

preview

بزرگترین مرجع کتابهای الکترونیک فارسی و انگلیسی
بزرگترین مرجع نرم افزارهای کاربردی و تخصصی
بزرگترین مرجع دایرود کلیپهای موبایل

www.IranMeet.com

INFO@IRANMEET.COM



وزارت علوم تحقیقات و فن آوری
دانشگاه پیام نور

زبان ماشین و برنامه سازی سیستم

(رشته مهندسی کامپیوتر)

مهندس داریوش نیکمهر

فهرست مطالب

عنوان	صفحه
پیشگفتار	۱
فصل اول : سیستم اعداد	
هدف کلی	۲
اهداف رفتاری	۲
۱-۱- مقادیر دودویی (Binary)	۳
۱-۲- جمع و تفریق در سیستم دوتائی	۵
۱-۳- بایت (Byte)	۸
۱-۴- مقادیر منفی	۸
۱-۵- گروه‌بندی بیت‌ها	۱۲
۱-۶- عملیات در سیستم شانزده‌تائی	۱۴
۱-۷- عملیات در سیستم هشت‌تائی (Octal)	۱۸
۱-۸- مقادیر اعشاری	۲۲
مروری بر مطالب فصل	۲۴
تمرین	۲۵
فصل دوم : معماری ریزپردازنده 80286	
هدف کلی	۲۷
اهداف رفتاری	۲۷
۲-۱- ریز پردازنده 80286	۲۷
۲-۱-۱- ثبات فلگ (Flag register)	۳۰
۲-۱-۲- ثبات IP	۳۳
۲-۱-۳- صف دستورالعمل (Instruction Queue)	۳۳
مروری بر مطالب فصل	۳۵
تمرین	۳۶

فصل سوم : برنامه‌نویسی

۳۷	هدف کلی
۳۷	اهداف رفتاری
۳۸	۳-۱- برنامه و دستورالعملها
۳۸	۳-۲- قانون نامگذاری
۳۹	۳-۳- متغیرها (Variables)
۳۹	۳-۴- برچسبها (Labels)
۴۰	۳-۵- ثابتها (Constants)
۴۲	۳-۶- فیلد عملیات
۴۲	۳-۷- فیلد عملوند
۴۳	۳-۸- فیلد ملاحظات (Comment)
۴۳	۳-۹- تکنیکهای آدرس‌دهی
۴۴	۳-۹-۱- آدرس دهی بلاواسطه
۴۴	۳-۹-۲- آدرس‌دهی مستقیم
۴۴	۳-۹-۳- آدرس‌دهی رجیستر
۴۵	۳-۹-۴- آدرس‌دهی غیرمستقیم رجیستر
۴۵	۳-۹-۵- آدرس‌دهی مبنا
۴۶	۳-۹-۶- آدرس‌دهی اندیس مستقیم
۴۷	۳-۹-۷- آدرس‌دهی اندیس مبنا
۴۸	مروری بر مطالب فصل
۴۹	تمرین

فصل چهارم : دستورالعملهای اساسی

۵۰	هدف کلی
۵۰	اهداف
۵۱	۴-۱- انتقال داده‌ها در حافظه
۶۰	۴-۲- دستورالعمل LEA
۶۱	۴-۳- مبادله داده‌ها
۶۳	۴-۴- جمع و تفریق

۸۰.....	۴-۵- ضرب دو مقدار
۸۸.....	۴-۶- ضرب دو مقدار 32 بیتی بدون علامت
۹۰.....	۴-۷- تقسیم دو مقدار
۹۶.....	۴-۸- دستورالعملهای کاهش و افزایش
۹۹.....	۴-۹- دستورالعمل محاسبه مکمل ۲
۱۰۱.....	مروری بر مطالب فصل
۱۰۲.....	تمرین

فصل پنجم : انشعاب و تکرار

۱۰۴.....	هدف کلی
۱۰۴.....	اهداف رفتاری
۱۰۴.....	۵-۱- دستورالعمل پرش غیر شرطی
۱۰۵.....	۵-۲- دستورالعملهای پرش شرطی
۱۰۹.....	۵-۳- دستورالعمل مقایسه
۱۱۳.....	۵-۴- دستورالعملهای تکرار
۱۱۸.....	مروری بر مطالب فصل
۱۱۹.....	تمرین

فصل ششم : عملیات بیتی

۱۲۲.....	هدف کلی
۱۲۲.....	اهداف رفتاری
۱۲۳.....	۶-۱- عملیات منطقی
۱۲۳.....	۶-۱-۱- دستورالعمل NOT
۱۲۳.....	۶-۱-۲- دستورالعمل AND
۱۲۵.....	۶-۱-۳- دستورالعمل OR
۱۲۶.....	۶-۱-۴- دستورالعمل XOR
۱۲۷.....	۶-۱-۵- دستورالعمل TEST
۱۳۲.....	۶-۲- عملیات شیفت
۱۳۳.....	۶-۲-۱- دستورالعمل SHL

۱۳۴.....	SHR دستور العمل
۱۳۶.....	SAL دستور العمل
۱۳۷.....	SAR دستور العمل
۱۳۸.....	عملیات چرخش (Rotate)
۱۳۸.....	ROL دستور العمل
۱۴۰.....	ROR دستور العمل
۱۴۱.....	RCL دستور العمل
۱۴۳.....	RCR دستور العمل
۱۴۵.....	عملیات فلگها
۱۴۶.....	تبدیل حروف
۱۴۸.....	مروری بر مطالب فصل
۱۴۹.....	تمرین

فصل هفتم: مکروها و روالها و وقفهها

۱۵۱.....	هدف کلی
۱۵۱.....	اهداف رفتاری
۱۵۲.....	پشته (Stack)
۱۵۲.....	PUSH دستور العمل
۱۵۳.....	POP دستور العمل
۱۵۴.....	PUSHF دستور العمل
۱۵۵.....	POPF دستور العمل
۱۵۵.....	روال (Procedures)
۱۵۷.....	مکروها (Macros)
۱۶۱.....	Macro directives دیرکتیوها
۱۶۵.....	EXITM دستور العمل
۱۶۶.....	IRP دستور العمل
۱۶۷.....	IRPC دستور العمل
۱۶۸.....	REPT دستور العمل
۱۷۱.....	LOCAL دیرکتیو

۱۷۵.....	۷-۳-۷- عملگرهای مکرو
۱۷۵.....	۷-۳-۸- عملگر &
۱۷۷.....	۷-۴- وقفه‌ها (Interrupts)
۱۷۷.....	۷-۴-۱- نحوه کار وقفه‌ها
۱۷۸.....	۷-۴-۲- منابع وقفه‌ها
۱۷۹.....	۷-۴-۳- وقفه‌های رزرو شده (Reserved Interrupts)
۱۷۹.....	۷-۴-۴- وقفه‌های سیستم
۱۸۰.....	۷-۴-۵- وقفه‌های DOS
۱۸۱.....	۷-۴-۶- دستورالعملهای وقفه
۱۸۴.....	۷-۴-۷- فراخوانی تابع وقفه نوع 21
۱۹۷.....	۷-۵- خواندن رشته‌ها
۱۹۹.....	۷-۶- عملیات date و time
۲۰۰.....	۷-۶-۱- اندازه‌گیری زمان اجرای برنامه‌ها
۲۰۱.....	۷-۶-۲- ایجاد تأخیر (Generating delays)
۲۰۴.....	۷-۷- کدهای اسکی و دودویی
۲۰۵.....	۷-۷-۱- تبدیل رشته‌های ASCII به دودویی
۲۱۳.....	مروری بر مطالب فصل
۲۱۴.....	تمرین

فصل هشتم : عملیات پردازش رشته‌ها

۲۱۶.....	هدف کلی
۲۱۶.....	اهداف رفتاری
۲۱۶.....	۸-۱- رشته (String)
۲۱۷.....	۸-۱-۱- دستورالعمل MOVS
۲۲۲.....	۸-۱-۲- دستورالعمل STOS
۲۲۵.....	۸-۱-۳- دستورالعمل LODS
۲۲۵.....	۸-۱-۴- دستورالعمل CMPS
۲۲۸.....	۸-۱-۵- دستورالعمل SCAS
۲۳۱.....	مروری بر مطالب فصل

۲۳۲.....	تمرین
فصل نهم : برنامه‌های نمونه	
۲۳۳.....	هدف کلی
۲۳۳.....	اهداف رفتاری
۲۳۴.....	۹-۱- اجزای یک برنامه.....
۲۳۴.....	۹-۲- یک برنامه نمونه.....
۲۳۶.....	۹-۳- نحوه اجرای برنامه.....
۲۳۷.....	۹-۴- برنامه‌های اسمبلی نوشته شده.....
۲۶۹.....	مروری بر مطالب فصل.....
فصل دهم : اسمبلی 80386	
۲۷۰.....	هدف کلی
۲۷۰.....	اهداف رفتاری
۲۷۰.....	۱۰-۱- ریز پردازنده 80386.....
۲۷۱.....	۱۰-۲- انواع داده‌ها.....
۲۷۲.....	۱۰-۳- محاسبه آدرس مؤثر (Effective Address).....
۲۷۲.....	۱۰-۴- معماری.....
۲۷۶.....	۱۰-۵- دستورالعملهای 80386.....
۲۷۹.....	مجموعه کامل دستورالعملهای 80386.....
۲۸۵.....	مروری بر مطالب فصل.....
ضمائم	
۲۸۶.....	ضمیمه ۱: عملگرها (OPERATORS).....
۲۹۰.....	ضمیمه شماره ۲: Instruction Set Summary.....
۲۹۳.....	ضمیمه شماره ۳: Instruction times.....
۳۰۰.....	ضمیمه شماره ۴: کد ماشین دستورالعمل‌ها.....
۳۰۲.....	ضمیمه شماره ۵: جدول کد اسکی.....
۳۰۳.....	ضمیمه شماره ۶: کد دستورالعمل‌ها.....
۳۱۸.....	سئوالات چهار گزینه‌ای.....
۳۶۸.....	واژه نامه.....

پیشگفتار

با لطف و عنایت پرودگار متعال کتاب زبان ماشین و برنامه سازی سیستم با توجه به نیاز دانشجویان دانشگاه پیام نور در رشته مهندسی کامپیوتر بصورت خودآموز و با مثالهای زیاد و ساده و روان تهیه گردیده است. مطالب ارائه شده با توجه به تجربیات تدریس در درس برنامه نویسی زبان اسمبلی در سالیان متمادی در دانشگاه می باشد. این کتاب در ده فصل آماده شده که هر فصل دارای اهداف فصل، تمرین و مروری بر مطالب فصل می باشد. در انتهای کتاب سؤالات چهار گزینه ای و نهایتاً واژه نامه گنجانده شده است. در نهایت از زحمات و همکاری آقای مهندس نوید نیک مهر در تهیه این کتاب قدردانی و سپاسگزاری نموده و این کتاب را به همسر مهربانم و فرزندان دلبندم که همواره مشوق اینجانب در کارهای علمی و پژوهشی می باشند تقدیم می نمایم.

داریوش نیک مهر

فصل اول

سیستم اعداد

هدف کلی

نمایش مقادیر در سیستم دودویی و نحوه تبدیل آنها به سایر سیستمها.

اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می شوید.

۱- مقادیر دودویی یا باینری.

۲- واحدهای مختلف اندازه گیری حافظه.

۳- نمایش اعداد منفی.

۴- تبدیل مقادیر باینری به سیستم دهدهی و برعکس.

۵- نمایش مقادیر در سیستم شانزده تایی.

۶- نمایش مقادیر در سیستم هشت تایی.

۷- تبدیل مقادیر از سیستم دهدهی به سیستم هشت تایی و برعکس.

۸- تبدیل مقادیر از سیستم شانزدهدهی به سیستم دهدهی و برعکس.

۹- تبدیل مقادیر از سیستم شانزدهدهی به سیستم مبنای هشت و برعکس.

۱-۱- مقادیر دودویی (Binary)

بشر با توجه به تعداد انگشت‌هایش از ده رقم 0,1,2,3,4,5,6,7,8,9 برای ایجاد مقادیر و اعداد و انجام محاسبات روی آنها استفاده می‌نماید. به بیانی دیگر بشر در یک سیستم دهدهی یا Decimal کار می‌کند. از طرف دیگر کامپیوتر در یک سیستم دودویی یا Binary کار می‌کند و فقط دو رقم 1 و 0 را می‌شناسد. در نتیجه هر مقداری که به کامپیوتر داده شود بایستی تبدیل به یک سری 0 و 1 گردد تا بتواند در کامپیوتر ذخیره و مورد استفاده در محاسبات قرار گیرد. برای تبدیل مقادیر از سیستم دهدهی به سیستم دودویی بایستی آن مقدار بطور متوالی بر 2 تقسیم نمائیم. بعنوان مثال عدد 50 را در نظر بگیرید.

مثال ۱-۱

مقدار	تقسیم بر	نتیجه	باقیمانده
50	2	25	0
25	2	12	1
12	2	6	0
6	2	3	0
3	2	1	1
1	2	0	1

عدد 50 معادل 110010 در سیستم دودویی می‌باشد.

به منظور تبدیل مقداری از سیستم باینری به سیستم دهدهی، ارقام عدد را می‌بایستی بترتیب از راست به چپ در 1، 2، 8، 16 ... ضرب نموده با هم جمع نمائیم. به عنوان مثال عدد 11010 در سیستم دودویی را در نظر بگیرید.

مثال ۱-۲

1	1	0	1	0
16	8	4	2	1
16*1+				16+
8*1				8
4*0				0
2*1				2
1*0				0
				26

بعبارت دیگر ارقام را بایستی بترتیب در 2^0 ، 2^1 ، 2^2 ، 2^3 ، 2^4 ... ضرب نمود.

1	1	0	1	0
2^4	2^3	2^2	2^1	2^0

مثال ۱-۳

عدد 37 را به سیستم دودویی تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
1	18	2	37
0	9	2	18
1	4	2	9
0	2	2	4
0	1	2	2
1	0	2	1

بنابراین مقدار 37 برابر با 100101 در سیستم دودویی می‌باشد.

مثال ۴-۱

عدد 1101101 را به سیستم دهدهی تبدیل نمائید.

1	1	0	1	1	0	1
64	32	16	8	4	2	1

64+

32

8

4

1

109

نتیجه میشود که عدد 1101101 در سیستم دودویی معادل 109 در سیستم دهدهی می‌باشد.

۲-۱- جمع و تفریق در سیستم دودویی

جمع و تفریق در سیستم دودویی شبیه جمع و تفریق در سیستم دهدهی می‌باشد با این تفاوت که به جای ده بر یک، دو بر یک (Carry) ایجاد می‌شود. فرض کنید دو مقدار 3 و 10 در سیستم دودویی با هم جمع نمائیم. ابتدا بایستی هر کدام از این مقادیر را به سیستم دودویی تبدیل نموده سپس آنها را با هم جمع نمائیم.

10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

ملاحظه می‌شود که 10 در سیستم دودویی برابر است با 1010.

از طرف دیگر مقدار 3 در سیستم دودویی را بدست می‌آوریم.

3	2	1	1
1	2	0	1

حال دو مقدار 11 و 1010 با هم جمع می‌نمائیم.

$$\begin{array}{r}
 1 \quad \text{Carry} \\
 1010+ \\
 11 \\
 \hline
 1101
 \end{array}$$

در مورد 1+1 بایستی در نظر داشت که نتیجه میشود 10. که یک carry یک به ستون بعدی منتقل می‌گردد.

مثال ۵-۱

مجموع دو مقدار 20 و 17 را بدست آورید.

ابتدا مقادیر 17 و 20 را به سیستم دودویی تبدیل می‌نمائیم.

20	2	10	0
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

مقدار 20 میشود 10100 در سیستم دودویی.

17	2	8	1
8	2	4	0
4	2	2	0
2	2	1	0
1	2	0	1

این نشان می دهد که 17 معادل 10001 در سیستم دودویی می باشد.

حال

$$\begin{array}{r}
 1 \quad \text{Carry} \\
 10001+ \\
 10100 \\
 \hline
 100101
 \end{array}$$

که این مقدار یعنی 100101 اگر به سیستم دهدهی تبدیل شود برابر است با

1	0	0	1	0	1
32	16	8	4	2	1

$$\begin{array}{r}
 32+ \\
 4 \\
 1 \\
 \hline
 37
 \end{array}$$

در مورد تفریق در سیستم دهدهی همانطوریکه ملاحظه می گردد در صورت

لزوم یک 1 در سیستم دهدهی قرض گرفته می شود.

مثال ۶-۱

$$\begin{array}{r}
 534 - \\
 281 \\
 \hline
 253
 \end{array}$$

ولی در سیستم دودویی در صورت لزوم یک 1 در سیستم دودویی قرض گرفته که borrow نامیده می شود. مثال

$$\begin{array}{r} 1011- \\ 0110 \\ \hline 0101 \end{array}$$

۳-۱- بایت (Byte)

در حافظه کامپیوتر فقط مقادیر 0 و 1 ذخیره میشود. به ارقام 0 و 1 بیت گفته میشود. بیت مخفف کلمات binary digit می باشد. به هر هشت بیت کنار هم در حافظه کامپیوتر بایت گفته میشود. بیت های یک بایت از 0 تا 7 شماره گذاری شده و بیت شماره 0 بیت کم ارزش ترین یا LSB و بیت شماره 7 بیت با بیشترین ارزش یا MSB می باشد.

7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	1

هر بایت 256 وضعیت مختلفه از 0 و 1 را ایجاد می نماید. بنابراین اعداد صحیح بین 0 تا 255 را می توان در یک بایت قرار داد. از طرف دیگر در کامپیوتر از 256 کارکتر مختلف می توان استفاده نمود. با استفاده از جدول کد ASCII می توان به هر کاراکتر یک کد منحصر بفرد بین 0 تا 255 تخصیص داد. بنابراین هر کاراکتر عملاً یک بایت اشغال می نماید.

۴-۱- مقادیر منفی

اعداد و مقادیر منفی در کامپیوتر با استفاده از روش مکمل 2 نمایش داده می شوند. برای نمایش یک مقدار منفی در کامپیوتر بایستی مراحل زیر را طی نمود.

- ۱- ابتدا عدد را بدون علامت تصور نموده آنرا به سیستم دودویی تبدیل نمائید.
- ۲- سپس آنقدر رقم 0 در سمت چپ نتیجه مرحله 1 قرار می‌دهیم تا تعداد ارقام آن مضربی از هشت گردد. چنانچه نتیجه مرحله 1 از هشت رقم بیشتر باشد بایستی آنقدر 0 در سمت چپ قرار دهیم تا شانزده رقمی گردد.
- ۳- سپس ارقام نتیجه مرحله 2 را مکمل می‌نمائیم یعنی 0 به 1 و 1 به 0 تبدیل می‌کنیم.
- ۴- نتیجه بدست آمده را در سیستم دودویی با 1 جمع می‌نمائیم.

مثال ۷-۱

عدد 26- را در نظر بگیرید. ابتدا عدد 26 را به سیستم دودویی تبدیل می‌نمائیم.

26	2	13	0
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

که میشود 11010.

حال نتیجه بدست آمده را هشت رقمی می‌نمائیم.

00011010

سپس 0 ها را به 1 و 1 ها را به 0 تبدیل می‌کنیم.

11100101

حال نتیجه بدست آمده را با 1 جمع می‌نمائیم.

$$\begin{array}{r} 11100101+ \\ 1 \\ \hline 11100110 \end{array}$$

عدد 11100110 در سیستم دودویی نمایش 26- می باشد که یک بایت اشغال می نماید. نکته مهمی که بایستی در نظر داشت این است که MSB اعداد منفی در روش مکمل 2 همیشه 1 می باشد.

مثال ۸-۱

عدد 35- را به سیستم دودویی تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
1	17	2	35
1	8	2	17
0	4	2	8
0	2	2	4
0	1	2	2
1	0	2	1

که نتیجه می شود 35 معادل 100011 در سیستم دودویی می باشد. حال نتیجه بدست آمده را هشت رقمی می نمائیم.

00100011

سپس 0ها را به ۱ و 1ها را به 0 تبدیل می کنیم.

11011100

حال نتیجه بدست آمده را با 1 جمع می کنیم

$$\begin{array}{r} 1101100 + \\ 1 \\ \hline 11011101 \end{array}$$

مقدار 11011101 در سیستم دودویی معادل 35- می باشد. که همانطوریکه ملاحظه می گردد بیت MSB آن برابر با یک می باشد.

مثال ۹-۱

عمل زیر را با استفاده از روش مکمل 2 انجام دهید.

$$\begin{array}{r} 27- \\ \underline{20} \end{array}$$

این عمل تفریق در حقیقت بمنزله جمع دو مقدار زیر می باشد.

$$27+(-20)$$

حال مقادیر 20- و 27 را به سیستم دودویی تبدیل نموده.

27	2	13	1
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

مقدار 27 معادل 11011 در سیستم دودویی می باشد. حال ابتدا مقدار 20 را

به سیستم دودویی تبدیل نمود.

20	2	10	0
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

مقدار 20 برابر است با 10100 در سیستم دودویی. حال 20- را در سیستم

دودویی بدست می آوریم. برای این کار ابتدا عدد را هشت رقمی نموده

00010100

سپس صفرها را به 1 و یکها را به صفر تبدیل می نمائیم.

11101011

آنگاه مقدار 1 به آن اضافه می‌نمائیم.

$$\begin{array}{r} 11101011+ \\ 1 \\ \hline 11101100 \end{array}$$

نتیجه می‌شود که مقدار 20- برابر است با 11101100 در سیستم دودویی.
حال دو مقدار 20- و 27 را در سیستم دودویی با هم جمع می‌نمائیم.

$$\begin{array}{r} 11101100 + \\ 11011 \\ \hline 100000111 \end{array}$$

با توجه به آنکه نتیجه جمع دو بایت بصورت یک بایت می‌باشد بیت 1 سمت چپ بایستی حذف گردد، نتیجه می‌شود 111 که برابر با 7 می‌باشد.

۵-۱- گروه‌بندی بیت‌ها

به هر هشت بیت کنار هم بایت گفته می‌شود. دو بایت کنار هم یعنی شانزده بیت متوالی را word می‌نامند(البته در بعضی از کامپیوترها هر کلمه می‌تواند شامل ۴ بایت باشد). بیت‌های یک word از 0 تا 15 شماره‌گذاری می‌گردد. در یک word بایت سمت راست را بایت مرتبه پائین (Low order byte) و بایت سمت چپ را بایت مرتبه بالا (High order byte) گفته می‌شود.

15	8	7	0
High order byte		Low order byte	

در یک Word بیت شماره 0 را LSB و بیت شماره 15 را MSB می‌نامند.
از طرف دیگر چهار بایت متوالی تشکیل یک Double word می‌دهند.

31	24	23	16	15	8	7	0

هر هشت بایت متوالی تشکیل یک Quadword می‌دهد و نهایتاً هر هشتاد و یک بیت متوالی یا ده بایت متوالی تشکیل یک Tenbyte می‌دهد. جدول ذیل مقادیری که در یک byte ، word ، double word قرار می‌گیرند را نشان می‌دهد.

جدول ۱-۱

نوع	مقادیر بدون علامت	مقادیر علامت دار
Byte	0 تا 255	-128 تا 127
Word	0 تا 65535	-32768 تا 32767
Double word	0 تا $2^{32}-1$	-2^{31} تا $2^{31}-1$

بایستی توجه داشت که عملیات باینری روی بیت‌ها انجام می‌شود. جدول عملگر جمع بصورت زیر می‌باشد.

جدول ۱-۲

بیت 1	بیت 2	نتیجه	دوبریک (carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

جدول عملگر تفریق نیز بصورت زیر می‌باشد.

جدول ۱-۳

بیت 1	بیت 2	نتیجه	یک قرضی (borrow)
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

۱-۶- عملیات در سیستم مبنای شانزده

ارقام در سیستم مبنای شانزده یا Hexadecimal عبارتند از 0 تا 15. بمنظور جلوگیری از ابهام، ارقام 10 تا 15 را بترتیب با حروف A تا F نشان داده میشوند.

A	10
B	11
C	12
D	13
E	14
F	15

برای تبدیل مقداری از سیستم دهدهی به سیستم مبنای شانزده آن عدد را بطور متوالی بر 16 تقسیم می‌نمائیم. بعنوان مثال عدد 174 را در نظر بگیرید.

مثال ۱-۱۰

باقیمانده	نتیجه	تقسیم بر	مقدار
E 14	10	16	174
A 10	0	16	10

که نتیجه می‌شود AE.

مثال ۱-۱۱

عدد 3740 را از سیستم دهدهی به سیستم مبنای شانزده تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
12 C	233	16	3740
9	14	16	233
14 E	0	16	14

چون 14 معادل E می باشد و C معادل 12 می باشد بنابراین جواب می شود E9C در سیستم مبنای شانزده.

مثال ۱۲-۱

مقدار 27845 را به سیستم مبنای شانزده تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
5	1740	16	27845
12 C	108	16	1740
11 B	6	16	108
6	0	16	6

مقدار 27845 برابر با 6CC5 در سیستم شانزدهدهی می باشد.
برای تبدیل مقداری از سیستم شانزدهدهی به سیستم دهدهی ارقام عدد را از سمت راست بترتیب در 1, 16, 16², 16³, ... ضرب نموده با هم جمع می نمائیم.

مثال ۱۳-۱

عدد 2AF5 را در نظر بگیرید.

5	F	A	2
1	16	16 ²	16 ³

$$\begin{array}{r}
 5 + \\
 15 * 16 \\
 10 * 256 \\
 2 * 4096 \\
 \hline
 10997
 \end{array}$$

که نتیجه منجر میشود به $2AF5$ که برابر با 10997 می باشد.

مثال ۱۴-۱

مقدار $4F2$ در سیستم مبنای شانزده چه مقدار در سیستم دهدهی می باشد؟ برای اینکار ابتدا رقم ۲ را در ۱، رقم F را در ۱۶ و رقم ۴ را در 16^2 ضرب می نمائیم. سپس مقادیر بدست آمده را با هم جمع می کنیم.

$$\begin{array}{r} 4 \quad F \quad 2 \\ 16^2 \quad 16 \quad 1 \\ \hline 4*16^2+ \\ F*16 \\ 2*1 \end{array}$$

که منجر میشود به

$$\begin{array}{r} 4*256+ \\ 15*16 \\ 2*1 \end{array}$$

که نتیجه می شود

$$\begin{array}{r} 1024+ \\ 240 \\ 2 \\ \hline 1266 \end{array}$$

مقدار $4F2$ در سیستم شانزدهدهی برابر با 1266 در سیستم دهدهی می باشد.

از طرف دیگر هر رقم در سیستم مبنای شانزده را می توان بوسیله چهار رقم در سیستم باینری نمایش داد.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

حال برای تبدیل یک مقدار در سیستم مبنای شانزده به سیستم دودویی می‌توان از جدول مذکور استفاده نموده و ارقام را با مقدار معادل آن جایگزین نمود. به عنوان مثال عدد 2FA5B را در نظر بگیرید. با جایگزینی هر رقم با چهار رقم معادل آن در سیستم دودویی نتیجه زیر حاصل می‌گردد.

00101111101001011011

به منظور تبدیل یک مقدار از سیستم دودویی به سیستم مبنای شانزده ابتدا ارقام را از سمت راست چهار تا چهار تا جدا نموده سپس با استفاده از جدول فوق مقادیر معادل را قرار می‌دهیم.

مثال ۱۵-۱

111011001011101

که ابتدا بصورت زیر در می‌آوریم.

0111 0110 0101 1101

که معادل 765D می‌باشد.

۱-۷- عملیات در سیستم مبنای هشت (Octal)

ارقام در سیستم مبنای هشت عبارتند از 0 تا 7. برای تبدیل مقداری از سیستم دهدهی به سیستم مبنای هشت بایستی آن مقدار را بطور متوالی بر هشت تقسیم نمود. بعنوان مثال عدد 125 را در نظر بگیرید.

مثال ۱-۱۶

باقیمانده	نتیجه	تقسیم بر	مقدار
5	15	8	125
7	1	8	15
1	0	8	1

که نتیجه می شود 175 در سیستم مبنای هشت.

مثال ۱-۱۷

بمنظور تبدیل مقداری از سیستم مبنای هشت به سیستم دهدهی، ارقام عدد را از سمت راست بترتیب در 1، 8، 8^2 ، 8^3 ، ... ضرب نموده نتایج حاصله را با هم جمع می نماییم. بعنوان مثال عدد 237 در سیستم مبنای هشت را در نظر بگیرید.

2 3 7

8^2 8 1

$7*1+$	7
$3*8$	24
$2*8^2$	128
	159

که نتیجه می شود 159 در سیستم دهدهی.

مثال ۱۸-۱

عدد 4260 را از سیستم دهدهی به سیستم مبنای هشت تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
4	532	8	4260
4	66	8	532
2	8	8	66
0	1	8	8
1	0	8	1

نتیجه می شود که 4260 در سیستم دهدهی معادل 10244 در سیستم مبنای هشت می باشد.

مثال ۱۹-۱

عدد 382 را به سیستم مبنای هشت تبدیل نمائید.

باقیمانده	نتیجه	تقسیم بر	مقدار
6	47	8	382
7	5	8	47
5	0	8	5

که نتیجه می شود 576 در سیستم مبنای هشت.

مثال ۲۰-۱

مقدار 4327 را از سیستم مبنای هشت به سیستم دهدهی تبدیل نمائید.
 برای اینکار ابتدا رقم 7 را در 1، رقم 2 را در 8، رقم 3 را در 8^2 و نهایتاً رقم 4 را در 8^3 ضرب می نمائیم سپس مجموع مقادیر بدست آمده را محاسبه می نمائیم.

4 3 2 7

 $8^3 8^2 8^1$

که نتیجه می شود

 $4*8^3 +$ $3*8^2$ $2*8$ $7*1$

که معادل است با

 $4*512 +$ $3*64$ $2*8$ $7*1$

که نهایتاً برابر است با

 $2048 +$

192

16

7

2263

بنابراین مقدار 4327 در سیستم مبنای هشت برابر است با 2263 در سیستم

دهدهی.

بایستی توجه نمود که هر رقم در سیستم مبنای هشت را می توان بوسیله سه

رقم در سیستم دودویی نمایش داد.

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

برای تبدیل مقداری از سیستم هشت تایی به سیستم دودویی کافی است که به جای هر رقم در سیستم مبنای هشت سه رقم معادل آنرا قرار داد. بعنوان مثال عدد 417 در سیستم مبنای هشت معادل 100001111 در سیستم دودویی می باشد. بمنظور تبدیل مقداری از سیستم دودویی به سیستم مبنای هشت کافی است که ارقام عدد از طرف راست سه تا سه تا جدا نموده و به جای آنها مقدار معادل در سیستم مبنای هشت قرار دهیم.

مثال ۲۱-۱

عدد 10110111010111 در سیستم دودویی در نظر بگیرید که می توان بصورت زیر جدا نمود.

010 110 111 010 111

که جواب نهائی میشود 26727 در سیستم مبنای هشت. از طرف دیگر برای تبدیل مقداری از سیستم مبنای هشت به سیستم مبنای شانزده و برعکس می بایستی ابتدا مقدار را به سیستم دودویی تبدیل نموده سپس به سیستم مبنای هشت یا مبنای شانزده تبدیل نمود.

مثال ۲۲-۱

عدد 2AFB5 را در نظر بگیرید.

2AFB5

2	A	F	B	5
0010	1010	1111	1011	0101

حال سه رقم سه رقم از سمت راست جدا نموده.

000 101 010 111 110 110 101

که نهایتاً برابر با 527665 در سیستم مبنای هشت می‌باشد.

۸-۱- مقادیر اعشاری

به منظور تبدیل یک مقدار اعشاری به سیستم دودویی ابتدا قسمت صحیح آنرا به طریق گفته شده به سیستم دودویی تبدیل نموده، سپس قسمت اعشاری آنرا جدا نموده بطور مکرر در 2 ضرب می‌نمائیم. بعنوان مثال عدد 14.725 را در نظر بگیرید. عدد 14 بصورت 1110 در سیستم دودویی می‌باشد. برای تبدیل قسمت اعشاری یعنی 0.725 به سیستم دودویی آنرا در 2 ضرب می‌نمائیم.

مثال ۲۳-۱

$$\begin{array}{r} 0.725* \\ 2 \\ \hline 1.450 \end{array}$$

قسمت صحیح یعنی 1 را جدا نموده، قسمت اعشار را در 2 ضرب

می‌نمائیم.

$$\begin{array}{r} 0.45* \\ 2 \\ \hline 0.90 \end{array}$$

قسمت صحیح یعنی 0 را جدا نموده، قسمت اعشار را در 2 ضرب می‌کنیم.

$$\begin{array}{r} 0.8* \\ 2 \\ \hline 1.6 \end{array}$$

و به همین روال کار را ادامه می‌دهیم.

$$\begin{array}{r} 0.6* \\ 2 \\ \hline 1.2 \end{array}$$

جواب می‌شود 1110.10111

برای تبدیل یک مقدار اعشاری از سیستم دودویی به سیستم دهدهی قسمت صحیح آنرا از سمت راست بترتیب در 2^0 ، 2^1 ، 2^2 ، 2^3 ، ... ضرب نموده با هم جمع می‌کنیم سپس قسمت اعشار آنرا بترتیب از سمت چپ در 2^{-1} ، 2^{-2} ، 2^{-3} ، 2^{-4} ، ... ضرب نموده با هم جمع می‌نمائیم. بعنوان مثال عدد 1101.01011 در سیستم دودویی را در نظر بگیرید.

مثال ۲۴-۱

1	1	0	1	.	0	1	0	1	1
8	4	2	1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

$$1*8 + 4*1 + 2*0 + 1*1 + 0*\frac{1}{2} + 1*\frac{1}{4} + 0*\frac{1}{8} + 1*\frac{1}{16} + 1*\frac{1}{32}$$

که خلاصه می‌شود

$$8 + 4 + 1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{32} = 13.34375$$

مروری بر مطالب فصل

در حافظه کامپیوتر فقط ارقام 0 و 1 ذخیره می‌گردد. به این ارقام 0 و 1 بیت گفته می‌شود. به هر هشت کنار هم بایت و به هر شانزده بیت کنار هم word گفته می‌شود. برای ذخیره یک مقدار در حافظه بایستی ابتدا آنرا بصورت یک‌سری بیت درآورد. و برای نمایش مقادیر روی صفحه مانیتور بایستی آنها را به سیستم ده‌تایی تبدیل نمود. در سیستم دهدهی فقط از ارقام 0 تا 9 استفاده می‌گردد. در سیستم دودویی فقط از ارقام 0 و 1 استفاده می‌گردد. اعداد منفی را می‌توان در حافظه کامپیوتر با استفاده از روش مکمل 2 نشان داد. استفاده از سیستم‌هایی مبنای شانزده و مبنای هشت نیز امکان پذیر می‌باشد. در سیستم مبنای شانزده از ارقام 0 تا 9 و A تا F استفاده می‌گردد. در سیستم مبنای هشت فقط از ارقام 0 تا 7 می‌توان استفاده نمود. مقادیر اعشاری را نیز می‌توان در حافظه کامپیوتر قرار داد.

☞ تمرین

- ۱- هر بایت از بیت تشکیل شده است.
- ۲- هر کلیو بایت معادل بایت می باشد.
- ۳- در هر Word اعداد 0 تا را می توان جا داد.
- ۴- در هر بایت اعداد - تا + را می توان قرار داد.
- ۵- در کامپیوتر برای نمایش اعداد منفی از استفاده می شود.
- ۶- در کامپیوتر به جای عمل تفریق از استفاده می گردد.
- ۷- در هر Word به تعداد وضعیت مختلفه 1 و 0 وجود دارد.
- ۸- عدد 20- را به سیستم دودویی تبدیل نمائید.
- ۹- عمل 18-25 را با استفاده از روش مکمل دو انجام دهید.
- ۱۰- عدد 1101101 را به سیستم دهدهی تبدیل نمائید.
- ۱۱- عدد 101101.11001 را به سیستم ده تائی تبدیل کنید.
- ۱۲- عدد 2FABC را به سیستم هشت تائی تبدیل نمائید.
- ۱۳- عدد 43271 در سیستم مبنای هشت را به سیستم دودویی تبدیل نمائید.
- ۱۴- مشخص نمائید که اگر MSB یک مقداری یک باشد آیا آن مقدار منفی است؟
- ۱۵- مشخص نمائید که آیا می توان عمل ضرب و تفریق و تقسیم را به عمل جمع تبدیل نمود؟
- ۱۶- اگر مقداری منفی باشد.
- الف- MSB آن صفر است.
- ب- MSB آن یک است.
- ج- LSB آن یک است.
- د- هیچکدام.

۱۷- اگر مقداری از نوع double word داشته باشیم به چند بایت حافظه نیاز است؟

- الف- 2
ب- 3
ج- 4
د- هیچکدام

۱۸- مقدار 7AB در سیستم شانزدهمی چه مقداری در سیستم دودویی می باشد؟

- الف- 11110101011
ب- 101110100111
ج- 11011011111
د- هیچکدام

۱۹- مقدار 6- در سیستم دهدهی معادل چه مقداری در سیستم دودویی می باشد؟

- الف- 11111001
ب- 11111010
ج- 11110111
د- هیچکدام

۲۰- تفاضل کد اسکی 'a' و 'A' چیست؟

- الف- 30
ب- 32
ج- 40
د- هیچکدام

فصل دوم

معماری ریزپردازنده 80286

هدف کلی

معرفی ریزپردازنده 80286

اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می شوید

۱- ریزپردازنده 80286 و عملیات آن.

۲- معماری ریزپردازنده ۸۰۲۸۶ و اجزاء تشکیل دهنده.

۳- ثبات ها

۴- فلگ ها

۱-۲- ریز پردازنده 80286

این ریزپردازنده دارای ویژگیهای پیشرفته ای برای عملکرد در سطح بالائی را

دارد. در 80286، همچنین استفاده از ویژگی های ذیل امکان پذیر می باشد.

۱- Multitasking

۲- Multiuser systems

این ریزپردازنده دارای هشت ثابت (Register) شانزده بیتی بنامهای AX، BX، CX، DX، SP، BP، SI، DI می باشد. چهار ثابت AX، BX، CX، DX را می توان بعنوان ثابتهای شانزده بیتی در نظر گرفت یا هر کدام را بعنوان دو ثابت هشت بیتی در نظر گرفت و استفاده نمود.

15	8	7	0	
AH		AL		AX
BH		BL		BX
CH		CL		CX
DH		DL		DX

AX	Accumulator
BX	Base
CX	Count
DX	Data

15	0	
		SP
		BP
		SI
		DI

SP	Stack pointer
BP	Base pointer
SI	Source index

DI Destination index

معمولاً از ثباتهای SP و BP در مورد عملیات روی پشته‌ها و از ثباتهای SI و DI بعنوان شاخص در ساختارهای پیچیده‌تر استفاده می‌گردد. ریزپردازنده همچنین دارای چهار ثبات شانزده‌بیتی معروف به ثباتهای سگمنت می‌باشد که بمنظور آدرس دهی از آنها استفاده می‌نماید. این ثبات‌ها بنامهای CS، DS، SS، ES می‌باشند.

15	0
	CS
	DS
	SS
	ES

CS	Code	Segment
DS	Data	Segment
SS	Stack	Segment
ES	Extra	Segment

سگمنت‌ها ناحیه‌های پیوسته در حافظه می‌باشند. اندازه هر سگمنت می‌تواند تا 64K بایت باشد.

بایستی توجه داشت که در ثبات CS آدرس شروع ناحیه‌ای از حافظه که کدهای دستورالعمل در آن قرار دارد گذاشته می‌شود. در ثبات DS آدرس شروع ناحیه‌ای از حافظه که داده‌ها در آن قرار دارد گذاشته می‌شود. در ثبات SS آدرس شروع ناحیه‌ای از حافظه که پشته در آن ایجاد می‌شود قرار داده می‌شود. و در صورتیکه در برنامه‌ها از دستورالعملهای پردازش

رشته‌ها استفاده شود آدرس ابتدای آن ناحیه در حافظه در ثبات ES قرار داده می‌شود.

۱-۲-۱- ثبات فلگ (Flag register)

ریزپردازنده 80286 دارای یک ثبات شانزده بیتی بنام ثبات فلگ می‌باشد. دوازده بیت از این ثبات فقط می‌تواند مورد استفاده برنامه نویس قرار بگیرد. هر کدام از این بیت‌ها نام خاصی دارد و مشخص کننده وضعیت خاصی می‌باشد.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
//	NT	IO	PL	OF	DF	IF	TF	SF	ZF	//	AF	//	PF	//	CF

CF	Carry Flag
PF	Parity Flag
AF	Auxiliary Flag
ZF	Zero Flag
SF	Sign Flag
OF	Overflow Flag
TF	Trap Flag
IF	Interrupt Enable Flag
IOPL	Input / output privilege Level Flag
NT	Nested Task Flag

زمانی که در عملیات جمع و تفریق (هشت یا شانزده بیتی) یک carry یا borrow ایجاد می‌گردد مقدار CF برابر با یک می‌شود. در غیر اینصورت مقدار فلگ CF برابر با صفر می‌شود. از فلگ CF در دستورالعملهای شیفت و چرخش نیز استفاده می‌گردد.

از فلگ PF اساساً در کاربردهای ارتباطات داده‌ای استفاده می‌شود. زمانیکه توازن فرد ایجاد شود مقدار آن یک می‌شود و در صورت ایجاد توازن زوج مقدار آن

صفر می‌گردد. عبارت دیگر اگر تعداد بیت‌های یک در بایت مرتبه پائین word زوج باشد مقدار PF برابر یک می‌گردد. در غیر اینصورت مقدار PF برابر با صفر می‌گردد.

در عمل جمع دو مقدار اگر کلیه بیت‌های نتیجه صفر گردد مقدار ZF برابر با یک می‌گردد. در غیر اینصورت مقدار ZF برابر با صفر می‌شود.

در عمل جمع دو مقدار باینری اگر از MSB نتیجه یک carry خارج گردد مقدار CF برابر با یک می‌شود. در غیر اینصورت مقدار CF برابر با صفر می‌شود.

در عمل جمع دو مقدار باینری اگر MSB نتیجه برابر با یک باشد مقدار فلگ SF برابر با یک می‌شود. در غیر اینصورت مقدار SF برابر با صفر می‌باشد.

در عمل جمع دو مقدار باینری اگر یک carry در بیت قبل از MSB نتیجه داشته باشیم ولی در بیت carry MSB نداشته باشیم و یا بالعکس مقدار فلگ OF 1 می‌شود. در غیر اینصورت مقدار فلگ OF صفر می‌گردد. زمانی که OF برابر با 1 می‌شود یعنی نتیجه محاسبه در محل تعیین شده در حافظه جا نمی‌گیرد و باعث می‌شود که تعدادی از بیت‌های نتیجه حذف شود.

فلگ AF در مورد عملیات BCD کاربرد دارد که در بخش‌های بعدی بحث می‌شود. در عمل جمع دو مقدار باینری مقدار فلگ AF یک می‌شود اگر از بیت شماره 3 یک carry خارج گردد.

مثال ۱-۲

دو مقدار 2345 و 3219 را در سیستم مبنای شانزده در نظر بگیرید و با هم جمع نمائید و مقدار فلگ‌ها را مشخص نمائید.

$$\begin{array}{r} 00110010\ 00011001+ \\ 0010001101000101 \\ \hline 0101010101011110 \end{array}$$

SF=0	MSB برابر با 0 می باشد.
ZF=0	تمام بیت ها 0 نمی باشند.
PF=0	تعداد بیت های 1 در هشت بیت اول فرد می باشد.
CF=0	در بیت 15، carry نداریم.
AF=0	در بیت شماره 3، carry نداریم.
OF=0	در بیت های 14 و 15، carry نداریم.

مثال ۲-۲

حال مجموع دو مقدار 5439 و 456A در سیستم مبنای شانزده را محاسبه نموده سپس مقدار فلگ ها را مشخص می نمائیم.

$$(5439)_{16} = (0101010000111001)_2$$

$$(456A)_{16} = (0100010101101010)_2$$

$$\begin{array}{r} 0101010000111001 + \\ 0100010101101010 \\ \hline 1001100110100011 \end{array}$$

که فلگ ها بصورت زیر تغییر می یابند.

ZF=0	تمام بیت های نتیجه 0 نیستند.
CF=0	carry در بیت شماره 15 وجود ندارد.
SF=1	MSB نتیجه برابر با یک می باشد.
OF=1	در بیت شماره 14 carry وجود دارد ولی در بیت شماره 15 وجود ندارد.
AF=1	در بیت شماره 3 یک carry وجود دارد.
PF=1	تعداد بیت های یک در بایت مرتبه پائین نتیجه زوج است.

نکته‌ای که بایستی توجه داشت این است که تمام دستورالعملها محتوی فلگ‌ها را تغییر نمی‌دهند. بعنوان مثال دستورالعمل MOV که برای انتقال داده‌ها استفاده می‌شود روی هیچ فلگی اثر ندارد.

۲-۱-۲- ثبات IP

ثبات IP یک ثبات شانزده‌بیتی می‌باشد که آدرس دستورالعمل بعدی که بایستی اجرا گردد در این ثبات قرار داده میشود. ریزپردازنده با استفاده از این آدرس دستورالعمل بعدی را مورد پردازش قرار می‌دهد. برنامه نویس به این ثبات دسترسی نداشته و محتوی آنرا نمی‌تواند تغییر دهد.

۲-۱-۳- صف دستورالعمل (Instruction Queue)

در ریزپردازنده 80286 یک صف دستورالعمل وجود دارد که طول آن 6 بایت می‌باشد و از این صف برای قراردادن تعدادی دستورالعمل که از حافظه به ریزپردازنده منتقل می‌شود استفاده می‌گردد. اینکار سرعت اجرای دستورالعملها را بیشتر می‌نماید. ریزپردازنده از طریق BUS به حافظه متصل می‌گردد. BUS در هر لحظه شانزده‌بیت را از حافظه به ریزپردازنده و یا برعکس منتقل می‌نماید. داده‌ها، داده‌های کنترلی و آدرس می‌توانند توسط BUS بین حافظه و ریزپردازنده جابه‌جا گردند.

