برنامه:

1. همانطوری که در ساختار اصولی برنامه‌های مقیم در حافظه بحث شد در این قسمت عملیات زیر انجام می‌شود
2. توسط دستورات 11 تا 15 آدرس روتین وقفه 08H با استفاده از سرویس 35H دستور INT 21H گرفته می‌شود و در محل OLDINT8 ذخیره می‌گردد.
3. توسط دستورات 16 تا 19 افست آدرس روتین جدید وقفه NEWISR در محل آدرس روتین وقفه 08 قرار می‌گیرد.
4. توسط دستورات 20 تا 25 روتین وقفه NEWISR در حافظه مقیم می‌گردد.

با ایجاد فایل com برنامه نوشته شده و اجرای آن، برنامه‌ی مذکور در حافظه مقیم می‌شود و هر30 ثانیه یک بار بوق شنیده می‌شود. البته به جای دستورات زدن بوق کامپیوتر (دستورات 5 تا 9) می‌توان هر تعداد و هر نوع دستوراتی را برای برنامه‌ی مقیم در حافظه نوشت، که هر 30 ثانیه برنامه مذکور اجرا گردد.

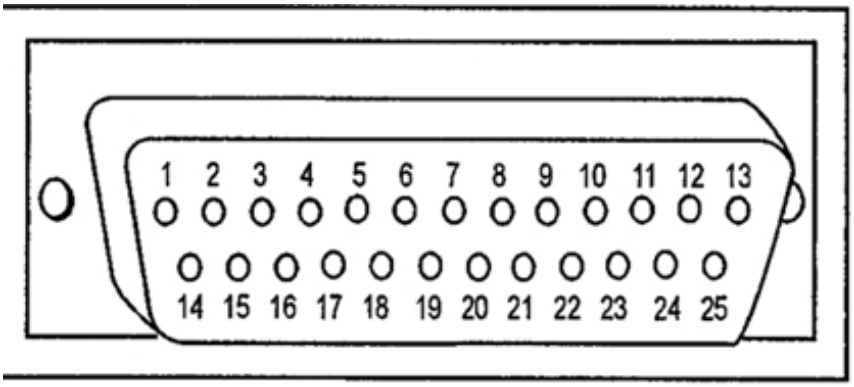
فصل هجدهم

طرز کار با پورت موازی

مقدمه

پورت موازی برای ارسال اطلاعات کامپیوتر به چاپگر پیش‌بینی شده است ولی می‌توان از آن برای خروج اطلاعات کامپیوتر و ارسال داده به مدارهای منطقی و دیجیتال، میکرو پروسسورها و میکروکنترلرها و غیره نیز استفاده نمود. پورت موازی مطابق شکل زیر در پشت کامپیوتر قرار دارد و پایه‌های آن نیز مطابق جدول زیر می‌باشد.

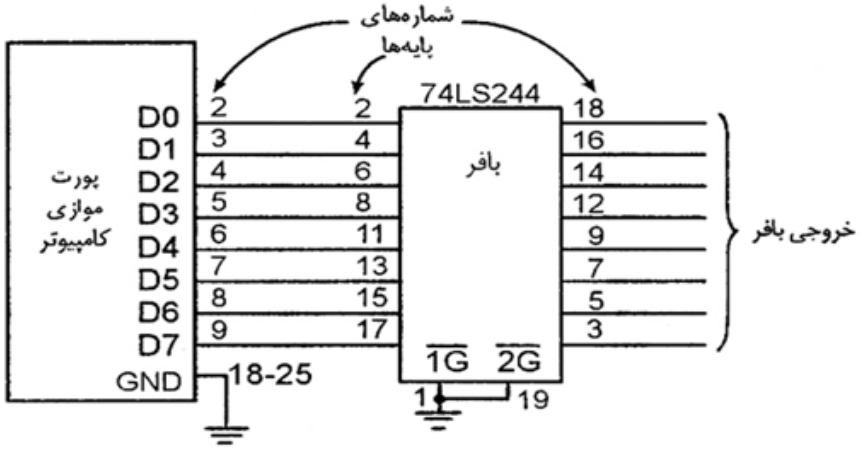
شماره پایه	نام‌پایه‌ها		
1	Strobe		سیگنال خروجی *
2	D0	بیت صفر داده	پایه‌های داده خروجی
3	D1	بیت ۱ داده	
4	D2	بیت ۲ داده	
5	D3	بیت ۳ داده	
6	D4	بیت ۴ داده	
7	D5	بیت ۵ داده	
8	D6	بیت ۶ داده	
9	D7	بیت ۷ داده	
10	Acknowledge		سیگنال تصدیق (ورودی)**
11	Busy		سیگنال مشغول (ورودی)
12	Out of Paper		کاغذ نیست (ورودی)
13	Select		چاپگر انتخاب شده (ورودی)
14	Auto Feed		کاغذ به‌طور خودکار بیاید (خروجی)
15	Error		اشتباه (ورودی)
16	Initialize Printer		مقدار اولیه دادن به چاپگر (خروجی)
17	Select Input		انتخاب ورودی
18 تا 23	زمین		