همانطوریکه میدانید بایتهای حافظه بصورت منحصر بفرد شماره‌گذاری شده و بعنوان آدرس مورد استفاده کامپیوتر قرار می‌گیرد. بایستی توجه داشت که انتقال داده‌ها فقط بصورت‌های ذیل میسر می‌باشد.

۱- انتقال از یک ثبات به محلی در حافظه.

۲- انتقال از محلی در حافظه به یک ثبات.

۳- انتقال یک مقدار ثابت به یک ثابت.

۴- انتقال یک مقدار ثابت به یک محل در حافظه.

۵- انتقال از یک ثابت به ثابت دیگر.

انتقال داده‌ها بوسیله دستور MOV انجام می‌شود که در فصل‌های بعدی تشریح خواهد شد.

مروری بر مطالب فصل

ریزپردازنده 80286 دارای ویژگیهای خاصی می باشد. این ریزپردازنده دارای هشت ثبات شانزده بیتی بنامها AX، BX، CX، DX، SP، BP، SI، DI می باشد. چهار ثبات AX، BX، CX، DX هر کدام بعنوان دو ثبات هشت بیتی می توانند مورد استفاده قرار گیرند بنامهای AL، AH، BL، BH، CL، CH، DL، DH. ثبات فلگ نیز شانزده بیتی بوده که هر بیت مشخص کننده وضعیت خاصی می باشد که توسط دستورالعملها تغییر می کنند. برخی از این فلگ ها عبارتند از OF، PF، CF، AF، ZF، DF که در برنامه ها کاربرد زیادی دارند. انتقال اطلاعات از حافظه به ثبات، از ثبات به حافظه، مقدار ثابت به حافظه، مقدار ثابت به ثبات، ثبات به ثبات امکان پذیر می باشد. ولی امکان انتقال اطلاعات بین قسمتی از حافظه و قسمت دیگری از حافظه نمی باشد.

⌘ تمرین

- ۱- تعداد ثباتهای هشت بیتی را مشخص نمائید.
- ۲- کار Flag Register را شرح دهید.
- ۳- مکانیزمهای انتقال داده‌ها را بنویسید.
- ۴- دو مقدار A2FB و CD1A را در مبنای شانزده با هم جمع نموده سپس مقدار فلگ‌های AF, OF, SF, ZF, PF, F و را مشخص نمائید.
- ۵- دو مقدار C2BF و A5B2 را در مبنای شانزده با هم جمع نموده سپس مقدار فلگ‌های OF, SF, PF, ZF, CF و AF را مشخص نمائید.
- ۶- مورد استفاده ثباتهای CS, DS, ES, SS را بنویسید.
- ۷- صف دستورالعمل و مورد استفاده آنرا بنویسید.
- ۸- دو مقدار 54C2 و 3271 در سیستم مبنای شانزده را از هم کم نموده سپس مقدار فلگ‌های AF, OF, SF, ZF, PF, CF را مشخص نمائید.
- ۹- آیا تمام دستورالعمل‌ها روی فلگ‌ها اثر دارند؟
- ۱۰- کار BUS را بنویسید.

فصل سوم

برنامه‌نویسی

هدف کلی

آشنائی با شکل کلی دستورالعملها و تکنیکهای انتقال داده .

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

۱- برنامه و شکل کلی دستورالعملها.

۲- نامگذاری فیلدها.

۳- انواع فیلدها و کاربرد آنها.

۴- تکنیکهای آدرس دهی.

۳-۱- برنامه و دستورالعملها

در زبان اسمبلی برنامه تشکیل شده است از تعدادی دستورالعملهای اجرایی که بیانگر عملیاتی است که بایستی انجام شود. این سری دستورالعملها همانطوریکه میدانیم SOURCE CODE یا کد منبع نامیده می شود. مانند هر زبان برنامه نویسی دیگر زبان اسمبلی شکل و قالب از پیش تعریف شده ای برای کد منبع دارد. هر دستورالعمل اسمبلی شامل چهار فیلد می باشد.

فیلد ملاحظات فیلد عملوند فیلد عملیات فیلد اسم

البته بایستی توجه داشت که در بعضی از دستورالعملها از تمام فیلدها استفاده نمی گردد.

۳-۲- قانون نامگذاری

نام در زبان اسمبلی حداکثر می تواند شامل 31 کاراکتر باشد. کاراکترها شامل حروف Z تا A و ارقام 9 تا 0 و سیمبلهای مخصوص @ ؟ . \$ - می باشد. موارد ذیل بایستی در نامگذاری رعایت گردد.

۱- اسم نمی تواند با یک رقم شروع گردد.

۲- اسم نبایستی یکی از کلمات ذخیره شده در اسمبلی باشد.

۳- در صورتیکه از * در نام استفاده گردد، بایستی اولین کاراکتر نام باشد. کلمات زیر اسامی مجاز در اسمبلی می باشند.

LOOP1	B@A2
X	.XY2
Y2A	SUM2
A_5B	ADDX
COUNT	

کلمات زیر مجاز نمی باشند.

LOOP	NEAR
LABEL	ADD
2AB	(5AX
FAR	A2.B

۳-۳- متغیرها (Variables)

نام متغیر مشخص کننده محلی از حافظه می باشد که بوسیله برنامه قابل دسترسی می باشد و محتوی آنرا در حین اجرای برنامه می توان تغییر داد. تعریف متغیر شامل آدرس، نوع داده و اندازه آن می باشد. از متغیرها می توان بعنوان عملوند در دستورالعملها استفاده نمود. برای متغیرها از نوع بایت از DB، متغیرهای از نوع word از DW و متغیرهای از نوع double word از DD استفاده می گردد.

۳-۴- برچسبها (Labels)

از برچسبها بعنوان آدرس دستورالعمل در برنامه های کاربردی استفاده می شود. از برچسبها به دو صورت استفاده می گردد. اگر برچسب در همان سگمنت کد باشد نوع آن NEAR در غیر اینصورت از نوع FAR می باشد. در صورتیکه نوع آن NEAR باشد می توان بعد از نام برچسب از : استفاده نمود و دیگر نیازی به کلمه NEAR نمی باشد.

LOOP1:

مثال ۳-۱

یا

LOOP1 LABEL NEAR

در صورتیکه برچسب از نوع FAR باشد استفاده از کلمه FAR الزامی

می باشد.

MYCODE LABEL FAR

۳-۵- ثابت‌ها (Constants)

ثابت‌ها مقادیری هستند که در دستورالعمل‌های برنامه‌ها مورد استفاده قرار می‌گیرند. ثابت‌ها از انواع ذیل می‌باشند.

۱- **Binary**: شامل یک سری 0 و 1 می‌باشد که در انتهای آنها حرف B قرار داده می‌شود.

مثال ۳-۲

110111 B
1000 B

۲- **Decimal**: شامل ارقام 0 تا 9 می‌باشد و بطور اختیاری می‌توان حرف D را به آخر آن اضافه نمود.

40
یا
40D

۳- **Hexadecimal**: شامل ارقام 0 تا 9 و حروف A تا F می‌باشد که در انتهای آنها حرف H اضافه می‌گردد.

مثال ۳-۴

32H
0FFH

اگر مقداری در سیستم مبنای شانزده با یکی از حروف A تا F شروع گردد بایستی 0 به ابتدای آن اضافه گردد. در این صورت کامپیوتر آنرا با نام یک برچسب یا متغیر اشتباه نمی‌گیرد.

۴- Octal: شامل ارقام 0 تا 7 می باشد که در انتهای آنها حرف O قرار می گیرد. می توان به جای O از حرف Q نیز استفاده نمود.

مثال ۳-۵

6O
24O
12Q

۵- Character: ثابت های کاراکتری شامل هر کاراکتر از کدهای ASCII می باشد که بین علامت نقل قول ' یا " قرار می گیرند.

مثال ۳-۵

'B'
"JOHN"
'BOB'

۶- Floating point: این نوع data نمایش مقادیر اعشاری بصورت نمائی می باشد.

مثال ۳-۶

SINE DD 0.332E-1

که بوسیله اکثر کامپیوترها حمایت نمی گردد.

۳-۶- فیلد عملیات

در فیلد عملیات نام دستورالعمل واقعی ریزپردازنده یا عملی که بایستی انجام شود ذکر می‌گردد. نام دستورالعمل بین 2 تا 6 کاراکتر می‌باشد.

مثال ۳-۸

MOV	REP
CMP	LOOP
REPNE	
LEA	

۳-۷- فیلد عملوند

این فیلد شامل آدرس dataهایی که بایستی بوسیله فیلد عملیات پردازش گردد می‌باشد. فیلد عملوند با حداقل یک فاصله از فیلد عملیات جدا میشود. بعضی از دستورالعملها فاقد عملوند می‌باشند. سایر دستورالعملها یک یا دو عملوند دارند که با کاما از هم جدا می‌شوند. مانند

CBW	عملوند ندارد
NOP	عملوند ندارد
CLC	عملوند ندارد
NOT AL	یک عملوند دارد
MOV AX, Y	دو عملوند دارد

در مواردی که فیلد عملوند دارای دو عملوند می باشد عملوند اول را عملوند مقصد و عملوند دوم را عملوند مبداء می نامند.

مثال ۳-۱۰

AND AX, X
که AX را عملوند مقصد و X را عملوند مبداء می نامند.

۳-۸- فیلد ملاحظات (Comment)

این فیلد آخرین فیلد دستورالعمل می باشد که شامل توضیحات در مورد دستورالعمل یا برنامه می باشد. این فیلد از سایر فیلدها توسط ; جدا می گردد.

مثال ۳-۱۱

MOV AH, 45H; Parameter for reading a character
دستورالعملها می توانند فقط شامل فیلد Comment باشند. در اینصورت دستورالعمل با ; شروع می شود.

مثال ۳-۱۲

; This is an assembly Program
; For calculating the n factorial.

۳-۹- تکنیکهای آدرس دهی

ریزپردازنده 80286 از هفت روش آدرس دهی استفاده می نماید که عبارتند از

۱- آدرس دهی بدون واسطه

۲- آدرس دهی ثبات

۳- آدرس دهی مستقیم

۴- آدرس دهی غیرمستقیم ثبات

۵- آدرس دهی مینا

۶- آدرس دهی اندیس مستقیم

۷- آدرس دهی اندیس مینا

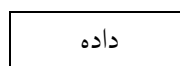
از این امکانات متنوع آدرس دهی برای عملوندها استفاده می شود.

۱-۹-۳- آدرس دهی بدون واسطه

در این مد آدرس دهی داده می تواند 8 بیت یا 16 بیت طول داشته باشد و بعنوان عملوند در دستورالعمل استفاده می گردد.

مثال ۱۳-۳

MOV BL, 10

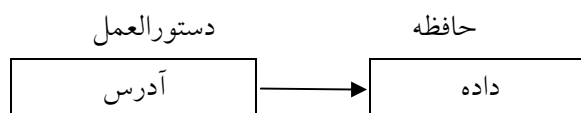


۲-۹-۳- آدرس دهی مستقیم

در این روش آدرس داده که شانزده بیت می باشد جزء دستورالعمل می باشد.

مثال

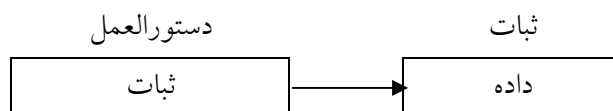
MOV AX, TABLE



۳-۹-۳- آدرس دهی ثبات

در این مد آدرس دهی داده در ثباتی قرار دارد که بوسیله دستورالعمل

مشخص می شود.



برای عملوند شانزده بیتی از ثباتهای AX, BX, CX, DX, BP, SI, DI استفاده می گردد.

مثال ۳-۱۵

MOV AX, CX

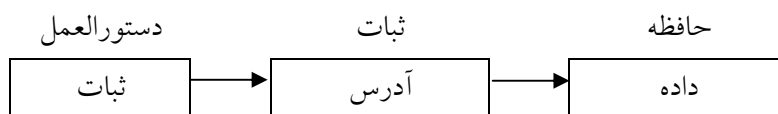
و در مورد عملوند هشت بیتی از ثباتهای AH, AL, BH, BL, CH, CL, DH, DL استفاده می گردد.

مثال ۳-۱۶

MOV DL, AL

۴-۹-۳- آدرس دهی غیرمستقیم ثبات

در این روش آدرس داده در یکی از ثباتهای BX, DI, SI قرار داده می شود.



MOV AX, [BX]

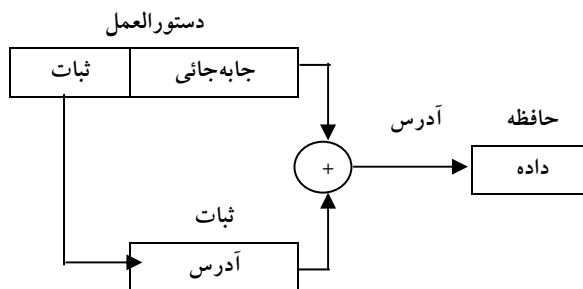
۵-۹-۳- آدرس دهی مبنا

در این روش آدرس داده در یکی از ثباتهای BX, BP, SI, DI قرار داده می شود. در این روش یک جابه جایی باندازه ۸ بیت یا ۱۶ بیت دارد.

MOV AX,[BX]+4

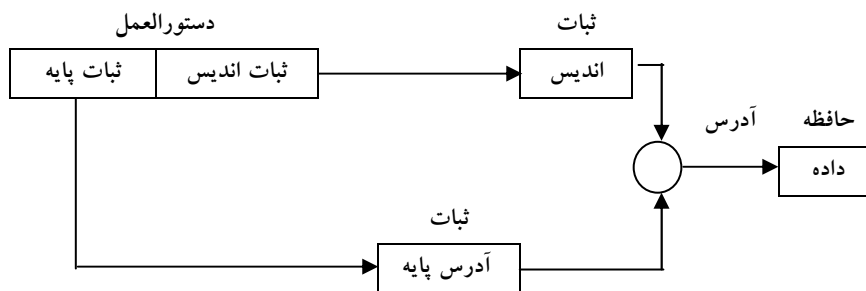
که 4، مقدار جابه‌جائی و آدرس داده در BX قرار داده شده است. دو دستور ذیل معادل دستور فوق می‌باشد.

MOV AX, 4[BX]
MOV AX, [BX+4]



۶-۹-۳- آدرس‌دهی اندیس مستقیم

دراین روش آدرس داده در یکی از ثباتهای BX و یا BP قرار داده می‌شود. و از ثباتهای SI و یا DI بعنوان اندیس استفاده می‌گردد. در حقیقت آدرس عبارتست از مجموع BX و یا BP با SI یا DI.



مثال ۳-۱۷

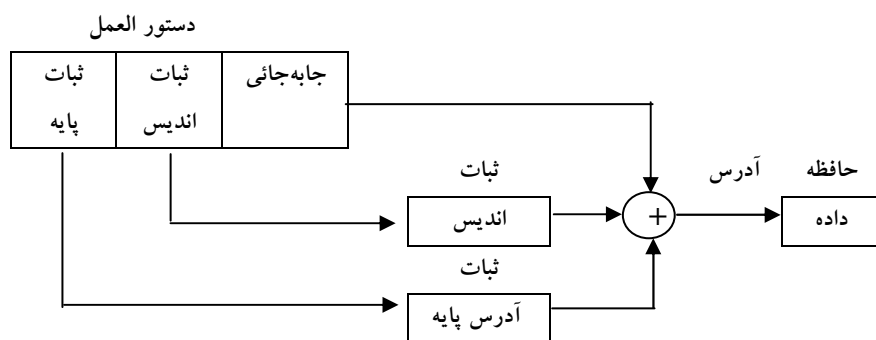
MOV AX,[BX][DI]

۷-۹-۳- آدرس دهی اندیس مبنا

در این روش آدرس داده شبیه قبل بوده با این تفاوت که یک جا به جایی هشت بیتی یا شانزده بیتی نیز وجود دارد.

مثال ۳-۱۸

MOV AX, VALUE [BX][DI]
MOV AX, [BX+2][DI]
MOV AX,[BX] [DI+2]



مروری بر مطالب فصل

هر دستورالعمل می‌تواند شامل چهار فیلد اسم، عملیات، عملوند و ملاحظات باشد. نام مشخصه‌ها در زبان اسمبلی شامل 31 کاراکتر می‌باشد و می‌بایستی با یکی از حروف شروع شد و نمی‌تواند یکی از کلمات ذخیره شده بوسیله زبان اسمبلی باشد. ثابتها را می‌توان در مبنای 2، 8، 10، 16 در برنامه استفاده نمود که بترتیب برای آنها از پسوندهای B، O، D، H استفاده می‌نمائیم. دستورالعملها می‌توانند بدون عملوند، یک عملوند، یا دو عملوند می‌باشند. فیلد ملاحظات با ; شروع می‌شود. ریزپردازنده 80286 از هفت تکنیک آدرس‌دهی استفاده می‌نماید که بترتیب عبارتند از :

۱- آدرس دهی بدون واسطه

۲- آدرس دهی ثابت

۳- آدرس دهی مستقیم

۴- آدرس دهی غیرمستقیم ثابت

۵- آدرس دهی مبنا

۶- آدرس دهی اندیس مستقیم

۷- آدرس دهی اندیس مبنا

یک دستورالعمل می‌تواند با ; شروع شده که جنبه ملاحظات و توضیحات برای برنامه دارد.

تمرین

۱- کدامیک از کلمات زیر نام مجاز در اسمبلی می‌باشد؟

BOOK	VARIABLE	\$ 5AC
2 NAME	FOR	. 2_A7
+ PLUS	NAME3	BOOK2

۲- فیلدهای یک دستورالعمل را نام ببرید.

۳- انواع ثابت‌ها را بنویسید و در مورد هر یک مثالی بدهید.

۴- تعداد حداقل و حداکثر عملوندها در یک دستورالعمل را بدهید.

۵- مدهای آدرس‌دهی را نام ببرید. و در مورد هر کدام مثالی بدهید.

۶- قانون نامگذاری مشخصه‌ها را بیان نمایید.

۷- در مورد ثابت‌ها در مبنای 16 از چه پسوندی استفاده می‌گردد.

۸- آیا استفاده از پسوند در مورد ثابتهای مبنای 10 اختیاری است؟

فصل چهارم

دستورالعملهای اساسی

هدف کلی

معرفی دستورالعملهای اساسی در زبان اسمبلی.

اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

۱- انتقال داده‌ها در حافظه

۲- دستورالعملهای جمع و تفریق

۳- دستورالعملهای جمع و تفریق مقادیر بزرگ

۴- ضرب و تقسیم

۵- دستورالعملهای ضرب مقادیر بزرگ

۶- دستورالعملهای کاهش و افزایش

۷- دستورالعمل محاسبه مکمل 2 یک مقدار.

۱-۶- انتقال داده‌ها در حافظه

انتقال داده‌ها بین مکانهای مختلف حافظه اصلی و ثباتها بوسیله دستورالعمل MOV انجام می‌شود. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

MOV dst , src

این دستورالعمل محتوی SRC را در dst قرار داده و محتوی SRC بدون تغییر باقی می‌ماند.

MOV CL, -30

-30 را در ثبات CL قرار می‌دهد.

MOV X, 25H

مقدار 37 را در مکان X در حافظه قرار می‌دهد.

MOV AX, BX

محتوی ثبات BX را در AX قرار می‌دهد و محتوی ثبات BX بدون تغییر باقی می‌ماند.

MOV DS, AX

محتوی ثبات AX را در DS قرار می‌دهد.

MOV AX, TABLE

محتوی حافظه TABLE را در ثبات AX قرار می‌دهد.

MOV TABLE, AX

محتوی ثبات AX را در مکان TABLE از حافظه اصلی قرار می‌دهد.

درمورد دستورالعمل MOV بایستی در نظر داشت که :

۱- هر دو عملوند یعنی sdt و src بایستی از نوع بایت یا هر دو از نوع word باشند.

۲- هر دو عملوند نمی‌توانند متغیر باشند. یعنی دستورالعمل زیر غلط می‌باشد.

```
MOV X, Y
```

۳- هیچکدام از عملوندها نمی‌توانند ثبات IP باشند.

۴- هیچکدام از عملوندها نمی‌توانند ثبات فلگ باشند.

۵- محتوی یک ثبات سگمنت را نمی‌توان مستقیماً بیک ثبات سگمنت دیگر منتقل نمود. این کار را بایستی بصورت غیر مستقیم انجام داد.

```
MOV AX, ES
```

```
MOV DS, AX
```

۶- عملوند dst نمی‌تواند ثبات CS باشد.

۷- در دستورالعمل MOV بجزء در مواردیکه src ثابت باشد حتماً یکی از عملوندها بایستی ثبات باشد.

۸- یک ثابت را نمی‌توان مستقیماً به یک ثبات سگمنت منتقل نمود.

این کار را بایستی بصورت زیر انجام داد.

```
MOV AX, DATA_SEG
```

```
MOV DS, AX
```

۹- دستورالعمل MOV روی هیچ فلگی اثر ندارد.

همانطور که میدانید هر مکان از حافظه یا متغیر دارای مشخصات زیر می‌باشد.

۱- نام، که از قانون نامگذاری تبعیت می‌نماید.

۲- آدرس، که نشان دهنده مکان متغیر در حافظه می‌باشد.

۳- مقدار، که محتوی آن مکان را نشان می‌دهد.

آدرس حافظه		
2401		
2402		
2403		
2404	65	X
2405		
2406		
2407		
2408		
2409	300	Y
2410		
2411		

در بالا حافظه اصلی را نشان می دهد که X متغیری از نوع بایت با محتوی 65 و آدرس آن 2404 می باشد. همچنین Y متغیری است از نوع word با محتوی 300 که آدرس آن 2408 می باشد. بایستی توجه داشت که آدرس هر data آدرس بایت شروع آن data می باشد. متغیر y چون بایتهای 2408 و 2409 را در حافظه اشغال می کند آدرس آن 2408 می باشد. حال چنانچه بخواهیم آدرس متغیری را در ثبات BX قرار دهیم از دستورالعمل زیر استفاده می گردد.

MOV BX , OFFSET Y

با اجرای این دستورالعمل آدرس 2408 در ثبات BX قرار می گیرد.

BX	Y
2408	300

با اجرای دستورالعمل

MOV BX, Y

محتوی Y یعنی 300 در ثبات BX قرار داده می شود.

BX	Y
300	300

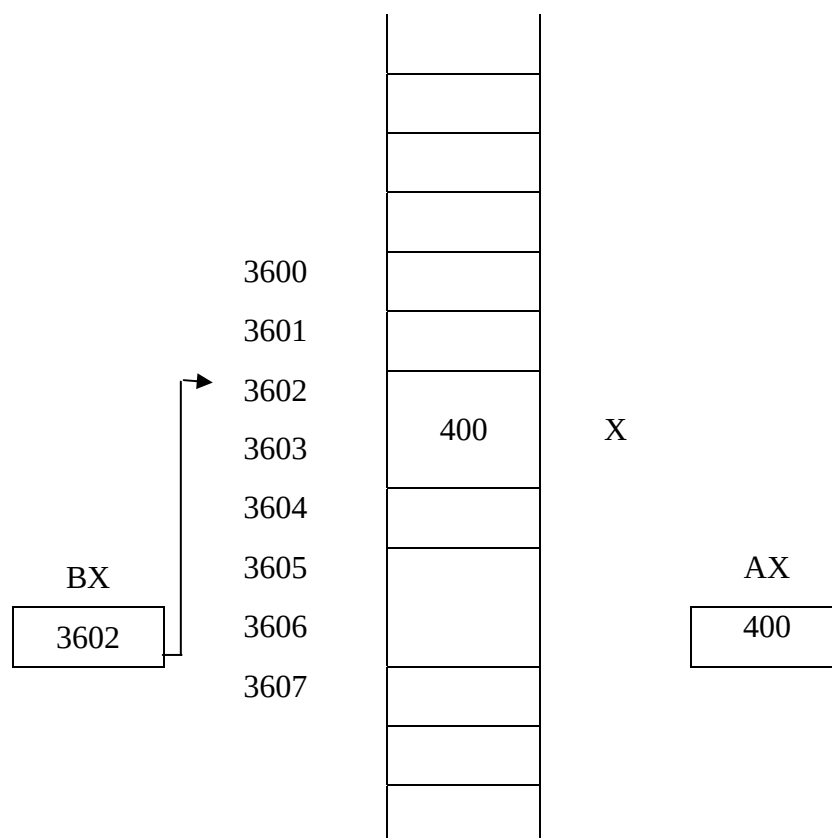
معمولاً آدرس متغیرها را در یکی از ثباتهای SI, DI, BP, BX قرار داده می شود. حال سه دستورالعمل زیر را در نظر بگیرید.

```

X      DW 400
MOV    BX, OFFSET X
MOV    AX, [BX]

```

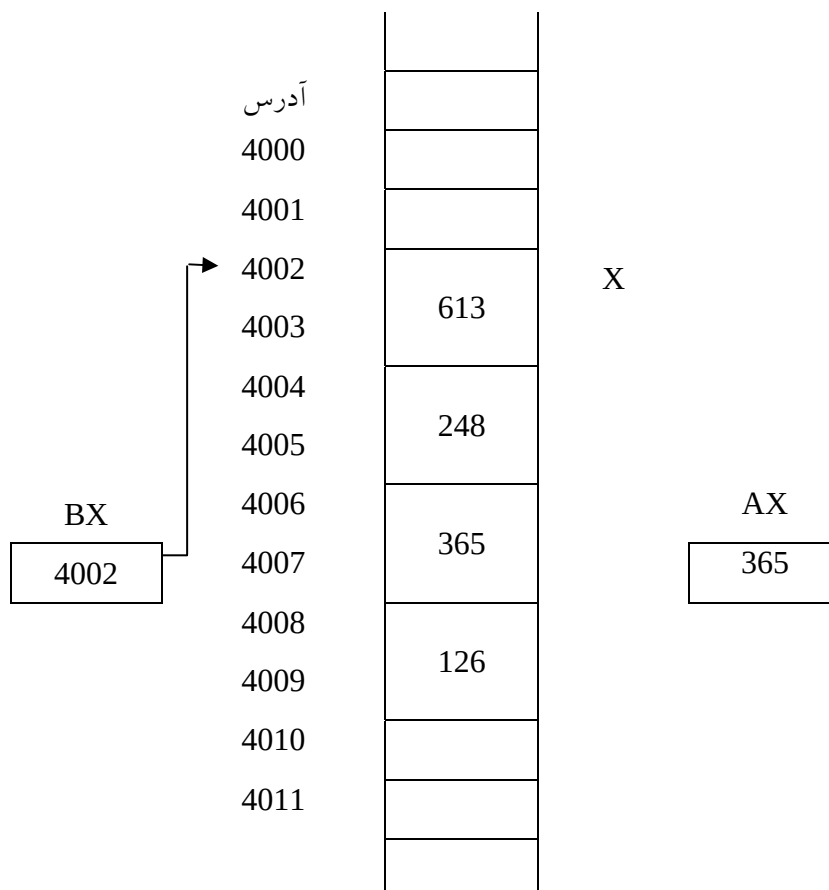
همانطور که در شکل زیر نشان داده شده است. X مکانی از حافظه است که یک word را اشغال نموده است. دستورالعمل دوم آدرس متغیر X را در ثبات BX قرار می دهد. دستورالعمل آخر محتوی مکانی از حافظه که بوسیله BX اشاره می شود را به ثبات AX منتقل می نماید.



مثال ۱-۴

دستورالعملهای ذیل را در نظر بگیرید.

```
X  DW 613,248,365,126
MOV BX, OFFSET X
MOV AX, [BX] +4
```



اولین دستورالعمل ایجاد یک آرایه چهار عنصری از نوع word می نماید با مقادیر 613 ، 248 ، 365 ، 126 بترتیب در آدرسهای 4002، 4004، 4006، 4008. دستورالعمل دوم آدرس متغیر X را در ثبات BX قرار می دهد. آخرین دستورالعمل 4 واحد به آدرس درون BX اضافه نموده به آدرس 4006 می رسد، سپس مقداری که در آدرس 406 حافظه قرار دارد یعنی 365 به ثبات AX منتقل می گردد. بایستی توجه داشت که محتوی ثبات BX پس از اجرای دستورالعمل های فوق 4002 می باشد.

بایستی توجه داشت که سه دستورالعمل ذیل معادلند

```
MOV AX, [BX]+4
MOV AX, 4[BX]
MOV AX, [BX+4]
```

مثال ۲-۴

دستورالعمل های ذیل را در نظر بگیرید.

```
X      DW  126,248,613,1260
MOV  DI, 4
MOV  AX, X[DI]
```

آدرس		
1A00		X
1A01	126	X+1
1A02		X+2
1A03	248	X+3
1A04		X+4
1A05	613	X+5
1A06		X+6
1A07	1260	X+7
1A08		X+8
1A09		

AX
613

DI
4

دستورالعمل اول ایجاد یک آرایه می نماید بنام X از نوع word با مقادیر 126, 248, 613, 1260 به ترتیب در آدرسهای 1A00, 1A02, 1A04, 1A06 از

حافظه. دستورالعمل دوم مقدار 4 را در ثبات DI قرار می‌دهد. آخرین دستورالعمل محتوی $X+4$ یعنی مقدار 613 را در ثبات AX قرار می‌دهد.

مثال ۳-۴

دستورالعمل ذیل را در نظر بگیرید.

```
MOV AX, X[BX][DI]
```

این دستورالعمل محتوی آدرسی که از مجموع مقادیر X ، BX ، DI بدست می‌آید را در ثبات AX قرار می‌دهد. در این دستور می‌توان بجای BX از BP و به جای DI از SI استفاده نمود.

مثال ۴-۴

```
MOV AX, X[BP][DI]
MOV AX, X[BX][SI]
MOV AX, X[BP][SI]
```

نهایتاً دستورالعمل MOV نیز می‌تواند به یکی از شکل ذیل باشد.

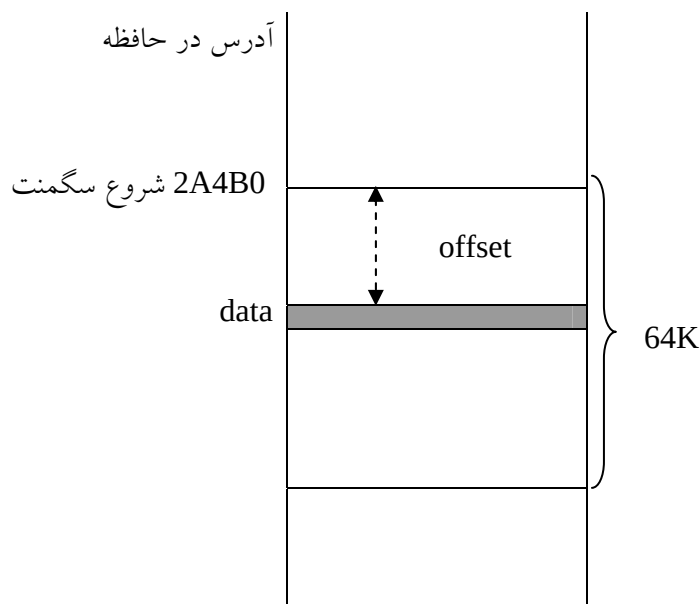
```
MOV AX, [BX][DI+2]
MOV AX, [BX+2][DI]
MOV AX, [BX][DI+2]
MOV AX, [BX+DI+2]
```

از این فرم‌های MOV برای استخراج داده‌ها از یک آرایه دو بعدی استفاده می‌گردد.

همانطور که قبلاً متذکر شدیم در برنامه از چهار سگمنت بنامهای زیر می‌توان استفاده نمود.

```
code    segment    CS
data    segment    DS
stack   segment    SS
extra   segment    ES
```

کامپیوتر آدرس شروع این سگمنت‌ها را طوری انتخاب می‌نماید که قابل تقسیم بر 16 باشند یعنی چهار بیت سمت راست آدرس شروع سگمنت‌ها 0000 باشد. در حقیقت آدرس شروع سگمنت‌ها بیست بیتی می‌باشد که چون چهار بیت سمت راست آنها 0000 می‌باشد به شانزده بیت تقلیل یافته و در ثباتهای CS، DS، SS، ES قرار داده میشوند. برا یافتن آدرس واقعی یا فیزیکی سگمنت data در حافظه بایستی محتوی DS را در 16 ضرب نمود. همانطوریکه نیز قبلاً بیان شد اندازه سگمنت‌ها تا 64K بایت می‌تواند باشد. آدرس هر data در حافظه چون این data segment قرار دارد می‌توان نسبت به ابتدای آن segment مشخص نمود که به این آدرس offset گفته میشود.



در حقیقت آدرس سگمنت 2A4B در ثبات DS قرار داده میشود. حال برای بدست آوردن آدرس فیزیکی یا واقعی data بطریق ذیل عمل می‌نمائیم.

$$DS * 16 + offset$$

۲-۴- دستورالعمل LEA

دستورالعمل LEA نیز آدرس متغیر را در یکی از ثباتهای SI, DI, BX, BP می‌دهد. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

LEA dst , src

1- src یک متغیر از نوع بایت یا word می‌باشد.

2- dst یکی از چهار ثبات BX, BP, SI, DI می‌باشد.

3- دستورالعمل LEA بر هیچ فلگی اثر ندارد.

مثال ۵-۴

X DW 613
LEA BX , X

آدرس حافظه		
4000		
4001		
4002		X
4003	613	
4004		
BX 4002		
4005		

در حقیقت عملکرد این دستور معادل دستور ذیل می‌باشد.

MOV BX,OFFSET X

مثال ۶-۴

LEA SI, COL[BX]

این دستورالعمل آدرس متغیر COL را با محتوی BX جمع نموده آدرس بدست آمده را در SI قرار می دهد.

مثال ۷-۴

X DB ?
LEA BX, X

متغیر X از نوع بایت تعریف گردیده و آدرس آن در ثبات BX قرار داده شده است.

بایستی توجه داشت که دستورالعمل LEA بر هیچ فلگی اثر ندارد.

۳-۴- مبادله داده ها

برای مبادله محتوی دو آدرس داده از دستورالعمل XCHG استفاده می گردد. شکل کلی دستورالعمل بصورت زیر می باشد.

XCHG dst, src

این دستورالعمل محتوی SRC با dst را مبادله می نماید یعنی محتوی SRC در dst قرار می گیرد و محتوی dst در src.

در مورد دستورالعمل XCHG بایستی در نظر داشت که:

۱- dst, src نمی تواند ثابت باشند.

۲- dst, src بایستی هر دو از نوع بایت یا هر دو از نوع word باشند.

۳- dst, src هر دو متغیر نمی توانند باشند.

۴- این دستورالعمل بر روی هیچ فلگی اثر ندارد.

مثال ۸-۴

```
MOV  AX, 1000
MOV  X, 3000
XCHG X, AX
```

قبل از اجرای دستورالعمل XCHG

AX	X
1000	3000

بعد از اجرای دستورالعمل XCHG

AX	X
3000	1000

مثال ۹-۴

```
XCHG  AX , BX
```

محتوی BX را در AX و محتوی AX را در X قرار می دهد.

```
X DB 65
Y DB 48
MOV  AL , X
XCHG AL, Y
MOV  X , AL
```

دستورالعملهای فوق باعث مبادله مقادیر X و Y که هر دو از نوع بایت

می باشند می شود. قبل از اجرای دستورالعمل های فوق

X	Y
65	48

پس از اجرای دستورالعملهای فوق

AL	X	Y
48	48	65

بایستی توجه داشت که دستورالعمل XCHG روی هیچ فلگی اثر ندارد.

۴-۴- جمع و تفریق

جمع دو مقدار بوسیله دستورالعمل ADD انجام می‌شود. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

ADD dst , src

این دستورالعمل محتوی SRC را با محتوی dst جمع نموده نتیجه را در dst قرار می‌دهد و مقدار SRC بدون تغییر می‌ماند.

dst ← dst + src

مثال ۱۰-۴

```
MOV AX , 613
MOV BX , 248
ADD AX , BX
```

دستورالعمل اول مقدار 613 را در AX قرار می‌دهد. دستورالعمل دوم مقدار 248 را در BX قرار می‌دهد. دستورالعمل سوم مجموع AX و BX را محاسبه و نتیجه را در AX قرار می‌دهد. که همانطور که ملاحظه می‌شود مقدار BX تغییر نمی‌کند.

قبل از اجرای دستورالعمل فوق

AX	BX
613	248

بعد از اجرای دستورالعملهای فوق مقادیر عبارتند از:

AX	BX
861	248

در مورد دستورالعمل ADD بایستی موارد زیر را در نظر داشت.

- ۱- هر دو عملوند بایستی از نوع بایت یا هر دو از نوع word باشند.
- ۲- هر دو عملوند نمی‌توانند از نوع متغیر باشند.

۳- به جزء در مواردیکه یکی از عملوندها ثابت باشد حتماً یکی از عملوندها بایستی از نوع ثبات باشد.

۴- دستورالعمل ADD بر روی فلگ‌های AF, CF, OF, PF, SF, ZF اثر دارد.

مثال ۱۱-۴

```
X      DB 13
MOV    AL, 10
ADD    X, AL
```

دستورالعمل اول X را از نوع بایت تعریف نموده و مقدار آنرا 13 قرار می‌دهد. دستورالعمل دوم مقدار 10 را در ثبات AL قرار می‌دهد. دستورالعمل سوم مقدار AL را با محتوی X جمع نموده نتیجه که می‌شود 23 را در X قرار می‌دهد. و محتوی AL نیز 10 می‌باشد.

مثال ۱۲-۴

```
X      DW 613,248,126
MOV    BX, OFFSET X
MOV    AX, 1000
ADD    AX, [BX + 2]
```

اولین دستورالعمل یک آرایه سه عنصری از نوع word ایجاد می‌نماید. مقادیر عناصر آرایه بترتیب عبارتند از 613، 248، 128 دستورالعمل دوم آدرس X را در BX قرار می‌دهد. دستورالعمل سوم مقدار 1000 را در AX قرار می‌دهد. آخرین دستورالعمل محتوی دومین عنصر آرایه را یعنی 248 با محتوی AX جمع نموده و نتیجه را در AX قرار می‌دهد.

2000			
2001			
2002		X	
2003	613		
2004			BX
2005	248		2002
2006			
2007	126		
2008			AX
2009			1248

مثال ۱۳-۴

```

X      DB      18
MOV    AL , -18
ADD    AL , X

```

دستورالعمل اول مقدار 18 را در X قرار می‌دهد. دستورالعمل دوم مقدار -18 را در AL قرار می‌دهد. آخرین دستورالعمل محتوی X را با محتوی AL جمع نموده، نتیجه را که 0 می‌شود در AL قرار می‌دهد. بایستی توجه داشت که پس از انجام عمل جمع مقدار فلگ ZF برابر با یک می‌شود.

مثال ۱۴-۴

```

X      DB 12,20,5,14,26,30
MOV    DI, 5
MOV    AL, 20
ADD    X[DI], AL

```

دستورالعمل اول یک آرایه 6 عنصری از نوع بایت با مقادیر
 به ترتیب 12، 20، 5، 14، 26، 30 تعریف می نماید. دستورالعمل دوم مقدار 5
 را در DI قرار می دهد. دستورالعمل سوم مقدار 20 را در AL قرار می دهد.
 آخرین دستورالعمل محتوی مکانی از حافظه که بوسیله $X+5$ مشخص می شود
 را با AL جمع می نماید. یعنی مقدار 30 را با 20 جمع نموده و نتیجه را در محل
 $X+5$ از حافظه قرار می دهد. و مقادیر ثباتهای AL و DI بدون تغییر
 باقی می ماند.

آدرس حافظه		
A100		
A101		
A102	12	X
A103	20	X+1
A104	5	X+2
A105	14	X+3
A106	26	X+4
A107	30	X+5
A108		
A109		
AL		DI
20		4

پس از اجرای دستورالعملهای فوق شکل حافظه بصورت زیر در می آید.

A100		
A101		
A102	12	X
A103	20	X+1
A104	5	X+2
A105	14	X+3
A106	26	X+4
A107	50	X+5
A108		
A109		

از دستورالعمل ADC نیز برای جمع مقادیر می توان استفاده نمود. فرم کلی این دستورالعمل بصورت زیر است.

ADC dst , src

که محتوی src را با محتوی dst جمع نموده نتیجه را با مقدار فلگ CF جمع نموده و نهایتاً نتیجه بدست آمده را در dst قرار می دهد.

$$dst \leftarrow dst + src + CF$$

در موقع استفاده از این دستورالعمل بایستی توجه داشت که :

۱- هر دو عملوند dst , src بایستی از نوع بایت یا هر دو از نوع word باشند.

۲- dst , src هر دو متغیر نمی توانند باشند.

۳- به جزء در مواردیکه یکی از عملوندهای src , dst ثابت باشد یکی از عملوندها بایستی حتماً از نوع ثابت باشد.

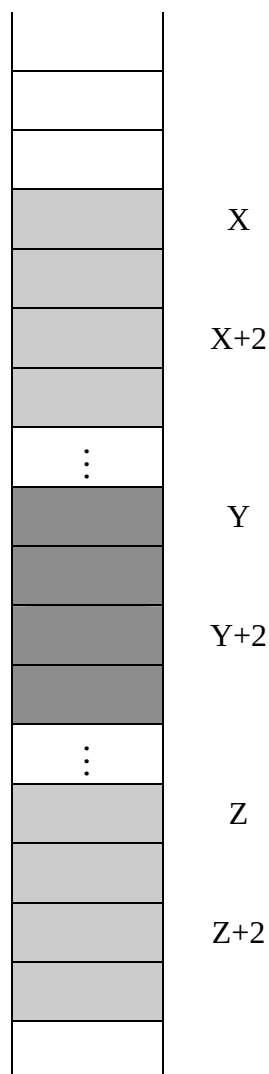
۴- دستورالعمل ADC روی فلگ‌های ZF , SF , AF , PF , OF , CF اثر دارد.

مثال ۱۵-۴

```
X DW ?
MOV AX, 1000
MOV X, 3000
ADC AX, X
```

دستورالعمل اول X را از نوع $word$ تعریف می‌نماید. دستورالعمل دوم مقدار 1000 را در ثبات AX قرار می‌دهد. دستورالعمل سوم مقدار 3000 را در متغیر X قرار می‌دهد. آخرین دستورالعمل محتوی X را با محتوی ثبات AX جمع نموده و چنانچه $CF=1$ باشد یک واحد به جمع بدست آمده اضافه نموده 4001 را در ثبات AX قرار می‌دهد. و چنانچه مقدار $CF=0$ باشد مقدار 4000 را در ثبات AX قرار می‌دهد.

یکی از کاربردهای مهم دستورالعمل ADC در محاسبه مجموع دو مقدار از نوع $double word$ می‌باشد. فرض کنید می‌خواهیم مجموع دو متغیر Y و X از نوع $double word$ را محاسبه نموده و نتیجه را در متغیر Z از نوع $double word$ قرار دهیم.



همانطور که در شکل ملاحظه می گردد، متغیر X از نوع double word را می توان بصورت دو تا word بنامهای X و $X+2$ در نظر گرفت. بطور مشابه همین عمل را در مورد Y و Z می توان انجام داد. برای جمع دو متغیر از نوع double word کافی است که ابتدا دو تا word با ارزش کمتر را جمع نموده با

استفاده از دستورالعمل ADD سپس دو تا word با ارزش بیشتر را با هم جمع نموده با استفاده از دستورالعمل ADC.

	31	16	15	0
X	word با ارزش بیشتر		Word کم ارزش	

	31	16	15	ADD	0
Y	word با ارزش بیشتر		Word کم ارزش		

	31	16	15	0
Z	word با ارزش بیشتر		Word کم ارزش	

قطعه برنامه‌ای که دو متغیر X و Y از نوع double word را با هم جمع نموده و نتیجه را در متغیر Z از نوع double word قرار می‌دهد بصورت ذیل می‌باشد:

```

X      DD      60000
Y      DD      40000
Z      DD      ?
MOV     AX , X
ADD     AX , Y
MOV     Z , AX
MOV     AX, X+2
ADC     AX , Y+2
MOV     Z +2, AX

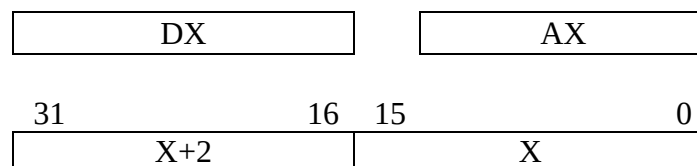
```

سه دستورالعمل اول متغیرهای X , Y , Z را از نوع double word تعریف نموده و مقدار X را برابر با 60000 و مقدار Y را برابر با 40000 قرار می‌دهد. سه دستورالعمل بعدی دو تا word با ارزش کم X و Y را با هم جمع نموده نتیجه را

در word با ارزش کم Z قرار می‌دهد. سه دستورالعمل بعدی دو تا word با ارزش بیشتر X و Y را با هم جمع نموده با استفاده از ADC و نتیجه را در word با ارزش بیشتر Z قرار می‌دهد. بایستی توجه داشت که در دستورالعملهای MOV، ADD، ADC هر دو عملوند نمی‌توانند متغیر باشند.

مثال ۱۶-۴

قطعه برنامه زیر مقدار X از نوع double word را با مقدار ثباتهای DX و AX جمع نموده نتیجه را در X قرار می‌دهد.



ابتدا محتوی X را با AX جمع نموده با استفاده از دستورالعمل ADD سپس محتوی X+2 را با DX با استفاده از دستورالعمل ADC جمع می‌نمائیم.

```

X      DD  40000
MOV    AX , 300
MOV    DX, 400
ADD    X , AX
ADC    X+2, DX
    
```

قطعه برنامه ذیل مجموع متغیرهای X و Y از نوع double word و عدد 24 را محاسبه و نتیجه را در متغیر W قرار می‌دهد. یعنی :

$$W \leftarrow X + Y + 24$$

برای اینکار ابتدا X ، Y ، W را بصورت double word تعریف نموده سپس مجموع مقادیر X و Y را بدست می آوریم. آنگاه عدد 24 را بصورت یک مقدار از نوع double word در نظر گرفته با مجموع قبلی بدست آمده جمع نموده و نهایتاً نتیجه نهائی را در W قرار می دهیم.

X	DD	?
Y	DD	?
Z	DD	?
MOV	AX , X	
MOV	DX,X+2	
ADD	AX,Y	
ADC	DX,Y+2	
ADD	AX,24	
ADC	DX ,0	
MOV	W, AX	
MOV	W+2 , DX	

برای انجام عمل تفریق از دستورالعمل SUB استفاده می گردد. فرم کلی دستورالعمل SUB عبارتست از

SUB	dst , src
dst ←	dst – src

مقدار src از dst کم می شود نتیجه در dst قرار می گیرد. در بکارگیری این دستورالعمل بایستی موارد زیر را در نظر داشت.

۱- هر دو عملوند بایستی از یک نوع باشند هر دو از نوع بایت یا هر دو از نوع word باشند.

۲- هر دو عملوند نبایستی از نوع متغیر باشند.

۳- به جزء در مواردیکه یکی از عملوندها ثابت باشد حتماً بایستی یکی از عملوندها از نوع ثبات باشد.

۴- دستورالعمل SUB بر روی فلگ‌های ZF، CF، OF، PF، AF، SF اثر دارد.

۵- محتوی عملوند dst در عمل SUB تغییر نمی‌کند.

مثال ۱۷-۴

```
MOV    AL, 10
MOV    BL, 6
SUB    AL, BL
```

دستورالعمل اول مقدار 10 را در AL قرار می‌دهد. دستورالعمل دوم مقدار 6 را در BL قرار می‌دهد و دستورالعمل سوم محتوی BL را از محتوی AL کم نموده نتیجه را در AL قرار می‌دهد و مقدار BL تغییر نمی‌کند.

قبل از اجرای دستورالعمل SUB

AL	BL
10	6

بعد از اجرای دستورالعمل SUB

AL	BL
4	6

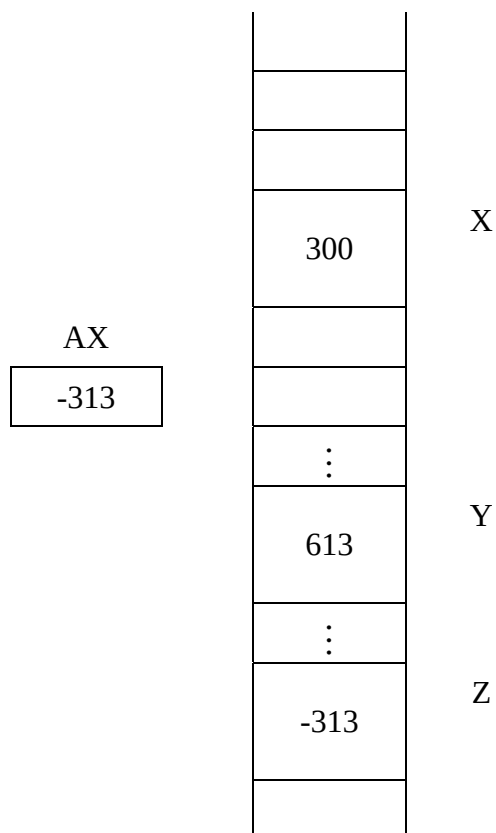
مثال ۱۸-۴

```

X      DW      300
Y      DW      613
Z      DW      ?
MOV    AX, X
SUB    AX, Y
MOV    Z, AX
```

سه دستورالعمل اول سه متغیر X با مقدار 300 و Y با مقدار 613 و Z از نوع word تعریف می‌نماید. دستورالعمل چهارم محتوی متغیر X را به AX منتقل می‌نماید. دستورالعمل بعدی محتوی متغیر Y را از محتوی ثابت AX کم نموده نتیجه را در AX قرار می‌دهد. آخرین دستورالعمل محتوی ثابت AX را به متغیر Z

منتقل می‌نماید. در نتیجه اجرای دستورالعمل SUB مقدار فلگ SF=1 می‌شود، زیرا نتیجه تفریق مقداری منفی است.



مثال ۱۹-۴

```

ARR      DB  26,126,64,13,40,60
MOV      SI, 4
MOV      AL, 20
SUB      AL, ARR [SI]

```

اولین دستورالعمل یک آرایه شش عنصری از نوع بایت با مقادیر 6,126,64,13,40,60 تعریف می‌نماید. دستورالعمل دوم مقدار 4 را در ثبات SI

قرار می‌دهد. دستورالعمل سوم مقدار 20 را در ثبات AL قرار می‌دهد. آخرین دستورالعمل، محتوی $ARR+4$ را از AL کم نموده نتیجه را که می‌شود 20- در AL قرار می‌دهد. و مقدار فلگ SF برابر با یک می‌شود.

	26	X
SI	126	X+1
4	64	X+2
AL	13	X+3
-20	40	X+4
	60	

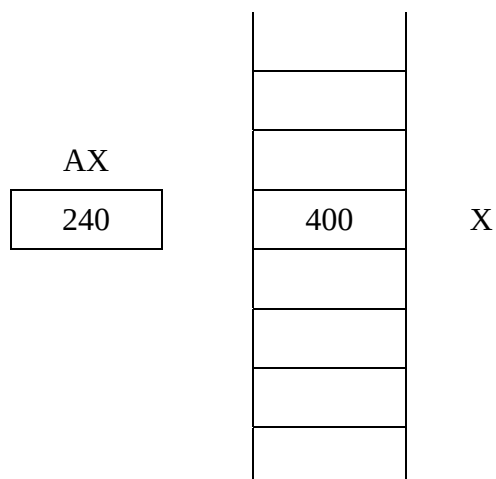
مثال ۲۰-۴

```

X      DW      613
MOV    AX, 248
SUB    AX, 48
SUB    X, AX

```

اولین دستورالعمل متغیر X را از نوع word با مقدار 613 تعریف نموده، دستورالعمل دوم مقدار 248 را در AX قرار می‌دهد. دستورالعمل سوم مقدار 48 را از محتوی ثبات AX کم نموده و نتیجه را که می‌شود 200 در AX قرار می‌دهد. آخرین دستورالعمل محتوی AX از محتوی متغیر X کم نموده نتیجه که می‌شود 400 را در متغیر X قرار می‌دهد.



از دستورالعمل SBB نیز برای عمل تفریق استفاده می‌گردد. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

SBB dst , src
 $\text{dst} \leftarrow \text{dst} - \text{src} - \text{CF}$

محتوی src از محتوی dst کم می‌شود سپس مقدار CF را از نتیجه کم نموده آنگاه نتیجه را در dst قرار می‌دهد.

مثال ۲۱-۴

MOV AX , 1000
 SBB AX , 800

دستورالعمل اول مقدار 1000 را در AX قرار می‌دهد. دستورالعمل دوم 800 را از محتوی AX کم نموده چنانچه مقدار فلگ CF برابر با یک باشد نتیجه می‌شود 199 در غیر اینصورت نتیجه می‌شود 200.

مواردیکه بایستی در استفاده از دستورالعمل SBB رعایت گردند عبارتند از:

۱- هر دو عملوند بایستی از نوع بایت یا هر دو عملوند از نوع word باشند.

۲- هر دو عملوند نمی‌توانند از نوع متغیر باشند.

۳- بجز در مواردیکه یکی از عملوندها ثابت باشد حتماً بایستی یکی از عملوندها از نوع ثبات باشد.

۴- دستورالعمل SBB روی فلگ‌های AF, PF, ZF, SF, OF, CF اثر دارد.

مثال ۲۲-۴

با در نظر گرفتن اینکه مقدار فلگ CF برابر با صفر می‌باشد قطعه برنامه زیر را اجرا نمائید.

MOV AL, 100
SBB AL, 60

دستورالعمل اول مقدار 100 را در ثبات AL قرار می‌دهد و چون دستورالعمل MOV روی هیچ فلگی اثر ندارد، مقدار فلگ CF بدون تغییر مقدار صفر می‌ماند. دستورالعمل دوم در اینجا چون مقدار CF برابر با صفر است دقیقاً مانند دستورالعمل SUB عمل نموده مقدار 60 را از محتوی AL کم نموده و نتیجه را که برابر با 40 می‌باشد در AL قرار می‌دهد.

قبل از اجرای دستور SBB

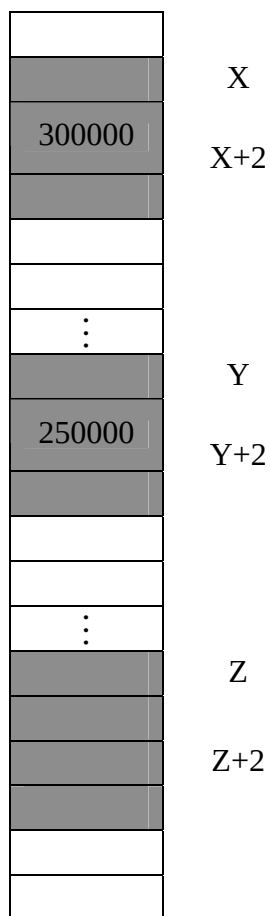
CF	AL
0	100

بعد از اجرای دستور SBB

CF	AL
0	40

از کاربردهای مهم دستورالعمل SBB در تفریق دو مقدار از نوع double word می‌باشد. بعنوان مثال دو متغیر Y و X را از نوع double word در نظر بگیرید قطعه برنامه زیر تفاضل آنها را محاسبه نموده نتیجه را در متغیر Z که از نوع double word می‌باشد قرار می‌دهد.

X	DD	300000
Y	DD	250000
Z	DD	?
MOV	AX, X	
SUB	AX, Y	
MOV	Z, AX	
MOV	AX, X+2	
SBB	AX, Y+2	
MOV	Z+2, AX	

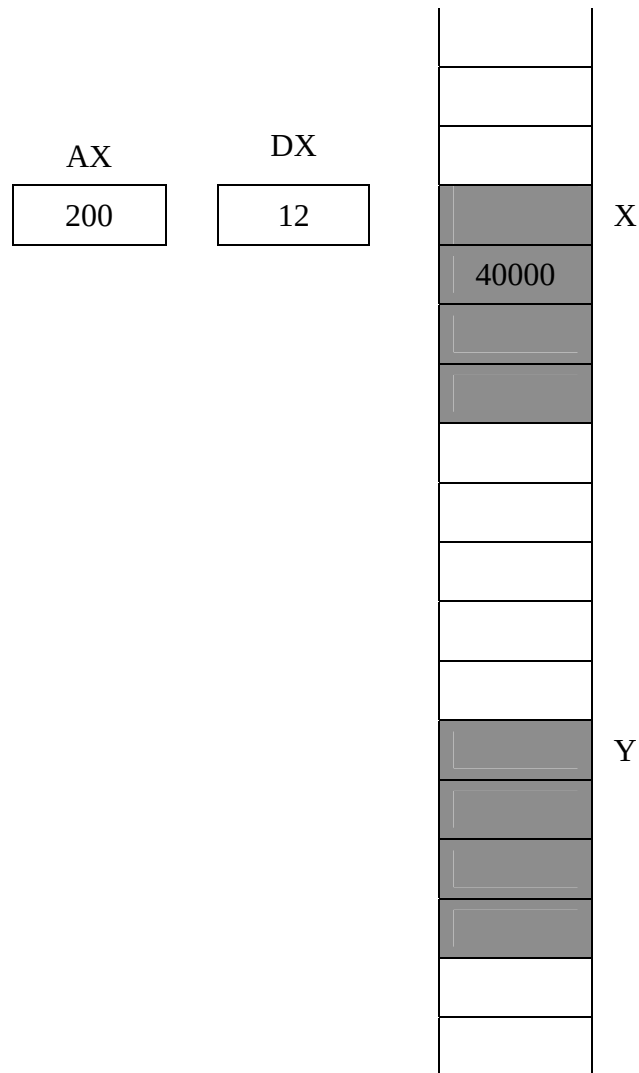


سه دستورالعمل اول متغیرهای X, Y, Z را از نوع double word تعریف می‌نماید سه دستورالعمل بعدی تفاضل دو Word بنامهای X و Y را محاسبه نموده و نتیجه را در دو بایت اول Z قرار می‌دهد. سه دستورالعمل آخر تفاضل دو بایت آخر X و دو بایت آخر Y را با استفاده از دستورالعمل SBB محاسبه نموده و نتیجه را در دو بایت آخر Z قرار می‌دهد.

مثال ۲۳-۴

قطعه برنامه‌ریز تفاضل مقدار متغیر X از نوع double word با محتوی ثباتهای $AX:DX$ محاسبه می‌نماید و نتیجه را در متغیر Y قرار می‌دهد.

X	DD	40000
Y	DD	?
MOV	AX, 200	
MOV	DX, 12	
SUB	DX, X	
SBB	AX, X+2	
MOV	Y, DX	
MOV	Y+2, AX	



۵-۴- ضرب دو مقدار

دستورالعمل MUL و IMUL برای ضرب دو مقدار استفاده می‌گردد.
 دستورالعمل MUL وقتی استفاده می‌گردد که عملوندها بصورت بدون علامت (unsigned) در نظر گرفته شوند. از دستور IMUL وقتی استفاده می‌گردد

که عملوندها بصورت علامت دار (signed) در نظر گرفته شوند. شکل کلی دستور MUL بصورت زیر می باشد.

MUL Opr

- در مورد عملوند opr نکات زیر را بایستی رعایت نمود.
- الف) عملوند opr بایستی از نوع بایت یا word باشد.
- ب) عملوند opr می تواند متغیر یا ثابت باشد.
- ج) عملوند نمی تواند ثابت باشد.
- د) دستورالعمل MUL روی فلگ های CF و OF اثر دارد.
- هـ) در این دستورالعمل مقادیر فلگ های SF, AF, ZF, PF تعریف نشده اند.
- و) چنانچه عملوند opr از نوع بایت باشد. محتوی ثبات AL در محتوی opr ضرب شده نتیجه در AX قرار می گیرد.
- ز) چنانچه عملوند opr از نوع word باشد محتوی ثبات AX در محتوی opr ضرب گردیده نتیجه در DX:AX قرار می گیرد.

مثال ۲۴-۴

```
MOV    BL, 11H
MOV    AL, 0B4H
MUL    BL
```

دستور العمل اول مقدار 11H یعنی 17 را در ثبات BL قرار می دهد.

BL

00010001

دستورالعمل دوم مقدار 0B4H را در ثبات AL قرار می دهد. همانطوریکه

میدانیم B4 در مبنای شانزده معادل 180 در مبنای ده می باشد.

AL

10110100

در اینجا گرچه MSB ثبات AL یک می باشد ولی چون از دستورالعمل MUL استفاده می شود محتوی AL را بصورت منفی در نظر نمی گیریم.

$$17 * 180 = 3060$$

مقدار 3060 در ثبات AX قرار می گیرد.

AX
3060

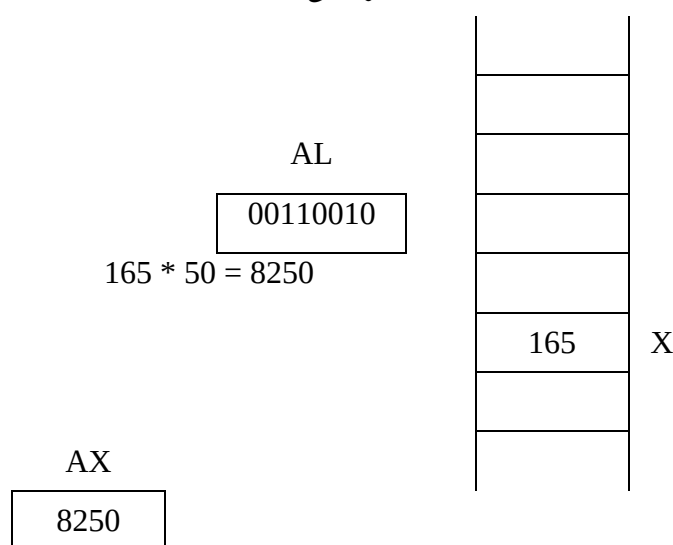
مثال ۲۵-۴

```

X      DB      ?
MOV    X, 0A5H
MOV    AL, 62O
MUL    X

```

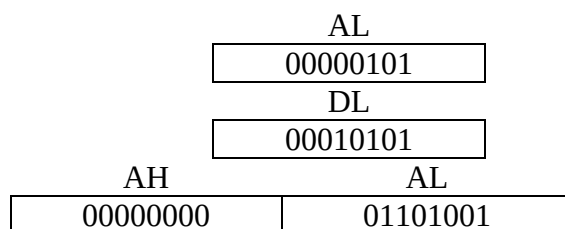
اولین دستورالعمل متغیر X را از نوع بایت تعریف نموده، دومین دستورالعمل 0A5H یعنی مقدار 165 را در متغیر X قرار داده و دستورالعمل سوم 62 در مبنای هشت یعنی مقدار 50 را در ثبات AL قرار می دهد.



مثال ۲۶-۴

```
MOV    AL, 5
MOV    DL, 21
MUL    DL
```

اولین دستورالعمل مقدار 5 را در ثبات AL قرار داده. دومین دستورالعمل مقدار 21 را در ثبات DL قرار داده. سومین دستورالعمل محتوی ثبات AL را در محتوی ثبات DL ضرب نموده نتیجه را در AX قرار می‌دهد.



در اینجا مقدار OF و CF هر دو صفر می‌شود که نشان دهنده اینست که نتیجه حاصلضرب دو بایت در یک بایت جای می‌شود و نتیجه در AL قرار می‌گیرد و مقدار ثبات AH صفر می‌باشد.

دستورالعمل MUL نیز برای محاسبه حاصلضرب دو مقدار از نوع word نیز می‌توان استفاده نمود. برای این کار یکی از عملوندها بایستی حتماً در ثبات AX قرار گیرد. بایستی توجه داشت که نتیجه حاصلضرب در ثباتهای DX:AX قرار می‌گیرد.

مثال ۲۷-۴

```
MOV    AX, 2000
MOV    BX, 15
MUL    BX
```

اولین دستورالعمل مقدار 2000 را در ثبات AX قرار می‌دهد. دومین دستورالعمل مقدار 15 را در ثبات BX قرار می‌دهد. سومین دستورالعمل حاصلضرب محتوی ثباتهای AX و BX را محاسبه نموده نتیجه حاصلضرب را در ثباتهای DX:AX قرار می‌دهد.

$$15 * 2000 = 30000$$

یعنی مقدار 30000 را در ثباتهای DX:AX قرار می‌دهد. در اینجا چون نتیجه در یک Word جا می‌شود مقدار فلگ‌های OF, CF برابر با صفر می‌شود.

مثال ۲۸-۴

```

X      DW      5000
MOV     AX, 3000
LEA     BX, X
MUL     [BX]
```

اولین دستورالعمل متغیر X را از نوع Word با مقدار 5000 تعریف نموده دومین دستورالعمل مقدار 3000 را در ثبات AX قرار می‌دهد. سومین دستورالعمل آدرس X را در ثبات BX قرار می‌دهد. آخرین دستورالعمل مقداری که توسط ثبات BX اشاره می‌شود یعنی 5000 را در محتوی AX یعنی 3000 ضرب نموده نتیجه در ثباتهای DX:AX قرار می‌دهد.

	2000		
BX	2001		
2002	2002	5000	X
	2003		
AX	2004		
3000	2005		

DX	AX
15000000	

دستورالعمل IMUL نیز برای حاصلضرب دو مقدار استفاده می گردد. با این تفاوت که عملوندها را بصورت علامتدار (Signed) در نظر می گیرد. مثال

```
MOV    AL, 11H
MOV    BL, 0B4H
MUL    BL
```

دستورالعمل اول مقدار 11 H یعنی 17 را در ثبات AL قرار می دهد.

AL
00010001

دستورالعمل دوم مقدار B4H یعنی 10110100 را در ثبات BL قرار می دهد.

BL
10110100

چون MSB محتوی BL برابر با یک می باشد محتوی BL را بصورت منفی در نظر می گیریم.

1	0	1	1	0	1	0	0
128	64	32	16	8	4	2	1

$$128 + 32 + 16 + 4 = 180$$

$$180 - 2^8 = 180 - 256 = -76$$

آخرین دستورالعمل مقدار -76 را در 17 ضرب نموده نتیجه یعنی -1292 در ثبات AX قرار می گیرد.

شکل کلی این دستورالعمل بصورت زیر می باشد.

IMUL opr

در مورد عملوند opr نکات زیر را بایستی رعایت نمود.

الف) عملوند بایستی از نوع بایت یا word باشد.

ب) عملوند بایستی از نوع متغیر یا ثبات باشد.

ج) عملوند نبایستی ثابت باشد.

د) دستورالعمل IMUL روی فلگ های CF, OF اثر دارد.

هـ) در مورد این دستورالعمل مقادیر فلگ های SF, AF, ZF و PF تعریف نشده می باشند.

و) چنانچه عملوند opr از نوع بایت باشد محتوی ثبات AL در محتوی opr ضرب شده نتیجه حاصل ضرب در AX قرار داده می شود.

ز) چنانچه عملوند opr از نوع word باشد محتوی ثبات AX در محتوی opr ضرب شده نتیجه در DX:AX قرار داده می شود.

مثال ۲۹-۴

```

X      DW      ?
MOV     X, -300
MOV     AX, 20
IMUL    X
    
```

اولین دستورالعمل X را از نوع word تعریف نموده، دومین دستورالعمل مقدار 300- را در متغیر X قرار داده، سومین دستورالعمل مقدار 20 را در ثبات AX قرار داده حاصلضرب یعنی 6000- را در ثباتهای DX:AX قرار می‌دهد. در اینجا مقدار فلگ‌های CF و OF برابر با صفر می‌شود که نتیجه می‌شود مقدار DX برابر با صفر می‌باشد و مقدار AX برابر با 6000- می‌باشد.

```

X      DB      10110110B
MOV     AL, 10010001B
IMUL    X
    
```

اولین دستورالعمل مقدار X را از نوع بایت بصورت زیر تعریف می‌نماید.

متغیر X

10110110

دومین دستورالعمل مقدار AL را بصورت زیر تعریف می‌نماید.

AL

10010001

آخرین دستورالعمل محتوی ثبات AL را در مقدار متغیر X ضرب نموده نتیجه را در AX قرار می‌دهد. چون از دستورالعمل IMUL استفاده گردیده و MSB متغیر X و ثبات AL برابر با یک می‌باشد مقادیر متغیر X و ثبات AL بصورت منفی در نظر گرفته می‌شود.

1	0	1	1	0	1	1	0
128	64	32	16	8	4	2	1

$$128 + 32 + 16 + 4 + 2 = 192$$

$$192 - 256 = -64$$

حال

1	0	0	1	0	0	0	1
128	64	32	16	8	4	2	1

$$128 + 16 + 1 = 145$$

$$145 - 256 = -91$$

۶-۴- ضرب دو مقدار 32 بیتی بدون علامت

از دستور MUL وقتی استفاده می‌شود که عملوندها هشت یا شانزده بیتی باشند. اما برای ضرب دو مقدار بدون علامت 32 بیتی بایستی از الگوریتم زیر استفاده نمود. همانطوریکه وقتی دو مقدار را روی کاغذ می‌خواهیم ضرب نمائیم برای جمع حاصلضرب‌های جزئی، آنها را یک ستون بطرف چپ شیفت می‌دهیم از این روش بایستی استفاده نمود برای ضرب مقادیر بزرگ. بعنوان مثال دو مقدار 124 و 103 را در نظر بگیرید.

مثال ۳۰-۴

	1	2	4	*
	1	0	3	
	3	7	2	
0	0	0		
1	2	4		
1	2	7	7	2

حاصلضرب جزئی ۱

حاصلضرب جزئی ۲

حاصلضرب جزئی ۳

حاصلضرب نهائی

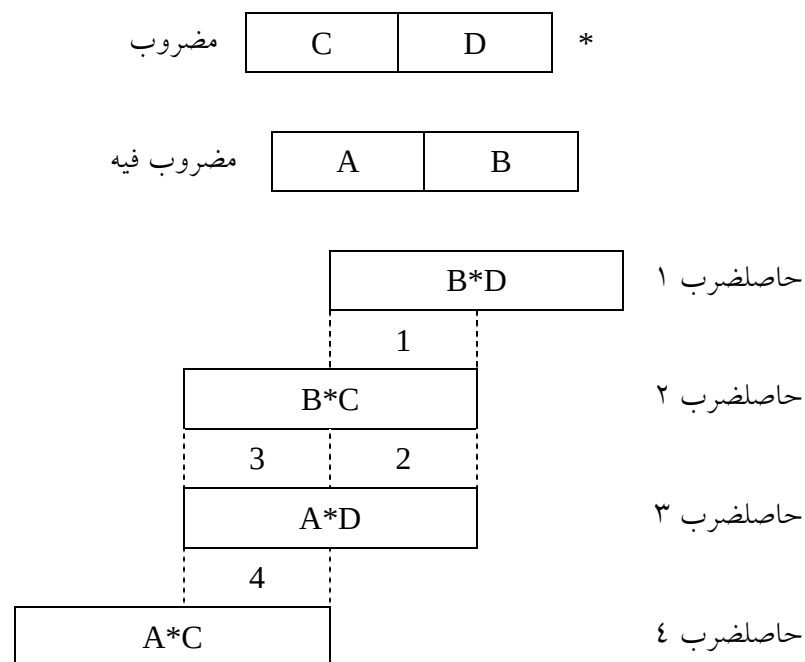
همانطور که دقت می کنید

$$103 * 124 = (3 * 124) + (0 * 124) + (100 * 124)$$

یا

$$103 * 124 = (3 * 1 * 124) + (0 * 10 * 124) + (1 * 100 * 124)$$

با این طریق می توان با استفاده از دستور MUL دو مقدار 32 بیتی را بدون علامت را در هم ضرب و به یک نتیجه 64 بیتی رسید. در شکل زیر B، C و A، هر کدام بصورت 2 بایت در نظر گرفته شده است.



حاصلضرب نهائی (64 بیتی) = مجموع

برنامه این الگوریتم در فصل نهم کتاب داده شده است.

۷-۴- تقسیم دو مقدار

با استفاده از دستورالعمل DIV می توان دو مقدار را بر هم تقسیم نمود. شکل کلی دستورالعمل DIV بصورت زیر می باشد:

DIV Opr

در مورد عملوند opr بایستی نکات زیر را رعایت نمود:

الف) opr بایستی از نوع بایت یا word باشد.

ب) opr نمی تواند ثابت باشد.

ج) opr بایستی از نوع ثبات یا متغیر باشد.

د) چنانچه opr از نوع بایت باشد محتوی محتوی ثبات AX بر opr تقسیم شده، خارج قسمت را در ثبات AL و باقیمانده را در ثبات AH قرار می دهد.

هـ) چنانچه opr از نوع word باشد محتوی ثباتهای DX:AX را بر opr تقسیم نموده، نتیجه تقسیم را در AX و باقی مانده را در DX قرار می دهد.

ز) دستورالعمل DIV هر دو عملوند را بصورت باقی مانده unsigned (بدون علامت) در نظر می گیرد.

مثال ۳۱-۴

```
MOV    AX , 130
MOV    BL , 5
DIV    BL
```

در اولین دستورالعمل محتوی AX می شود 130، دومین دستورالعمل مقدار

5 را در ثبات BL قرار می دهد. آخرین دستورالعمل محتوی AX را بر محتوی BL

تقسیم نموده نتیجه تقسیم را در AL و باقیمانده را در AH قرار می دهد.

AX

0000000010000010

BL

00000101

پس از اجرای دستورالعمل تقسیم داریم که

AL

خارج قسمت

00011010

AH

باقیمانده

00000000

BL

00000101

بایستی توجه داشت که دستورالعمل DIV بر روی هیچ فلگی اثر ندارد و مقدار فلگ‌های AF, OF, PF, SF, ZF, CF تعریف نشده می‌باشند. در ضمن بایستی توجه داشت که مقدار عملوند opr بدون تغییر باقی می‌ماند.

مثال ۳۲-۴

```
X      DB  10110100B
MOV    AX, 0400H
DIV    X
```

اولین دستورالعمل مقدار 10110100B را در متغیر X قرار می‌دهد.

X

10110100

1	0	1	1	0	1	0	0
128	64	32	16	8	4	2	1
$128 + 32 + 16 + 4 = 180$							

دومین دستورالعمل مقدار 0400H را در ثبات AX قرار می‌دهد.

AX

0000010000000000

محتوی ثبات AX برابر با 1024 می‌باشد. آخرین دستورالعمل مقدار 1024 را بر 180 تقسیم نموده مقدار خارج قسمت یعنی 5 را در ثبات AL و باقیمانده یعنی 124 را در ثبات AH قرار می‌دهد. پس از انجام عمل تقسیم داریم که

AL

00000101

AH

01111100

متغیر X

10110100

مثال ۳۳-۴

X	DW 2600
MOV	AX, 00A2H
MOV	DX, 0B1CH
DIV	X

DX

AX

0000101100001100	0000000010100010
------------------	------------------

اولین دستورالعمل متغیر X را از نوع Word با مقدار 2600 تعریف نموده، دومین دستورالعمل مقدار 00A2H را در ثبات AX قرار داده و دستورالعمل سوم مقدار B1C در سیستم 16 تایی را در ثبات AX قرار داده و نهایتاً آخرین دستورالعمل محتوی AX: DX یعنی 0B1C00A2 در سیستم 16 تایی را بر

2600 تقسیم نموده خارج قسمت را در AX و باقیمانده در ثبات DX قرار می دهد و مقدار X بدون تغییر یعنی مقدار 2600 باقی می ماند.
دستورالعمل IDIV مشابه دستورالعمل DIV می باشد با این تفاوت که عملوندها را بصورت علامتدار در نظر می گیرد. شکل کلی این دستورالعمل بصورت زیر می باشد.

IDIV Opr

در مورد استفاده از دستورالعمل IDIV بایستی نکات زیر را در نظر داشت.
الف) عملوند opr بایستی از نوع بایت یا word باشد.
ب) عملوند opr نمی تواند ثابت باشد.
ج) عملوند opr بایستی از نوع ثبات یا متغیر باشد.
د) چنانچه opr از نوع بایت باشد محتوی ثبات AX بر مقدار opr تقسیم نموده خارج قسمت را در ثبات AL و باقیمانده را در ثبات AH قرار می دهد.
هـ) چنانچه opr از نوع word باشد محتوی ثبات DX:AX را بر opr تقسیم نموده و نتیجه تقسیم را در AX و باقیمانده را در ثبات DX قرار می دهد.
ز) دستورالعمل IDIV، هر دو عملوند را بصورت Signed (علامتدار) در نظر می گیرد.

مثال ۳۴-۴

```
MOV    BL, 0B4H
MOV    AX, 0400H
IDIV   BL
```

BL

10110100

AX

0000010000000000

اولین دستورالعمل مقدار B4 در سیستم مبنای 16 را در ثبات BL قرار می‌دهد چون از دستورالعمل IDIV استفاده شده است عملوندها را بصورت علامت دار در نظر می‌گیرد. یعنی اگر MSB عملوند برابر با یک باشد آن را منفی تلقی می‌نماید بنابراین مقدار ثبات BL را بصورت زیر در نظر می‌گیرد.

$$\begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \end{array}$$

$$128 + 32 + 16 + 4 = 180$$

$$180 - 256 = -76$$

دومین دستورالعمل مقدار 400H را در ثبات AX قرار می‌دهد، که مقدار آن برابر با 1024 می‌باشد. آخرین دستورالعمل مقدار 1024 را بر -76 تقسیم نموده، نتیجه تقسیم برابر با 13- می‌باشد که در ثبات AL قرار داده می‌شود و باقیمانده را که برابر با 36 می‌باشد در ثبات AH قرار می‌دهد و مقدار BL بدون تغییر باقی می‌ماند. مقادیر ثباتها پس از اجرای دستورالعملها عبارتند از :

AL
11110011
AH
00100100
BL
10110100

بایستی توجه داشت که دستورالعمل IDIV روی هیچ فلگی اثر ندارد و مقادیر فلگ‌های AF, CF, OF, PF, ZF, SF تعریف نشده می‌باشند.

مثال ۳۵-۴

```
MOV    AX, 2ACH
MOV    DX, 0B2H
MOV    BX, 004AH
IDIV   BX
```

اولین دستورالعمل مقدار 2AC در سیستم مبنای 16 را در ثبات AX و مقدار B2 در سیستم مبنای 16 را در ثبات DX قرار داده محتوی ثباتهای DX:AX یعنی 00B202AC در سیستم 16 مبنای را بر 004A در سیستم مبنای 16 تقسیم نموده نتیجه تقسیم را در AX و خارج قسمت را در DX قرار می‌دهد.

مثال ۳۶-۴

```
      X    DB    0A2H
MOV    AX, 0502H
IDIV   X
```

اولین دستورالعمل مقدار A2 در سیستم 16 تائی یعنی 10100010 در سیستم دودویی را در متغیر X قرار می‌دهد.

متغیر X

10100010

1	0	1	0	0	0	1	0
128	64	32	16	8	4	2	1

$$128 + 32 + 2 = 162$$

$$162 - 256 = -94$$

دستورالعمل دوّم مقدار 0502H را در ثبات AX قرار می‌دهد.

AX

0000010100000010

1	0	1	0	0	0	0	0	0	1	0
1024	512	256	128	64	32	16	8	4	2	1

$$1024 + 256 + 2 = 1282$$

آخرین دستورالعمل مقدار 1282 را بر 94- تقسیم نموده خارج قسمت که برابر با 13- می باشد را در ثبات AL و باقیمانده را که معادل 60 می باشد در ثبات AH قرار می دهد و مقدار X همان مقدار قبلی یعنی A2H را دارد.

متغیر X

10100010

AL

11110011

AH

00111100

۸-۴- دستورالعملهای کاهش و افزایش

با استفاده از دستورالعمل DEC می توان یک واحد از مقدار عملوند کم نمود. شکل کلی دستورالعمل بصورت زیر می باشد.

DEC opr

نکات ذیل را بایستی در موقع استفاده از این دستورالعمل در نظر داشت.

الف) opr بایستی از نوع word یا بایت باشد.

ب) opr نمی تواند ثابت باشد.

ج) این دستورالعمل فقط روی فلگ های SF ، OF ، ZF ، AF ، PF اثر دارد.

مثال ۳۷-۴

```
MOV    AX, 3000
DEC    AX
```


دستورالعمل اول مقدار 3000 را در ثبات AX قرار می دهد. دستورالعمل دوم یک واحد از محتوی AX کم نموده نتیجه را در AX قرار می دهد.

قبل از اجرای DEC

AX
3000

بعد از اجرای DEC

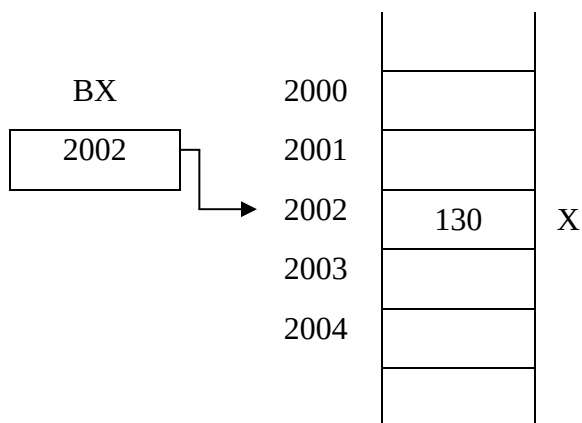
AX
2999

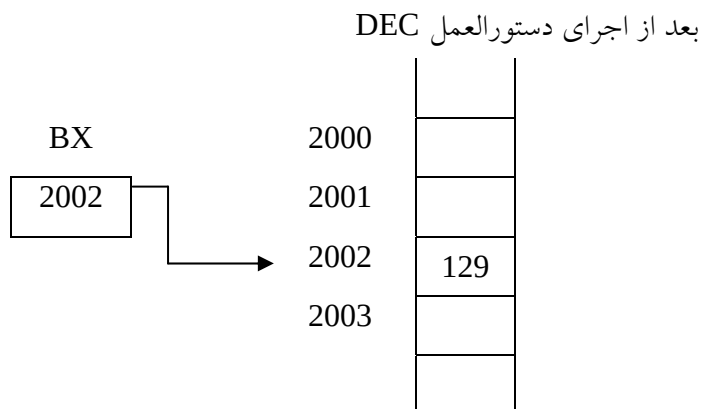
مثال ۳۸-۴

```

X      DB      130
LEA    BX, X
DEC    [BX]
    
```

اولین دستورالعمل مقدار 130 را در متغیر X قرار داده، دومین دستورالعمل آدرس متغیر X را در ثبات BX قرار می دهد. آنگاه محتوی محلی که بوسیله BX اشاره می شود را یکی کاهش می دهد.





دستورالعمل INC باعث می‌شود که یک واحد به عملوند اضافه گردد. شکل

کل این دستورالعمل عبارتست از

INC Opr

در مورد استفاده از این دستورالعمل بایستی نکات ذیل را رعایت نمود.

الف) opr نمی‌تواند ثابت باشد.

ب) opr بایستی از نوع ثبات یا متغیر باشد.

ج) opr بایستی از نوع بایت یا word باشد.

د) این دستورالعمل روی فلگ‌های SF ، OF ، ZF ، AF و PF اثر دارد.

مثال ۳۹-۴

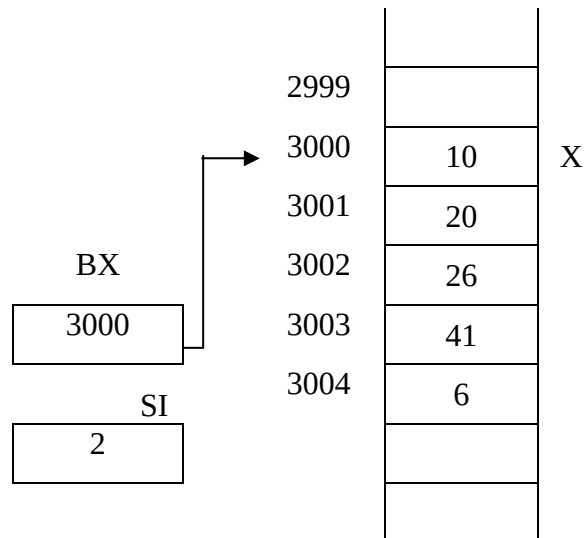
```
MOV    AL, 100
INC     AL
```

مقدار AL را به 101 افزایش می‌دهد.

مثال ۴۰-۴

```
      X      DB  10,20,26,44,6
MOV    SI, 2
MOV    BX, OFFSET X
INC     [BX][SI]
```

اولین دستورالعمل یک آرایه 5 عنصری از نوع بایت بنام X ایجاد می‌نماید. مقادیر عناصر آرایه عبارتند از بترتیب 6، 40، 26، 20، 10. دستورالعمل دوم مقدار 2 را در رجیستر SI قرار می‌دهد. دستورالعمل سوم آدرس متغیر X را در ثبات BX قرار می‌دهد. آخرین دستورالعمل یک واحد به محتوی خانه‌ای از حافظه که بوسیله محتوی $BX+2$ اشاره می‌کند اضافه می‌گرداند.



در حقیقت مقدار خانه حافظه با آدرس 3002 از 26 به 27 افزایش می‌یابد.

۹-۴- دستورالعمل محاسبه مکمل ۲

برای پیدا نمودن مکمل 2 یک مقدار، از دستورالعمل NEG استفاده می‌گردد. شکل کلی آن بصورت زیر می‌باشد.

NEG Opr

الف) مقدار مکمل 2 عملوند opr را محاسبه نموده در opr قرار می‌دهد.

ب) opr می‌تواند از نوع بایت یا word باشد.

ج) opr می تواند ثبات یا متغیر باشد.

د) opr ثابت نمی تواند باشد.

هـ) روی فلگ های SF ، OF ، CF ، ZF ، AF و PF اثر دارد.

مثال ۴۱-۴

```
MOV    AX, -100
NEG     AX
```

مقدار AX را به 100 تغییر می دهد.

```
      X      DB      ?
MOV     X, 26
NEG     X
```

مقدار X که از نوع بایت می باشد نهایتاً برابر با -26 می باشد.

در فصل نهم نحوه نوشتن برنامه و اجزای آن بیان گردیده است.

مروری بر مطالب فصل

در این فصل دستورالعمل ADD برای جمع نمودن دو مقدار از نوع بایت یا word و دستورالعمل SUB برای تفریق کردن دو مقدار از نوع بایت یا word داده شد. شکل کلی آنها بصورت زیر می باشد.

ADD	dst , src
SUB	dst , src

لازم به ذکر است که این دو دستورالعمل روی فلگ‌های محاسباتی اثر می‌گذارند و هر دو عملوند نمی‌تواند ثابت یا متغیر باشند. از دستورالعملهای ADC و SBB بترتیب برای جمع و تفریق دو مقدار از نوع double word استفاده می‌گردد. فرق این دستورالعملها با دو دستورالعمل بالا فقط در استفاده از مقدار CF می‌باشد. برای ضرب دو مقدار بدون علامت از نوع بایت یا word از دستورالعمل MUL و برای ضرب دو مقدار با علامت از نوع بایت یا word از دستورالعمل IMUL استفاده می‌گردد. دستورالعملهای ضرب یک عملوندی می‌باشند و این عملوند ثابت نمی‌تواند باشد. عملوند دیگر از ثبات‌های AL یا AX استفاده می‌گردد. شکل کلی عبارتند از :

MUL	Opr
IMUL	Opr

بهمین ترتیب برای تقسیم دو مقدار از نوع بایت یا word از دستورالعملهای DIV و IDIV بر حسب آنکه عملوندها بدون علامت یا با علامت در نظر گرفته شدند استفاده می‌گردد. دستورالعملهای DEC و INC یک عملوندی بوده و عملوند نمی‌تواند ثابت باشد و باعث بترتیب کاهش یا افزایش یک واحد به عملوند می‌باشد. دستورالعمل NEG نیز یک عملوندی بوده باعث تغییر علامت عملوند می‌گردد.

‡ تمرین

- ۱- دستورالعمل ADD روی کدام فلگ اثر دارد؟
 - ۲- دستورالعمل DEC روی کدام فلگ اثر ندارد؟
 - ۳- در دستورالعمل IDIV مقادیر کدام فلگ تعریف نشده می‌باشد؟
 - ۴- الگوریتمی را ارایه دهید که دو مقدار از نوع double word را در هم ضرب نماید.
 - ۵- قطعه برنامه‌ای برای الگوریتم تمرین 4 ارایه کنید.
 - ۶- دستورالعملهای لازم برای محاسبه مجموع ثباتهای AX، BX، CX و DX ارایه کنید.
 - ۷- به چند طریق می‌توان مقدار ثبات AX را صفر نمود؟ دستورالعملهای لازم را ارایه کنید.
 - ۸- دستورالعملها MUL بر چه فلگ‌هایی اثر ندارد؟
 - ۹- قطعه برنامه‌ای بنویسید که مجموع عناصر آرایه 5 عنصری X از نوع word را محاسبه نماید.
 - ۱۰- قطعه برنامه‌ای بنویسید که مجموع عناصر آرایه 5 عنصری X از نوع double word را محاسبه نماید.
 - ۱۱- دستورالعمل معادل LEA مشخص کنید.
 - ۱۲- دستورالعمل معادل INC BX مشخص کنید.
 - ۱۳- قطعه برنامه زیر را به اسمبلی تبدیل نمائید.
- ```
long int x,y,z,w;
w=x+y-z+30;
```
- ۱۴- قطعه برنامه‌ای بنویسید که حاصلضرب مقادیر ثباتهای AL، BL، CL و DL را محاسبه نماید.
  - ۱۵- قطعه برنامه زیر را به اسمبلی تبدیل نمائید.

VAR

```
X, Y, Z, W : 1 ..20 ;
W := X + Y * Z - W + 100;
```

۱۶- قطعه برنامه‌ای بنویسید که حاصلضرب عناصر آرایه چهار عنصری X از نوع بایت را مشخص نماید؟ در صورتیکه با مشکلی روبرو شدید ذکر نمائید.

۱۷- قطعه برنامه معادل زیر به اسمبلی بدهید.

```
int x, y, z, w;
w = x - y + z - 200;
```

۱۸- چنانچه بخواهیم محتوی AL را بیک word تبدیل نموده و نتیجه را در AX قرار دهیم محتوی AH چیست؟

۱۹- در مورد تمرین 18 چنانچه MSB ثبات AL برابر یک باشد محتوی AH چیست؟

۲۰- دستورالعمل لازم برای انجام محاسبه زیر را مشخص کنید.

```
int x,y,z,w;
w = x/y * z-10;
```

# فصل پنجم

## انشعاب و تکرار

### هدف کلی

آشنائی با دستورالعملهای انشعاب و تکرار در زبان اسمبلی

### اهداف رفتاری

پس از مطالعه این فصل با مفاهیم و مطالب زیر آشنا می شوید.

۱- پرش غیر شرطی

۲- پرشهای شرطی

۳- مقایسه

۴- انواع دستورالعملهای تکرار

### ۱-۵- دستورالعمل پرش غیر شرطی

دستورالعمل پرش غیر شرطی در زبان اسمبلی JMP می باشد. این

دستورالعمل معادل دستورالعمل GOTO در سایر زبانهای برنامه نویسی می باشد.



شکل کلی این دستورالعمل بصورت زیر می باشد. این دستور روی هیچ فلگی اثر ندارد.

JMP            آدرس

مثال ۵-۱

JMP            LAB2

با اجرای این دستورالعمل کنترل به LAB2 منتقل می گردد. بایستی توجه داشت که کنترل بدون هیچ گونه قید و شرطی به LAB2 منتقل می گردد.

مثال ۵-۲

```
MOV AL, 5
ADD AL, BL
JMP LAB1
MUL BL
INC BL
LAB1: SUB CX, 2
 :
```

در قطعه برنامه فوق ابتدا مقدار 5 در ثبات AL قرار می گیرد، سپس مقدار BL به آن اضافه گردید. آنگاه کنترل به LAB1 منتقل می گردد و دستورالعمل SUB به بعد اجرا می گردد. بایستی توجه داشت که دو دستور MUL و INC اجرا نمی شوند.

## ۵-۲- دستورالعملهای پرش شرطی

دستورالعملهای پرش شرطی وقتی اجرا می گردد که در برنامه شرطی برقرار گردد. شکل کلی این دستورالعملها بصورت زیر می باشد.

JX                      آدرس

که X یک رشته یک تا سه کارکتری می باشد.

مثال ۳-۵

JZ                      LAB2

در صورتیکه مقدار ZF برابر با یک باشد کنترل به LAB2 در برنامه منتقل می گردد.

مثال ۴-۵

JS                      LAB5

در صورتیکه مقدار SF برابر با یک باشد کنترل به LAB5 در برنامه منتقل می گردد.