ارسال اطلاعات توسط کامپیوتر به چاپگر و همچنین سیگنال‌ها، تماماً به وسیله سرویس‌های دستور وقفه INT 17H کنترل می‌شوند و نیازی به دخالت کاربر نیست. هر کامپیوتر می‌تواند چهار پورت موازی به نام‌های LPT1، LPT2، LPT3و LPT4 داشته باشد که چاپگرها به آن متصل باشند و دستورات وقفه INT 17H آدرس این پورت‌ها را با شماره‌های 0 و 1 و 2 و 3 می‌شناسد.

پورت‌های موازی برای ارسال اطلاعات

از پورت‌های موازی برای ارسال اطلاعات به یک بافر یا ثبات (به جای چاپگر) جهت کارهای صنعتی نیز استفاده می‌شود. در این صورت کامپیوتر اطلاعات را به جای چاپگر، به یک بافر می‌فرستد که می‌توان از خروجی بافر برای فرمان مدارهای منطقی، میکرو پروسسورها، میکروکنترلرها و بالاخره ارتباط کامپیوتر با خارج استفاده نمود.



برای ارسال اطلاعات به پورت موازی یا بافر خروجی، باید آدرس آن را در دست داشت. آدرس پورت موازی کامپیوتر در ناحیه داده BIOS از آدرس 0040:0008 تا 0040:000F قرار دارند. موقعی که کامپیوتر را روشن می‌کنیم، برنامه راه‌اندازی یا بوتینگ کامپیوتر بررسی می‌کند که کدامیک از پورت‌های موازی LPT1، LPT2، LPT3و LPT4 در کامپیوتر نصب شده‌اند. مثلاً در صورتی که در کامپیوتر فقط پورت LPT1 قرار داده شده باشد، آدرس آن را از ناحیه حافظه پیدا می‌کند و روی مانیتور می‌نویسد. به عنوان مثال روی مانیتور آدرس پورت موازی LPT1 برابر 378H را می‌نویسد.

پورت‌های موازی	آدرس‌های ناحیه داده BIOS
LPT1	0040:0008-0040:0009
LPT2	0040:000A-0040:000B
LPT3	0040:000C-0040:000D
LPT4	0040:000E-0040:000F

البته با نرم افزار Debug نیز توان آدرس پورت‌های موازی را با فرمان:

-D 40: 08 L8

بدست آورد که در این صورت جواب نرم افزار به صورت:

0040: 0008 78 03 00 00 00 00 00 00

می‌شود که نشان می‌دهد آدرس LPT1 برابر 378H می‌باشد.

حال می‌توان اطلاعات را از ثبات AL مستقیماً به پورت موازی با آدرس 378H ارسال نمود. به عنوان مثال اگر بخواهیم عدد 08H به پورت موازی ارسال شود کافیست دستورات زیر را بنویسیم:

```
MOV AL, 08H
MOV DX, 378H
OUT DX, AL
```

با توجه به مطالب فوق می‌توان هر نوع اطلاعات را از پورت موازی به خارج از کامپیوتر ارسال نمود.

تمرین: برنامه‌ای بنویسید که خروجی‌های پورت موازی D0 تا D7 را از کامپیوتر ارسال نماید.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0