مثال ۵-۵

JNO                    LAB20

چنانچه مقدار OF برابر با صفر باشد کنترل به LAB20 در برنامه منتقل می گردد. از این دستورالعمل معمولاً پس از اجرای عملیات ریاضی استفاده می شود.

مثال ۶-۵

```
MOV AX, -100
ADD AX, BX
SUB AX, CX
JNZ LABNEXT
:
LABNEXT: MOV CX, 10
:
```

در قطعه برنامه بالا ابتدا مقدار 100- در ثبات AX قرار داده می شود و سپس مقدار BX به آن افزوده می گردد و سپس مقدار CX از آن کسر می گردد. حال چنانچه نتیجه محاسبه یعنی مقدار AX مخالف صفر باشد کنترل به آدرس LABNEXT در برنامه منتقل می گردد.

جدول ذیل انواع دستورالعمل های پرش شرطی را نشان می دهد.

جدول ۱-۵

| شرط تست | نام دیگر دستورالعمل | نام دستورالعمل | عمل                            |
|---------|---------------------|----------------|--------------------------------|
| ZF=1    | JE                  | JZ             | انشعاب روی صفر                 |
| ZF=0    | JNE                 | JNZ            | انشعاب روی مخالف صفر           |
| SF=1    |                     | JS             | انشعاب روی علامت منفی          |
| SF=0    |                     | JNS            | انشعاب روی علامت غیر منفی      |
| OF=1    |                     | JO             | انشعاب روی سرریزی              |
| OF=0    |                     | JNO            | انشعاب روی عدم سرریزی          |
| PF=1    | JPE                 | JP             | انشعاب روی ایجاد بیت توازن     |
| PF=0    | JPO                 | JNP            | انشعاب روی عدم ایجاد بیت توازن |
| CF=1    |                     | JC             | انشعاب روی ایجاد بیت نقلی      |
| CF=0    |                     | JNC            | انشعاب روی عدم ایجاد بیت نقلی  |

در جدول بالا حروف مخفف کلمات زیر می باشند.

|   |          |        |
|---|----------|--------|
| Z | ZERO     |        |
| S | SIGN     |        |
| N | NOT      |        |
| P | PARITY   |        |
| O | OVERFLOW |        |
| O | ODD      | در JPO |
| E | EQUAL    |        |
| J | JUMP     |        |
| E | EVEN     | در JPE |
| C | CARRY    |        |

بایستی توجه داشت که دستورالعملهای پرش در حقیقت نقش دستورالعمل IF در سایر زبانهای برنامه‌نویسی را دارد.

#### مثال ۷-۵

```
TOT DW ?
MOV TOT, 0
MOV CX, 10
BEGIN: ADD TOT, CX
 DEC CX
 JNZ BEGIN
```

در قطعه برنامه بالا متغیر TOT از نوع word تعریف گردیده و مقدار آن برابر با صفر قرار داده شده است. مقدار اولیه CX نیز برابر با 10 می‌باشد. قطعه برنامه نقش یک حلقه تکرار دارد که مقادیر 1 تا 10 را با هم جمع می‌نماید یعنی تا مادامیکه مقدار CX مخالف صفر می‌باشد، مقدار CX با TOT جمع می‌گردد و یک واحد از CX کم می‌گردد.

### مثال ۸-۵

```

X DW ?
MOV AX,X
SUB AX,100
NEG AX
JNS ACT2
:
ACT2: ADD BX,AX
:

```

مقدار X در ثبات AX قرار داده شده آنگاه 100 واحد کاهش داده شده سپس مقدار AX در منفی یک ضرب شده حال چنانچه مقدار AX منفی نباشد کنترل به ACT2 منتقل می گردد. در غیر اینصورت اجرای دستورالعملهای بعدی ادامه می یابد.

### ۳-۵- دستورالعمل مقایسه

دستورالعمل مقایسه در زبان اسمبلی CMP می باشد. شکل کلی دستورالعمل بصورت زیر می باشد.

CMP opr1 , opr2

الف) opr1 و opr2 هر دو از نوع بایت یا word می باشند. این دستورالعمل مقادیر عملوندها را تغییر نمی دهد.

ب) opr1 و opr2 می توانند هر دو ثبات باشند.

ج) opr1 و opr2 هر دو نمی توانند متغیر باشند.

د) opr1 و opr2 هر دو نمی توانند ثابت باشند.

هـ) دستورالعمل CMP مانند دستورالعمل SUB عمل می کند، با این تفاوت که نتیجه درجائی ذخیره نمی گردد بلکه مقادیر فلگها را تغییر می دهد.

ز) این دستورالعمل روی فلگهای AF ، OF ، SF ، PF ، ZF ، SF اثر دارد.

## مثال ۵-۹

CMP AX, BX

این دستورالعمل دو مقدار AX, BX را با هم مقایسه می‌نماید. در حقیقت مقدار BX را از AX کم نموده و برحسب نتیجه بدست آمده مقادیر فلگها را تعیین می‌نماید.

## مثال ۵-۱۰

CMP AL, 10  
JZ LAB2

مقدار AL را با 10 مقایسه نموده در صورتیکه برابر باشند کنترل به LAB2 منتقل می‌گردد.

تعدادی دستورالعمل‌های پرش شرطی وجود دارند که با دستورالعمل CMP استفاده می‌گردند. دستورالعملهای پرش زیر وقتی استفاده می‌گردند که عملوندها بصورت بدون علامت (Unsigned) در نظر گرفته شوند.

## جدول ۵-۲

| نام  | نامهای دیگر | شرط                             | فلگها                          |
|------|-------------|---------------------------------|--------------------------------|
| JB   | JNAE, JC    | $\text{Opr 1} < \text{Opr2}$    | $\text{CF} = 1$                |
| JNB  | JAE, JNC    | $\text{Opr 1} \geq \text{Opr2}$ | $\text{CF} = 0$                |
| JBE  | JNA         | $\text{Opr 1} \leq \text{Opr2}$ | $\text{CF} \vee \text{ZF} = 1$ |
| JNBE | JA          | $\text{Opr 1} > \text{Opr2}$    | $\text{CF} \vee \text{ZF} = 0$ |

دستورالعملهای پرش زیر وقتی استفاده می‌شوند که عملوندها بصورت علامتدار (Signed) در نظر گرفته شوند.

## جدول ۵-۳

| نام  | نامهای دیگر | شرط                             | فلگها                                             |
|------|-------------|---------------------------------|---------------------------------------------------|
| JL   | JNGE        | $\text{Opr1} < \text{Opr 2}$    | $\text{SF} \oplus \text{OF} = 1$                  |
| JNL  | JGE         | $\text{Opr 1} \geq \text{Opr2}$ | $\text{SF} \oplus \text{OF} = 0$                  |
| JLE  | JNG         | $\text{Opr1} \leq \text{Opr2}$  | $(\text{SF} \oplus \text{OF}) \vee \text{ZF} = 1$ |
| JNLE | JG          | $\text{Opr1} > \text{Opr2}$     | $(\text{SF} \oplus \text{OF}) \vee \text{ZF} = 0$ |

حروف مخفف کلمات ذیل می باشند.

|   |              |
|---|--------------|
| B | Below        |
| A | Above        |
| G | Greater than |
| E | Equal to     |
| L | Less than    |
| C | Carry        |
| N | Not          |

مقصود از علامت  $\vee$  عملگر OR و مقصود از علامت  $\oplus$  عملگر Exclusive OR می باشد.

MOV AX, [BX]

مقدار اول در AX با مقدار دوم مقایسه می شود ; CMP AX, [DI]

آیا مقدار اول کمتر یا مساوی مقدار دوم می باشد؟ ; JBE DONE

در غیر اینصورت مبادله مقادیر ; XCHG AX, [DI]

MOV [BX], AX

DONE:

:

در قطعه برنامه بالا دو مقدار از حافظه که بوسیله ثباتهای DI و BX مشخص می شوند را بترتیب صعودی مرتب می نماید.

## مثال ۱۱-۵

```

CMP AL , 10 ;
JAE LAB1
:
LAB1: JA LAB 2
:
LAB2:
:

```

اگر محتوی AL بزرگتر از 10 باشد کنترل به LAB2، اگر محتوی AL مساوی 10 باشد کنترل به LAB1 در غیر اینصورت کنترل به دستورالعمل بعد از دستورالعمل JAE منتقل می‌گردد.

```

CMP AL , BL
JE ZERO

```

کنترل به آدرس ZERO منتقل می‌گردد اگر مقادیر BL و AL مساوی می‌باشند.

## مثال ۱۲-۵

```

MOV AX , -100
CMP BX , AX
JG LAB2

```

عملوندهای CMP علامت دار در نظر گرفته می‌شوند.



مثال ۱۳-۵

```
MOV AX, 100
CMP BX, AX
JA LAB2
```

عملوندهای CMP بدون علامت در نظر گرفته می‌شوند.

#### ۴-۵- دستورالعملهای تکرار

هر وقت بخواهیم تعدادی دستورالعمل بصورت مکرر اجرا گردد از دستورالعملهای تکرار بایستی استفاده نمائیم. دستورالعمل تکرار در زبان اسمبلی **LOOP** می باشد که شکل کلی آن بصورت زیر می باشد.

LOOP      آدرس

هر وقت کنترل بدستور LOOP میرسد ابتدا مقدار ثبات CX یک واحد کاهش یافته سپس محتوی ثبات CX با صفر مقایسه می‌گردد و در صورتیکه محتوی ثبات CX مخالف صفر باشد کنترل به آدرس داده شده منتقل می‌گردد. تعداد دفعات تکرار عملاً بایستی در ثبات CX قرار داد. دستورالعمل LOOP روی هیچ فلگی اثر ندارد.

مثال ۵-۱۴

```

MOV CX, 10
LABI:
 ⋮
LOOP LABI

```

این قطعه برنامه معادل برنامه پاسکال زیر می باشد یعنی دامنه تکرار ده بار اجرا می گردد.

```

FOR I:=1 TO 10 DO
 BEGIN
 :
 END ;

```

قطعه برنامه زیر مجموع عناصر آرایه X که از نوع Word و N عنصری می باشد را محاسبه نموده نتیجه را در متغیر TOTAL قرار می دهد.

```

N DW ?
TOTAL DW ?
X DW مقادیر عناصر آرایه
MOV CX, N
MOV AX, 0 ; مجموع برابر با صفر
MOV SI, AX ; برابر با صفر
START_LOOP: ADD AX,X [SI]; جمع عناصر
 ADD SI, 2
 LOOP START_LOOP
 MOV TOTAL, AX

```

قطعه برنامه زیر آرایه N عنصری A از نوع word را بصورت صعودی  
بروش حبابی مرتب می نماید.

```
Bubble ; Sort
N DW ?
MOV CX , N
DEC CX
LOOP 1: MOV DI , CX
 MOV BX , 0
LOOP2: MOV AX , A[BX]
 CMP AX, A[BX+2]
 JGE CONTINUE
 XCHG AX, A[BX+2]
 MOV A [BX] , AX
CONTINUE: ADD BX,2
 LOOP LOOP2
 MOV CX , DI
 LOOP LOOP1
```

شکل دیگر دستور تکرار بصورت زیر می باشد.

```
LOOPNE آدرس
یا
LOOPNZ آدرس
```

کار دستورالعمل LOOPNE یا LOOPNZ مانند دستورالعمل LOOP می باشد با این تفاوت که شرط تکرار آن است که مقدار CX مخالف صفر و مقدار ZF برابر با صفر باشد. این دستورالعمل روی هیچ فلگی اثر ندارد.

## مثال ۱۵-۵

```

ARR DB
N DW
MOV CX , N
MOV SI , -1
MOV AL, 20H; ASCII code for blank
NEXT: INC SI
 CMP AL, ARR[SI]; test for blank
 LOOPNE NEXT
 JNZ NOT_FOUND

```

قطعه برنامه بالا رشته داده شده N عنصری ARR از نوع بایت را جستجو می‌نماید که آیا کارکتر blank یا فاصله در رشته وجود دارد یا خیر؟ توجه داشته باشید که دستور تکرار بالا وقتی متوقف می‌شود که عناصر رشته همه مورد بررسی قرار گرفته باشند یا به کارکتر فاصله رسیده باشیم. شکل دیگر دستورالعمل تکرار بصورت زیر می‌باشد.

```

LOOPE آدرس
یا
LOOPZ آدرس

```

دستورالعمل LOOPE یا LOOPZ مانند دستورالعمل LOOP عمل می‌نماید با این تفاوت که شرط تکرار آن است که مقدار CX مخالف صفر و مقدار ZF برابر با یک باشد. این دستورالعمل روی هیچ فلگی اثر ندارد.

```

; BX = offset of the starting address
; DX = offset of the ending address
; BX = offset of nonzero (if found)
; BX = DI (if not found)

```

```

SUB DI , BX
INC DI ; تعداد بایت = (DI)-(BX)+1
MOV CX , DI
DEC BX
NEXT: INC BX ; point to next location
 CMP BYTE PTR [BX], 0 ; Compare it to zero
 LOOPE NEXT ; go compare next byte
 JNZ NZ_FOUND; Nonzero byte found?
 : ; NO.
NZ_FOUND:
 : ; Yes.

```

برنامه فوق یک بلوک از حافظه که آدرس شروع آن توسط ثبات BX و آدرس انتهای آن توسط ثبات DI مشخص شده را بایت به بایت جستجو نموده برای یافتن عنصری که مخالف صفر باشد.

شکل دیگر دستور تکرار بصورت زیر می باشد.

JCXZ                      آدرس

در حقیقت این دستورالعمل یک نوع دستورالعمل پرش می باشد که براساس فلگها عمل نمی کند بلکه براساس محتوی ثبات CX عمل می کند. چنانچه محتوی CX مساوی صفر باشد کنترل به آدرس داده شده منتقل می شود. این دستورالعمل روی هیچ فلگی اثر ندارد. نهایتاً جدول ذیل را داریم.

جدول ۴-۵

| عمل                             | نام         | نام دیگر | شرط تکرار             |
|---------------------------------|-------------|----------|-----------------------|
| <b>LOOP</b>                     | <b>LOOP</b> |          | <b>CX &lt; &gt; 0</b> |
| LOOP While equal or zero        | LOOPE       | LOOPZ    | CX < > 0 and ZF=1     |
| LOOP while not equal or nonzero | LOOPNE      | LOOPNZ   | CX < > 0 and ZF=0     |
| Branch on CX                    | JCXZ        |          | CX=0                  |

## مروری بر مطالب فصل

در این فصل دستورالعملهای پرسش تشریح گردیده این دستورالعملها بر اساس فلگها عمل می نمایند. ضمناً دستورالعملهای تکرار نیز مطرح گردیده است که در دستورالعملها تکرار تعداد دفعات تکرار در ثبات CX قرار می گیرد. دستورالعمل مقایسه CMP می باشد که مانند دستورالعمل SUB عمل نموده ولی نتیجه در جایی ذخیره نمی گردد بلکه روی فلگها اثر می گذارد.

## ☞ تمرین

- ۱- دستورالعمل CMP مشابه کدام دستورالعمل می باشد؟
- ۲- آیا استفاده از دستورالعمل CMP X, Y مجاز می باشد؟
- ۳- دستورالعمل JNA چه موقعی استفاده می شود؟
- ۴- دستورالعمل JLE چه موقعی مورد استفاده قرار می گیرد؟
- ۵- تفاوت دستورالعمل LOOP با LOOPZ چیست؟
- ۶- اگر در ابتدا محتوی CX را برابر با صفر قرار دهیم. حلقه تکرار چند بار اجرا می گردد؟
- ۷- دستورالعملهای اسمبلی معادل قطعه برنامه پاسکال زیر بدهید.  
  

```
S := 0 ;
FOR I:=1 TO N DO
 S := S+I ;
```
- ۸- قطعه برنامه ای به زبان اسمبلی معادل قطعه برنامه زیر بدهید.

```

X: ARRAY [1..10] OF INTEGER;
S, I: INTEGER;

S:= 0 ;

I:= 1 ;

WHILE I <= 10 DO
BEGIN

 S:=S +X [I];
 I:=I + 1 ;

END ;

```

۹-قطعه برنامه‌ای بنویسید که یک آرایه  $N$  عنصری از نوع بایت را در نظر بگیرد، تعداد عناصر مثبت آنرا مشخص نماید در متغیر TED قرار دهد.

۱۰-آیا می‌توان قطعه برنامه‌ای نوشت که  $N$  مقدار را بروش مرتب سازی درجی بصورت صعودی مرتب نماید؟

۱۱-یک قطعه برنامه بدهید که آرایه  $N$  عنصری  $X$  از نوع بایت را از نظر مکانی وارون نماید.

۱۲-قطعه برنامه‌ای بدهید که آرایه  $N$  عنصری  $X$  از نوع word را در نظر گرفته عناصر مخالف صفر آنرا در آرایه دیگری بنام  $Y$  قرار دهد (در صورت لزوم انتهای آرایه  $Y$  خالی بماند).

۱۳-قطعه برنامه‌ای بنویسید که مقدار صحیح و مثبت  $N$  را در نظر گرفته فاکتوریل آنرا مشخص نماید.



۱۴-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را در نظر گرفته کوچکترین عنصر آرایه را مشخص نماید.

۱۵-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع double word را در نظر گرفته کوچکترین عنصر آرایه را مشخص نماید.

۱۶-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را در نظر گرفته اندیس بزرگترین عنصر آرایه را مشخص نماید.

۱۷-قطعه برنامه‌ای بنویسید که مجموع زیر را محاسبه نماید.

$$1 + 2^2 + 3^2 + 4^2 + \dots + N^2$$

۱۸-قطعه برنامه‌ای بنویسید که مشخص نماید آیا عدد N اول می‌باشد یا خیر؟

۱۹-قطعه برنامه‌ای بنویسید که مشخص نماید آیا عدد N کامل می‌باشد یا خیر؟

۲۰-قطعه برنامه‌ای بنویسید که مقدار N از نوع بایت را گرفته  $N^X$  را محاسبه نماید.

۲۱-قطعه برنامه‌ای بنویسید که آرایه N عنصری X از نوع word را گرفته میانه آنرا مشخص نمود در MID قرار دهد.

# فصل ششم

## عملیات بیتی

### هدف کلی

آشنایی با عملیات روی بیت‌ها.

### اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا خواهید شد.

- ۱- عملیات منطقی و عملگرهای وابسته
- ۲- عملیات شیفت بیت‌ها
- ۳- عملیات چرخش بیت‌ها
- ۴- عملیات مربوط به فلگ‌ها
- ۵- تبدیل حروف کوچک به بزرگ و بالعکس.

## ۶-۱- عملیات منطقی

از دستورالعملهای منطقی برای انجام عملیات منطقی استفاده می‌شود. این دستورالعملها بصورت بیتی روی عملوندها عمل می‌نماید. عملیات منطقی عبارتند از NOT، AND، OR، XOR و TEST.

### ۶-۱-۱- دستورالعمل NOT

شکل کلی دستورالعمل NOT بصورت زیر می‌باشد.

NOT opr

الف) opr می‌تواند از نوع word یا بایت باشد.

ب) opr می‌تواند متغیر یا ثابت باشد.

ج) این دستورالعمل بیت‌های opr را از 0 به 1 و از 1 به 0 تبدیل می‌نماید. بعبارت دیگر مکمل 1 عملوند را می‌دهد.

د) دستورالعمل NOT روی هیچ فلگی اثر ندارد.

مثال ۶-۱

```
MOV DL, 8AH
NOT DL
```

قبل از اجرای دستور NOT

DL

|          |
|----------|
| 10001010 |
|----------|

بعد از اجرای دستور NOT

DL

|          |
|----------|
| 01110101 |
|----------|

بنابراین محتوی ثابت DL به 75H تغییر می‌نماید.

## ۶-۱-۲ دستورالعمل AND

شکل کلی دستورالعمل AND بصورت زیر می باشد.

AND dst , src

الف) عملوندهای src و dst هر دو از نوع بایت یا word می باشند.

ب) عملوندهای src و dst هر دو متغیر یا هر دو ثابت نمی توانند باشند.

ج) وقتی عملوند src ثابت باشد عملوند dst بایستی از نوع ثبات یا متغیر باشد.

د) بیت های dst و src نظیر به نظیر مطابق جدول ذیل and می شوند و نتیجه در dst قرار می گیرد.

جدول ۶-۱

| بیت اول | بیت دوم | بیت دوم and بیت اول |
|---------|---------|---------------------|
| 0       | 0       | 0                   |
| 0       | 1       | 0                   |
| 1       | 0       | 0                   |
| 1       | 1       | 1                   |

## مثال ۶-۲

```
MOV AL , 5BH
MOV DH, 4DH
AND AL, DH
```

مقادیر ثباتها قبل از اجرای دستورالعمل AND عبارتست از:

AL 01011011

DH 01001101

مقادیر ثباتها پس از اجرای دستورالعمل AND عبارتند از:

DH 

|          |
|----------|
| 01001101 |
|----------|

 بدون تغییر

AL 

|          |
|----------|
| 01001001 |
|----------|

 نتیجه

### ۳-۱-۶- دستورالعمل OR

شکل کلی دستورالعمل OR بصورت زیر می باشد.

OR dst , src

الف) عملوندهای dst و src از نوع بایت یا word می باشند.

ب) عملوندهای dst و src هر دو متغیر یا هر دو ثابت نمی توانند باشند.

ج) وقتی عملوند src ثابت باشد عملوند dst بایستی از نوع متغیر یا ثابت باشد.

د) بیت های dst و src نظیر به نظیر مطابق جدول ذیل or می شود و نتیجه در dst قرار می گیرد.

جدول ۲-۶

| بیت دوم or بیت اول | بیت دوم | بیت اول |
|--------------------|---------|---------|
| 0                  | 0       | 0       |
| 1                  | 1       | 0       |
| 1                  | 0       | 1       |
| 1                  | 1       | 1       |

### مثال ۳-۶

```
MOV BL , 0A5H
MOV AL, 2AH
OR AL , BL
```

مقادیر ثابت ها قبل از اجرای دستورالعمل OR

BL 

|          |
|----------|
| 10100101 |
|----------|

AL 

|          |
|----------|
| 00101010 |
|----------|

مقادیر ثباتها بعد از اجرای دستورالعمل OR

BL 

|          |
|----------|
| 10100101 |
|----------|

AL 

|          |
|----------|
| 10101111 |
|----------|

**۴-۱-۶- دستورالعمل XOR**

شکل کلی دستورالعمل XOR بصورت زیر می باشد.

XOR dst , src

الف) src و dst هر دو از نوع بایت یا word می باشند.

ب) src و dst هر دو متغیر یا ثابت نمی تواند باشند.

ج) بیت های src و dst نظیر به نظیر با استفاده از جدول ذیل xor گردیده نتیجه در

dst قرار می گیرد و مقدار src بدون تغییر باقی می ماند.

**جدول ۳-۶**

| بیت دوم xor بیت اول | بیت دوم | بیت اول |
|---------------------|---------|---------|
| 0                   | 0       | 0       |
| 1                   | 1       | 0       |
| 1                   | 0       | 1       |
| 0                   | 1       | 1       |

مثال ۴-۶

```
MOV CL, 2DH
MOV AL, 0C2H
XOR AL, CL
```

مقادیر ثباتها قبل از اجرای دستورالعمل XOR

AL 

|          |
|----------|
| 11000010 |
|----------|

CL 

|          |
|----------|
| 00111101 |
|----------|

مقادیر ثباتها بعد از اجرای دستورالعمل XOR

CL 

|           |
|-----------|
| 0011 1101 |
|-----------|

AL 

|           |
|-----------|
| 1111 1111 |
|-----------|

۵-۱-۶- دستورالعمل TEST

شکل کلی دستورالعمل TEST بصورت زیر می باشد:

```
TEST opr1, opr2
```

الف) opr1 و opr2 هر دو از نوع بایت یا word می باشد.

ب) opr1 و opr2 هر دو ثابت یا هر دو متغیر نمی توانند باشند.

ج) این دستورالعمل مانند دستورالعمل AND عمل می نماید ولی نتیجه را در جایی

ذخیره نمی کند یعنی دو عملوند بدون تغییر باقی می مانند و فقط مقادیر فلگ ها را

تغییر می دهد.

## مثال ۶-۵

```
MOV AL , 25
MOV DH , 0E4H
TEST AL , DH
```

مقادیر ثباتها قبل از اجرای دستورالعمل TEST

AL 

|          |
|----------|
| 00011001 |
|----------|

DH 

|          |
|----------|
| 11100100 |
|----------|

مقادیر ثباتها پس از اجرای دستورالعمل TEST

AL 

|          |
|----------|
| 00011001 |
|----------|

DH 

|          |
|----------|
| 11100100 |
|----------|

بایستی توجه کرد که دستورالعمل TEST باعث می شود که مقدار ZF برابر

با یک گردد.

## مثال ۶-۶

```
MOV AL , 0ABH
NOT AL
TEST AL, 10100101B
JZ YES
:
YES:
:
```

قطعه برنامه فوق مشخص می نماید که آیا مقادیر بیت های 7، 5، 2 و 0 ثبات

AL برابر با یک می باشد یا خیر؟ مقدار 10100101B عملاً MASK می باشد که



در بیت‌هایی که می‌خواهیم برای یک بودن تست شود مقدار یک و در سایر بیت‌ها مقدار صفر را قرار می‌دهیم.

AL    10101011

پس از اجرای دستور NOT

AL    01010100

MASK 10100101

در این مثال پس از اجرای دستورالعمل TEST مقدار ثبات AL بدون تغییر باقی ماند و فقط مقدار فلگ ZF برابر با یک می‌شود.

مثال ۶-۷

```

OR DL, 00000101B
XOR DL, 01000010B
AND DL, 11100111B
MOV AL, DL
NOT AL
TEST AL, 10000010B
JZ EXIT
:
EXIT:
:

```

قطعه برنامه فوق ابتدا بیت‌های شماره 0 و 2 ثبات DL را یک می‌کند و بیت‌های شماره 4 و 3 را به صفر تبدیل می‌کند و بیت‌های شماره 1 و 6 را مکمل می‌نماید در ضمن چنانچه بیت‌های شماره 7 و 1 برابر با یک باشند کنترل به EXIT منتقل می‌نماید.

## مثال ۸-۶

قطعه برنامه زیر بیت‌های شماره فرد ثبات AL را مکمل می‌نماید. یعنی 1 به 0 و 0 به 1 تبدیل می‌نماید.

```
MOV AL, 0C7H
MOV MASK, 10101010B
XOR AL, MASK
```

|      |          |
|------|----------|
| AL   | 11000111 |
| MASK | 10101010 |

پس از اجرای دستورالعمل XOR مقادیر AL و MASK عبارتند از:

|      |          |
|------|----------|
| AL   | 01101101 |
| MASK | 10101010 |

## مثال ۹-۶

قطعه برنامه زیر بیت‌های شماره زوج ثبات AL را به یک تبدیل می‌نماید.

```
MOV AL, 0A6H
MOV MASK, 55H
OR AL, MASK
```

|      |          |
|------|----------|
| AL   | 10100110 |
| MASK | 01010101 |

پس از اجرای دستورالعمل OR

|      |          |
|------|----------|
| AL   | 11110111 |
| MASK | 01010101 |

مثال ۱۰-۶

قطعه برنامه زیر بیت‌های شماره فرد AL را به صفر تبدیل می‌نماید.

```
MOV AL, 0C7H
MOV MASK, 55H
AND AL, MASK
```

|      |          |
|------|----------|
| AL   | 11000111 |
| MASK | 01010101 |

پس از اجرای دستورالعمل AND

|      |          |
|------|----------|
| AL   | 01000101 |
| MASK | 01010101 |

در قطعه برنامه زیر اگر بیت‌های شماره 1 و 14 یا بیت‌های شماره 9 و 7 ثبات AX برابر با یک باشند کنترل به TASK 1 و اگر بیت 3 یا 4 برابر با یک باشند کنترل به TASK2 در غیر اینصورت کنترل به TASK3 منتقل می‌گردد.

|        |           |
|--------|-----------|
| NOT    | AX        |
| TEST   | AX, 4002H |
| JZ     | TASK1     |
| TEST   | AX, 280H  |
| JZ     | TASK1     |
| NOT    | AX        |
| TEST   | AX, 18H   |
| JNZ    | TASK2     |
| TASK3: | :         |
| TASK1: | :         |
| TASK2: | :         |

## ۶-۲- عملیات شیفت

عملیات شیفت باعث تغییر مکان بیت‌های یک بایت یا یک word بطرف چپ یا راست می‌شود. دستورالعملهای متعددی برای این کار مورد استفاده قرار می‌گیرند که عبارتند از:

|       |                  |     |
|-------|------------------|-----|
| Shift | Logical Left     | SHL |
| Shift | Logical Right    | SHR |
| Shift | Arithmetic Left  | SAL |
| Shift | Arithmetic Right | SAR |

### ۶-۲-۱- دستورالعمل SHL

شکل کلی دستورالعمل SHL بصورت زیر می‌باشد.

SHL opr , cnt

الف) cnt تعداد بیت‌هایی می‌باشد که بطرف چپ شیفت داده می‌شود. در صورتیکه cnt مخالف یک باشد از ثبات CL استفاده می‌نمائیم.

ب) opr می تواند از نوع بایت یا word باشد.

ج) opr می تواند متغیر یا ثابت باشد.

د) opr ثابت نمی تواند باشد.

هـ) روی فلگهای OF، SF، ZF، PF و CF اثر دارد.

ز) بیت های opr را با اندازه cnt بیت بطرف چپ شیفت می دهد و از طرف راست با صفر پر می شود.



مثال ۱۱-۶

```
SHL AX, CL
SHL BL, CL
SHL AL, 1
```

```
STC
MOV CL, 3
MOV DL, 8DH
SHL DL, CL
```

دستورالعمل STC مقدار فلگ CF را به یک تبدیل می نماید.

قبل از اجرای دستورالعمل SHL

|    |          |    |   |
|----|----------|----|---|
| CL | 00000011 | CF | 1 |
| DL | 10001101 |    |   |

بعد از اجرای دستورالعمل SHL

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 01101000 |
| CF | 0        |

## ۶-۲-۲ دستورالعمل SHR

شکل کلی دستورالعمل SHR بصورت زیر می باشد.

SHR opr, cnt

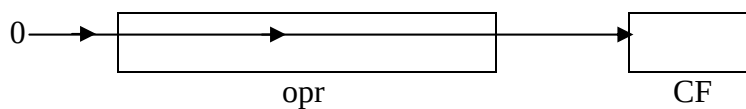
الف) opr می تواند از نوع بایت یا word باشد.

ب) opr می تواند متغیر یا ثابت باشد. ولی ثابت نمی تواند باشد.

ج) روی فلگهای CF، ZF، PF اثر دارد.

د) اگر مقدار cnt برابر با یک باشد خودش را می نویسم در غیر اینصورت از ثبات CL استفاده می نمائیم.

هـ) بیت های opr را با اندازه cnt بیت بطرف راست شیفت داده و از طرف چپ با صفر پر می نمائیم.



مثال ۶-۱۲

```
SHR DL, 1
SHR AL, CL
```

مثال ۶-۱۳

```
STC
MOV CL, 3
MOV DL, 8DH
SHR DL, CL
```

محتوی ثبات DL را سه بیت بطرف راست شیفت می دهد.

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10001101 |
| CF | 1        |

مقادیر پس از اجرای دستورالعمل SHR عبارتند از

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 00010001 |
| CF | 1        |

۶-۲-۳- دستورالعمل SAL

شکل کلی دستورالعمل SAL بصورت زیر می باشد.

SAL opr, cnt

الف) opr می تواند از نوع بایت یا word باشد.

ب) cnt اگر یک باشد مقدار یک را می نویسم در غیر اینصورت از ثبات CL استفاده می نمائیم.

ج) opr ثابت نمی تواند باشد.

د) روی فلگهای OF، SF، ZF، PF و CF اثر دارد.

هـ) بیت های opr را باندازه cnt بیت بطرف چپ شیفت می دهد و از طرف راست با صفر پر می شود.



مثال ۶-۱۴

```
SAL DL, 1
SAL AL, CL
SAL [BX], CL
SAL ARR [SI], CL
```

مثال ۶-۱۵

```
STC
MOV CL, 3
MOV DL, 8DH
SAL DL, CL
```

قبل از اجرای دستورالعمل SAL

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10001101 |
| CF | 1        |

بعد از اجرای دستورالعمل SAL

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 01101000 |
| CF | 0        |



#### ۶-۲-۴- دستورالعمل SAR

شکل کلی دستورالعمل SAR بصورت زیر می باشد.

SAR opr , cnt

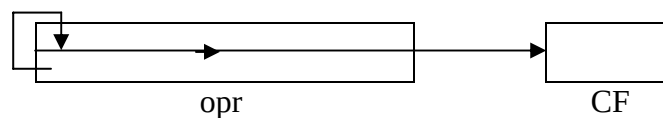
الف) opr می تواند از نوع ثبات یا متغیر باشد.

ب) opr نمی تواند ثابت باشد.

ج) opr می تواند از نوع بایت یا word باشد.

د) cnt اگر معادل یک باشد مقدار 1 در غیر اینصورت از ثبات CL استفاده می نمائیم.

هـ) بیت های opr را باندازه cnt بیت بطرف راست شیفت داده و از سمت چپ با MSB پر می نماید.



#### مثال ۶-۱۶

```
SAR DL, 1
SAR AX, CL
SAR [BX], CL
```

#### مثال ۶-۱۷

```
MOV CL, 3
STC
MOV DL, 8DH
SAR DL, CL
```

مقادیر قبل از اجرای دستورالعمل SAR

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10001101 |
| CF | 1        |

مقادیر بعد از اجرای دستورالعمل SAR

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 11110001 |
| CF | 1        |

### ۶-۳- عملیات چرخش (Rotate)

عملیات چرخش باعث دور زدن بیت‌های یک بایت یا word می‌شوند. عبارت دیگر مانند دستورالعمل‌های شیفت باعث خارج شدن بیت‌ها از بایت یا word نمی‌شود. دستورالعمل‌های چرخش عبارتند از:

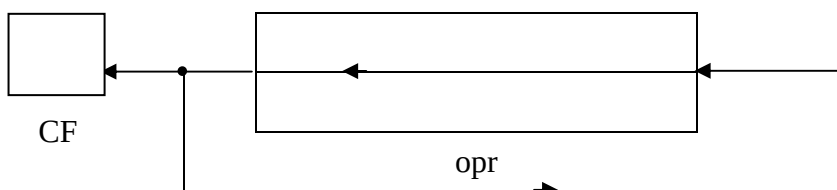
|                            |     |
|----------------------------|-----|
| Rotate Left                | ROL |
| Rotate Right               | ROR |
| Rotate Left through Carry  | RCL |
| Rotate Right through Carry | RCR |

#### ۶-۳-۱- دستورالعمل ROL

شکل کلی دستورالعمل ROL بصورت زیر می‌باشد.

ROL opr, cnt

- الف) opr می‌بایستی از نوع بایت یا word باشد.
- ب) opr می‌بایستی متغیر یا ثابت باشد.
- ج) اگر مقدار cnt برابر با یک باشد عدد 1 را می‌نویسم در غیر اینصورت از ثابت CL استفاده می‌کنیم.
- د) روی فلگ CF اثر دارد.
- هـ) این دستورالعمل باندازه cnt بیت از سمت چپ چرخش می‌دهد.



مثال ۶-۱۸

```
ROL DL, CL
ROL BX, 1
ROL [BX], CL
```

مثال ۶-۱۹

```
MOV CL, 3
MOV DL, 8DH
STC
ROL DL, CL
```

مقادیر قبل از اجرای دستورالعمل ROL

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10001101 |
| CF | 1        |

مقادیر بعد از اجرای دستورالعمل ROL

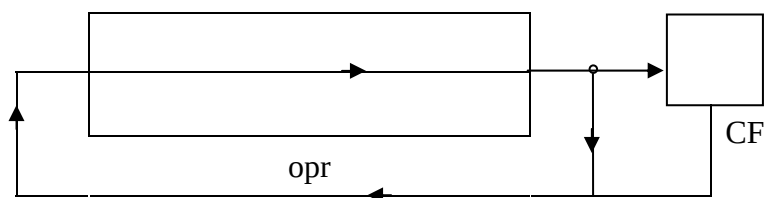
|    |          |
|----|----------|
| CL | 00000011 |
| DL | 01101100 |
| CF | 0        |

## ۲-۳-۶- دستورالعمل ROR

شکل کلی دستورالعمل ROR بصورت زیر می باشد:

ROR opr , cnt

- الف) opr از نوع بایت یا word می باشد.
- ب) opr می بایستی از نوع متغیر یا ثابت باشد. opr ثابت نمی تواند باشد.
- ج) cnt اگر معادل یک باشد عدد 1 را می نویسم در غیر اینصورت از ثابت CL استفاده می نمائیم.
- د) روی فلگ CF اثر دارد.
- هـ) باندازه cnt بیت بطرف راست چرخش می دهد.



مثال ۶-۲۰

```
ROR DL, 1
ROR BX, CL
ROR AL, CL
ROR X[DI], CL
```

مثال ۶-۲۱

```
MOV CL, 3
STC
MOV DL, 8DH
ROR DL, CL
```

مقادیر قبل از اجرای دستورالعمل ROR

|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10001101 |
| CF | 1        |

مقادیر بعد از اجرای دستورالعمل ROR

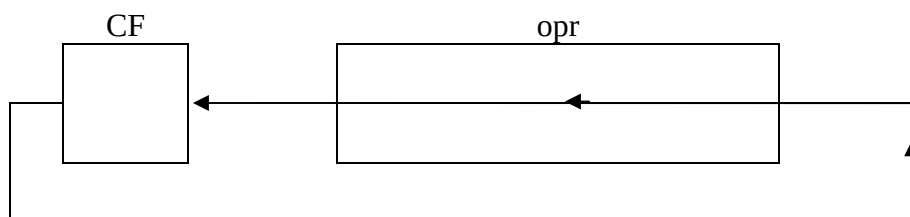
|    |          |
|----|----------|
| CL | 00000011 |
| DL | 10110001 |
| CF | 1        |

۶-۳-۳ - دستورالعمل RCL

شکل کلی دستورالعمل RCL بصورت زیر می باشد.

```
RCL opr , cnt
```

- الف) چنانچه مقدار cnt یک باشد مقدار 1 نوشته می شود در غیر اینصورت از ثبات CL استفاده می گردد.
- ب) opr بایستی از نوع بایت یا word باشد.
- ج) opr بایستی متغیر یا ثبات باشد. opr مقدار ثابت نمی تواند باشد.
- د) روی فلگ CF اثر دارد.
- هـ) دستورالعمل RCL باندازه cnt بیت از بیت های opr را از طرف چپ و از طریق بیت CF چرخش می دهد.



مثال ۶-۲۲

```

RCL DL, 1
RCL BX, CL
RCL AL, CL
RCL BL, 1
RCL [BX], CL

```

مثال ۶-۲۳

```

MOV CL, 3
MOV DL, 8DH
STC
RCL DL, CL

```

مقادیر قبل از اجرای دستورالعمل RCL

|    |          |
|----|----------|
| DL | 10001101 |
| CL | 00000011 |
| CF | 1        |

مقادیر بعد از اجرای دستورالعمل RCL

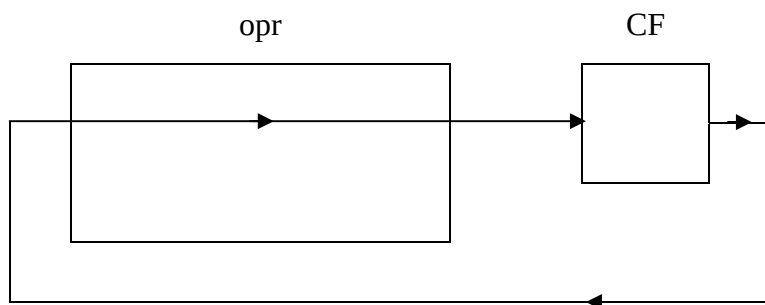
|    |          |
|----|----------|
| CL | 00000011 |
| DL | 01101110 |
| CF | 0        |

#### ۴-۳-۶- دستورالعمل RCR

شکل کلی دستورالعمل RCR بصورت ذیل می باشد:

RCR opr , cnt

- الف) opr بایستی از نوع word یا بایت باشد.
- ب) opr بایستی ثابت یا متغیر باشد و ثابت نمی تواند باشد.
- ج) اگر مقدار cnt برابر با یک باشد بایستی مقدار 1 را نوشت در غیر این صورت از ثابت CL استفاده نمود.
- د) روی فلگ CF اثر دارد.
- هـ) دستورالعمل RCR باندازه cnt بیت بطرف راست از طریق فلگ CF چرخش می دهد.



مثال ۶-۲۴

```
RCR DL, 1
RCR [BX], CL
RCR AL, CL
RCR X[BX][DI], CL
RCR [BX], 1
```

مثال ۶-۲۵

```
MOV CL, 3
MOV DL, 8DH
STC
RCR DL, CL
```

مقادیر قبل از اجرای دستورالعمل RCR

|    |                     |
|----|---------------------|
| CL | <div>00000011</div> |
| DL | <div>10001101</div> |
| CF | <div>1</div>        |

مقادیر بعد از اجرای دستورالعمل RCR

|    |                     |
|----|---------------------|
| CL | <div>00000011</div> |
| DL | <div>01110001</div> |
| CF | <div>1</div>        |



## ۶-۴- عملیات فلگ‌ها

دستورالعملهای مربوط به عملیات روی فلگ‌ها در ذیل داده شده‌اند. این دستورالعملها عبارتند از:

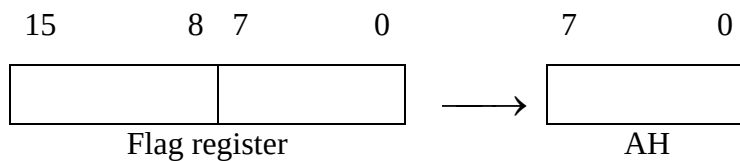
جدول ۶-۴

|                  |     |                |
|------------------|-----|----------------|
| Clear Carry      | CLC | CF صفر می‌شود  |
| Complement Carry | CMC | CF مکمل می‌شود |
| Set Carry        | STC | CF یک می‌شود   |
| Clear Direction  | CLD | DF صفر می‌شود  |
| Set Direction    | STD | DF یک می‌شود   |
| Clear Interrupt  | CLI | IF صفر می‌شود  |
| Set Interrupt    | STI | IF یک می‌شود   |

همانطوریکه ملاحظه می‌شود این دستورالعملها فاقد عملوند می‌باشند. این دستورالعمل‌ها فقط روی فلگ مربوطه اثر دارند. بعنوان مثال CLC فقط روی فلگ CF اثر می‌کند و مقدار قبلی آنرا تغییر می‌دهد. از طرف دیگر دو دستورالعمل LAHF و SAHF نیز مربوط به عملیات فلگ‌ها می‌باشند که در ذیل توصیف می‌گردند. شکل کلی دستورالعمل LAHF بصورت زیر می‌باشد.

### LAHF

- الف) این دستورالعمل فاقد عملوند می‌باشد.
- ب) بایت کم ارزش (بیت‌های 0 تا 7) ثبات فلگ را به AH منتقل می‌نماید.
- ج) روی هیچ فلگی اثر ندارد.
- د) فقط مقادیر بیت‌های شماره 7، 6، 4، 2، 0 منتقل می‌گردد.



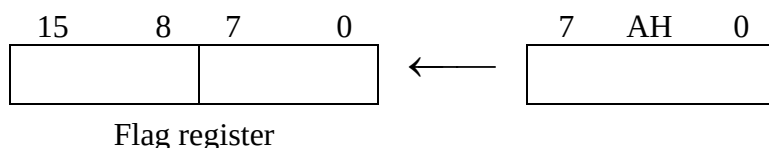
شکل کلی دستورالعمل SAHF بصورت زیر می باشد.

### SAHF

الف) این دستورالعمل فاقد عملوند می باشد.

ب) روی هیچ فلگی اثر ندارد.

ج) محتوی ثابت AH را به بایت کم ارزش (بیت های 0 تا 7) ثابت فلگ منتقل می کند.



### ۵-۶- تبدیل حروف

همانطوریکه میدانید به هر کارکتر یک کد اسکی بین 0 تا 255 تخصیص می یابد. کداسکی برای حروف بزرگ A تا Z بترتیب از 65 تا 90 یا از 41H تا 5AH می باشد. حروف کوچک نیز دارای کد اسکی بین 97 تا 122 یا 61H تا 7AH می باشند.

|   |     |          |
|---|-----|----------|
| A | 65  | 01000001 |
| A | 97  | 01100001 |
| Z | 90  | 01011010 |
| Z | 122 | 01111010 |

همانطوریکه ملاحظه می کنید تفاوت نمایش بین A و a یا Z و z در بیت شماره 5 آنها می باشد. بنابراین نتیجه می گیریم برای تبدیل حروف کوچک به بزرگ کافی است که بصورت زیر عمل نمائیم.

AND 00100000B , حرف کوچک

و برای تبدیل حروف بزرگ به حروف کوچک کافی است که بصورت زیر عمل شود

OR 00100000B , حرف بزرگ

#### مثال ۲۶-۶

در رشته STR حروف کوچک را به بزرگ تبدیل نمائید. بایستی کنترل شود که کارکتر از نوع حروف کوچک می باشد سپس به حروف بزرگ متناظر تبدیل گردد.

|       |      |                                  |
|-------|------|----------------------------------|
|       | STR  | DB 'Change to uppercase letters' |
|       | LEA  | BX , STR+1                       |
|       | MOV  | CX, 26                           |
| LAB1: | MOV  | AX, [BX]                         |
|       | CMP  | AH, 61H                          |
|       | JB   | LAB2                             |
|       | CMP  | AH, 7AH                          |
|       | JA   | LB2                              |
|       | AND  | AH, 0DFH                         |
|       | MOV  | [BX], AH                         |
| LAB2: | INC  | BX                               |
|       | LOOP | LAB1                             |

## مروری بر مطالب فصل

در این فصل عملیات بیتی مورد بحث قرار گرفت. ابتدا عملیات منطقی مانند AND، OR، NOT، XOR، TEST بحث گردید. این دستورالعملها روی بیتها عمل می نمایند و همانطوریکه گفته شد به جزء دستورالعمل NOT سایر دستورالعملها دارای دو عملوند می باشند. TEST شبیه AND می باشد ولی نتیجه در جایی ذخیره نمی گردد. عملیات بعدی دستورالعملهای شیفت و چرخش می باشند که این دستورالعملها نیز روی بیتها عمل می کنند و چنانچه تعداد بیتهای مورد عمل بیش از یک باشد بایستی تعداد بیتها را در ثبات CL قرار دهیم. دستورالعملهای شیفت عبارتند از SHL، SHR، SAL، SAR و دستورالعملهای چرخش عبارتند از ROL، ROR، RCL، RCR. در انتهای فصل عملیات مربوط به تغییر فلگها مورد بحث قرار گرفته و در نهایت با استفاده از چندین مثال کاربرد مطالب فوق نشان داده شده است که از جمله تبدیل حروف کوچک در یک متن به حروف بزرگ می باشد.

## ⌘ تمرین

۱- اشکال دستورالعملهای زیر را مشخص نمایید.

```
ROL DL, 2
SHL BX, CX
MOV X, [BX]
```

۲- مقدار CF و BL پس از اجرای دستورالعملهای ذیل چیست؟

```
MOV CL, 4
MOV BL, 0BBH
CLC
SAL BL, CL
XOR BL, CL
SAR BL, CL
```

۳- مقدار CF و BX پس از اجرای دستورالعملهای ذیل چیست؟

```
MOV CL, 3
MOV BX, 2ACH
STC
ROL BX, CL
INC CL
RCL BX, CL
```

۴- دستورالعملهای ذیل روی چه فلگ‌هایی اثر دارند؟

RCR, SHL, MOV, TEST, AND, XOR

۵- برنامه‌ای بنویسید که یک رشته را گرفته تعداد حروف کوچک آنرا مشخص نماید.

۶- برنامه‌ای بنویسید که بیت‌های شماره فرد ثبات AX را به یک و بیت‌های زوج ثبات AX را به صفر تبدیل نماید.

۷- برنامه‌ای بنویسید که مشخص نماید آیا بیت‌های شماره ۷، ۵، ۲ ثبات BL برابر با صفر می‌باشد یا خیر؟

۸- برنامه‌ای بنویسید که مشخص نماید آیا بیت‌های شماره زوج ثبات AX برابر با یک می‌باشد یا خیر؟

۹- برنامه‌ای بنویسید که بیت‌های ثبات AX را آنقدر بطرف چپ شیفت دهد تا MSB آن برابر با یک گردد. در صورتیکه در ابتدا MSB برابر با یک می‌باشد کنترل به EXIT منتقل گردد.

۱۰- پس از اجرای دستورالعمل‌های ذیل مقادیر ثباتها چیست؟

```
MOV AX, 4BCH
MOV DX, 0F2BCH
XOR AX, DX
SUB DX, 8
NOT DX
ADD AX, 16
AND DX, AX
```

۱۱- با چه دستوری مقادیر CF, DF, IF برابر با یک می‌شود.

۱۲- به چند طریق می‌توان مقدار ثبات AX را برابر با صفر قرار داد؟

۱۳- برنامه‌ای بنویسید که مقدار بیت شماره n ام ثبات AX را مشخص نماید.

# فصل هفتم

## ماکروها و روالها و وقفه‌ها

### هدف کلی

معرفی پشته، مکرو، روال، وقفه‌ها.

### اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا می‌شوید.

۱-تعریف پشته، کاربرد و دستورالعملهای مربوطه.

۲-تعریف روال، کاربرد، نحوه ایجاد و فراخوانی آنها.

۳-تعریف ماکرو، نحوه ایجاد، کاربرد و فراخوانی آنها.

۴-عملگرهای ماکرو.

۵-وقفه‌ها و دستورالعملهای مربوطه.

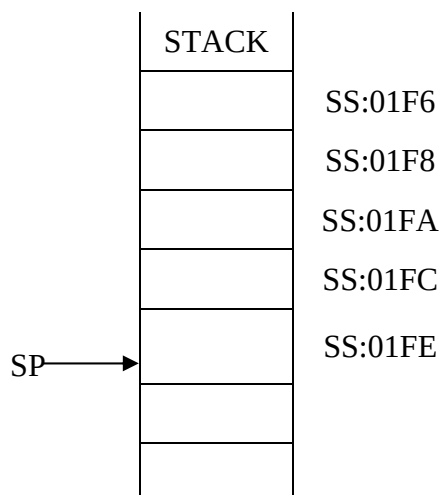
۶-روالهای تبدیل باینری به اسکی و بالعکس.

۷-اندازه‌گیری زمان اجرای برنامه‌ها.

۸- ایجاد تأخیر.

### ۷-۱- پشته ( Stack )

پشته قسمتی از حافظه اصلی یا RAM می باشد. پشته خاصیت LIFO دارد یعنی عنصری که آخر وارد پشته می شود اولین عنصری است که از پشته خارج می شود. مقداری که پشته در خود ذخیره می نماید از نوع word می باشد. ثبات SP به عنصر top پشته اشاره می نماید.



ثبات SP همیشه به آخرین word ای که وارد پشته شده اشاره می نماید. عملیاتی که روی پشته انجام می شوند عبارتند از PUSH و POP که در ذیل تشریح می گردد. همانطوریکه قبلاً گفته شد ثبات SS به ابتدای سگمنت پشته اشاره می نماید.

#### ۷-۱-۱- دستورالعمل PUSH

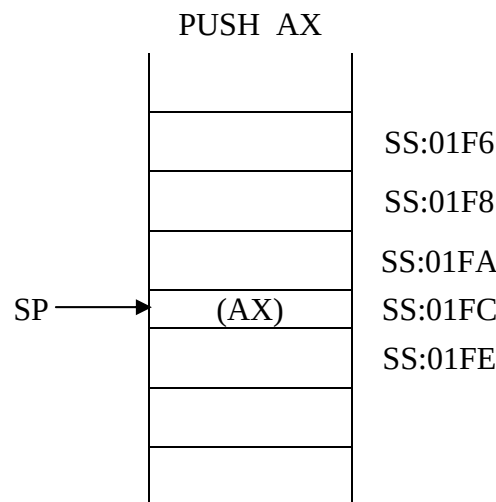
شکل کلی دستورالعمل PUSH بصورت زیر می باشد.

PUSH opr



- الف) opr بایستی از نوع word باشد.
- ب) opr بایستی متغیر یا ثبات باشد. ثابت نمی تواند باشد.
- ج) opr می تواند یکی از ثباتهای DS، SS، ES باشد.
- د) دستورالعمل PUSH روی هیچ فلگی اثر ندارد.
- ه) این دستورالعمل opr را در پشته قرار می دهد و ثبات SP را باندازه دو واحد کاهش می دهد.

مثال ۷-۱



## ۷-۱-۲- دستورالعمل POP

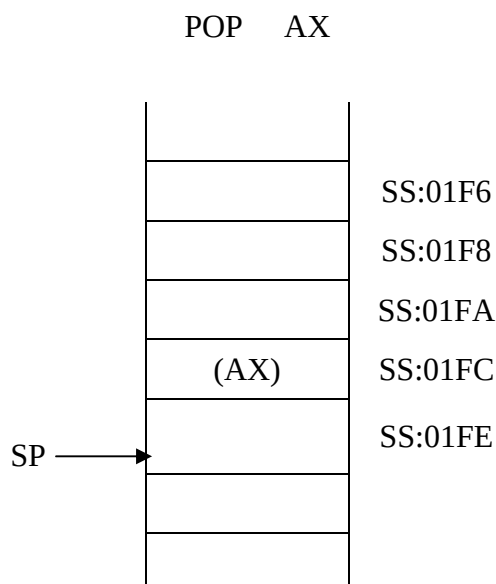
شکل کلی دستورالعمل POP بصورت زیر می باشد.

POP dst

- الف) dst بایستی از نوع word باشد.
- ب) dst بایستی متغیر یا ثبات باشد.
- ج) dst می تواند یکی از ثباتها DS، SS، ES باشد.

د) دستورالعمل POP روی هیچ فلگی اثر ندارد.  
 ه) این دستورالعمل عنصری را که بوسیله SP مشخص می‌شود در dst قرار داده و ثابت SP را باندازه دو واحد افزایش می‌دهد.

مثال ۷-۲



مثال ۷-۳

```

PUSH AX
PUSH BX
PUSH ES
PUSH DI
 ⋮
POP DI
POP ES
POP BX
POP AX

```

۷-۱-۳- دستورالعمل **PUSHF**

شکل کلی دستورالعمل PUSHF بصورت زیر می باشد.

PUSHF

الف) این دستورالعمل فاقد عملوند می باشد.

ب) این دستورالعمل روی هیچ فلگی اثر ندارد.

ج) این دستورالعمل محتوی ثبات فلگ را در پشته قرار می دهد. و ثبات SP را به اندازه دو واحد کاهش می دهد.

مثال ۷-۴

PUSHF

#### ۷-۱-۴- دستورالعمل POPF

شکل کلی دستورالعمل POPF بصورت زیر می باشد.

POPF

الف) این دستورالعمل فاقد عملوند می باشد.

ب) این دستورالعمل مقداری که در پشته بوسیله ثبات SP اشاره می شود را به ثبات

فلگ منتقل نموده و محتوی ثبات SP را دو واحد افزایش می دهد.

ج) روی کلیه فلگ ها اثر دارد.

مثال ۷-۵

POPF

#### ۷-۲- روال (Procedures)

برای راحتی نوشتن برنامه‌ها، برنامه‌ها را می‌توان به تعدادی روال تقسیم نمود و نوشتن هر روال را به یک برنامه‌نویس واگذار کرد. شکل کلی روال‌ها بصورت زیر می‌باشد.

```
name PROC NEAR
 :
 RET
name ENDP
```

الف) PROC مخفف کلمه Procedure می‌باشد.

ب) name نام روال می‌باشد.

ج) RET مخفف کلمه return می‌باشد که کنترل را به برنامه فراخوان این روال برمی‌گرداند.

د) ENDP مخفف end of procedure می‌باشد که انتهای روال را مشخص می‌کند.

ه) دستورالعمل RET روی هیچ فلگی اثر ندارد.

ز) NEAR مشخصه روال می‌باشد. کلیه روال‌های یک برنامه بایستی دارای مشخصه NEAR باشند.

مثال ۶-۷

```
SAVEREG PROC NEAR
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
RET
SAVEREG ENDP
```

روال فوق برای ذخیره کردن مقادیر ثباتهای مشخص شده استفاده می‌شود.  
بمنظور فراخوانی یک روال از دستور CALL استفاده می‌گردد. شکل کلی دستور  
CALL بصورت زیر می‌باشد.

CALL Name

الف) دستور CALL روی هیچ فلگی اثر ندارد.  
ب) name اسم روالی می‌باشد که توسط دستورالعمل CALL فراخوانی می‌شود.  
پس از فراخوانی روال به محض رسیدن به دستور RET کنترل به دستورالعمل بعد  
از دستور CALL برمی‌گردد.

مثال ۷-۷

CALL SAVEREG

بایستی توجه داشت که زبان اسمبلی برنامه اصلی را بعنوان یک روال با  
مشخصه FAR در نظر می‌گیرد که به محض رسیدن به RET برنامه اصلی، کنترل  
به سیستم عامل برمی‌گردد.

بنابراین در یک برنامه فقط یک روال اصلی با مشخصه FAR وجود دارد و برنامه می‌تواند شامل تعدادی روال با مشخصه NEAR باشد که در روال اصلی توسط دستورهای CALL فراخوانی گردند. متذکر می‌شویم که آدرسهای برگشت روال‌ها در پشته ذخیره می‌گردند. یک روال را می‌توان بارها فراخوانی نمود. در فصل دهم استفاده از روال‌ها نشان داده شده است. یک روال را می‌توان بصورت بازگشتی یا Recursive فراخوانی نمود.

### ۳-۷- ماکروها (Macros)

ماکروها عملاً قسمت‌هایی از برنامه مبداء (source program) می‌باشند که با استفاده از نام آنها می‌توان در هر جای برنامه مبداء درج نمود. در حقیقت ماکروها دنباله‌ای از دستورالعملهای اسمبلی می‌باشند که ممکن است چندین بار در برنامه ظاهر شوند. به جای چندین بار تایپ نمودن این دنباله از دستورالعملها می‌توان فقط نام آنها را در برنامه درج نمود. شکل کلی ماکروها بصورت زیر می‌باشد.

```
name MACRO
 :
 ENDM
```

الف) name عبارتست از نام macro

ب) ENDM انتهای macro را مشخص می‌نماید.

ج) بین MACRO و ENDM دنباله‌ای از دستورالعملهای برنامه مبداء قرار می‌گیرد.

د) در زمان ترجمه برنامه به زبان ماشین ابتدا در برنامه به جای macro ها دستورالعملهای وابسته به ماکروها قرار داده می‌شوند سپس به زبان ماشین ترجمه می‌شوند.

ه) تفاوت بین روال و ماکرو در این است که ماکرو در برنامه در زمان اجرای برنامه وجود خارجی ندارد بلکه روال در زمان اجرای برنامه وجود داشته و فراخوانی می‌گردد.

ز) macro ها بایستی در ابتدای برنامه تعریف شوند.

مثال ۸-۷

```
TOT MACRO
 MOV AX,X
 ADD AX,Y
 ENDM
```

ماکرو TOT مجموع محتوی دو متغیر Y و X را محاسبه و در ثبات AX قرار می‌دهد.

حال برنامه زیر را در نظر بگیرید که از این ماکرو استفاده می‌نماید.

```
 X DW ?
 Y DW ?
 S DW ?
MOV X, 1000
MOV Y, 3000
TOT
MOV S , AX
SUB X, 10
TOT
ADD S, AX
 :
```

ابتدا این برنامه قبل از اجرا یعنی در زمان ترجمه بصورت زیر در می‌آید.

```

X DW ?
Y DW ?
S DW ?
MOV X, 1000
MOV Y, 3000
; macro جایگزین
MOV AX, X
ADD AX, Y
MOV S, AX
SUB X, 10
; macro جایگزین
MOV AX, X
ADD AX, Y
;
ADD S, AX
:

```

ماکروها می‌توانند دارای پارامتر باشند.

#### مثال ۹-۷

```

ADD_W MACRO T1, T2, SUM
 MOV AX , T1
 ADD AX, T2
 MOV SUM, AX
 ENDM

```

ماکرو داده شده دارای سه پارامتر بنامهای T1, T2, SUM می‌باشد. در موقع استفاده از این ماکرو بایستی سه مقدار را بعنوان جایگزینی این پارامترها بدهیم.  
اگر بنویسیم

```
ADD_W PRICE, TAX, COST
```

دستورالعملهای ذیل به جای این دستورالعمل قرار می‌گیرد.



```
MOV AX, PRICE
ADD AX, TAX
MOV COST, AX
```

چنانچه تعداد پارامترها در فراخوانی ماکرو کمتر از تعداد پارامترهای ماکرو باشد ایجاد خطا می نماید.

#### مثال ۱۰-۷

ماکرو EXCHG داده شده در ذیل دو مقدار از نوع word را جابه جا می نماید. لازم به توضیح است که به منظور جلوگیری از پاک شدن مقدار قبلی ثبات AX از دستورالعملهای PUSH و POP استفاده گردیده است.

```
EXCHG MACRO W1, W2
PUSH AX
MOV AX, W1
XCHG AX, W2
MOV W1, AX
POP AX
ENDM
```

حال اگر ماکرو را بصورت زیر فراخوانی نمائیم.

```
EXCHG X , Y
```

دستورالعملهای ذیل به جای آن در source برنامه قرار می گیرد.

```

PUSH AX
MOV AX, X
XCHG AX, Y
MOV X, AX
PUSH AX

```

### ۱-۳-۷- دیرکتیوها Macro directives

در Macro Assembler می‌توان از directive های متعددی در ماکروها

استفاده نمود که بشرح زیر می‌باشند.

```

LOCAL directive
EXITM directive
IFB directive
IFNB directive
IRP directive
IRPC directive
REPT directive

```

در ذیل هر یک از موارد بالا مورد بحث قرار می‌گیرد.

شکل کلی IFB بصورت زیر می‌باشد. IFB مخفف IF BLANK می‌باشد.

```

IFB < argument >
:
ENDIF

```

یا

```

IFB < argument >
:
ELSE
:
ENDIF

```

IFB، درست است اگر، مقدار argument در موقع فراخوانی ماکرو blank یا خالی باشد.

مثال ۷-۱۱

```
ADD_B MACRO N1, N2, N3, N4
MOV AL, N1
IFB <N2>
ADD AL, 20
ELSE
```

```

ADD AL, N2
ENDIF
IFB < N3 >
ADD AL, 30
ELSE
ADD AL, N3
ENDIF
IFB < N4 >
ADD AL, 5
ELSE
ADD AL, N4
ENDIF
ENDM

```

حال اگر ماکرو فوق را بصورت زیر فراخوانی نمائیم.

```
ADD_B NUM1, , , NUM4
```

دستورالعمل ذیل جایگزین می گردد.

```

MOV AL, NUM1
ADD AL, 20
ADD AL, 30
ADD AL, NUM4

```

به جای دستورالعمل ADD \_ B NUM1, NUM2 دستورالعملهای ذیل

قرار می گیرد.

```

MOV AL, NUM1
ADD AL, NUM2
ADD AL, 30
ADD AL, 5

```

شکل کلی IFNB بصورت زیر می باشد. IFNB درست است  
اگر، مقدار argument در موقع فراوانی ماکرو blank نباشد. IFNB به معنی  
IF NOT BLANK می باشد.

```
IFNB < argument >
:
ELSE
:
ENDIF
```

یا

```
IFNB < argument >
:
ENDIF
```

مثال ۷-۱۲

```
ADD_B MACRO X1, X2, X3, X4
 PUSH AX
 MOV AX, X1
 IFNB < X2 >
 ADD AX, X2
 ENDIF
 IFNB < X3 >
 ADD AX, X3
 ENDIF
 IFNB < X4 >
 ADD AX, X4
 ENDIF
ENDM
```

حال اگر دستورالعمل زیر را بنویسیم.

ADD\_B     N1, , N3, N4

دستورالعملهای ذیل جایگزین می گردد.

```
PUSH AX
MOV AX, N1
ADD AX, N3
ADD AX, N4
```

## ۷-۳-۲- دستورالعمل EXITM

شکل کلی دستورالعمل EXITM بصورت زیر می باشد:

EXITM

از EXITM فقط در ماکروها استفاده می گردد. EXITM باعث توقف بسط ماکرو می شود بسته به نتایجی که بوسیله directive های شرطی (IF) بوجود می آید.

## مثال ۷-۱۳

```
IFB < NUM1 >
EXITM
ENDIF
```

چنانچه در موقع فراخوانی ماکرو مقدار NUM1 داده نشود بسط ماکرو متوقف می شود. و از کلیه دستورالعملهایی که در ماکرو بعد از EXITM داده شده صرف نظر می گردد.

### ۷-۳-۳ دستورالعمل IRP

شکل کلی دستورالعمل IRP بصورت زیر می باشد.

```
IRP dummy, < argument-list >
 :
 :
ENDM
```

به تعداد آرگومان‌هایی که در < argument -list > وجود دارد دستورالعمل‌های بین IRP و ENDM تکرار خواهد شد. در هر بار تکرار بترتیب به جای dummy مقادیر داخل < argument-list > قرار داده می شود.

مثال ۷-۱۴

```
IRP VALUE, <1,2,3,5,11,13,17,19,23>
DW VALU*VALUE*VALUE
ENDM
```

ایجاد یک آرایه 9 عنصری از نوع word با مقادیر مکعب نه عدد اول می نماید. یعنی با مقادیر 1, 8, 27, 125, 1331, ..., 12167.

|       |
|-------|
|       |
| 1     |
| 8     |
| 27    |
| 125   |
| 1331  |
| 2197  |
| 4913  |
| 6859  |
| 12167 |
|       |

در حقیقت ایجاد دستورالعملهای ذیل می نماید.

```
DW 1 * 1 * 1
DW 2 * 2 * 2
DW 3 * 3 * 3
DW 5 * 5 * 5
DW 11 * 11 * 11
DW 13 * 13 * 13
DW 17 * 17 * 17
DW 19 * 19 * 19
DW 23 * 23 * 23
```

#### ۴-۳-۷- دستورالعمل IRPC

شکل کلی دستورالعمل IRPC بصورت زیر می باشد.

```
IRPC dummy , string
 ⋮
 ENDM
```

دستورالعملهای بین IRPC و ENDM به تعداد کارکترهای رشته string تکرار می گردد. در هر دفعه تکرار دستورالعملها به جای dummy بترتیب یکی از کارکترهای رشته string جایگزین می گردد. شبیه IRP می باشد با این تفاوت که آرگومانهایش بجای اعداد، متغیرهای رشته ای هستند.

مثال ۵-۱۷

```
IRPC CHAR, 0123456789
 DB CHAR
 ENDM
```

ایجاد یک رشته ده بیتی می نماید با مقادیر کد اسکی برای ارقام 0 تا 9.



|    |
|----|
|    |
| 48 |
| 49 |
| 50 |
| 51 |
| 52 |
| 53 |
| 54 |
| 55 |
| 56 |
| 57 |
|    |
|    |

در حقیقت ایجاد دستورالعملهای زیر می‌نماید.

```
DB '0'
DB '1'
DB '2'
DB '3'
DB '4'
DB '5'
DB '6'
DB '7'
DB '8'
DB '9'
```

#### ۵-۳-۷- دستورالعمل REPT

شکل کلی دستورالعمل REPT بصورت زیر می‌باشد.

```
REPT expression
:
ENDM
```

دستورالعملهای بین REPT و ENDM را به تعداد دفعاتی که توسط expression مشخص می‌گردد تکرار می‌کند.

مثال ۷-۱۶

```

ALLOCATE MACRO TLABEL , LENGTH
 TLABEL EQU THIS BYTE
 VALU = 0
 REPT LENGTH VALUE = VALUE + 1
 DB VALUE

ENDM

ENDM

```

دقت کنید که در اینجا دو تا ENDM وجود دارد. یکی انتهای REPT و دیگری انتهای MACRO را مشخص می‌نماید. این ماکرو LENGTH بایت حافظه را تخصیص می‌دهد و مقادیر آنرا بترتیب 1 تا LENGTH قرار می‌دهد. پس از تعریف نمودن ماکرو ALLOCATE، می‌توان از این ماکرو با استفاده از دستورالعملهای زیر یک آرایه 40 بایتی بنام TABLE 1 ایجاد نمود.

```

DATA_SEG GSEGMENT PARA DATA 'DATA'
ALLOCATE TABLE1 , 40
DATA_SEG ENDS

```

از دیرکتیو EQU برای تخصیص ساده اسمی به اعداد، آدرسهای ترکیبی پیچیده و ... استفاده می‌گردد.

مثال ۷-۱۷

```
K EQU 1024
```

در برنامه هر جا از نام K استفاده می‌گردد به جای آن 1024 قرار داده می‌شود.

مثال ۷-۱۸

SPEED EQU RATE

در برنامه هر جا از کلمه SPEED استفاده می‌شود به جای آن RATE قرار داده می‌شود.

مثال ۷-۱۹

TABLE EQU DS:[BP] [SI]

در برنامه هر جا از نام TABLE استفاده گردیده باشد به جای آن DS:[BP][SI] جایگزین می‌گردد.

مثال ۷-۲۰

COUNT EQU CX

هر جا از نام COUNT در برنامه استفاده شده باشد به جای آن ثبات CX قرار می‌دهد.

مثال ۷-۲۱

DBL\_SPEED EQU 2\*SPEED

در برنامه هر جا از نام DBL\_SPEED استفاده شده باشد به جای آن عبارت  $2 * SPEED$  قرار می‌گیرد.

مثال ۷-۲۲

MINS\_DAY EQU 60\*24

در برنامه هر جا از نام MINS\_DAY استفاده شده باشد به جای آن  
 $60 * 24$  قرار می گیرد.  
 دیرکتیو = مانند EQU می باشد با این تفاوت که می توان آنرا مجدداً تعریف  
 نمود یا به مقدار قبلی آن ارجاع نمود.

مثال ۷-۲۳

CONST=56      معادل EQU می باشد ;  
 CONST=75      مقدار آنرا مجدداً تعریف می نماید ;  
 CONST=CONST      به مقدار قبلی مراجعه می نماید ;  
                          +1

### ۷-۳-۶- دیرکتیو LOCAL

شکل کلی دیرکتیو LOCAL بصورت زیر می باشد.

LOCAL dummy-list

این دیرکتیو باعث می شود که اسمبلر برای هر عنصر در dummy-list  
 ایجاد یک سیمبل منحصر بفرد بنماید.  
 در حقیقت در ماکروها وقتی از label یا برچسب استفاده می نمائیم بسط آنها  
 دچار اشکال می گردد. بعنوان مثال ماکرو زیر را در نظر بگیرید.

```
 WAIT MACRO COUNT
 PUSH CX
 MOV CX , COUNT
NEXT : LOOP NEXT
 POP CX
 ENDM
```

حال اگر در برنامه سه بار از این macro استفاده گردد.

```
 :
 WAIT COUNT1
 :
 WAIT COUNT2
 :
 WAIT COUNT3
 :
```

و ماکروها را بسط دهیم برنامه به صورت زیر نمایان می گردد.

```
 :
 PUSH CX
 MOV CX, COUNT 1
NEXT : LOOP NEXT
 POP CX
 :
 PUSH CX
 MOV CX, COUNT 2
ENXT : LOOP NEXT
 POP CX
 :
 PUSH CX
 MOV CX , COUNT 3
NEXT : LOOP NEXT
 POP CX
 :
```

همانطوریکه مشاهده می‌نمائید دارای سه label یا آدرس NEXT می‌باشد که ایجاد ابهام می‌نماید. برای رفع این اشکال از دیرکتیو LOCAL استفاده می‌گردد.

```

 WAIT MACRO COUNT
LOCAL NEXT
 PUSH CX
 MOV CX , COUNT
NEXT : LOOP NEXT
 POP CX
ENDM

```

حال اگر در برنامه داده شده ماکرو WAIT را بسط دهیم برنامه بصورت زیر در می‌آید.

```

 :
 PUSH CX
 MOV CX , COUNT
NEXT00 : LOOP NEXT00
 POP CX

 :
 PUSH CX
 MOV CX , COUNT
NEXT01: LOOP NEXT01
 POP CX

 :
 PUSH CX
 MOV CX , COUNT
NEXT02: LOOP NEXT02
 POP CX

 :

```

بایستی توجه داشت که در تعریف یک ماکرو می‌توان ماکروهای مختلفی را فراخوانی نمود. در صورتیکه در تعریف یک ماکرو خود آن ماکرو را فراخوانی نمائیم ماکرو را بازگشتی یا recursive macro نامیده میشود.

## مثال ۲۴-۷

```

ADD_W MACRO N1, N2, N3, N4, N5
 IFB < N1 >
 MOV AX, 0 ;; initialize sum
 ELSE
 ADD_W N2, N3, N4, N5
 ADD AX, N1
 ENDIF
ENDM

```

در تعریف بالا چنانچه آرگومانی وجود نداشته باشد، مجموع برابر صفر و در غیر اینصورت مجموع تمام آرگومانها به جزء آرگومان اولی را محاسبه نموده و سپس اولین آرگومان به آن اضافه می‌نمائیم.

همانطوریکه قبلاً متذکر شدیم ماکروهائی که در یک برنامه مورد استفاده قرار می‌گیرند بایستی در ابتدای برنامه تعریف نمود. از طرف دیگر می‌توان یک macro library ایجاد نمود. macro library یک فایل روی دیسک می‌باشد بنام MACRO.LIB که شامل تمام تعاریف ماکروهائی مورد استفاده در برنامه‌های مختلفه می‌باشد. چنانچه از macro library بخواهیم استفاده نمائیم کافی است که دستورالعمل زیر را به عنوان اولین دستورالعمل برنامه قرار دهیم و دیگر نیازی به تعریف ماکروهائی مورد نیاز برنامه در ابتدای برنامه نمی‌باشد.

```
INCLUDE MACRO.LIB
```

چون فایل MACRO.LIB ممکن است دارای ماکروهائی باشد که مورد نیاز برنامه‌ای که می‌خواهد اجرا شود نباشد برای جلوگیری از اتلاف فضای حافظه می‌توان ماکروهائی غیر ضروری برای برنامه را در دستور PURGE قرار داد تا آنها را به حافظه منتقل نکند.

مثال ۷-۲۵

```
INCLUDE MACRO.LIB
PURGE SHOW, CLS, LOCATE
```

ماکروهای SHOW, CLS, LOCATE به حافظه منتقل نمی گردند چون در برنامه این ماکروها فراخوانی نشده اند.

### ۷-۳-۷- عملگرهای ماکرو

دو عملگر ماکرو در Macro Assembler عبارتند از

```
&
;;
```

شکل کلی عملگر ;; عبارتست از

```
;; comment
```

عملگر ;; باعث می شود که اسمبلر comments ها را در موقع بسط ماکرو حذف نماید.

برنامه های بدون comments فضای کمتری را اشغال نموده و سریعتر اجرا می شوند. چنانچه بخواهید comment ای را در موقع بسط ماکرو حذف نکنید بایستی از ; استفاده گردد.

```
; comment
```

### ۷-۳-۸- عملگر &

شکل کلی عملگر & بصورت زیر می باشد.

```
text & text
```

این عملگر دو text یا سیمبل را بهم وصل نموده در کنار یکدیگر قرار می دهد.



مثال ۲۶-۷

```
DEF_TABLE MACRO SUFFIX, LENGTH
TABLE &SUFFIX DB LENGTH DUP (?)
ENDM
```

حال اگر ماکرو فوق را به صورت زیر فراخوانی نمائیم.

```
DEF_TABLE A, 5
```

آنگاه اسمبلر آنرا به TABLEA DB 5 DUP(2) تبدیل می نماید.

ماکرو زیر باعث نمایش یک کاراکتر روی صفحه مانیتور می شود.

```
SHOW MACRO character
;; Display the specified character.
PUSH AX
```

```
PUSH DX
```

```
MOV AH,2 ;; select, display option
```

```
MOV DL, 'character'
```

```
INT 21H ;; call type 21 interrupt
```

```
POP DX
```

```
POP AX
```

```
ENDM
```

به منظور نمایش کارکتر \* روی صفحه مانیتور از دستورالعملهای ذیل

استفاده می گردد:

```
MOV AH, 2
MOV DL, '*'
INT 21H
```

#### ۴-۷- وقفه‌ها (Interrupts)

ریزپردازنده در یک کامپیوتر برنامه‌ها را بسادگی اجرا نمی‌کند. بلکه بعنوان تنظیم کننده سیستم درگیر چیزهائی میشود که اتفاق می‌افتند. بعنوان مثال وقتی که کلیدی روی صفحه کلید فشار می‌دهید ریزپردازنده بایستی دریابد که کدام کلید فشار داده شده و عمل مناسب آن کلید را انجام دهد. بعنوان مثال وقتی کلید Ctrl-Break را فشار می‌دهید عملی که بایستی انجام شود کاملاً متفاوت است با وقتی که کلید T را فشار می‌دهید. وقتی که data از disk به حافظه یا بالعکس منتقل می‌گردد این مسئولیت ریزپردازنده است که دستورالعملهای مناسب برای اینکار را اجرا نماید. همانطوریکه گفته شد ریزپردازنده در کلیه کارهای کامپیوتر نقشی ایفاء می‌نماید. حال سوالی که مطرح می‌شود اینست که ریزپردازنده چگونه با وسائل جانبی درگیر می‌گردد؟ حقیقت امر این است که ریزپردازنده‌ها و وسائل جانبی بطرق مختلف با هم ارتباط برقرار می‌کنند. کاری که انجام می‌شود بدین صورت است که ریزپردازنده شروع به اجرای برنامه می‌نماید و به اجرای برنامه ادامه می‌دهد تا زمانی که یک وسیله جانبی مانند صفحه کلید، دیسک، یا مانیتور به ریزپردازنده اعلام نماید که به کمک ریزپردازنده نیاز دارد. البته وسائل جانبی در حقیقت با ریزپردازنده صحبت نمی‌کنند بلکه آنها سیگنال نیاز به کمک خود را از طریق وقفه یا Interrupt ارسال می‌نمایند.

#### ۱-۴-۷- نحوه کار وقفه‌ها

وقتی که یک وسائل جانبی اقدام به ارسال سیگنال وقفه می‌نماید شماره شناسائی خود را که type code نامیده می‌شود نیز ارسال می‌کند. هر وسیله جانبی

از قبیل keyboard ، disk drive ، floppy drive و ... دارای type code مختلفی می باشد. در حقیقت 256 نوع مختلف type code وجود دارد. هر وقت درخواست کمکی از طرف یک وسیله جانبی شود (Interrupt) ریزپردازنده اگر کاری که در حال انجام آن می باشد بتواند موقتاً رها نماید، اینکار را انجام داده و به کمک وسیله جانبی می رود (البته با حفظ موقعیت فعلی). پس از تکمیل کار وسیله جانبی مجدداً کار قبلی خود را از سر می گیرد. در صورتیکه ریزپردازنده نتواند کار فعلی خود را رها نماید پس از تکمیل این کار به کمک وسیله جانبی می رود.

همانطور که متذکر شدیم 256 تا type code وجود دارد به شماره های 255 تا 0. ریزپردازنده از این type code استفاده نموده آدرسی را در ابتدای حافظه محاسبه نموده و از آدرس محاسبه شده آدرس دیگری را می خواند. این آدرس جدید Interrupt Vector نام دارد که در حقیقت آدرس برنامه ای است که کار آن وقفه را عهده دار می باشد. در مورد وسیله جانبی استاندارد، برنامه های سرویس دهنده وقفه ها (Interrupt servicing programs) در تراشه ROM ذخیره می گردد. در بسیاری از کامپیوترها این ROM به Basic Input/Output system یا BIOS معروفست.

حال که ریزپردازنده می داند کدام برنامه را بایستی اجرا کند این کار را انجام داده یعنی برنامه را اجرا نموده و پس از اتمام اجرای برنامه به کار قبلی خود برگشته و اجرای آن را دنبال می کند.

## ۲-۴-۷- منابع وقفه ها

وقفه هایی که تاکنون بیان شده است وقفه های خارجی (External Interrupt) می باشد که توسط وسایل یا تجهیزات جانبی فعال می گردند. این نوع وقفه ها قسمتی از 256 نوع وقفه را پوشش می دهد. سایرین می توانند یکی از دو نوع وقفه ذیل باشند.

۱- برنامه‌ها همچنین می‌توانند با استفاده از دستورالعمل‌های خاص ایجاد وقفه در برنامه، وقفه‌ها را فعال نمایند.

۲- در موارد خاص ریزپردازنده حتی می‌تواند به خودش وقفه بدهد. بعنوان مثال وقتی که شما سعی در تقسیم بر صفر دارید.

### ۳-۴-۷- وقفه‌های رزرو شده (Reserved Interrupts)

از 256 وقفه، 32 تای اول یعنی شماره 0 تا 31 بوسیله Intel رزرو گردیده است. وقفه‌های نوع 32 تا 255 برای موارد دیگر استفاده می‌گردند. نمونه‌ای از وقفه‌های رزرو شده در ذیل داده شده است:

|          |                                     |
|----------|-------------------------------------|
| Type 0,  | Divide Error                        |
| Type 1,  | Single – Step                       |
| Type 2,  | Nonmaskable Interrupts              |
| Type 3,  | Breakpoint                          |
| Type 4,  | Overflow                            |
| Type 5,  | Bound Range Exceeded                |
| Type 6,  | Invalid Table Limit Too Small       |
| Type 7,  | Processor Extension Not Available   |
| Type 8,  | Interrupt Table Limit Too Small     |
| Type 9,  | Processor Extension Segment Overrun |
| Type 13, | Segment Overrun                     |
| Type 16  | Processor Extension Error           |

### ۴-۴-۷- وقفه‌های سیستم

در کامپیوترها، 1024 بایت اول حافظه یعنی محل‌های حافظه با آدرس 0 تا 3FF تخصیص به جدولی دارد که این جدول بنام interrupt vector table معروفست. این جدولی است با آدرس‌های 32 بیتی که به interrupt service routines در کامپیوتر اشاره می‌کنند. 256 وقفه مختلف به شماره‌های 0 تا 255 یا FF تا 0 در مبنای شانزده وجود دارد. ریزپردازنده Intel ، 32 وقفه اول یعنی

وقفه‌های شماره 1FH تا 0 را برای استفاده خودش در نظر می‌گیرد. 32 وقفه بعدی یعنی شماره‌های 3FH تا 20H برای استفاده سیستم عامل DOS در نظر گرفته شده است.

وقفه از طریق دستورالعمل‌های وقفه در برنامه یا تجهیزات خارجی (external devices) در سیستم فعال می‌گردد.

وقتی که ریزپردازنده یک وقفه دریافت می‌نماید شماره وقفه را در 4 ضرب نموده تا آدرس interrupt vector در جدول را بدست آورده سپس محتوی آدرس بدست آمده را در ثبات IP و ثبات CS قرار می‌دهد و شروع به اجرای دستورالعملها در آن آدرس می‌نماید. بعنوان مثال اگر وقفه از نوع 4AH باشد.

$$4AH * 4 = 128H$$

محلی از حافظه که آدرس آن 128H می‌باشد شامل آدرس Interrupt service routine وقفه نوع 4AH می‌باشد.

#### ۵-۴-۷- وقفه‌های DOS

همانطور که قبلاً متذکر شدیم وقفه‌های نوع 3FH تا 20H برای سیستم عامل DOS رزرو گردیده است. اکثر این وقفه‌ها در سیستم عامل DOS کاربرد دارند بنابراین در اینجا آنها را مورد بحث قرار نمی‌دهیم. فقط دقت داشته باشید که نوع 21 دارای گزینه‌هایی برای برقراری ارتباط با keyboard، display، printer، disk و وسائل ارتباطی غیر همزمان (asynchronous communications device) می‌باشد که در بخش بعدی این فصل مورد بحث قرار می‌گیرد.

| شماره وقفه | نام                          |
|------------|------------------------------|
| 20         | Terminate Program            |
| 21         | Function Calls               |
| 22         | Terminate                    |
| 23         | Ctrl-Break Exit Address      |
| 24         | Critical Error Handler       |
| 25         | Absolute Disk Read           |
| 26         | Absolute Disk Write          |
| 27         | Terminate, But Stay Resident |
| 28         | Reserved for Dos             |

#### ۶-۴-۷- دستورالعملهای وقفه

در زبان اسمبلی در مورد وقفه‌های برنامه می‌توان از دستورالعملهای ذیل

استفاده نمود:

INT  
INTO  
IRET

شکل کلی دستورالعمل INT بصورت زیر است:

INT Interrupt-type

الف) interrupt – type یکی از نوع‌های وقفه 0 تا 255 می‌باشد.

ب) مقدار فلگهای IF و TF را صفر قرار می‌دهد. و روی سائز فلگها اثر ندارد.

ج) محتوی ثابت فلگ (flag register) را وارد stack می‌نماید.

د) محتوی ثابت CS را وارد stack می‌نماید.

ه) آدرس interrupt vector را محاسبه می‌نماید (شماره وقفه را در 4 ضرب می‌کند).

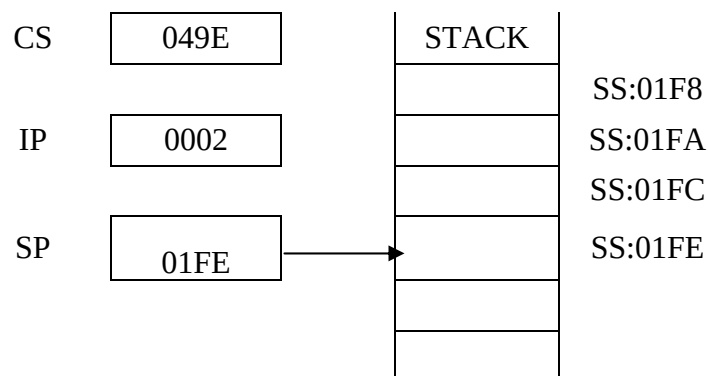
و) محتوی ثابت IP را وارد stack می‌نماید.

ز) محتوی word اول interrupt vector را در ثبات IP و محتوی word دوم آنرا در ثبات CS قرار می‌دهد.

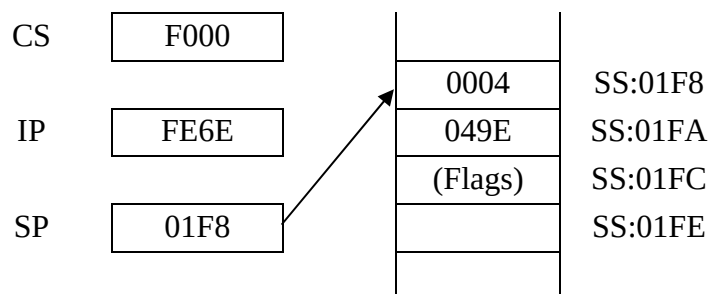
مثال ۲۷-۷

INT 1 AH

مقادیر ثباتها و پشته قبل از اجرای دستور فوق:



مقادیر ثباتها و پشته بعد از اجرای دستور فوق:



|      |      | Interrupt Vectors |
|------|------|-------------------|
| 19 { | 0064 |                   |
|      | 0066 |                   |
| 1A { | 0068 | FE6E              |
|      | 006A | F000              |
| 1B { | 006C |                   |
|      | 006E |                   |
|      |      |                   |
|      |      |                   |

شکل کلی دستورالعمل INTO بصورت زیر می باشد. این دستورالعمل یک دستورالعمل وقفه شرطی است و مخفف کلمات Interrupt If Overflow می باشد.

#### INTO

- الف) دارای عملوند نمی باشد.
- ب) مقادیر فلگهای TF و IF را صفر می کند.
- ج) این دستورالعمل ایجاد وقفه می نماید اگر مقدار فلگ OF برابر یک باشد.
- د) این دستورالعمل وقفه نوع 4 را فعال می نماید.
- شکل کلی دستورالعمل IRET بصورت زیر می باشد. IRET مخفف Interrupt Return می باشد.

#### IRET

- الف) روی تمام فلگها اثر دارد.
- ب) این دستورالعمل دارای عملوند نمی باشد.
- ج) این دستورالعمل برای وقفه ها همان کاری را انجام می دهد که RET برای procedure call. یعنی باعث برگشت کنترل به برنامه اصلی می شود.



به‌همین دلیل IRET بایستی آخرین دستورالعملی باشد که ریزپردازنده در یک Interrupt Service Routine اجرا می‌نماید.

د) IRET باعث می‌شود که سه مقدار 16 بیتی از پشته خارج شده و بترتیب در ثباتهای Flag register, CS, IP قرار گیرد.

ه) مقادیر سایر ثباتها ممکن است از بین برود مگر اینکه Interrupt service routine صریحاً آنها را ذخیره نماید.

#### ۷-۴-۷- فراخوانی تابع وقفه نوع 21

در صفحه بعد قسمتی از فراخوانی‌های تابع وقفه نوع 21H یعنی Type 21H function calls برای سیستم عامل DOS داده شده است.

## جدول ۷-۱

**Function calls with the Type 21 interrupt.**

| AH                                           | Operation                                 | Input Values                                           | Results*                                                                                |
|----------------------------------------------|-------------------------------------------|--------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <b>Asynchronous Communications Functions</b> |                                           |                                                        |                                                                                         |
| 3                                            | Wait for Asynchronous Input character     | None                                                   | AL=Character                                                                            |
| 4                                            | Output a character to asynchronous device | DL = Character                                         | None                                                                                    |
| <b>File Management Functions</b>             |                                           |                                                        |                                                                                         |
| D                                            | Reset default disk drive                  | None                                                   | None                                                                                    |
| E                                            | Select default disk drive                 | DL=Drive Number (0=A,1=B, 2 = C)                       | AL=Number of disk drive (2 for single drive)                                            |
| 19                                           | Get default drive code                    | None                                                   | AL=Default drive code ( 0 = A, 1 = B, 2=C)                                              |
| 2E                                           | Set verify state (See also function 54)   | DL=0<br>AL=0 to turn verify off<br>=1 t turn verify on | None                                                                                    |
| 30                                           | Get DOS version number                    | None                                                   | AL=Version number (3 or 2)<br>(Version 1.x returns 0)<br>AH= Revision number<br>BX,CX=0 |

**Function calls with the Type 21 interrupt (continued).**

| AH                                | Operation                     | Input Values                                                                | Results*                            |
|-----------------------------------|-------------------------------|-----------------------------------------------------------------------------|-------------------------------------|
| <b>Interrupt Vector Functions</b> |                               |                                                                             |                                     |
| 25                                | Set interrupt vector          | DS:DX=Vector address<br>AL =Interrupt number                                | None                                |
| 35                                | Read interrupt vector address | AL =Interrupt number                                                        | ES:BX= Vector address               |
| <b>Directory Functions</b>        |                               |                                                                             |                                     |
| 39                                | Create a directory (MKDIR)    | DS:DX=Address Of ASCIIZ String for directory                                | None **                             |
| 3A                                | Remove a directory (RMDIR)    | DS:DX=Address Of ASCIIZ String for directory                                | None **                             |
| 3B                                | Change the directory (CHDIR)  | DS:DX=Address Of ASCIIZ String for new directory                            | None **                             |
| 47                                | Get current directory         | DL=Drive Number (0 = default, 1=A, etc.)<br>DS:SI=Address Of 64-byte buffer | DS:SI = Address of ASCIIZ string ** |

**Function calls with the Type 21 interrupt (continued).**

| AH                                        | Operation                 | Input Values                                                                                                                   | Results*                                                                                                                       |
|-------------------------------------------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>Extended File Management Functions</b> |                           |                                                                                                                                |                                                                                                                                |
| 36                                        | Get free disk space       | DL=Drive number<br>(0=default, 1=A, etc.)                                                                                      | AX=0FFFFH if invalid<br><br>=Sectors per cluster<br>BX=No. of free clusters<br>DX=Total no. of clusters<br>CX=Bytes per sector |
| 3C                                        | Create a file             | DS:DX=Address Of<br>ASCII string<br>CX=Attribute of file                                                                       | AX=File handle **                                                                                                              |
| 3D                                        | Open a file               | DS:DX=Address of<br>ASCII string<br>AL=0 to open for reading<br>= 1 to open for writing<br>= 2 to open for reading and writing | AX=File handle **                                                                                                              |
| 3E                                        | Close a file handle       | BX=File handle                                                                                                                 | None **                                                                                                                        |
| 3F                                        | Read from file or device  | BX=File handle<br><br>CX=No. of bytes to read<br><br>DS:DX=Buffer address                                                      | AX=No. of bytes read**<br><br>= 0 if read from end of file                                                                     |
| 40                                        | Write to a file or device | BX=File handle                                                                                                                 | AX=No. of bytes written **                                                                                                     |

**Function calls with the Type 21 interrupt (continued).**

| AH                                                    | Operation                                  | Input Values                                                                     | Results*                                     |
|-------------------------------------------------------|--------------------------------------------|----------------------------------------------------------------------------------|----------------------------------------------|
| <b>Extended File Management Functions (continued)</b> |                                            |                                                                                  |                                              |
|                                                       |                                            | CX=No. of bytes to write<br>DS:DX=Buffer address                                 |                                              |
| 41                                                    | Delete a file                              | DS:DX=Address of ASCIIZ string                                                   | None**                                       |
| 43                                                    | Get file attribute                         | AL=0<br>DS:DX=Address of ASCIIZ string<br>for file                               | CX=Attribute**                               |
| 43                                                    | Set file attribute                         | AL=1<br>DS:DX=Address of ASCIIZ string<br>for file                               | None **                                      |
|                                                       |                                            | CX=Attribute                                                                     |                                              |
| 54                                                    | Get verify state (See<br>also function 2E) | None                                                                             | AL=0 if verify is off<br>= 1 if verify is on |
| 56                                                    | Rename a file                              | DS:DX=Address of ASCIIZ string<br>for old name<br>ES:DI=Address of ASCIIZ string | None **                                      |

**Function calls with the Type 21 interrupt (continued).**

| AH                                                    | Operation                     | Input Values                                                                                     | Results*                  |
|-------------------------------------------------------|-------------------------------|--------------------------------------------------------------------------------------------------|---------------------------|
| <b>Extended File Management Functions (continued)</b> |                               |                                                                                                  |                           |
|                                                       | For new<br>Name               |                                                                                                  |                           |
| <b>Process Management functions</b>                   |                               |                                                                                                  |                           |
| 31                                                    | Keep process                  | AL=Return code<br>DX=Memory size, in paragraphs                                                  | None                      |
| 4B                                                    | Load and execute a<br>program | AL=0<br><br>DS:DX=Address of ASCIIZ<br>string for program<br>ES:BX=Address of parameter<br>block | None **                   |
| 4B                                                    | Load overlay                  | AL=3<br>DS:DX=Address of ASCIIZ<br>string for program<br>ES:BX=Address of parameter<br>block     | None **                   |
| 4C                                                    | End process                   | AL=Return code                                                                                   | None                      |
| 4D                                                    | Get return code of child      | None                                                                                             | AX=Return code<br>process |

**Function calls with the Type 21 interrupt (continued).**

| AH                                              | Operation             | Input Values                                                  | Results*                                                                  |
|-------------------------------------------------|-----------------------|---------------------------------------------------------------|---------------------------------------------------------------------------|
| <b>Process Management Functions (continued)</b> |                       |                                                               |                                                                           |
| 62                                              | Get PSP               | None                                                          | BX=Segment address of PSP                                                 |
| <b>Memory Management Functions</b>              |                       |                                                               |                                                                           |
| 48                                              | Allocate memory       | BX=Number of paragraphs requested                             | AX=Segment address of allocated memory**                                  |
| 49                                              | Free allocated memory | ES=Segment address of memory to be freed                      | None **                                                                   |
| 4A                                              | Set block             | BX=Number of paragraphs<br>ES= Segment address of memory area | None **                                                                   |
| <b>Get Extended Error Function</b>              |                       |                                                               |                                                                           |
| 59                                              | Get extended error    | BX=0                                                          | AX= Extended code<br>BH= Error class<br>BL= Suggested action<br>CH= Locus |

\*Besides the registers listed in this column, only AX and the flags are affected.

\*\*If an error occurs, these function calls return CF=1 and an error code in AX. See the following table for the meanings of the error codes.

**Function Call Error Reports**

Most of the Directory and Extended File Management functions return CF= 0 if the operation is successful and CF= 1 if an error occurred. Along with CF=1, they return an error code in AX; the following table tells what these codes mean.

## جدول ۷-۲

Error codes for DOS function calls.

| Code | Meaning                                   |
|------|-------------------------------------------|
| 1    | Invalid function number                   |
| 2    | File not found                            |
| 3    | Path not found                            |
| 4    | Too many open files (no handles left)     |
| 5    | Access denied                             |
| 6    | Invalid handle                            |
| 7    | Memory control blocks destroyed           |
| 8    | Insufficient memory                       |
| 9    | Invalid memory block address              |
| 10   | Invalid environment                       |
| 11   | Invalid format                            |
| 12   | Invalid access code                       |
| 13   | Invalid data                              |
| 15   | Invalid drive was specified               |
| 16   | Attempted to remove the current directory |
| 17   | Not same device                           |
| 18   | No more files                             |
| 19   | Disk is write-protected                   |
| 20   | Bad disk unit                             |
| 21   | Drive not ready                           |
| 22   | Invalid disk command                      |
| 23   | CRC error                                 |
| 24   | Invalid length (during disk operation)    |
| 25   | Seek error                                |
| 26   | Not an MS-DOS disk                        |
| 27   | Sector not found                          |
| 28   | Out of paper                              |
| 29   | Write fault                               |
| 30   | Read fault                                |
| 31   | General failure                           |
| 32   | Sharing violation                         |



**Error codes for DOS function calls (continued).**

| <b>Code</b> | <b>Meaning</b>                      |
|-------------|-------------------------------------|
| 33          | Lock violation                      |
| 34          | Invalid disk change                 |
| 35          | FCB unavailable                     |
| 50          | Network request not supported       |
| 51          | Remote computer not listening       |
| 52          | Duplicate name on network           |
| 53          | Network name not found              |
| 54          | Network busy                        |
| 55          | Network device no longer exists     |
| 56          | Net BIOS command limit exceeded     |
| 57          | Network adapter hardware error      |
| 58          | Incorrect response from network     |
| 59          | Unexpected network error            |
| 60          | Incompatible remote adapt           |
| 61          | Print queue full                    |
| 62          | Queue not full                      |
| 63          | Not enough space for print file     |
| 64          | Network name was deleted            |
| 65          | Access denied                       |
| 66          | Network device type incorrect       |
| 67          | Network name not found              |
| 68          | Network name limit exceeded         |
| 69          | Net BIOS session limit exceeded     |
| 70          | Temporarily paused                  |
| 71          | Network request not accepted        |
| 72          | Print or disk redirection is paused |
| 80          | File exists                         |
| 82          | Cannot make                         |
| 83          | Interrupt 24 failure                |
| 84          | Out of structures                   |
| 85          | Already assigned                    |
| 86          | Invalid password                    |
| 87          | Invalid parameter                   |
| 88          | Net write fault                     |

With DOS Microsoft has also included a Get Extended Error function call (AH=59H) that provides more comprehensive error.

دستورالعملهای ذیل باعث پاک شدن صفحه مانیتور می شود.

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

دستورالعملهای ذیل مکان نما یا Cursor را در سطر 13 ستون 40 قرار می دهد. همانطور که می دانید صفحه مانیتور بصورت بصورت یک صفحه شطرنجی در نظر گرفته می شود که دارای 80 ستون و 25 سطر می باشد. شماره سطرها و ستونها از صفر شروع می شوند.

```
MOV AH, 2
MOV BH, 0
MOV DH, 13 ; row
MOV DL, 40 ; column
INT 10 H
```

دستورالعملهای ذیل یک کارکتر را از صفحه کلید گرفته، آنرا در ثبات AL قرار می دهد و سپس روی صفحه مانیتور نمایش می دهد.

```
MOV AH, 1
INT 21H
```

دستورالعملهای ذیل یک کارکتر را از صفحه کلید گرفته، آنرا در ثبات AL قرار می دهد. (روی صفحه مانیتور نمایش نمی دهد)

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

دستورالعملهای ذیل باعث می شوند که کنترل به سیستم عامل DOS

برگردد.

```
MOV AX, 4C00H
INT 21H
```

دستورالعملهای ذیل باعث می شود که کارکتر A روی صفحه مانیتور  
نمایش داده شود.

```
MOV AH, 02H
MOV DL, 65
INT 21H
```

دستورالعملهای ذیل باعث می شود مکان نما بابتدای سطر بعد منتقل  
گردد.

```
MOV AH, 02H
MOV DL, 0DH
INT 21H
MOV DL, 0AH
INT 21H
```

بایستی توجه داشت که دستور MOV AH, 02H پس از دستور  
INT 21H نیازی نمی باشد.

برنامه ذیل جهت پاسخگوئی Y یا N به یک پیغام می باشد.

```
GET_KEY: MOV AH, 1
 INT 21H

 CMP AL, 'Y'
 JE YES
 CMP AL, 'N'
 JE NO
 JNE GET_KEY
```

در ذیل جدول مربوط به عملیات صفحه کلید برای وقفه نوع 21 داده شده است.

جدول ۷-۳

**Keyboard operations with Type 21 interrupt.**

| AH | Operation                                                                | Input Values                                           | Results                                                              |
|----|--------------------------------------------------------------------------|--------------------------------------------------------|----------------------------------------------------------------------|
| 1  | Wait for keyboard character, then display it (with Ctrl-Break check)     | None                                                   | AL=Character                                                         |
| 6  | Read keyboard character (no Ctrl-Break check)                            | DL=0FFH                                                | AL=Character, if available<br>=0, if no character is available       |
| 7  | Wait for keyboard character, but do not display it (no Ctrl-Break check) | None                                                   | AL=Character                                                         |
| 8  | Same as function 7, but with Ctrl-Break check                            | None                                                   | AL=Character                                                         |
| A  | Read keyboard string into buffer                                         | DS:DX=Buffer address<br>First buffer byte= Buffer size | Second buffer byte<br>=Number of chars. read                         |
| B  | Read keyboard status                                                     | None                                                   | AL=0FFH if no character is available<br>=0 if character is available |
| C  | Clear keyboard buffer and call a keyboard function                       | AL=Keyboard function number (1, 6, 7, 8, or A)         | Per keyboard function                                                |

جدول مربوط به عملیات صفحه نمایش برای وقفه 21 در ذیل داده شده است.

جدول ۷-۴

**Video operations with Type 21 interrupt.**

| AH | Operation                                      | Input Values                                       | Results                  |
|----|------------------------------------------------|----------------------------------------------------|--------------------------|
| 2  | Display a character<br>(with Ctrl-Break check) | DL= Character                                      | Cursor follows character |
| 5  | Print a character                              | DL=Character                                       | None                     |
| 6  | Display a character<br>(no Ctrl-Break check)   | DL=Character                                       | Cursor follows character |
| 9  | Display a string                               | DS:DX= String address.<br>String must end with \$. | Cursor follows string    |

## ۵-۷- خواندن رشته‌ها

در بسیاری از برنامه‌ها نیاز به وارد نمودن نام یا آدرس یا بطور کلی رشته‌ای از طریق صفحه کلید می‌باشد. برای اینکار شما بایستی در data segment فضائی باندازه ماکزیمم تعداد کارکتهائی که ممکن است از صفحه کلید داده شود بعلاوه 2 روزه نمود. این شامل

الف) یک فضا از نوع بایت که مقدار آن معادل تعداد ماکزیمم کارکتهای ورودی بعلاوه یک می‌باشد.

ب) یک فضا از نوع بایت که مقدار آن بوسیله تابع A یا 10H مشخص می‌گردد و عبارتست از تعداد کارکتهای واقعی که از صفحه کلید داده شده است.

ج) یک بلوک از بایتهای بدون مقدار اولیه که شامل کارکتهائی است که عملاً از طریق صفحه کلید وارد گردیده است.

بنابراین شکل کلی رشته بصورت زیر می‌باشد.

Stringname DB key-plus-one, keys-plus-one DUP(?)

بعنوان مثال برای رزرو فضائی تا 50 کارکتر دستورالعمل زیر را بایستی در data segment قرار داد.

USER\_STRING DB 51,51 DUP (?)

حال برای دریافت رشته از صفحه کلید دستورالعملهای زیر را بایستی اجرا نمود.

```
LEA DX, USER_STRING; make DX point to buffer
MOV AH, 0AH; read the string
INT 21H
```

برنامه کامل آن بصورت زیر می باشد.

```
READ_KEYS PROC FAR
 PUSH AX
 MOV AX, DATA_SEG
 MOV DS, AX
 LEA DS, AX
 LEA DX, USER_STRING
 MOV AH, 0AH
 INT 21H
 SUB CH, CH; Read Character count into CX
 MOV CL, USER_STRING+1
 ADD DX, 2; Make DX point to text
 POP AX
 RET
READ_KEYS ENDP
```

در بسیاری از موارد در برنامه ها پیغامی نمایش داده میشود و سپس از کاربر انتظار پاسخگوئی می باشد. بعنوان مثال شما از کاربر می خواهید که نامش را وارد نماید. برای اینکار ابتدا پیغامی که می خواهید روی صفحه مانیتور نمایش داده شود را در Data segment تعریف می نمائید. سپس با استفاده از روال READ\_KEYS داده شده در بالا اینکار را انجام می دهید.

مثال ۲۸-۷

```
DATA_SEG PARA DATA 'DATA'
GET_NAME DB 'Please enter your name: $'
DATA_SEG ENDS
```

و سپس در code segment دستورالعملهای ذیل را بایستی بدهیم.

```

LEA DX, GET_NAME; display the prompt
MOV AH, 9
INT 21h
CALL READ_KEYS; Read the response

```

بایستی توجه داشت که DS:DX اشاره می کند به رشته ای که شامل

نام می باشد و محتوی CX برابر با تعداد کاراکترهای رشته می باشد.

## ۷-۶- عملیات date و time

جدول ذیل مربوط به فراخوانی های تابع وقفه نوع 21 در زمینه زمان

و تاریخ می باشد.

### جدول ۷-۵

| AH                                                    | Operation | Input Values                                                    | Results *                                                |
|-------------------------------------------------------|-----------|-----------------------------------------------------------------|----------------------------------------------------------|
| <b>Note :</b> All time and date values are in binary. |           |                                                                 |                                                          |
| 2A                                                    | Get date  | None                                                            | CX=Year<br>DH=Month<br>DL=Day                            |
| 2B                                                    | Set date  | CX=Year (1980-2099)<br>DH=Month<br>DL=Day                       | AL=0 if date is valid<br>=FF if date is invalid          |
| 2C                                                    | Get time  | None                                                            | CH=Hours<br>CL=Minutes<br>DH=Seconds<br>DL=1/100 seconds |
| 2D                                                    | Set time  | CH=Hours (0-23)<br>CL=Minutes<br>DH=Seconds<br>DL=1/100 Seconds | AL=0 if time is valid<br>= FF if time is invalid         |

\* Besides the registers listed in this column, only AX and the flags are affected.



### ۱-۶-۷- اندازه‌گیری زمان اجرای برنامه‌ها

با توجه باینکه می‌توان زمان را برحسب صدم ثانیه در کامپیوترها مطرح نمائیم از این ویژگی برای اندازه‌گیری زمان اجرای یک برنامه یا قسمتی از یک برنامه استفاده می‌شود. برای اینکار قبل از شروع اجرای برنامه زمان یا time را می‌خوانیم آنگاه برنامه را اجرا نموده سپس زمان یا time را می‌خوانیم. تفاضل دو زمان خوانده شده زمان اجرا را نشان می‌دهد. در ذیل برنامه‌ای داده شده که زمان اجرای یک روال بنام SORT را اندازه می‌گیرد.

#### Calculate execution time.

```
; This sequence calculates the execution time of a program
; Called SORT.
; Results: CH= Hours
; CL= Minutes
; DH= Seconds
; DL= 1/100 Seconds
```

#### ; Put these temporary locations in the data segment:

```
HRS DB ?
MINS DB ?
SECS DB ?
HSECS DB ?
```

#### ; Here is the timing sequence:

```
MOV AH, 2CH ; Read the start time
INT 21H
MOV HRS,CH ; and save it
MOV MINS,CL
MOV SECS,DH
MOV HSECS,DL
CALL SORT ; Execute the procedure
MOV AH, 2CH ; Read the end time
INT 21H
```

```

SUB DL,HSECS : Calculate the difference
JNC SUB_SECS
ADD DL, 100
DEC DH
SUB_SECS: SUB DH, SECS
JNC SUB_MINS
ADD DH, 60
DEC CL
SUB_MINS: SUB CL, MINS
JNC SUB_HRS
ADD CL, 60
DEC CH
SUB_HRS: SUB CH, HRS
JNC DONE
ADD CH, 24
DONE: RET

```

## ۷-۶-۲- ایجاد تأخیر (Generating delays)

در بعضی از برنامه‌ها نیاز به ایجاد تأخیر در تولید صوت از طریق speaker یا در نمایش اشکال گرافیکی روی صفحه نمایش می‌باشد.

برنامه‌هایی که ایجاد تأخیر می‌نمایند بصورت زیر عمل می‌نمایند.

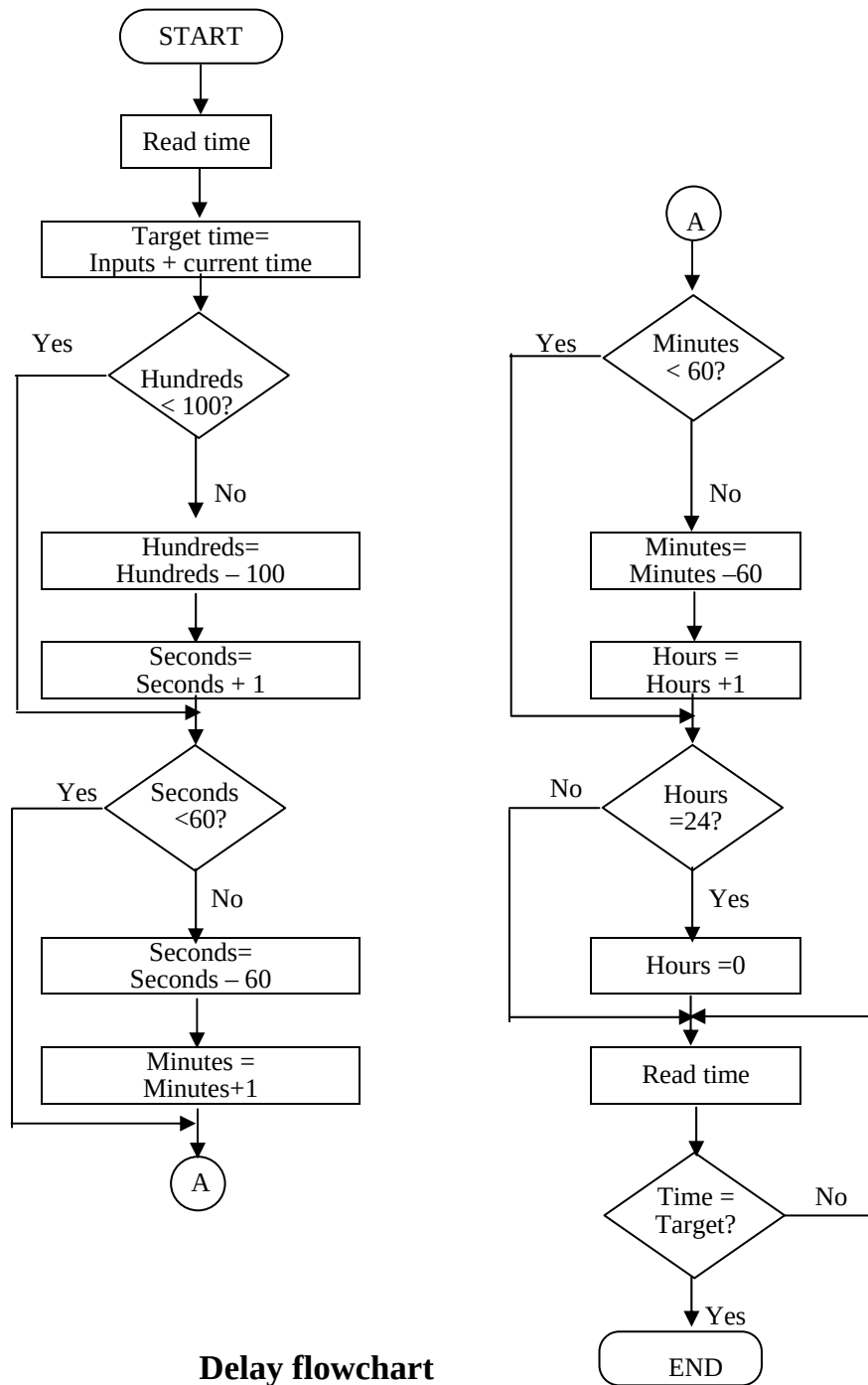
الف) وقت فعلی (Current time) را می‌خوانند.

ب) مقدار تأخیر را به وقت فعلی اضافه نموده تا زمان هدف (target time) بدست آید.

ج) زمان هدف را تنظیم می‌نمایند بطوریکه ساعت آن از 23 و دقیقه و ثانیه آن از 59 بیشتر نشود.

د) زمان یا time را مکرراً می‌خوانند تا زمان فعلی بیشتر از زمان هدف شود.

نمودار ذیل مراحل ایجاد تأخیر را نشان می‌دهد. مدت زمان تأخیر برحسب دقیقه، ثانیه، و صدم ثانیه داده می‌شود.



برنامه ایجاد تأخیر نیز در ذیل داده شده است.

Generate a delay.

```
; Make the processor wait for a specified period.
; Inputs: AL= Minutes
; BH= Seconds
; BL=1/100 Seconds
; All registers are preserved.

; Assemble with: MASM DELAY;
; Link with: LINK callprog + DELAY;

CSEG PUBLIC DELAY
 SEGMENT PARA PUBLIC 'CODE'
 ASSUME CS: CSEG
DELAY PROC FAR
 PUSH AX ; Save affected registers
 PUSH BX
 PUSH CX
 PUSH DX
 MOV AH, 2CH ; Read current time
 INT 21H
; Add the current time to the input values.
 MOV AH, CH ; Hours
 ADD AL, CL ; Minutes
 ADD BH, DH ; Seconds
 ADD BL, DL ; Hundredths
; Propagate any carryover.
 CMP BL, 100 ;Hundredths must be <100
 JB SECS
 SUB BL, 100
 INC BH
SECS: CMP BH, 60 ; Seconds must be <60
 JB MINS
 SUB BH, 60
 INC AL
MINS: CMP AL, 60 ; Minutes must be < 60
 JB HRS
 SUB AL, 60
 INC AH
HRS: CMP AH, 24 ; Hours must be < 24
 JNE CHECK
 SUB AH, AH
; wait for interval to elapse.
CHECK: PUSH AX ; Read the time again
```

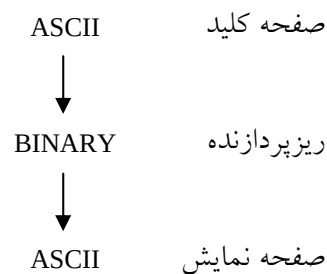
```

 MOV AH, 2CH
 INT 21H
 POP AX
 CMP CX, AX ; Compare hours and minutes
 JA QUIT
 JB CHECK
 CMP DX, BX ; Compare seconds and hunds
 JB CHECK
QUIT: POP DX ; Restore registers
 POP CX
 POP BX
 POP AX
 RET ; Return to calling program
DELAY ENDP
CSEG ENDS
 END

```

## ۷-۷- کدهای اسکی و دودوئی

مقادیر یا اعدادی که در برنامه‌ها بعنوان ورودی از طریق صفحه کلید داده می‌شوند اگر بخواهیم مورد استفاده ریزپردازنده قرار گیرد جهت انجام عملیات ریاضی بایستی به دودوئی یا باینری تبدیل گردند. و چنانچه بخواهیم نتایج محاسبات را روی صفحه نمایشگر یا دستگاه چاپگر به نمایش درآوریم یا چاپ نمائیم بایستی تبدیل به اسکی گردند.



### ۷-۷-۱- تبدیل رشته‌های ASCII به دودویی

همانطوریکه می‌دانیم کارکترهای 0 تا 9 دارای کداسکی 48 تا 57 می‌باشند.

شرح ذیل:

| ASCII Value (Hex) | Decimal Digit |
|-------------------|---------------|
| 30                | 0             |
| 31                | 1             |
| 32                | 2             |
| 33                | 3             |
| 34                | 4             |
| 35                | 5             |
| 36                | 6             |
| 37                | 7             |
| 38                | 8             |
| 39                | 9             |

از طرف دیگر همانطوریکه می‌دانیم هر عدد را بصورت یک سری از توانهای 10 می‌توان نمایش داد.

مثال ۲۹-۷

$$472 = (2 * 1) + (7 * 10) + (2 * 100)$$

یا

$$472 = 2 * 10^2 + 7 * 10^1 + 2 * 10^0$$

و با توجه به آنکه در موقع ورود اعداد، در هر لحظه فقط یک رقم را وارد می‌نمائیم الگوریتم تبدیل بایستی وزن رقم را مدنظر قرار دهد. بعنوان مثال اگر کاربر عدد 95 را تایپ نماید به محض دریافت 9 الگوریتم بایستی آنرا در 10 ضرب نماید قبل از آنکه با عدد 5 جمع نماید. بطور کلی فرآیند تبدیل بایستی بطریق ذیل عمل نماید.

الف) الگوریتم تبدیل بایستی اولین رقم (با ارزشترین) را با حذف چهار بیت مرتبه بالا (بیت‌های 7 تا 4) کد اسکی به دودوئی تبدیل نماید و مقدار باینری بدست آمده را نگهداری نماید.

ب) الگوریتم بایستی ارقام بعدی را به دودوئی تبدیل نموده و نتیجه قبلی به دست آمده در مرحله الف را در 10 ضرب نموده با نتیجه بدست آمده در این مرحله جمع نماید.

### مثال ۳۰-۷

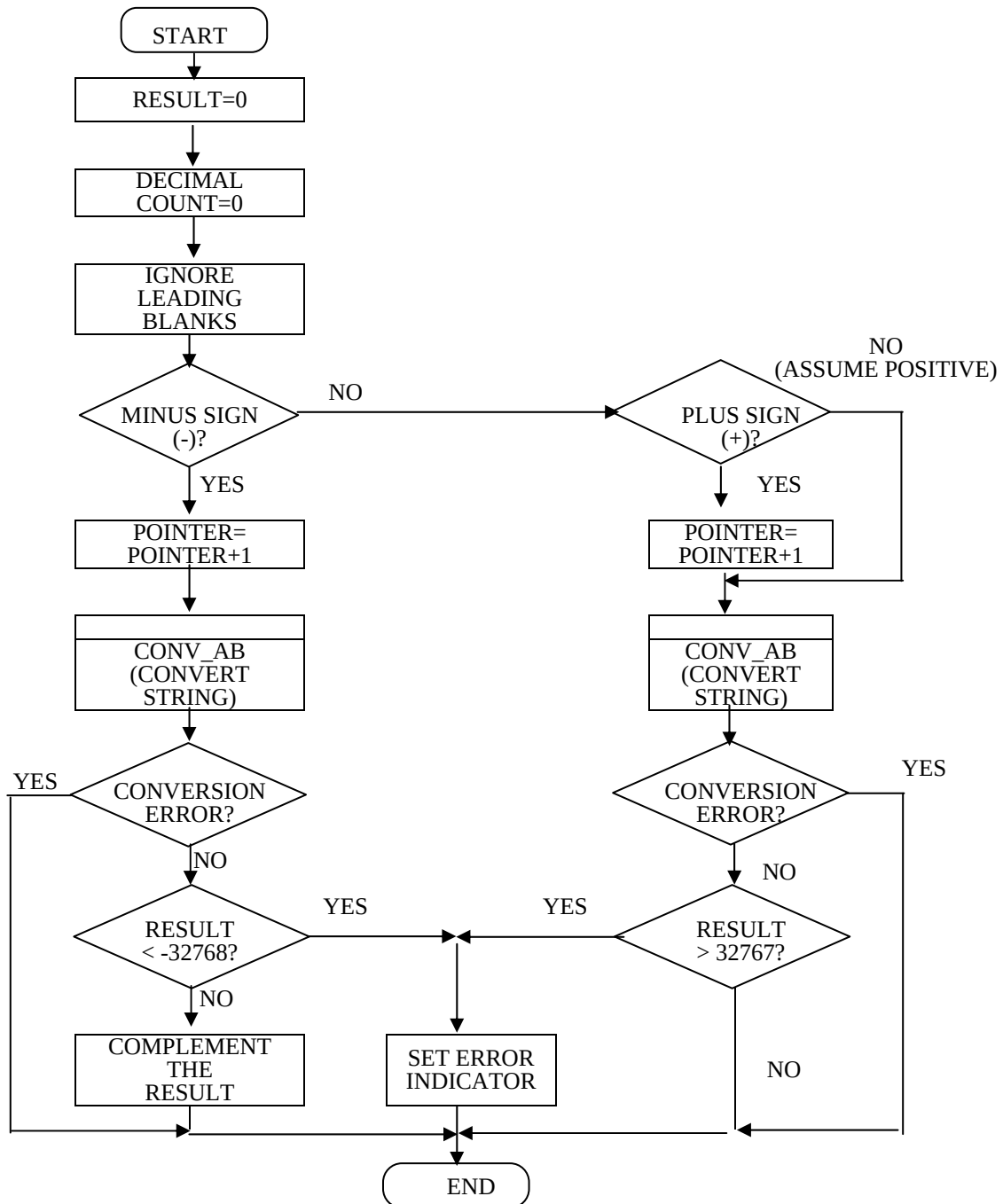
عدد 726 را در نظر بگیرید.

|          |          |          |
|----------|----------|----------|
| 00110111 | 00110010 | 00110110 |
| 7        | 2        | 6        |

حال با توجه به الگوریتم داده شده

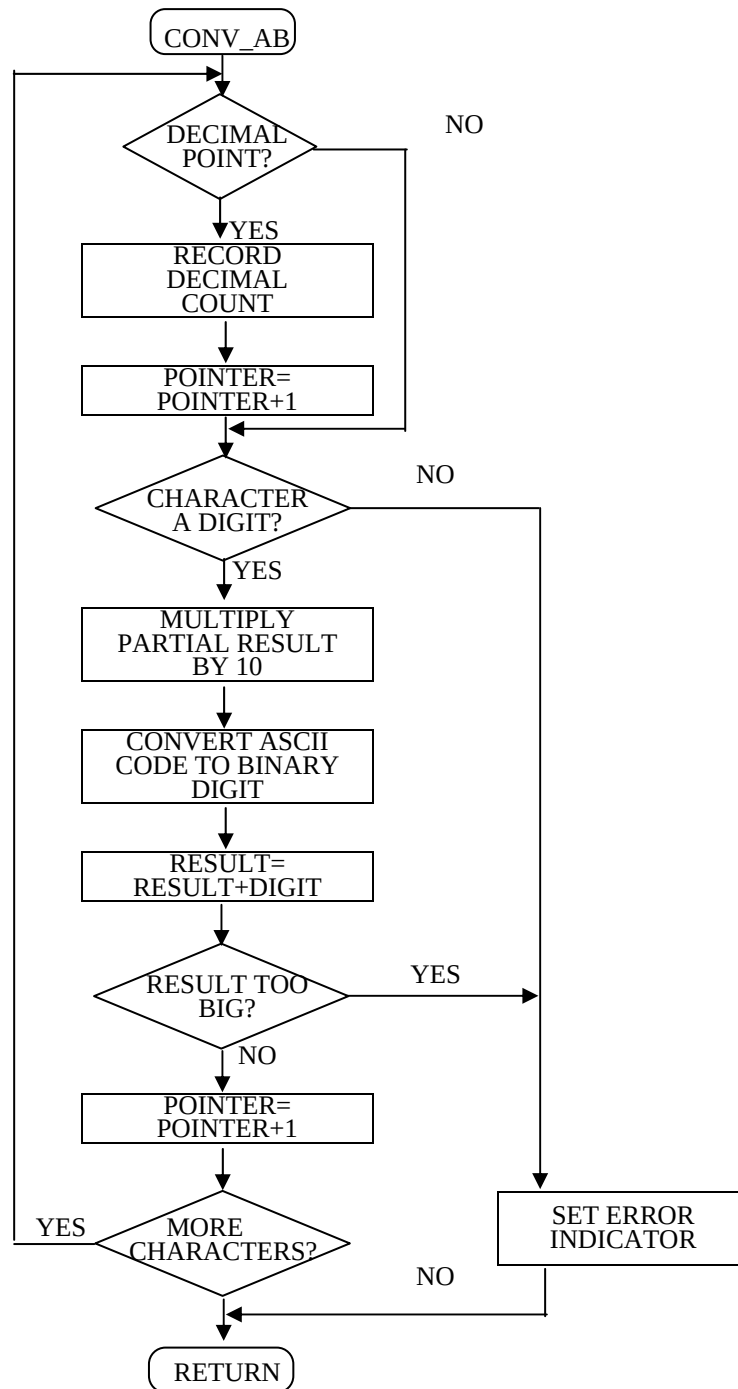
|            |                |     |
|------------|----------------|-----|
| 0111       | 7              |     |
| 0010       | 2              |     |
| 1000110    | $7 \times 10$  | 70  |
| 1001000    | $70 + 2$       | 72  |
| 0110       | 6              |     |
| 1011010000 | $72 \times 10$ | 720 |
| 1011010110 | $720 + 6$      | 726 |

از طرف دیگر الگوریتم تبدیل بایستی قادر باشد که بتواند اعداد منفی را که بصورت رشته‌ای از کداسکی داده می‌شود به دودوئی تبدیل نماید. نمودار ذیل تبدیل یک مقدار بین  $+32767$  تا  $-32768$  را انجام می‌دهد. در حقیقت نتیجه یک مقدار شانزده بیتی می‌باشد. روال CONV\_AB عملاً کار تبدیل را بعهدہ دارد.





Algorithm to convert an ASCII string to binary.



**Convert an ASCII String to Binary**

```

; Converts an ASCII string to its 16-bit, two's –complement
; Binary equivalent.
; Inputs: DS:DX= starting address of string
; CX= Character count
; Results: CF= 0 indicates no error
; AX= Binary value
; DX=count of digits after decimal point
; DI=0FFH
; CF=1 indicates error
; DS:DI=Address of non-convertible character
; DX and CX are unaffected.

; Assemble with: MASM ASC_BIN;
; Link with: LINK callprog + ASC_BIN;
PUBLIC ASCII_BIN

CSEG SEGMENT PARA 'CODE'
 ASSUME CS : CSEG
ASCII_BIN PROC FAR
 PUSH BX ;Save BX and CX
 PUSH CX
 MOV BX,DX ;Put offset in BX
 SUB AX,AX ; To start, result =0
 SUB DA,DX ; decimal count =0
 MOV DI,0FFH ;assume no bad characters
 CMP CX,7 ; String too long?
 JA NO_GOOD ; If so, go set CF and exit
BLANKS: CMP BYTE PTR [BX] ; ' ' Scan past leading blanks
 JNE CHK_NEG
 INC BX
 LOOP BLANKS
CHK_NEG: CMP BYTE PTR [BX] ; '-' Negative number?
 JNE CHK_POS
 INC BX ; If so, increment pointer
 DEC CX ;decrement the count,
 CALL CONV_AB ; convert the string
 JC THRU
 CMP AX,32768 ; Is the number too small?
 JA NO_GOOD
 NEG AX ; No. complement the result
 JS GOOD
 CMP BYTE PTR [BX], '+' ; Positive number?
 JNE GO_CONV
 INC BX ; If so, increment pointer

```

```

GO_CONV: DEC CX ; Decrement the count
 CALL CONV_AB ; Convert the string
 JC THRU
 CMP AX,32767 ; Is the number too big?
 JA NO_GOOD
 GOOD: CLC
 JNC THRU
NO_GOOD: STC ; If so, set carry flag
 THRU: POP CX ; Restore registers
 POP BX
 RET ; and exit
ASCII_BIN ENDP

```

**; This procedure performs the actual conversion.**

```

CONV_AB PROC
 PUSH BP ;Save scratch registers
 PUSH BX
 MOV BP,BX ;Put pointer in BP
 SUB BX,BX ; and clear BX
 CHK_PT: CMP DX,0 ; was a decimal point found?
 JNZ RANGE ; If so, skip following check
 CMP BYTE PTR DS:[BP], '.' ; Decimal point?
 JNE RANGE
 DEC CX ;If so, decrement count,
 MOV DX,CX ;and record it in DX
 JZ END_CONV ; Exit if CX=0
 INC BP ; Increment pointer
 RANGE: CMP BYTE PTR DS:[BP], '0' ; If the character is not a digit ...
 JB NON_DIG
 CMP BYTE PTR DS:[BP], '9'
 JBE DIGIT
 NON_DIG: MOV DI,BP ; put its address in DI,
 STC ;set the carry Flag,
 JC END_CONV ; and exit
 DIGIT: IMUL AX,10 ;Multiply the digit in AX by 10
 MOV BL, DS:[BP] ;Fetch ASCII code
 AND BX,0FH ; save only high bits,
 ADD AX,BX ;and update partial result
 JC END_CONV ; Exit if result is too big
 INC BP ;Otherwise, increment BP
 LOOP CHK_PT ; and continue
CONV_AB ENDP

```

```

END_CONV: CLC ; When done, clear carry flag
 POP BX ; Restore registers
 POP BP
 RET ; and return to caller
CONV_AB ENDP
CSEG ENDS
 END

```

به منظور بررسی درستی جواب بطریق ذیل عمل می‌نمائیم.

```

CALL ASCII_BIN ; Call the conversion procedure
JNC VALID ; Is the answer valid?
OR DI, DI ; No. find the error condition
JNZ INV_CHAR
OR AX, AX
JNZ RANGE_ER

```

```

: ; String was too long

```

```

RANGE_ER: ; Number out-of-range
INV_CHAR: ; Invalid character
VALID: ; The answer is valid

```

به منظور نمایش نتایج روی صفحه نمایش یا چاپ آنها با استفاده از دستگاه چاپگر بایستی ابتدا آنها را از باینری به ASCII تبدیل نمود. برنامه زیر این کار را انجام می‌دهد.

#### **BIN\_ASC- Convert Binary to ASCII**

```

; Converts a signed binary number to a six-byte ASCII
; String (sign plus five digits) in the data segment.
; Inputs: AX= Number to be converted
; DS: DX= starting address of string buffer
; Results: DS:DX= Starting address of string
; CX= Character count
; Other registers are preserved.

; Assemble with: MASM BIN_ASC;
; Link with: LINK callprog + BIN_ASC;

```

```

CSEG PUBLIC BIN_ASCII
 SEGMENT PARA PUBLIC 'CODE'
 ASSUME CS: CSEG
BIN_ASCII PROC FAR
 PUSH DX ; Save the caller's registers

 PUSH BX
 PUSH SI
 PUSH AX
 MOV BX, DX ; Put offset in BX
 MOV CX, 6 ; Fill buffer with spaces
FILL_BUFF: MOV BYTE PTR [BX], ' '
 INC BX
 LOOP FILL_BUFF
 MOV SI, 10 ; Get ready to divide by 10
 OR AX, AX ; If value is negative,
 JNS CLR_DVD
 NEG AX ; make it positive
CLR_DVD: SUB DX, DX ; Clear upper half of dividend
 DIV SI ; Divide AX by 10
 ADD DX, '0' ; Convert remainder to ASCII digit
 DEC BX ; Back up through buffer
 MOV [BX], DL ; Store char. In the string
 INC CX ; Count converted character
 OR AX, AX ; All done?
 JNZ CLR_DVD ; If not, get next digit
 POP AX ; Yes. Get original value

 OR AX, AX ; was it negative?
 JNS NO_MORE
 DEC BX ; Yes. Store sign
 MOV BYTE PTR [BX], '-'
 INC CX ; and increase character count
NO_MORE: POP SI ; Restore registers
 POP BX
 POP DX

 RET ; and exit
BIN_ASCII ENDP
CSEG ENDS
 END

```

## مروری بر مطالب فصل

در این فصل پشته یا Stack تعریف گردیده پشته خاصیت LIFO دارد یعنی آخرین ورودی اولین خروجی از پشته می‌باشد. پشته در حقیقت قسمتی از حافظه را اشغال می‌نماید. عملیاتی که روی پشته انجام می‌شود عبارتست از PUSH و POP.

در این فصل همچنین نحوه تعریف ماکرو داده شده هر ماکرو با کلمه macro شروع و به endm ختم می‌شود. از ماکروها برای سهولت در نوشتن و تایپ برنامه‌ها استفاده می‌شود. ماکروها را در ابتدای برنامه بایستی تعریف نمود. امکان استفاده از روال نیز در زبان اسمبلی وجود دارد. روال‌ها در موقع اجرای برنامه فراخوانی می‌شوند. تفاوت ماکرو و روال در اینست که روال در زمان اجرای برنامه فراخوانی می‌شود در صورتیکه ماکروها در زمان ترجمه جایگزین می‌گردند. از تعدادی عملگر در ماکروها استفاده می‌شود که در این فصل بحث گردیده. معمولاً در اکثر برنامه‌ها از وقفه استفاده می‌گردد. در این فصل توابع مربوط به وقفه بصورت کامل بحث شده است. برای دادن داده‌ها به کامپیوتر بایستی از فرم ASCII آنها را به باینری تبدیل نمود و همچنین برای نمایش داده‌ها بایستی از فرم باینری آنها را به شکل ASCII تبدیل نمود که برنامه‌های مخصوص آنها در این فصل تهیه و نوشته شده است. همچنین نحوه محاسبه زمان اجرای یک برنامه نیز بحث گردیده است.

## ‡ تمرین

- ۱- تفاوت macro با procedure چیست؟
- ۲- macroها در چه قسمتی از برنامه قرار می گیرند؟
- ۳- Procedureها در چه قسمتی از برنامه قرار می گیرند؟
- ۴- کار عملگرهای ماکرو چیست؟
- ۵- کار REPT, EXITM, IFNB چیست؟ یک مثال برای هر کدام ارایه کنید.
- ۶- کار LOCAL چیست؟ یک مثال ارایه کنید.
- ۷- یک Macro بنویسید که مقادیر ثباتها را ذخیره نماید.
- ۸- Interrupt Vector چیست؟
- ۹- در چه قسمتی از حافظه جدول Interrupt vector قرار دارد؟
- ۱۰- برنامه ای بنویسید که یک پیغام دلخواه را نمایش دهد.
- ۱۱- روالی بنویسید که دو مقدار صحیح را گرفته مجموع آنها را نمایش دهد.
- ۱۲- روالی بنویسید که فاصله زمانی بین فشار دو دکمه را محاسبه نماید.
- ۱۳- روالی بنویسید که صفحه مانیتور را پاک نموده و مکان نما را در سطر 10 ستون 40 قرار داده آنگاه کارکتر \* را نمایش دهد.
- ۱۴- یک Macro بنویسید که مقدار N از نوع بایت را گرفته مجموع زیرا را محاسبه نماید.  
$$1+2+3+\dots+N$$

۱۵- یک Procedure بنویسید که وقت را بصورت یک عدد شش رقمی گرفته مشخص نماید که پس از گذشت 5 ساعت و 55 دقیقه و 50 ثانیه وقت چیست و آنرا نمایش دهد.

۱۶- یک Procedure بنویسید که آرایه N عنصر X از نوع بایت را بصورت نزولی مرتب نماید.

۱۷- یک Macro بنویسید که مینیمم N مقدار از نوع word را مشخص نماید.

۱۸- یک Macro بنویسید که مشخص نماید عدد صحیح و مثبت N از نوع word اول می باشد یا خیر؟

۱۹- یک Macro بنویسید که دو مقدار M , N از نوع word و مثبت را گرفته کوچکترین مضرب مشترک آنها را محاسبه نماید.

۲۰- اگر N یک عدد صحیح و مثبت و مجذور کامل باشد یک Macro بدهید که جذر آنرا محاسبه نماید.



# فصل هشتم

## عملیات پردازش رشته‌ها

### هدف کلی

معرفی رشته‌ها و پردازش آنها.

### اهداف رفتاری

پس از مطالعه این فصل با مفاهیم زیر آشنا خواهید شد.

۱- تعریف رشته (String).

۲- انتقال یا جابجایی رشته‌ها.

۳- مقایسه رشته‌ها.

۴- بررسی یا جستجوی رشته‌ها.

۵- سایر عملیات مربوط به رشته‌ها.

### ۸-۱- رشته (String)

رشته عبارت است از یک بلوک پشت سر هم از بایت‌ها یا word ها در حافظه اصلی کامپیوتر. طول رشته می‌تواند تا 64K بایت باشد. بایستی توجه داشت که با دستورالعملهایی که تاکنون بحث نموده‌ایم می‌توان عملیات روی رشته‌ها را انجام داد. ولی استفاده از دستورالعملهای پردازش رشته‌ای کارآمدتر می‌باشد. عملیات پردازش رشته‌ای عبارتند از:

|         |          |      |
|---------|----------|------|
| Move    | جاب‌جایی | MOVS |
| Compare | مقایسه   | CMPS |
| Load    | بارکردن  | LODS |
| Scan    | جستجو    | SCAS |
| Store   | ذخیره    | STOS |

همانطوریکه قبلاً بیان گردید علاوه بر Data segment در بعضی از دستورالعملهای پردازش رشته‌ای استفاده از segment دیگری بنام Extra segment نیز ضروری می‌باشد.

```

EXTRA_SEG SEGMENT PARA 'EXTRA'
:
EXTRA_SEG ENDS

```

#### ۸-۱-۱- دستورالعمل MOVS

از این دستورالعمل برای کپی نمودن یک رشته از محلی از حافظه به محل دیگری از حافظه استفاده می‌گردد. رشته‌ای که از آن می‌خواهیم کپی تهیه نمائیم

رشته مبدأ یا Source و رشته بدست آمده را رشته مقصد یا Destination می نامند.

شکل کلی این دستورالعمل عبارتست از

MOVS

الف) چنانچه رشته از نوع بایت باشد از MOVSB و اگر رشته از نوع word باشد از

MOVSW استفاده می گردد.

ب) یک عنصر از رشته مبدأ را به رشته مقصد انتقال می دهد.

ج) دستورالعمل MOVS بر هیچ فلگی اثر ندارد.

قبل از استفاده از این دستورالعمل بایستی آدرس شروع رشته مقصد را در

ثبات DI قرار داده و رشته مقصد را در Extra segment تعریف نمود. همچنین

آدرس شروع رشته مبدأ را در ثبات SI قرار داده و رشته مبدأ را در

Data segment تعریف نمود. ضمناً مقدار فلگ DF مشخص کننده این است که

عمل جابه جایی از اولین عنصر به طرف آخرین عنصر می باشد یا بالعکس. چنانچه

مقدار DF برابر با صفر باشد عمل جابه جایی از اولین عنصر به طرف آخرین عنصر

و چنانچه مقدار DF برابر با یک باشد عمل جابه جایی از آخرین عنصر بطرف اولین

عنصر انجام می شود.

#### مثال ۸-۱

```
MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
CLD ; Forward movement
MOVSB
```

دستورالعملهای فوق آدرس شروع رشته مبدأ را در ثبات SI قرار داده و

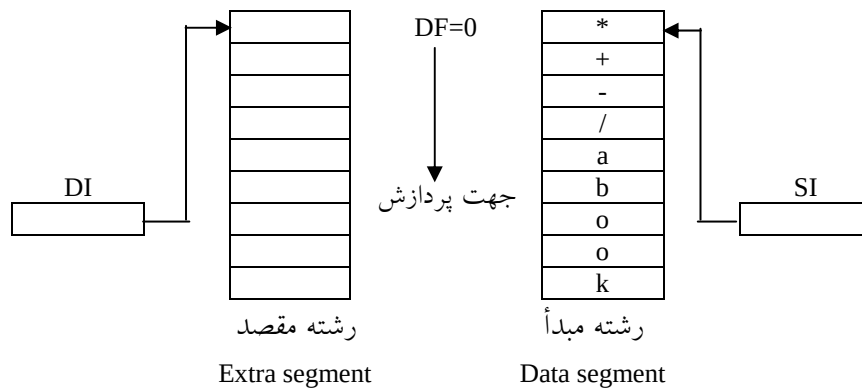
آدرس شروع رشته مقصد را در ثبات DI قرار داده و جهت حرکت از اولین عنصر

به طرف آخرین عنصر مشخص نموده است. دستورالعمل MOVSB باعث می شود

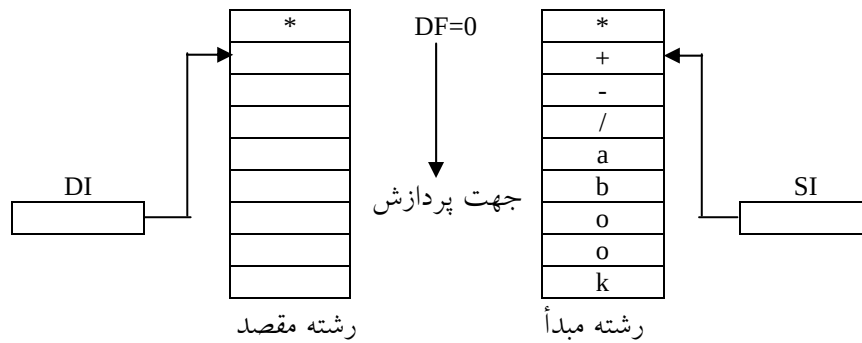
که یک عنصر از رشته مبدأ که در آدرس DS:SI قرار دارد به آدرس ES:DI کپی

گردد و مقدار DI و SI یک واحد افزایش یابند.

مقادیر ثباتها و رشته‌ها قبل از اجرای دستورالعمل MOSV

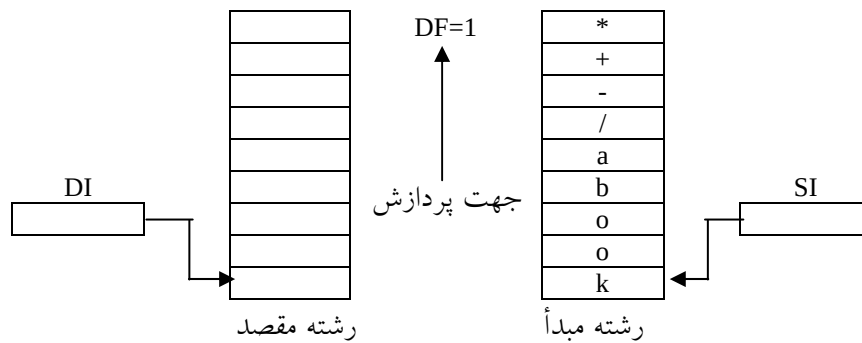


مقادیر ثباتها و رشته‌ها پس از اجرای دستورالعمل MOVS عبارتند از :

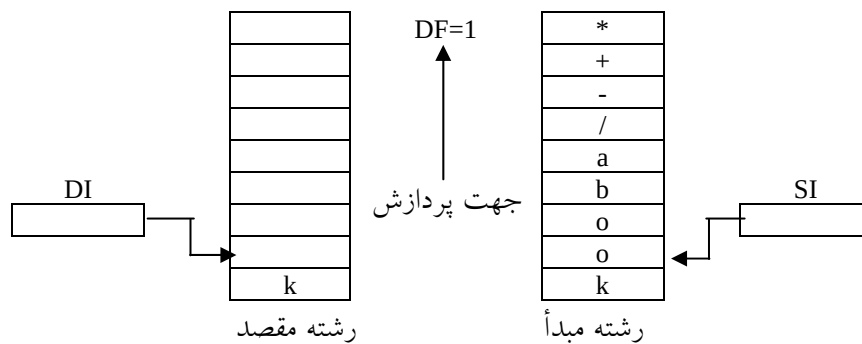


چنانچه به جای دستورالعمل CLD از دستورالعمل STD استفاده گردد مقادیر

ثباتها و رشته‌ها بصورت زیر در می‌آید:

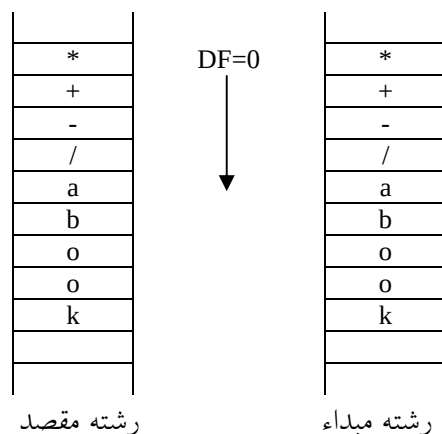


مقادیر ثابتهای SI و DI یک واحد کاهش می یابند.



حال برای انتقال سایر عناصر رشته بایستی تعداد عناصر رشته را در ثبات CX قرار داد و از پیشوند REP استفاده نمود.

```
MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
CLD
MOV CX, 9 ; تعداد عناصر رشته
REP MOVSB
```



پیشوند REP تا مادامیکه مقدار CX مخالف صفر می باشد باعث اجرای دستورالعمل MOVSb می گردد. به محض صفر شدن مقدار ثبات CX اجرای دستورالعمل MOVSb متوقف می گردد. همانطور که قبلاً گفته شد به جای دستورالعمل های

```
MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
```

از دستورالعمل های زیر می توان استفاده نمود.

```
LEA SI, SOURCE_STR
LEA DI, DEST_STR
```

ضمناً همانطوریکه گفته شد دستورالعمل های پردازش رشته ای در مقابل استفاده از دستورالعمل های معمولی کارآمد و مؤثرتر می باشند. برنامه فوق را می توان با استفاده از دستورالعمل های ذیل نیز نوشت.

```

MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
CLD
MOV CX, 9
JCXZ LAB5
LAB1: MOVSB
LOOP LAB1
LAB5:
 :
```

قطعه برنامه زیر باعث کپی شدن 100 بایت از رشته‌ای بنام SOURCE\_D به DEST\_D می‌شود. هر دو رشته در data segment قرار دارند.

```

PUSH DS
POP ES
CLD
LEA SI, SOURCE_D
LEA DI, DEST_D
MOV CX, 100
REP MOVSB
```

دستورالعملهای ذیل یک بایت از آدرس HERE به آدرس THERE منتقل می‌کند. بایستی توجه داشت که هر دو رشته در Extra segment قرار دارند.

```

LEA SI, ES: HERE
LEA DI, ES: THERE
MOVSB
```

## ۲-۱-۸- دستورالعمل STOS

این دستورالعمل باعث می‌شود که یک بایت یا یک word را از ثبات AL یا ثبات AX به یک عنصر رشته مقصد منتقل نماید. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

STOS

الف) این دستورالعمل بر روی هیچ فلگی اثر ندارد.

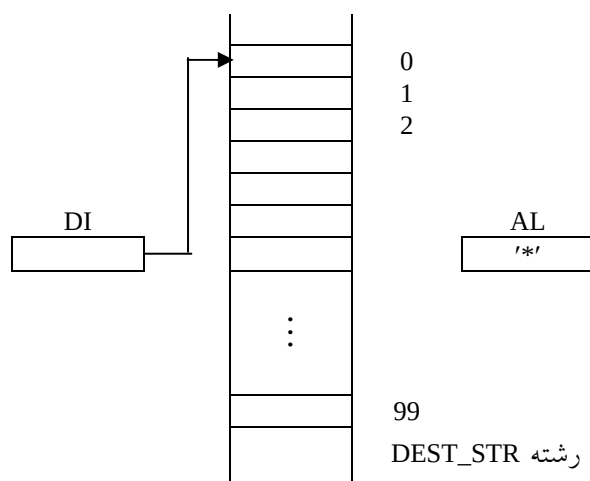
ب) چنانچه رشته از نوع بایت باشد از دستورالعمل STOSB و چنانچه از نوع word باشد از دستورالعمل STOSW استفاده می‌گردد.

- ج) مقداری که در رشته قرار می‌گیرد چنانچه از نوع بایت باشد در ثبات AL قرار داده می‌شود. و اگر از نوع word باشد در ثبات AX قرار داده می‌شود.
- د) از دستورالعملهای CLD و STD برای مشخص نمودن جهت پردازش می‌توان استفاده نمود.
- ه) از پیشوند REP نیز می‌توان استفاده نمود.
- ز) رشته را بایستی در extra segment تعریف نمود و آدرس شروع آنرا در ثبات DI قرار داد.

### مثال ۲-۸

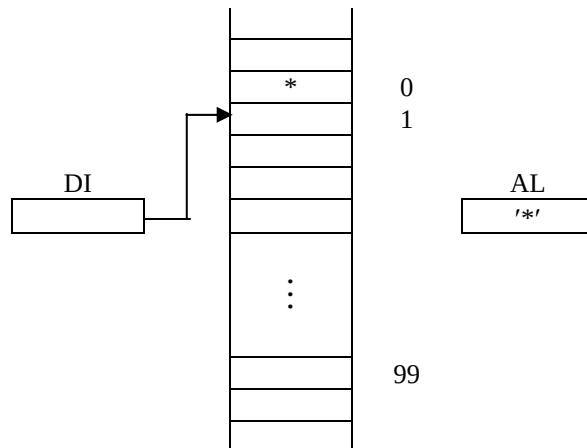
```
DEST_STR DB 100 DUP (?)
MOV AL, '*'
CLD
LEA DI, DEST_STR
STOSB
```

- مقدار \* را در عنصری از رشته که آدرس آن بوسیله ES:DI مشخص می‌شود قرار می‌دهد.





پس از اجرای دستورالعمل STOSB یک واحد به ثبات DI اضافه می‌گردد.



از پیشوند REP نیز می‌توان استفاده نمود. این پیشوند مادامیکه محتوی CX مخالف صفر می‌باشد باعث اجرای دستورالعمل STOSB می‌گردد.

مثال ۳-۸

```
DSTR DB 100 DUP(?)
MOV CX, 50
MOV AL, '*'
MOV DI, OFFSET DSTR
CLD
REP STOSB
```

قطعه برنامه فوق باعث می‌شود که مقدار 50 عنصر اول رشته DSTR برابر با

کارکتر \* گردند.

مثال ۴-۸

```
CLD
LEA DI, W_STRING
MOV AX, 0
MOV CX, 200
REP STOSW
```

قطعه برنامه فوق مقدار دویست word اول رشته W\_STRING را معادل صفر قرار می‌دهد. دقت داشته باشید که در اینجا افزایش DI باندازه دو واحد می‌باشد چون رشته از نوع word می‌باشد.

### ۳-۱-۸- دستورالعمل LODS

این دستورالعمل یک عنصر رشته مبداء را در AL یا AX قرار می‌دهد بسته به اینکه رشته از نوع بایت باشد یا word. شکل کلی این دستورالعمل عبارتست از

LODS

الف) چنانچه رشته از نوع بایت باشد از LODSB و چنانچه از نوع word باشد از LODSW استفاده می‌گردد.

ب) این دستورالعمل بر روی هیچ فلگی اثر ندارد.

ج) استفاده از پیشوند REP با این دستورالعمل امکان پذیر می‌باشد.

د) رشته بایستی در data segment تعریف گردد و آدرس شروع رشته در ثبات SI قرار داده می‌شود.

### مثال ۵-۸

LEA SI, SOURCE\_STR  
LODSB

اولین عنصر رشته SOURCE\_STR در داخل ثبات AL قرار می‌گیرد.

### ۴-۱-۸- دستورالعمل CMPS

این دستورالعمل دو رشته مبداء و مقصد را با هم مقایسه می‌نماید. دستورالعمل CMPS مانند دستورالعمل CMP عمل می‌نماید. عنصر رشته مبداء را از عنصر متناظر رشته مقصد کم نموده و فلگ‌ها را براساس نتیجه بدست

آمده تنظیم می نماید. این دستورالعمل باعث تغییر مقدار هیچکدام از عملوندها نمی شود. شکل کلی دستورالعمل بصورت زیر می باشد.

#### CMPS

الف) چنانچه رشته ها از نوع بایت باشند از CMPSB و چنانچه از نوع word باشند از CMPSW استفاده می گردد.

ب) از پیشوندهای REPE یا REPZ می توان استفاده نمود.

ج) از پیشوندهای REPNE یا REPNZ می توان استفاده نمود.

د) رشته مبدا را بایستی در data segment تعریف نمود و آدرس شروع آنرا در ثبات SI قرار داد.

ه) رشته مقصد را بایستی در Extra Segment تعریف نموده و آدرس شروع آنرا در ثبات DI قرار داد.

#### مثال ۶-۸

```
MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
CMPSB
```

دو عنصر اول رشته های SOURCE\_STR , DEST\_STR را با هم مقایسه می نماید. از پیشوندهای REPE یا REPZ با مفهوم repeat equal یا repeat zero با دستورالعمل CMPS می توان استفاده نمود. وقتی از پیشوند REPE یا REPZ استفاده می نمائیم شرط تکرار دستورالعمل اینست که  $CX < 0$  and  $ZF=1$  باشد.

## مثال ۷-۸

```

LEA SI, STR1
LEA DI, STR2
MOV CX, 100
CLD
REPE CMPSB
JZ FOUND
:
FOUND
:

```

قطعه برنامه فوق 100 بایت از دو رشته STR1 و STR2 را با هم مقایسه می‌نماید. از پیشوند REPNE یا REPZ با مفهوم repeat not equal یا repeat not zero با دستورالعمل CMPS می‌توان استفاده نمود. شرط تکرار دستورالعمل CMPS آن است که  $CX \neq 0$  and  $ZF=0$  باشد. قطعه برنامه زیر دو رشته STR1 و STR2 را با هم مقایسه می‌کند در صورتیکه مساوی باشند کنترل به SAME منتقل می‌گردد. صفت LENGTH مشخص کننده طول رشته می‌باشد.

```

MOV SI, OFFSET STR1
MOV DI, OFFSET STR2
MOV CX, LENGTH STR3
CLD
NEXT: CMPSB
 JNE EXIT
 LOOP NEXT
 JMP SAME
EXIT:
:
SAME:
:

```

## ۵-۱-۸- دستورالعمل SCAS

از دستورالعمل SCAS برای جستجوی یک رشته جهت وجود داشتن یا نداشتن یک عنصر رشته‌ای معین بکار می‌رود. شکل کلی بصورت زیر می‌باشد.

### SCAS

- الف) رشته مورد جستجو بایستی رشته مقصد باشد. یعنی رشته در extra segment تعریف شده و آدرس شروع آن در ثبات DI قرار گیرد.
- ب) از پیشوندهای REPE و REPNE می‌توان برای این دستورالعمل استفاده نمود.
- ج) عنصر مورد جستجو چنانچه از نوع بایت باشد در ثبات AL و در صورتیکه از نوع word باشد در ثبات AX قرار داده می‌شود.
- د) از فلگ DF برای تعیین جهت پردازش رشته استفاده می‌گردد.
- ه) چنانچه رشته از نوع بایت باشد SCASB و چنانچه از نوع word باشد از SCASW استفاده می‌گردد.

### مثال ۸-۸

قطعه برنامه زیر در رشته STRG به جستجوی \* می‌پردازد.

```
STRG DB 50 DUP (?)
MOV AL, '*'
MOV CX, 50
LEA DI, STRG
CLD
REPNE SCASB
```

دستورالعملهای ذیل یک رشته 80 کارکتری که به کارکتر فاصله ختم شده را جستجو نموده در صورتیکه کلیه عناصر رشته کارکتر فاصله باشد کنترل به NOT\_FOUND منتقل گردیده در غیر اینصورت با اولین عنصر مخالف blank شروع نموده و یک زیر رشته 30 کارکتری از رشته اولیه را به رشته SYMBOL انتقال می‌دهد.

```

MOV DI, OFFSET LINE
MOV CX, 80
MOV AL, 20H ; ASCII FOR BLANK
CLD
NEXT: SCAS LINE
 LOOPE NEXT
 JE NOT_FOUND
 MOV SI, DI
 DEC SI
 MOV DI, OFFSET SYMBOL
 MOV CX, 31
 FILL: STOS SYMBOL
 LOOP FILL
 MOV DI, OFFSET SYMBOL
 MOV CX, 31
 JMP SCANE
 MOVE: STOS SYMBOL
SCANE: LODS LINE
 CMP AL, 20H
 LOOPNE MOVE
 :

```

#### مثال ۹-۸

قطعه برنامه زیر رشته STRING را در نظر می‌گیرد و بدنبال کارکتر & می‌گردد که به کارکتر blank یا فاصله تبدیل نماید.

```

STRLEN EQU 15
STRING DB 'The time & is now'
CLD
MOV AL, '&'
MOV CX, STRLEN
LEA DI, STRING
REPNE SCASB
JNZ NOT_FOUND
DEC DI
MOV BYTE PTR[DI], 20H
:
NOT_FOUND:
:

```



## مروری بر مطالب فصل

دستورالعمل‌های پردازش رشته‌ای برای پردازش رشته‌ها بکار می‌روند. این دستورالعمل‌ها عبارتند از جابه‌جایی MOVS، مقایسه CMPS، بارکردن LODS، جستجو SCAS و ذخیره کردن STOS. معمولاً از دو رشته مبدأ و مقصد استفاده می‌گردد. آدرس شروع رشته مقصد را در رجستر DI و آدرس شروع رشته مبدأ را در رجستر SI قرار داده میشوند. رشته مبدأ را در data segment و رشته مقصد را در extra segment تعریف می‌نمائیم. فلگ DF جهت پردازش را مشخص می‌کند. از پیشوندهای REP، REPE، REPNE، REPZ، REPZ می‌توان با دستورالعمل‌های پردازش رشته‌ای استفاده نمود.



## ‡ تمرین

- ۱-پیشوند REP در مورد کدام دستورالعملهای رشته‌ای کاربرد دارد؟
- ۲-پیشوند REPZ در مورد کدام دستورالعملهای رشته‌ای کاربرد دارد؟
- ۳-دستورالعملهای STOS و SCAS روی کدام فلگ‌ها اثر دارند؟
- ۴-برنامه‌ای بنویسید که یک رشته 100 کاراکتری با استفاده از الگوی -----  
- زیر ایجاد نماید. (راهنمایی 20 مرتبه الگو را کپی نماید).
- ۵-رشته 100 کاراکتری STRG را در نظر گرفته کلیه کاراکترهای blank آنرا به \* تبدیل نمائید.
- ۶-رشته 100 کاراکتری STRG را در نظر بگیرید کلیه کاراکترهای آنرا به \* تبدیل نمائید.
- ۷-رشته 100 کاراکتری STRG را در نظر بگیرید اولین کاراکتر \* در رشته را به & تبدیل نمائید.
- ۸-رشته 100 کاراکتری STRG1 را در نظر گرفته 20 کاراکتر وسط رشته را به رشته STRG2 منتقل نمائید.
- ۹-رشته 50 کاراکتری STRG1 و رشته 100 کاراکتری STRG2 را در نظر بگیرید مشخص نمائید که آیا 50 کاراکتر آخر رشته STRG2 معادل STRG1 می‌باشد یا خیر؟
- ۱۰-رشته 50 کاراکتری STRG را در نظر بگیرید کلیه عناصر \* در آنرا حذف نمائید.  
(کاراکتر بعدی را جایگزین \* نمائید)

# فصل نهم

## برنامه‌های نمونه

### هدف کلی

نحوه نوشتن و ایجاد برنامه معرفی و چند برنامه نمونه.

### اهداف رفتاری

پس از مطالعه این فصل با مطالب زیر آشنا می‌شوید.

۱- اجزای مختلفه یک برنامه.

۲- نحوه نوشتن یک برنامه و اجرای آن.

## ۹-۱- اجزای برنامه

همانطوریکه گفته شد در نوشتن برنامه‌ها می‌توان از چهار سگمنت زیر استفاده نمود.

STACK SEGMENT  
DATA SEGMENT  
EXTRA SEGMENT  
CODE SEGMENT

در STACK SEGMENT پشته مورد نیاز برنامه اعلان می‌گردد. در DATA SEGMENT کلیه متغیرهای مورد نیاز برنامه اعلان و تعریف می‌شود. در EXTRA SEGMENT کلیه متغیرهای برنامه که جهت پردازش دستورالعملهای رشته‌ای مورد نیاز می‌باشد اعلان و تعریف می‌گردد. CODE SEGMENT شامل کلیه دستورالعملهای برنامه می‌باشد. در زبان اسمبلی در حقیقت برنامه بعنوان یک روال از نوع FAR نوشته می‌شود که وقتی بدستور RET برسیم کنترل به سیستم عامل برمی‌گردد.

## ۹-۲- یک برنامه نمونه

در ذیل یک برنامه نمونه داده شده است. این برنامه مقادیر یک آرایه چهار بایتی را وارون نموده و به یک آرایه دیگر منتقل می‌نماید.

```
STACK_SEG SEGMENT PARA STACK 'STACK'
DW 32 DUP (?)
STACK_SEG ENDS
DATA_SEG SEGMENT PARA 'DATA'
SOURCE DB 10, 20, 30, 40
DEST DB 4 DUP (?)
DATA_SEG ENDS
CODE_SEG SEGMENT PARA 'CODE'
```

```

ASSUME CS:CODE_SEG, DS:DATA_SEG, SS:STACK_SEG
OUR_PROG PROC FAR
;Set up the stack to contain the proper
; Valves so this program can return to Dos.
PUSH DS; Put return segment address on stack
MOV AX , 0
PUSH AX ; put zero return address on stack
; Initialize the data segment address
MOV AX, DATA_SEG ; Initialize DS
MOV DS , AX
;Initialize DEST With zeroes.

MOV DEST, 0 ;First byte
MOV DEST +1, 0 ;Second byte
MOV DEST +2, 0 ;Third byte
MOV DEST +3, 0 ;Fourth byte
;Copy SOURCE table into DEST table in reverse order.
MOV AL, SOURCE
MOV DEST +3, AL
MOV AL, SOURCE+1
MOV DEST+2, AL
MOV AL, SOURCE+2
MOV DEST+1, AL
MOV AL, SOURCE+3
MOV DEST, AL
RET ; Far return to DOS
OUR_PROG ENDP
CODE_SEG ENDS
END OUR_PROG

```

همانطوریکه ملاحظه می شود. شکلی کلی SEGMENT بصورت زیر می باشد.

```

 اسم سگمنت SEGMENT
 :
 اسم سگمنت ENDS

```

دستورالعمل ASSUME باعث می شود که آدرس شروع سگمنت CODE در

ثبات CS، آدرس شروع سگمنت DATA در ثبات DS و آدرس شروع سگمنت

STACK در ثبات SS قرار گیرد.

دستورالعمل END انتهای برنامه را مشخص می نماید. در موقع ترجمه برنامه به زبان ماشین به محض آنکه مترجم به دستور END رسید ترجمه برنامه متوقف می گردد.

### ۳-۹- نحوه اجرای برنامه

دستورالعملهای برنامه را در یک فایل با پسوند ASM قرار داده سپس دستور زیر را می دهیم.

ASM. اسم برنامه MASM > C:\

در این فاز برنامه از نظر نحوی بررسی شده چنانچه اشتباهات نحوی داشته باشد اسمبلر اشتباهات را متذکر می شود. چنانچه اشتباهی وجود داشته باشد آنرا رفع نموده مجدداً دستور MASM را می دهیم. چنانچه برنامه دارای اشتباه نحوی نباشد اسمبلر پیغام

NO WARNINGS  
NO ERRORS

را می دهد. آنگاه فرمان زیر را می دهید.

اسم برنامه LINK > C:\ASSEMBLY

در دستورالعمل فوق نیازی به پسوند ASM نمی باشد. چنانچه مشکلی در LINK باشد گزارش می شود که بایستی آنرا رفع نمود و برای اینکار مجدداً از دستور LINK بایستی استفاده نمود. حال برای اجرای برنامه کفایت که دستور ذیل را بدهیم.

اسم برنامه > C:\ASSEMBLY

در اینجا نیز نیازی به پسوند ASM . نمی باشد.

#### ۹-۴- برنامه‌های اسمبلی نوشته شده

در این بخش تعدادی برنامه که بزبان اسمبلی نوشته شده ارائه می‌گردد.  
اولین برنامه ابتدا صفحه مانیتور را پاک نموده سپس کارکتر A را در سطح 25 و  
ستون 13 صفحه مانیتور قرار می‌دهد.

```
STACK_SEG SEGMENT PARA STACK 'STACK'
DW 32 DUP (?)
STACK_SEG ENDS
;

DATA_SEG SEGMENT PARA 'DATA'
MESSAGE DB 'Nikmehr', 13, '$'
DATA_SEG ENDS
CODE_SEG SEGMENT PARA 'CODE'
BEGIN PROC FAR
ASSUME SS:STACK_SEG,CS:CODE_SEG, DS:DATA_SEG
START:
PUSH DS
SUB AX, AX
PUSH AX
MOV AX, DATA_SEG
MOV DS, AX
;
MOV AH, 00
MOV AL, 03
INT 10 H
;
MOV AH, 2
MOV BH, 0
MOV DH, 13 ; row
MOV DL, 25 ; column
INT 10H
;
;
MOV AH, 2
MOV DL, 65
INT 21 H
;
RET
BEGIN ENDP
CODE_SEG ENDS
END START
```

برنامه بعدی ارقام 0 تا 9 را بدنبال هم روی صفحه مانیتور ظاهر می سازد.  
 دستورالعمل NOP به معنی NO OPERATION می باشد و عملاً کاری انجام نمی دهد.

```

STACK_SEG SEGMENT PARA STACK 'STACK'
DW 32 DUP (?)
STACK_SEG ENDS
;
;
CODE_SEG SEGMENT PARA STACK 'STACK'
BEGIN PROC FAR
ASSUME SS:STACK_SEG, CS:CODE_SEG
START:
PUSH DS
SUB AX,AX
PUSH AX
;
MOV AH, 00
MOV AL, 03
INT 10 H
;
MOV AH, 2
MOV BH, 0
MOV DH, 13 ;ROW
MOV DL, 25 ; COLUMN
INT 10 H
;
;character input without echo
;
MOV CX, 10
LOOP1:NOP
MOV AH, 7
INT 21H
;
MOV AH, 2
MOV DL, AL
INT 21H
LOOP LOOP1
;
RET
BEGIN ENDP
CODE-SEG ENDS
END START

```



برنامه بعدی یک مقدار باینری را روی صفحه مانیتور نمایش می دهد.

```
STACK_SEG SEGMENT PARA 'STACK'
DW 32 DUP(?)
STACK_SEG
ENDS
;
;
DATA_SEG SEGMENT PARA 'DATA'
ASCII_VAL DB 8 DUP (?)
BINARY_VAL
DW?
ASCII_LENGTH DW 8
DATA_SEG
ENDS
CODE_SEG SEGMENT PARA 'CODE'
BEGIN PROC
FAR
ASSUME SS:STACK_SEG,
CS:CODE_SEG,DS:DATA_SEG
START:
PUSH DS
SUB AX,AX
PUSH AX
MOV AX, DATA_SEG
MOV DS,AX
;
MOV AH, 00
MOV AL, 03
INT 10 H
;
MOV AH, 2
MOV BH,0
MOV DH, 13 ;ROW
MOV DL, 25 ; COLUMN
INT 10H
;
MOV BINARY_VAL, 32456
CALL BINARYTOASCII
; displaying & printing binary numbers
;
LEA SI, ASCII_VAL
MOV CX, ASCII_LENGTH
LOOP1: MOV AH,2
MOV DL, [SI]
INT 21H
; PRINTING
MOV AH,5
MOV DL, [SI]
INT 21H
INC SI
LOOP LOOP1
```

```

;
;
;
RET
BEGIN ENDP
;
;
;

```

# **BINARYTOASCII PROC NEAR**

```

 MOV CX, 10
 LEA SI, ASCII_VAL+7
 MOV AX, BINARY_VAL

LABEL12:
 CMP AX, 10
 JB LABEL13
 XOR DX, DX
 DIV CX
 OR DL, 30H
 MOV [SI], DL
 DEC SI
 JMP LABEL12

LABEL13:
 OR AL, 30H
 MOV [SI], AL
 RET
BINARYTOASCII ENDP
CODE-SEG ENDS
END START

```

خروجی برنامه عبارتند از

```

3
2
4
5
6
32456

```

برنامه بعدی نحوه استفاده از Macro در برنامه‌ها را نشان می‌دهد.

#### **NIK MACRO**

ASSUME SS:STACK\_SEG, CS:CODE\_SEG, DS:DATA\_SEG

START:

PUSH DS

SUB AX, AX

PUSH AX

MOV AX, DATA\_SEG

MOV DS, AX

;

ENDM

STACK\_SEG SEGMENT PARA STACK 'STACK'

DW 32 DUP (?)

STACK\_SEG ENDS

;

;

DATA\_SEG SEGMENT PARA 'DATA'

MESSAGE DB 'DARYOUSH NIKMEHR' ,13, '\$'

DATA\_SEG ENDS

CODE\_SEG SEGMENT PARA 'CODE'

NIK

BEGIN PROC FAR

MOV AH, 00

MOV AL, 03

INT 10H

;

MOV AH, 2

MOV BH, 0

MOV DH, 13 ; ROW

```
MOV DL, 25; COLUMN
INT 10H
;
;
MOV AH, 2
MOV DL, '*'
INT 21H
;
;
RET
BEGIN ENDP
CODE_SEG
ENDS
END START
```

برنامه بعدی نحوه قرار دادن روالهای از نوع NEAR در برنامه‌ها را در  
CODE SEGMENT نشان می‌دهد.

```
CODE_SEG SEGMENT PARA 'CODE'
BEGIN PROC FAR
ASSUME CS:CODE_SEG , ...
START:
 :
CALL PROCB
 :
CALL PROCC
 :

RET ; کنترل به سیستم عامل برمی‌گردد
BEGIN ENDP
PROCB PROC NEAR
 :

RET ; کنترل به دستورالعمل بعد از CALL PROCB منتقل شود
PROCB ENDP
PROCC PROC NEAR
 :

RET ; کنترل به دستورالعمل بعد از PROCC منتقل شود
PROCC ENDP
CODE_SEG ENDS
END START
```

در ذیل برنامه ضرب دو مقدار 32 بیتی داده شده است؛ در این برنامه این دو مقدار بصورت بدون علامت در نظر گرفته شده‌اند.

```
; Multiplies two 32-bit unsigned number and generates a
; 64-bit product.
; Inputs: CX:BX=Multiplier
; DX:AX=Multiplicand
; Result : Product in DX, CX, BX, and AX (high to low order).

; To assemble: MASM MULU32;
; To Link: LINK callprog+MULU 32;
```

```
 PUBLIC MULU32

DSEG SEGMENT PARA 'DATA'
HI_MCND DW ?
LO_MCND DW ?
HI_PP1 DW ?
LO_PP1 DW ?
HI_PP2 DW ?
LO_PP2 DW ?
HI_PP3 DW ?
LO_PP3 DW ?
HI_PP4 DW ?
LO_PP4 DW ?
DSEG ENDS

CSEG SEGMENT PARA 'CODE'
 ASSUME CS:CSEG,DS:DSEG
MULU32 PROC FAR
 PUSH DS ;Save caller's DS and DI
```

```
PUSH DI
MOV DI, DSEG ;Initialize DS
MOV DS,DI
MOV HI_MCND,DX ;Save multiplicand in memory
MOV LO_MCND,AX
MUL BX ;Form partial product #1
MOV HI_PP1,DX ;and save it in memory
MOV LO_PP1, AX
MOV AX, HI_MCND ;Form partial product #2
MUL BX
MOV HI_PP2, DX ;and save it in memory
MOV LO_PP2, AX
MOV AX,LO_MCND ;Form partial product #3
MUL CX
MOV HI_PP3,DX ;and save it in memory
MOV LO_PP3, AX
MOV AX, HI_MCND ;Form partial product #4
MUL CX
MOV HI_PP4, DX ;and save it in memory
MOV LO_PP4, AX
```

; Add the partial products to form the final 64-bit product.

```
MOV AX,LO_PP1 ;Low 16 bits
MOV BX,HI_PP1 ;Form mid-lower 16 bits
ADD BX,LO_PP2 ; with sum #1
ADC HI_PP2,0
ADD BX,LO_PP3 ; and sum #2
MOV CX,HI_PP2 ;Form mid-upper 16 bits
ADC CX,HI_PP3 ;with sum #3
```

```

 ADC HI_PP4, 0
 ADD CX,LO_PP4 ;and sum #4
 MOV DX,HI_PP4 ;Form high 16 bits
 ADC DX,0 ;including propagated carry
 POP DI ;Restore caller's registers
 POP DS
 RET
MULU32 ENDP
CSEG ENDS
END
```



برنامه داده شده در ذیل حاصلضرب دو مقدار 32 بیتی علامتدار را تعیین می نماید.

```
; Multiplies two 32-bit signed numbers and generates
; A 64-bit product.
; Inputs: CX:BX = Multiplier
; DX:AX = Multiplicand
; Result : Product in DX, CX, BX, and AX (high to low order)
; Calls MULU32

; To assemble: MASM MULS 32;
; To link: LINK callprog+MULS32+MULU32;

 EXTRN MULU32 : FAR
 PUBLIC MULS 32
DSEG SEGMENT PARA 'DATA'
NEG_IND DB ?
DSEG ENDS

CSEG SEGMENT PARA 'CODE'
MULS32 PROC FAR
 ASSUME CS:CSEG , DS:DSEG
; Initialize the data segment address
 PUSH DS ;Save caller's DS and DI
 PUSH DI
 MOV DI, DSEG ;Initialize DS
 MOV DS , DI

 MOV NEG_IND,0 ;Negative indicator=0
 CMP DX, 0 ;Multiplicand negative?
 JNS CHKCX ;No. Go check multiplier
 NOT AX ;Yes. 2s-comp.multiplicand
 NOT DX
 ADD AX,1
 ADC DX,0
 NOT NEG_IND ;and 1s-comp. Indicator
CHKCX: CMP CX,0 ;Multiplier negative?
 JNS GOMUL ;No. Go multiply
 NOT BX ;Yes. 2s-comp. Multiplier
 NOT CX
 ADD BX,1
 ADC CX,0
 NOT NEG_IND ;and 1s-comp. Indicator
```

```
GOMUL: CALL MULU32 ;Perform unsigned multiplication
 CMP NEG_IND,0 ;Does product have right sign?
 JZ DONE ;Yes. Exit.
 NOT AX ;No. 2s-comp. Product
 NOT BX
 NOT CX
 NOT DX
 ADD AX,1
 ADC BX,0
 ADC CX,0
 ADC DX,0
DONE: POP DI ;Restore caller's registers
 POP DS
 RET
MULS32 ENDP
CSEG ENDS
END
```

برنامه ذیل تقسیم دو مقدار را انجام می دهد.

```

; This divide procedure determines the correct quotient
; And remainder, regardless of overflow.
; Inputs: BX = Divisor
; DX:AX=Dividend
; Results : BX:AX=Quotient
;
;
; DX = Remainder
; To assemble: MASM DIVUO;
; To link: LINK callprog+DIVUO;

PUBLIC DIVUO
CSEG SEGMENT PARA 'CODE'
ASSUME CS: CSEG
DIVUO PROC FAR
CMP BX,0 ; Divisor = 0?
JNZ DVROK
DVROK: INT 0 ; Yes. Abort the divide
PUSH ES ; Save working registers
PUSH DI
PUSH CX
MOV DI,0 ; Fetch current INT 0 vector
MOV ES, DI
PUSH ES:[DI] ; and save it on the stack
PUSH ES:[DI+2]
LEA CX, OVR_INT ;Make INT 0 vector
MOV ES:[DI], CX ;point to OVR_INT
MOV CX, SEG OVR_INT
MOV ES:[DI+2], CX
DIV BX ;Perform the division

```

```

RESTORE: SUB BX, BX ;If no overflow, BX=0
 POP ES: [DI+2] ;Restore INT 0 vector
 POP ES: [DI]
 POP CX ;Restore registers
 POP DI
 POP ES
 RET

; This interrupt service routine executes if the divide
; Operation produces overflow.

OVR_INT: POP CX ;Modify ret. Addr. Offset
 LEA CX, RESTORE ; to skip SUB BX,BX
 SUSH CX
 SUSH AX
 MOV AX,DX ;Set up 1st dividend, 0-Y1
 SUB DX,DX
 DIV BX ;Q1 is in AX, R1 is in DX
 POP CX ;Pop orig. AX into CX
 PUSH AX ;Save Q1 on stack
 MOV AX,CX ;Set up 2nd dividend, R1-Y0
 DIV BX ;Q0 is in AX, R0 is in DX
 POP BX ; Final quotient is in BX:AX
 IRET

DIVUO ENDP
CSEG ENDS
 END

```

برای محاسبه جذر یک مقدار 32 بیتی از روش تکراری نیوتون استفاده می‌شود. روش نیوتون بدین صورت است که اگر A جذر تقریب مقدار N باشد آنگاه A1 تقریب بهتری برای جذر N می‌باشد.

$$A1 = (N / A + A) / 2$$

بعنوان مثال اگر N=10000 باشد و A=200 آنگاه اولین تقریب برابر با  $10000 / 200 + 2 = 52$  می‌باشد.

$$10000/52 = 192, (192+52) / 2 = 122$$

$$10000/122 = 81, (122+81) / 2 = 101$$

$$10000/101 = 99, (101+99) / 2 = 100$$

$$10000 / 100 = 100$$

برنامه زیر جذر یک مقدار 32 بیتی را محاسبه می‌نماید.

```
; Culculates the square root of a 32-bit integer.
;
; Input : DX:AX = Integer
;
; Result : BX = square root
;
; The original number in DX:AX is unaffected.
```

```
; To assemble: MASM SQRT 32;
;
; To link: LINK callprog+SQRT32;
```

```
 PUBLIC SQRT32
CSEG SEGMENT PARA 'CODE'
 ASSUME CS: CSEG
SQRT32 PROC FAR
 PUSH BP ;Save contents of BP
 PUSH DX ;and source number DX:AX
 PUSH AX
```

```

 MOV BP,SP ;BP points to AX on the stack
 MOV BX, 200 ;As a first approx,
 DIV BX ;divide source number by 200,
 ADD AX, 2 ;then add 2
NXT_APP: MOV BX, AX ;Save this approx. in BX
 MOV AX, [BP] ;Read source number again
 MOV DX, [BP+2]
 DIV BX ;Divide by Last approx.
 ADD AX,BX ;Average last two approxs.
 SHR AX,1
 CMP AX,BX ;Last two approxs. Identical?
 JE DONE
 SUB BX,AX ;No. Check for diff. Of 1
 CMP BX,1
 JE DONE
 CMP BX,-1
 JNE NXT_APP
DONE: MOV BX,AX ;Put result in BX
 POP AX ;Restore source number
 POP DX
 POP BP ;and scratch register BP
 RET
SQRT32 ENDP
CSEG ENDS
END

```

برنامه زیر یک عنصر از یک لیست نامرتب را حذف می‌نماید.

```

; Deletes the value in AX from an unordered list in the
; Extra segment, if that value is in the list.
; Inputs: DI = starting address of the list
; First location = Length of list (words)
; Results: None
; DI and Ax are unaltered.

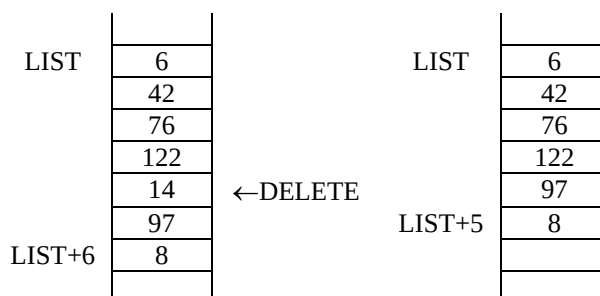
; Assemble with : MASM DEL_UL;
; Link with: LINK callprog + DEL_UL;
PUBLIC DEL_UL
CSEG SEGMENT PARA 'CODE'
ASSUME CS:CSEG
DEL_UL PROC FAR
CLD ;Make DF=0, to scan forward
PUSH BX ;Save scratch register BX
PUSH DI ; and starting address
MOV CX, ES:[DI] ;Fetch element count
ADD DI, 2 ;Make DI point to 1st data el.
REPNE SCASW ;Value in the list?
JE DELETE ;If so, go delete it.
POP DI ; Otherwise, restore registers
POP BX
RET ;and exit.

; The following instructions delete an element from the list,
; As follows:
; (1) If the element lies at the end of the list,
; Delete it by decreasing the element count by 1.

```

; (2) Otherwise, delete the element by moving all  
; Subsequent elements up by one position.

```
DELETE: JCXZ DEC_CNT ;If (CX)= 0, delete last el.
NEXT_EL: MOV BX,ES:[DI] ;Move one element up. In list
 MOV ES,[DI-2], BX
 ADD DI, 2 ;Point to next element
 LOOP NEXT_EL ;Repeat until all els. Moved
DEC_CNT: POP DI ;Decrease el. Count by 1
 DEC WORD PTR ES:[DI]
 POP BX ;Restore contents of BX
 RET ;and exit
DEL_UL ENDP
CSEG ENDS
 END
```





برنامه ذیل مقدار ماکزیمم و مینیمم یک لیست نامرتب را مشخص می نماید.

```

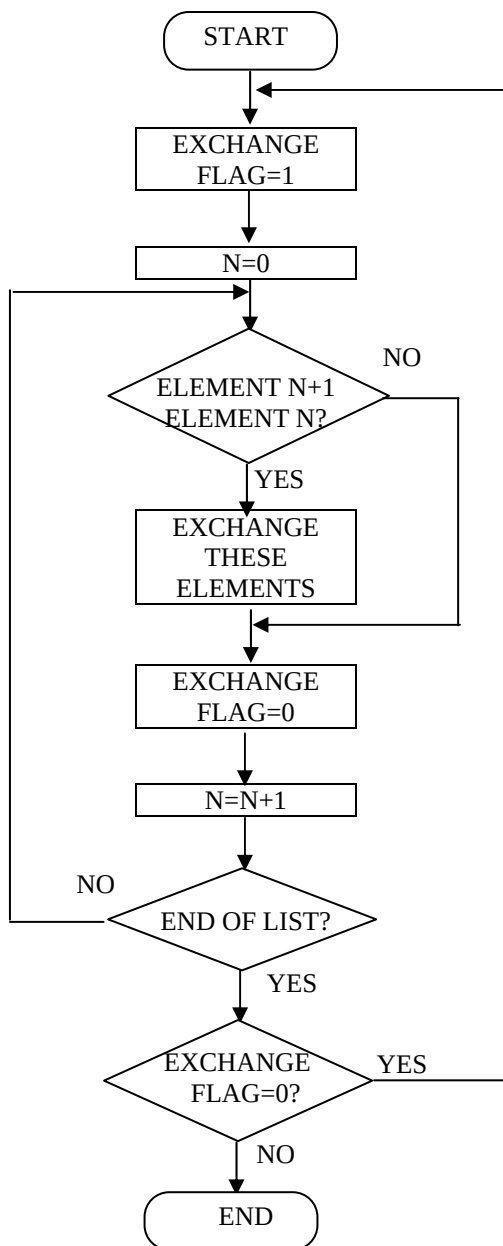
; Finds the maximum and minimum words in an unordered
; List in the extra segment.
; Inputs: ES: DI = Starting address of the list
; First location = Length of list (words)
; Results : AX = Maximum
; BX = Minimum
; DI is unaltered.

; Assemble with : MASM MINMAX;
; Link with: LINK callprog + MINMAX;

PUBLIC MINMAX
CSEG SEGMENT PARA 'CODE'
ASSUME CS: CSEG
MINMAX PROC FAR
PUSH CX
PUSH DI ;Save starting address
MOV CX, ES:[DI] ;Fetch element count
DEC CX ;Get ready for count-1 compares
ADD DI,2 ;point to first element
MOV BX, ES:[DI] ;Declare it both minimum
MOV AX,BX ;and maximum
CHKMIN: ADD DI,2 ;Point to next element
CMP ES:[DI], BX ;Compare element to minimum
JAE CHKMAX ;New minimum found?
MOV BX,ES:[DI] ;Yes. Put is in BX
JMP SHORT NEXTEL
CHKMAX: CMP ES:[DI], AX ;Compare element to maximum
JBE NEXTEL ;New maximum found?
MOV AX, ES:[DI] ;Yes. Put it in AX
NEXTEL: LOOP CHKMIN ;Check entire list
POP DI ;Restore starting address
POP CX
RET ; and exit
MINMAX ENDP
CSEG ENDS
END

```

در ذیل الگوریتم مرتب سازی حبابی داده شده است:



برنامه زیر تعدادی عناصر را بصورت صعودی بروش حبابی مرتب می نماید.

```
; Arranges the 16-bit elements of a list in the extra
; Segment in ascending order, using bubble sort.
; Inputs: ES:DI = starting address of the list
; First location = Length of list (words)
; DI is unaltered.
```

```
; Assemble with: MASM B_SORT;
; Link with: LINK callprog + B_SORT;
```

```
DSEG SEGMENT PARA 'DATA'
SAVE_CNT DW ?
START_ADDR DW ?
DSEG ENDS
```

```
 PUBLIC B_SPRT
CSEG SEGMENT PARA 'CODE'
 ASSUME CS:CSEG,DS:DSEG
B_SORT PROC FAR
 PUSH DS ;Save caller's registers
 SUSH CX
 SUSH AX
 SUSH BX
 MOV AX, DSEG ;Initialize DS
 MOV DS,AX
 MOV START_ADDR,DI ;Save starting address
 MOV CX,ES:[DI] ;Fetch element count
 DEC CX ;Get ready for count-1 compares
 MOV SAVE_CNT,CX ;Save this value in memory
```

```

INIT: MOV BX,1 ;Exchange flag (BX) = 1
 MOV CX,SAVE_CNT ;and load this count into CX
 MOV DI,START_ADDR ;Load start address into DI
NEXT: ADD DI,2 ;Address a data element
 MOV AX,ES:[DI] ;and load it into AX
 CMP ES:[DI+2],AX ;Is next el. <this el.?
 JAE CONT ;No. Go check next pair
 XCHG ES:[DI+2],AX ;Yes. Exchange these elements.
 MOV ES:[DI],AX
 SUB BX,BX ;and make exchange flag 0
CONT: LOOP NEXT ;Process entire list
 CMP BX,0 ;Any exchanges made?
 JE INIT ;If so, process list again
 MOV DI,START_ADDR ;If not, restore registers
 POP BX
 POP AX
 POP CX
 POP DS
 RET ;and exit
B_SORT ENDP
CSEG ENDS
END

```

برنامه زیر روش بهتری برای مرتب نمودن عناصر یک آرایه بر روش حبابی ارائه می دهد.

```
; Arranges the 16-bit elements of a list in the extra
; Segment in ascending order, using bubble sort.
; Inputs: ES:DI = starting address of the list
; First location = Length of list (words)
; DI is unaltered.
```

```
; Assemble with: MASM BUBBLE;
; Link with: LINK callprog + BUBBLE;
```

```
DSEG SEGMENT PARA 'DATA'
SAVE_CNT DW ?
START_ADDR DW ?
DSEG ENDS

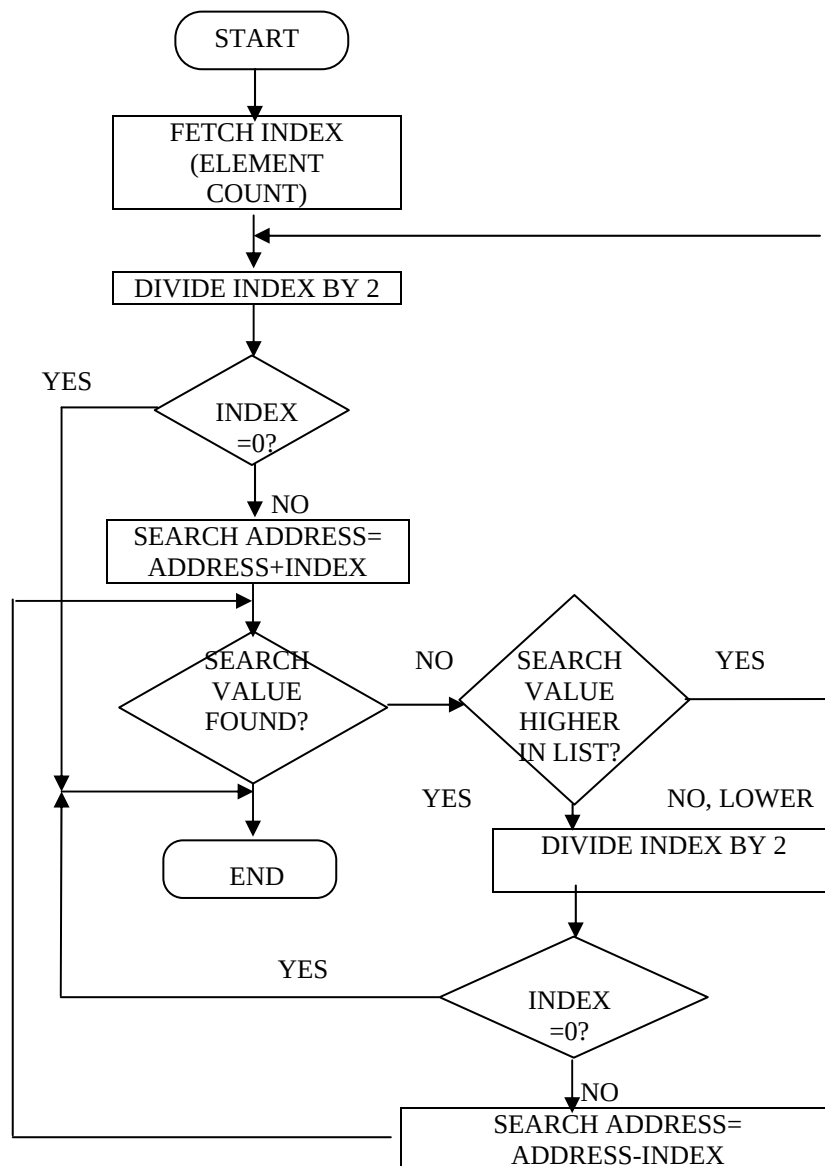
 PUBLIC BUBBLE
CSEG SEGMENT PARA 'CODE'
 ASSUME CS:CSEG,DS:DSEG
BUBBLE PROC FAR
 PUSH DS ;Save caller's registers
 PUSH CX
 PUSH AX
 PUSH BX
 MOV AX,DSEG ;Initialize DS
 MOV DS,AX
 MOV START_ADDR,DI
 MOV CX,ES:[DI] ;Fetch element count
 MOV SAVE_CNT,CX ;Save this value in memory
```

```

INIT: MOV BX,1 ;Exchange flag (BX) = 1
 DEC SAVE_CNT ;Get ready for count-1 compares
 JZ SORTED ;Exit if SAVE_CNT is 0
 MOV CX,SAVE_CNT ;and load this count into CX
 MOV DI, START_ADDR ;Load start address into DI
NEXT: ADD DI,2 ;Address a data element
 MOV AX,ES:[DI] ;and load it into AX
 CMP ES:[DI+2],AX ;Is next el. <this el.?
 JAE CONT ;No. Go check next pair
 XCHG ES:[DI+2],AX ;Yes. Exchange these elements.
 MOV ES:[DI],AX
 SUB BX,BX ;and make exchange flag 0
CONT: LOOP NEXT ;Process entire list
 CMP BX,0 ;Any exchanges made?
 JE INIT ;If so, process list again
SORTED: MOV DI, START_ADDR ;If not, restore registers
 POP BX
 POP AX
 POP CX
 POP DS
 RET ;and exit
BUBBLE ENDP
CSEG ENDS
END

```

در ذیل الگوریتم جستجوی دودوئی مطرح گردیده است.



برنامه ذیل روش جستجوی دودوئی برای مقادیر 16 بیتی ارائه می دهد.

```
; Searches an ordered list in the extra segment for the
; Word value contained in AX.
; Inputs : ES:DI = starting address of the list
; First location = Length of list (words)
; Results: if the value is in the list,
; CF=0
; SI=offset of matching element
; If the value is not in the list,
; CF = 1
; SI = offset of last element compared
; AX and DI are unaffected.
```

```
; Assemble with: MASM B_SEARCH;
; Link with: LINK callprog+B_SEARCH;
```

```
DSEG SEGMENT PARA 'DATA'
START_ADDR DW ?
DSEG ENDS
```

```
 PUBLIC B_SEARCH
CSEG SEGMENT PARA 'CODE'
 ASSUME CS:CSEG,DS:DSEG
B_SEARCH PROC FAR
 PUSH DS ;Save caller's DS register
 PUSH AX
 MOV AX,DSEG ;Initialize DS
 MOV DS,AX
 POP AX
```



```

; Find out if AX lies beyond the boundaries of the list.
CMP AX,ES:[DI+2] ;Search value <or=first el.?
JA CHK_LAST ;No. Go check last element
LEA SI,ES:[DI+2] ;Yes. Fetch addr. Of first el.
JE EXIT ;If value = 1 st element, exit
STC ; If value < 1 st element, set CF
JMP EXIT ;and then exit
CHK_LAST: MOV SI,ES:[DI] ;Point to last element
SHL SI,1
ADD SI,DI
CMP AX,ES:[SI] ;Search value > or = last el.?
JB SEARCH ;No. Go search list
JE EXIT ;Yes. Exit if value = last el.
STC ;If value > last element, set CF
JMP EXIT ;and then exit
; Search for value within the list.
SEARCH: MOV START_ADDR,DI ;Save starting address in memory
MOV SI, ES:[DI] ;Fetch index
EVEN_IDX: TEST SI,1 ;Force index to an even value
JZ ADD_IDX
INC SI
ADD_IDX: ADD DI,SI ;Calculate next search address
COMPARE: CMP AX, ES:[DI] ;Search value found?
JE ALL_DONE ; If so, exit
JA HIGHER ; Otherwise, find correct half
```

```

; These instructions are executed if the search value is lower
; In the list.
 CMP SI , 2 ;Index = 2?
 JNE IDX _ OK
NO_MATCH: STC ;If so, set CF
 JE ALL_DONE ;and exit
 IDX_OK: SHR SI ,1 ;If not, divide index by 2
 TEST SI, 1 ;Force index to an even value
 JZ SUB_IDX
 INC SI
 SUB_IDX: SUB DI, SI ;Calculate next address
 JMP SHORT COMPARE ; Go check this element

; These instructions are executed if the search value is higher
; In the list.

 HIGHER: CMP SI,2 ;Index = 2?
 JE NO_MATCH ;If so, go set CF and exit
 SHR SI,1 ;If not, divide index by 2
 JMP SHORT EVEN_IDX ;Go check next element

; Following are exit instructions.
ALL_DONE: MOV SI,DI ;Move compare address into SI
 MOV DI,START_ADDR ;Restore starting address
 EXIT: POP DS
 RET ;and exit
B_SEARCH ENDP
CSEG ENDS
 END

```

برنامه زیر یک عنصر را به یک لیست مرتب اضافه می‌نماید.

```
; Adds the element in AX to an ordered list in the
; Extra segment, if that value is not already in the list.
; Inputs: DI = starting address of the list
; First location = List length (words).
; Result : None
; DI and AX are unaltered.
; The B_SEARCH procedure (Example 5-6) is used to conduct
; The search.
```

```
; Assemble with: MASM ADD_2_OL;
; Link with: LINK callprog + ADD_2_OL +B_SEARCH;
```

```
EXTRN B_SEARCH:FAR
PUBLIC ADD_TO_OL
CSEG SEGMENT PARA 'CODE'
ASSUME CS:CSEG
ADD_TO_OL PROC FAR
PUSH CX ; Save caller's registers
PUSH SI
PUSH BX
CALL B_SEARCH ; Is the value in the list?
JNC GOODBYE ; If so, exit
MOV BX,SI ; if not, copy compare addr. To BX
MOV CX,ES:[DI] ; Find address of last element
SHL CX,1
ADD CX,DI ; and put it in CX
PUSH CX ; Save this address on the stack
SUB CX,SI ; Calculate no. of words to be moved
SHR CX,1
```

```

 CMP AX,ES:[SI] ; should compare el. Be moved, too?
 JA EXCLUDE
 INC CX ; Yes. Increase move count by 1
 JNZ CHECK_CNT
EXCLUDE: ADD BX,2 ; No. Adjust insert pointer
CHECK_CNT: CMP CX,0 ; Move count = 0?
 JNE MOVE_ELS
 POP SI ; If so, store value at end of list
 MOV ES:[SI+2],AX
 JMP SHORT INC_CNT ; Then go increase element count
MOVE_ELS: POP SI ; Load start address for move
 PUSH BX ; Save insert address on stack
MOVE_ONE: MOV BX, ES:[SI] ; Move one element down in list
 MOV ES:[SI+2], BX
 SUB SI,2 ; Point to next element
 LOOP MOVE_ONE ; Repeat until all are moved
 POP BX ; Retrieve insert address
 MOV ES:[BX], AX ; Insert AX in the list
INC_CNT: INC WORD PTR ES:[DI] ; Add 1 to element count
GOODBYE: POP BX ; Restore registers
 POP SI
 POP CX
 RET ; and exit
ADD_TO_OL ENDP
CSEG ENDS
 END

```

برنامه داده شده در ذیل یک مقدار را از لیست مرتب داده شده حذف می نماید.

```
; Deletes the value in AX from the ordered list in the
; Extra segment, if the value is in the list.
; Inputs: ES: DI = Starting address of the list
; First location = Length of list (words)
; AX and DI are unaffected.
; The B_SEARCH procedure (Example 5-6) is used to conduct
; The search.
```

```
; Assemble with: MASM DEL_OL;
; Link with: LINK callprog + DEL_OL + B_SEARCH;
```

```
EXTRN B_SEARCH : FAR
PUBLIC DEL_OL
CSEG SEGMENT PARA 'CODE'
DEL_OL PROC FAR
ASSUME CS,CSEG
PUSH CX ; Save caller's registers
PUSH SI
PUSH BX
CALL B_SEARCH ; Is the value in the list?
JC ADIOS ; If not, exit
MOV CX,ES:[DI] ; If so, find addr. Of last element
SHL CX,1
ADD CX,DI ; and put it in CX
CMP CX,SI ; Is the last el. To be deleted?
JE CNT_M1 ; Yes. Go decrement el. Count
SUB CX,SI ; No. calculate move count
SHR CX,1
```

```
MOVEM: MOV BX,ES:[SI+2] ; Move one element up in list
 MOV ES:[SI], BX
 ADD SI,2 ; Point to next element
 LOOP MOVEM ; Repeat until all are moved
CNT_M1: DEC WORD PTR ES:[DI] ; Decrease element count by 1
ADIOS: POP BX ; Restore registers
 POP SI
 POP CX
 RET ; and exit
DEL_OL ENDP
CSEG ENDS
END
```

## مروری بر مطالب فصل

در این فصل نحوه نوشتن برنامه‌ها به زبان اسمبلی و نحوه اجرای آنها بحث گردیده است. ضمناً چندین برنامه بصورت نمونه نوشته شده که می‌تواند الگوئی برای برنامه نویسی دانشجویان باشد.

# فصل دهم

## اسمبلی 80386

### هدف کلی

در این فصل ریزپردازنده 80386 و زبان اسمبلی وابسته بحث می گردد.

### اهداف رفتاری

پس از مطالعه این فصل با موارد زیر آشنا می شوید.

۱- ریزپردازنده 80386 .

۲- انواع data هائی که حمایت می نماید و محاسبه آدرس موثر

۳- معماری و ثباتهای آن.

۴- آشنائی با مجموعه دستورالعملهای 80386.



## ۱-۱۰- ریز پردازنده 80386

ریز پردازنده 80386 یک ریزپردازنده 32 بیتی می باشد که برای سیستم عاملهائی که عمل چند وظیفه ای (Multitasking) را انجام می دهند طراحی شده است. با توجه به ثباتهای 32 بیتی آن می تواند آدرسها و انواع داده های 32 بیتی را حمایت نماید. این ریزپردازنده قادر است تا چهار گیگابایت حافظه فیزیکی و 64 ترابایت (Terabyte) حافظه مجازی را آدرس دهی نماید. با توجه به پهنای باند بالای bus، امکانات استفاده pipelining، تراشه تبدیل آدرس، این ریزپردازنده زمان اجرای متوسط دستورالعملها را خیلی کم و به حداقل می رساند. این ریز پردازنده قادر است که 3 تا 4 میلیون دستورالعمل را در یک ثانیه اجرا نماید.

## ۲-۱۰- انواع داده ها

تراشه ریز پردازنده 80386 چندین نوع داده علاوه بر آنهایی که بوسیله 8086/80286 حمایت می شوند را پشتیبانی می نماید. این ریزپردازنده مقادیر صحیح با علامت و بدون علامت 32 بیتی را حمایت می نماید. همچنین اطلاعات بیتی از 1 تا 32 بیت را پشتیبانی می نماید. این ریزپردازنده انواع اشاره گرهای استاندارد استفاده شده بوسیله 8086/80286 و اشاره گرهای 32 و 48 بیتی را حمایت می نماید. بطور کلی 80386 از عملوندهای 8، 16، 32 بیتی می تواند استفاده نماید.

در ریزپردازنده 80386 بایستی به نکات ذیل توجه نمود.

الف) یک بایت از هشت بیت پیوسته تشکیل شده است.

ب) یک word از شانزده بیت پشت سر هم تشکیل شده است.

ج) یک double word از 32 بیت کنار هم تشکیل شده است.

د) یک quad word از 64 بیت پشت سر هم تشکیل شده است.

که بوسیله دیرکتیوهای زیر در زبان اسمبلی مورد استفاده قرار می گیرند.

|    |                    |
|----|--------------------|
| DB | Define byte        |
| DD | Define double word |
| DQ | Define quad word   |
| DW | Define word        |

### ۳-۱۰- محاسبه آدرس مؤثر (Effective Address)

زمانیکه می خواهیم به سگمنت بزرگتر از 64k دسترسی پیدا نمائیم آدرس مؤثر می تواند 32 بیتی باشد. آدرس مؤثر از مجموع ثبات مبنا، ثبات شاخص و یک جابجائی بدست می آید.

### ۴-۱۰- معماری

ریزپردازنده 80386 امکان استفاده از 32 ثبات که آنها را می توان به هفت دسته زیر تقسیم نمود را فراهم می نماید.

۱- ثباتهای مصرف عمومی (General-purpose registers).

۲- ثباتهای سگمنت (Segment registers).

۳- فلگها و اشاره گر دستورالعمل (Instruction pointer and flags).

۴- ثباتهای کنترل (Control registers).

۵- ثباتهای آدرس سیستم (System address registers).

۶- ثباتهای تست (Test registers).

ثباتهای مصرف عمومی امکان ذخیره 8، 16، 32 بیت داده را دارند.

تعداد این ثباتها هشت تا می باشد و اسامی آنها عبارتند از:

|     |               |
|-----|---------------|
| EAX | Accumulator   |
| EBX | Base          |
| ECX | Count         |
| EDX | Data          |
| ESP | Stack pointer |
| EBP | Base pointer  |

|     |                   |
|-----|-------------------|
| ESI | Source Index      |
| EDI | Destination Index |

|     |    |    |              |   |
|-----|----|----|--------------|---|
|     | 31 | 16 | 15           | 0 |
| EAX |    |    | [AH] AX [AL] |   |
| EBX |    |    | [BH] BX [BL] |   |
| ECX |    |    | [CH] CX [CL] |   |
| EDX |    |    | [DH] DX [DL] |   |

در حقیقت بیت‌های 0 تا 15 ثبات EAX همان ثبات AX می باشد که خود نیز به دو ثبات هشت بیتی AL و AH تقسیم می گردند. همینطور در مورد سایر ثباتهای EBX، ECX، EDX.

80386 دارای چهار ثبات 32 بیتی بنامهای ESP، EBP، ESI، EDI

می باشد که ثباتهای SP، BP، SI، DI بترتیب شانزده بیت اول این ثباتها می باشند

SP Stack Pointer  
BP Base Pointer  
SI Source Index  
DI Destination Index

|    |    |    |    |     |
|----|----|----|----|-----|
| 31 | 16 | 15 | 0  |     |
|    |    |    | SP | ESP |
|    |    |    | BP | EBP |
|    |    |    | SI | ESI |
|    |    |    | DI | EDI |

80386 دارای شش ثبات سگمنت 16 بیتی می باشد. اسامی این ثباتها عبارتند از:

CS Code Segment  
DS Data Segment  
SS Stack Segment  
ES Extra Segment  
FS  
GS

|    |    |   |
|----|----|---|
|    | 15 | 0 |
| CS |    |   |
| DS |    |   |
| SS |    |   |
| ES |    |   |
| FS |    |   |
| GS |    |   |

80368 دارای یک ثبات اشاره گر دستورالعمل می باشد که 32 بیتی است بنام EIP. محتوی ثبات EIP آدرس مؤثر دستورالعمل بعدی که بایستی اجرا شود می باشد. 16 بیت اول ثبات EIP بنام IP می باشد که بصورت جداگانه می توان به آن دسترسی پیدا نمود.

|     |    |    |    |   |
|-----|----|----|----|---|
|     | 31 | 16 | 15 | 0 |
| EIP |    |    |    |   |
|     |    |    | IP |   |

ثبات فلگ یا فلگ ثبات در ریزپردازنده 80386 به 32 بیت توسعه یافته است. ثبات فلگ بنام EFLAGS می باشد. البته بایستی توجه داشت که 16 بیت اول EFLAGS همان فلگ ثبات مطرح شده در فصلهای قبل می باشد.

|       |    |    |       |   |
|-------|----|----|-------|---|
|       | 31 | 16 | 15    | 0 |
| EFLAG |    |    |       |   |
|       |    |    | Flags |   |

ساختار ثبات EFLAGS در ذیل داده شده است.

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|    | NT | IO | PL | OF | DF | IF | TF | SF | ZF | // | AF | // | PF | // | CF |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    | VM | RF |

که

RF Resume Flag  
VM Virtual Mode Flag

ریزپردازنده 80386 دارای سه ثبات کنترل 32 بیتی بنامهای CR3, CR2, CR0 می باشد. ثبات CR0 شامل شش فلگ از قبل تعریف شده برای کنترل ریزپردازنده و وضعیت آن می باشد.

|     |    |    |    |   |
|-----|----|----|----|---|
|     | 31 | 16 | 15 | 0 |
| CR0 |    |    |    |   |
| CR2 |    |    |    |   |
| CR3 |    |    |    |   |

همچنین ریزپردازنده 80386 دارای چهار ثبات با کاربرد مخصوص بنامهای GDTR ، IDTR ، LDTR ، TR می باشد. که GDTR و IDTR ثباتهای 32 بیتی و LDTR ، TR ثباتهای شانزده بیتی می باشند.

|    |    |    |   |      |
|----|----|----|---|------|
| 31 | 16 | 15 | 0 |      |
|    |    |    |   | GDTR |
|    |    |    |   | IDTR |
|    |    |    |   | LDTR |
|    |    |    |   | TR   |

GETR Global descriptor table register  
IDTR Interrupt descriptor table register  
LDT Local descriptor table register  
TR Task state segment table register

## ۵-۱۰- دستورالعملهای 80386

بایستی توجه داشت که کلیه مطالبی که در مورد ریزپردازنده 80286 در فصلهای قبل مطرح شد توسط ریزپردازنده 80386 کاملاً می‌تواند بدون هیچگونه تغییری مورد استفاده قرار گیرند یعنی کلیه برنامه‌هایی که تاکنون نوشته شده می‌توان بوسیله ریزپردازنده 80386 اجرا نمود. ضمناً فهرستی از دستورالعملهای 80386 در انتهای این فصل داده شده است. بایستی توجه داشت که در ریزپردازنده 80386 عملوندها می‌توانند 32 بیتی باشند که اینکار در مورد 80286 امکان پذیر نبود.

مثال

```
ADD AX, BX
ADD EBX, ECX
```

که دومین دستورالعمل محتوی ECX که 32 بیتی می‌باشد با محتوی EBX که 32 بیتی می‌باشد جمع نموده نتیجه در EBX قرار می‌دهد.

مثال دیگر

```
ADD AL, 4
ADD EAX, 98765432H
```

که مقدار ثابت داده شده را با محتوی EAX جمع می‌نماید.

یا در مورد AND می‌توانیم دستورالعملهای ذیل را بکار ببریم.

```
AND AX, BX
AND EAX, ECX
```

در مورد دستورالعمل CBW، همانطوریکه قبلاً گفته می‌شود می‌توان برای تبدیل بایت به word استفاده نمود. برای این کار مقدار مورد نظر را در AL قرار می‌دهیم و پس از اجرای دستورالعمل CBW نتیجه در AX قرار می‌گیرد. اما برای

تبدیل word به double word کافی است که مقدار مورد نظر را در ثبات AX قرار داده پس از اجرای دستورالعمل CBW نتیجه در EAX قرار می گیرد. در مورد ADC می توانیم دستورالعملهای ذیل را استفاده نمائیم.

```
ADC AL , 4
ADC AX , 298
ADC EBX , 22334455H
ADC TABLE [SI] , 2
ADC NUMBE, 12345678
ADC DL , BL
ADC SI , X
ADC X, SI
```

در مورد CMP می توانیم از دستورالعملهای ذیل استفاده نمائیم:

```
CMP BL , 5
CMP EAX , 0FFFF0000H
CMP AL , 7
CMP AX , BX
CMP EDX , EAX
CMP EBX , Y
```

در مورد MOV از دستورالعملهای ذیل بعنوان مثال می توانیم استفاده نمائیم:

```
MOV AX , Y
MOV ECX , X
MOV X , AL
MOV BX , ES
MOV TABLE [BX] , SS
MOV DS , AX
MOV EAX , ECX
MOV CX, [BP][SI]
MOV EAX, 12345678H
```

در مورد دستورالعمل ضرب بعنوان مثال می توانیم از دستورالعمل ذیل استفاده نمائیم:

```
MOV EAX, 0FCAB1234H
MUL EBX
```

که نتیجه در EAX : EDX ذخیره می گردد.

در مورد NEG می توان از دستورالعملهای ذیل استفاده نمود:

|     |     |
|-----|-----|
| NEG | AL  |
| NEG | EBX |

در مورد NOT می توانیم بعنوان مثال دستورالعملهای ذیل را بکار ببریم:

|     |     |
|-----|-----|
| NOT | AL  |
| NOT | AX  |
| NOT | EBX |

در مورد PUSH و POP می توانیم بعنوان مثال از دستورالعملهای ذیل استفاده نمائیم:

|      |     |
|------|-----|
| POP  | CX  |
| POP  | EAX |
| POP  | SS  |
| PUSH | BX  |
| PUSH | EAX |



مجموعه کامل دستورالعملهای 80386

جدول ۱-۱۰

| Instruction | Meaning                          | Assembler Format |               |
|-------------|----------------------------------|------------------|---------------|
| AAA         | ASCII Adjust after Add           | AAA              |               |
| AAD         | ASCII Adjust before Divide       | AAD              |               |
| AAM         | ASCII Adjust after Multiply      | AAM              |               |
| AAS         | ASCII Adjust after Subtract      | AAS              |               |
| ADC         | Add with Carry                   | ADC              | Dest,src      |
| ADD         | Add                              | ADD              | Dest,src      |
| AND         | Logical AND                      | AND              | Dest,src      |
| ARPL        | Adjust RPL Field of Selector     | ARPL             | Sel,reg       |
| BOUND       | Check Array Bounds               | BOUND            | Reg,bound     |
| BSF         | Bit Scan Forward                 | BSF              | Dest,src      |
| BSR         | Bit Scan Reverse                 | BSR              | Dest,src      |
| BT          | Bit Test                         | BT               | Base,offset   |
| BTC         | Bit Test and complement          | BTC              | Base,offset   |
| BTR         | Bit Test and Reset               | BTR              | Base,offset   |
| BTS         | Bit Test and Set                 | BTS              | Base,offset   |
| CALL        | Call Procedure                   | CALL             | Dest          |
| CBW         | Convert Byte to Word             | CBW              |               |
| CDQ         | Convert Double Word to Quad Word | CDQ              |               |
| CLC         | Clear Carry Flag                 | CLC              |               |
| CLD         | Clear Direction Flag             | CLD              |               |
| CLI         | Clear Interrupt Flag             | CLI              |               |
| CLTS        | Clear Task-Switched Flag         | CLTS             |               |
| CMC         | Complement Carry Flag            | CMC              |               |
| CMP         | Compare                          | CMP              | Dest,src      |
| CMPS        | Compare Strings                  | CMPS             | Dest,src      |
| CWD         | Convert Word to Double Word*     | CWD              |               |
| CWDE        | Convert Word to Double Word*     | CWDE             |               |
| DAA         | Decimal Adjust after Add         | DAA              |               |
| DAS         | Decimal Adjust after Subtract    | DAS              |               |
| DEC         | Decrement by 1                   | DEC              | Dest          |
| DIV         | Unsigned Divide                  | DIV              | Acc,src       |
| ENTER       | Make Stack Frame for Procedure   | ENTER            | Storage,level |
| ESC         | Escape                           | ESC              |               |
| HLT         | Halt                             | HLT              |               |
| IDIV        | Signed Divide                    | IDIV             | Acc,src       |
| IMUL        | Signed Multiply                  | IMUL             | Acc,src       |
| IN          | Input from Port                  | IN               | Acc,port      |
| INC         | Increment by 1                   | INC              | Dest          |
| INT         | Software Interrupt (Trap)        | INT              | Inttype       |
| INTO        | Interrupt If Overflow            | INTO             |               |
| IRET        | Interrupt Return                 | IRET             |               |

| Instruction | Meaning                      | Assembler Format |         |
|-------------|------------------------------|------------------|---------|
| JA          | Jump If Above                | JA               | Dest    |
| JAE         | Jump If Above or Equal       | JAE              | Dest    |
| JB          | Jump If Below                | JB               | Dest    |
| JBE         | Jump If Below or Equal       | JBE              | Dest    |
| JC          | Jump If Carry                | JC               | Dest    |
| JCXZ        | Jump If CX Is Zero           | JCXZ             | Dest    |
| JE          | Jump If Equal                | JE               | Dest    |
| JECXZ       | Jump If ECX Is Zero          | JECXZ            | Dest    |
| JG          | Jump If Greater              | JG               | Dest    |
| JGE         | Jump If Greater or Equal     | JGE              | Dest    |
| JL          | Jump If Less                 | JL               | Dest    |
| JLE         | Jump If Less or Equal        | JLE              | Dest    |
| JMP         | Jump Unconditionally         | JMP              | Dest    |
| JNA         | Jump If Not Above            | JNA              | Dest    |
| JNAE        | Jump If Not Above or Equal   | JNAE             | Dest    |
| JNB         | Jump If Not Below            | JNB              | Dest    |
| JNBE        | Jump If Not Below or Equal   | JNBE             | Dest    |
| JNC         | Jump If No Carry             | JNC              | Dest    |
| JNE         | Jump If Not Equal            | JNE              | Dest    |
| JNG         | Jump If Not Greater          | JNG              | Dest    |
| JNGE        | Jump If Not Greater or Equal | JNGE             | Dest    |
| JNL         | Jump If Not Less             | JNL              | Dest    |
| JNLE        | Jump If Not Less or Equal    | JNLE             | Dest    |
| JNO         | Jump if No Overflow          | JNO              | Dest    |
| JNP         | Jump if Parity Odd           | JNP              | Dest    |
| JNS         | Jump if Sign Positive        | JNS              | Dest    |
| JNZ         | Jump if Not Zero             | JNZ              | Dest    |
| JO          | Jump if Overflow             | JO               | Dest    |
| JP          | Jump if parity Even          | JP               | Dest    |
| JPE         | Jump if parity Even          | JPE              | Dest    |
| JPO         | Jump if parity Odd           | JPO              | Dest    |
| JS          | Jump if Sign Negative        | JS               | Dest    |
| JZ          | Jump if Zero                 | JZ               | Dest    |
| LAHF        | Load Flags into AH Register  | LAHF             |         |
| LAR         | Load Access Rights Byte      | LAR              | Reg,src |
| LDS         | Load DS Register             | LDS              | Reg,src |
| LEA         | Load Effective Address       | LEA              | Reg,src |
| LEAVE       | Leave Procedure              | LEAVE            |         |
| LES         | Load ES Register             | LES              | Reg,src |
| LFS         | Load FS Register             | LFS              | Reg,src |
| LGS         | Load GS Register             | LGS              | Reg,src |
| LGDT        | Load GDT Register            | LGDT             | Src     |
| LIDT        | Load IDT Register            | LIDT             | Src     |

| Instruction | Meaning                    | Assembler Format |            |
|-------------|----------------------------|------------------|------------|
| LLDT        | Load LDT Register          | LLDT             | Src        |
| LMSW        | Load Machine Status Word   | LMSW             | Src        |
| LOCK        | Lock Bus                   | LOCK             |            |
| LODS        | Load String                | LODS             | Src        |
| LOOP        | Loop with CX Counter       | LOOP             | Dest       |
| LOOPE       | Loop If Equal              | LOOPE            | Dest       |
| LOOPNE      | Loop If Not Equal          | LOOPNE           | Dest       |
| LOOPNZ      | Loop If Not Zero           | LOOPNZ           | Dest       |
| LOOPZ       | Loop If Zero               | LOOPZ            | Dest       |
| LSL         | Load Segment Limit         | LSL              | Reg,src    |
| LSS         | Load SS Register           | LSS              | Reg,src    |
| LTR         | Load Task Register         | LTR              | Src        |
| MOV         | Move Data                  | MOV              | Dest,src   |
| MOV         | Move to/from Special Regs  | MOV              | Dest,src   |
| MOVS        | Move String                | MOVS             | Dest,src   |
| MOVSB       | Move with Sign-Extend      | MOVSB            | Reg,src    |
| MOVZB       | Move with Zero-Extend      | MOVZB            | Reg,src    |
| MUL         | Unsigned Multiply          | MUL              | Acc,src    |
| NEG         | 2's Complement Negation    | NEG              | Dest       |
| NOP         | No Operation               | NOP              |            |
| NOT         | 1's Complement Negation    | NOT              | Dest       |
| OR          | Logical Inclusive OR       | OR               | Dest,src   |
| OUT         | Output to Port             | OUT              | Port,acc   |
| OUTS        | Output String              | OUTS             | DX,src     |
| POP         | Pop Operand off Stack      | POP              | Dest       |
| POPA        | Pop All General Registers  | POPA             |            |
| POPF        | Pop Flags off Stack        | POPF             |            |
| PUSH        | Pop Operand Onto Stack     | PUSH             | Src        |
| PUSHA       | Push All General Registers | PUSHA            |            |
| PUSHF       | Push Flags onto Stack      | PUSHF            |            |
| RCL         | Rotate Left through Carry  | RCL              | Dest,count |
| RCR         | Rotate Right through Carry | RCR              | Dest,count |
| REP         | Repeat                     | REP              |            |
| REPE        | Repeat while Equal         | REPE             |            |
| REPNE       | Repeat while Not Equal     | REPNE            |            |
| REPZ        | Repeat while Not Zero      | REPZ             |            |
| REPZ        | Repeat while Zero          | REPZ             |            |
| RET         | Repeat from Procedure      | RET              |            |
| ROL         | Rotate Left                | ROL              | Dest,count |
| ROR         | Rotate Right               | ROR              | Dest,count |
| SAHF        | Store AH Register in Flags | SAHF             |            |
| SAL         | Shift Arithmetic Left      | SAL              | Dest,count |
| SAR         | Shift Arithmetic Right     | SAR              | Dest,count |

| Instruction | Meaning                      | Assembler Format |                 |
|-------------|------------------------------|------------------|-----------------|
| SBB         | Subtract with Borrow         | SBB              | Dest,src        |
| SCAS        | Compare String               | SCAS             | Dest            |
| SETcc       | Set Byte on Condition        | SETcc            | Dest            |
| SGDT        | Store GDT Register           | SGDT             | Dest            |
| SHL         | Shift Logical Left           | SHL              | Dest, count     |
| SHLD        | Double Precision Shift Left  | SHLD             | Dest, src,count |
| SHR         | Shift Logical Right          | SHR              | Dest,count      |
| SHRD        | Double Precision Shift Right | SHRD             | Dest,src,count  |
| SIDT        | Store IDT Register           | SIDT             | Dest            |
| SLDT        | Store LDT Register           | SLDT             | Dest            |
| SMSW        | Store Machine Status Word    | SMSW             | Dest            |
| STC         | Set Carry Flag               | STC              |                 |
| STD         | Set Direction Flag           | STD              |                 |
| STI         | Set Interrupt Flag           | STI              |                 |
| STOS        | Store String                 | STOS             | Dest            |
| STR         | Store Task Register          | STR              | Dest            |
| SUB         | Subtract                     | SUB              | Dest,src        |
| TEST        | Logical Compare              | TEST             | Dest,src        |
| VERR        | Verify Segment for Reading   | VERR             | Sel             |
| VERW        | Verify Segment for Writing   | VERW             | Sel             |
| WAIT        | Wait until BUSY#Negated      | WAIT             |                 |
| XCHG        | Exchange Operand, Register   | XCHG             | Dest,src        |
| XLAT        | Table Lookup                 | XLAT             | Source-table    |
| XOR         | Logical Exclusive OR         | XOR              | Dest,src        |

\*CWD sign extends register AX into registers DX and AX, whereas CWDE sign extends AX into EAX.

جدول ۱۰-۲

|                 |                                    |
|-----------------|------------------------------------|
| DEC             | Subtract 1                         |
| INC             | Add 1                              |
| NOT             | Logical NOT (complement or invert) |
| ROL             | Rotate left                        |
| ROR             | Rotate right                       |
| SBB             | Subtract with borrow               |
| SHL             | Shift logical left                 |
| SHR             | Shift logical right                |
| SUB             | Subtract                           |
| TEST            | Bit test                           |
| Program control |                                    |
| CALL            | Call subroutine                    |
| INT             | Interrupt (trap)                   |
| JA              | Jump if above                      |
| JAE             | Jump if above or equal             |
| JB              | Jump if below                      |
| JBE             | Jump if below or equal             |
| JC              | Jump if carry                      |
| JE              | Jump if equal                      |
| JMP             | Jump unconditionally               |
| JNC             | Jump if not carry                  |
| JNE             | Jump if not equal                  |
| JNS             | Jump if not sign                   |
| JNZ             | Jump if non zero                   |
| JS              | Jump if sign                       |
| JZ              | Jump if zero                       |
| RET             | Return from subroutine             |

## جدول ۱۰-۳

## Frequently Used 80386 Instructions

|      |                        |
|------|------------------------|
| ADC  | Add with carry         |
| ADD  | Add                    |
| AND  | Logical AND            |
| CALL | Call subroutine        |
| CMP  | Compare                |
| DEC  | Subtract 1             |
| IN   | Input                  |
| INC  | Add 1                  |
| INT  | Interrupt (trap)       |
| JA   | Jump if above          |
| JAE  | Jump if above or equal |
| JB   | Jump if below          |
| JBE  | Jump if below or equal |
| JC   | Jump if carry          |
| JE   | Jump if equal          |
| JMP  | Jump unconditionally   |
| JNC  | Jump if not carry      |
| JNE  | Jump if not equal      |
| JNS  | Jump if sign positive  |
| JNZ  | Jump if not zero       |
| JS   | Jump if sign negative  |
| LEA  | Load effective address |
| MOV  | Move                   |
| NOT  | Logical NOT            |
|      | (complement or invert) |
| OUT  | Output                 |
| POP  | Load from stack        |
| PUSH | Store on stack         |
| RET  | Return from subroutine |
| ROL  | Rotate left            |
| ROR  | Rotate right           |
| SBB  | Subtract with borrow   |

## مروری بر مطالب فصل

در این فصل در مورد ریز پردازنده ۸۰۳۸۶ بحث گردید. همانطوری که گفته شد این ریز پردازنده یک ریزپردازنده ۳۲ بیتی می باشد و تمام دستورالعمل هایی که روی ریزپردازنده ۸۰۲۸۶ اجرا می شود روی این ریزپردازنده نیز قابل اجرا می باشد. ثبات های ۳۲ بیتی آن عبارتند از EAX ، EBX ، ECX ، EDI . در ضمن می توان از ثبات های AX ، BX ، CX ، DX ، AL ، AH ، BL ، BH ، CL ، CH ، DH ، DL و ... استفاده نمود. در نهایت جدول کلی فرمت دستورالعملها برای ریزپردازنده ۸۰۳۸۶ داده شده است که با مرور آن می توان برنامه های مختلفی را به راحتی نوشت و بر روی این ریزپردازنده اجرا نمود

## ضمیمه ۱

## عملگرها (OPERATORS)

| Operator          | Function                                                                          |
|-------------------|-----------------------------------------------------------------------------------|
| <b>Arithmetic</b> |                                                                                   |
| +                 | Format: value1 + value2<br>Adds value1 and value2.                                |
| -                 | Format: value1 – value2<br>Subtracts value2 from value1.                          |
| *                 | Format: value1 * value2<br>Multiplies value2 by value1.                           |
| /                 | Format: value1 / value2<br>Divides value1 by value2, and returns the quotient.    |
| <b>MOD</b>        | Format: value1 MOD value2<br>Divides value1 by value2, and returns the Remainder. |
| <b>SHL</b>        | Format: value SHL expression<br>Shifts value left by expression bit positions.    |
| <b>SHR</b>        | Format: value SHR expression<br>Shifts value right by expression bit positions.   |



| Operator                 | Function                                                                                              |
|--------------------------|-------------------------------------------------------------------------------------------------------|
| <b>Logical</b>           |                                                                                                       |
| <b>AND</b>               | Format: value1 AND value2<br>Takes logical AND of value1 and value2.                                  |
| <b>OR</b>                | Format: value1 OR value2<br>Takes logical inclusive-OR of value1 and value2.                          |
| <b>XOR</b>               | Format: value1 XOR value2<br>Takes logical exclusive-OR of value1 and Value2.                         |
| <b>NOT</b>               | Format: NOT value<br>Reverses the state of each bit in value; that is, it takes the one's complement. |
| <b>Relational</b>        |                                                                                                       |
| <b>EQ</b>                | Format: operand1 EQ operand2<br>True if the two operands are identical.                               |
| <b>NE</b>                | Format: operand1 NE operand2<br>True if the two operands are not identical.                           |
| <b>LT</b>                | Format: operand1 LT operand2<br>True if operand1 is less than operand2.                               |
| <b>GT</b>                | Format: operand1 GT operand2<br>True if operand1 is greater than operand2.                            |
| <b>LE</b>                | Format: operand1 LE operand2<br>True if operand1 is less than or equal to operand2.                   |
| <b>GE</b>                | Format: operand1 GE operand2<br>True if operand1 is greater than or equal to operand2.                |
| <b>Value - Returning</b> |                                                                                                       |
| <b>\$</b>                | Format: \$<br>Returns the current value of the location counter.                                      |
| <b>SEG</b>               | Format: SEG variable<br>Or<br>SEG label<br>Returns the segment value of variable or label.            |

| Operator                           | Function                                                                                                                                                                                                                                                                           |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OFFSET</b>                      | Format: OFFSET variable<br>Or<br>OFFSET label<br>Returns the offset value of variable or label.                                                                                                                                                                                    |
| <b>LENGTH</b>                      | Format: LENGTH variable<br>Returns the length in units (bytes or words) for any variable defined using DUP.                                                                                                                                                                        |
| <b>TYPE</b>                        | Format: TYPE variable<br>Or<br>TYPE label<br>For variables, TYPE returns 1 (BYTE), 2(WORD), or 4 (DOUBLEWORD). For labels, it returns -1 (NEAR) or-2(FAR).                                                                                                                         |
| <b>Value-Returning<br/>SIZE</b>    | Format: SIZE variable<br>Returns the product of LENGTH times TYPE.                                                                                                                                                                                                                 |
| <b>Attribute</b>                   | Format: type PTR expression                                                                                                                                                                                                                                                        |
| <b>PTR</b>                         | Overrides the type (BYTE or WORD) or distance (NEAR or FAR) of a memory address operand. Type is the new attribute and expression is the identifier whose attribute is to to be overridden.                                                                                        |
| <b>DS:<br/>ES:<br/>SS:<br/>CS:</b> | Format: seg –reg: addr-expr<br>or<br>seg-reg: label<br>or<br>seg-reg: variable<br>Overrides the segment attribute of a label, variable, or address expression.                                                                                                                     |
| <b>SHORT</b>                       | Format: JMP SHORT labe1<br>Tells the assembler that the JMP target label is no farther than 127 bytes past the next instruction.                                                                                                                                                   |
| <b>THIS</b>                        | Format: THIS attribute<br>or<br>THIS type<br>Creates a memory address operand of either distance attribute (NEAR or FAR) or either type attribute (BYTE or WORD) at an offset equal to the current value of the location counter and a segment attribute of the enclosing segment. |

| Operator         | Function                                                                                                                    |
|------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>HIGH</b>      | Format: HIGH value<br>or<br>HIGH expression<br>Returns the high-order byte of a 16-bit numeric value or address expression. |
| <b>Attribute</b> |                                                                                                                             |
| <b>LOW</b>       | Format: LOW value<br>or<br>LOW expression<br>Returns the low-order byte of a 16-bit numeric value or address expression.    |

## ضمیمه شماره ۲

## Instruction Set Summary

For each instruction, it shows the general assembler format and which flags are affected. In the Flags column, -means unchanged, \* means may have changed, and? Means undefined.

| mnemonic | Assembler | format                    | Flags |    |    |    |    |    |    |    |    |
|----------|-----------|---------------------------|-------|----|----|----|----|----|----|----|----|
|          |           |                           | OF    | DF | IF | TF | SF | ZF | AF | PF | CF |
| AAA      | AAA       |                           | ?     | -  | -  | -  | ?  | ?  | *  | ?  | *  |
| AAD      | AAD       |                           | ?     | -  | -  | -  | *  | *  | ?  | *  | ?  |
| AAM      | AAM       |                           | ?     | -  | -  | -  | *  | *  | ?  | *  | ?  |
| AAS      | AAS       |                           | ?     | -  | -  | -  | ?  | ?  | *  | ?  | *  |
| ADC      | ADC       | Destination,source        | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| ADD      | ADD       | Destination,source        | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| AND      | AND       | Destination,source        | 0     | -  | -  | -  | *  | *  | ?  | *  | 0  |
| BOUND    | BOUND     | Reg16,source              | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| CALL     | CALL      | Target                    | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| CBW      | CBW       |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| CLC      | CLC       |                           | -     | -  | -  | -  | -  | -  | -  | -  | 0  |
| CLD      | CLD       |                           | -     | 0  | -  | -  | -  | -  | -  | -  | -  |
| CLI      | CLI       |                           | -     | -  | 0  | -  | -  | -  | -  | -  | -  |
| CMC      | CMC       |                           | -     | -  | -  | -  | -  | -  | -  | -  | *  |
| CMP      | CMP       | Destination,source        | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| CMPS     | CMPS      | Dest-string,source-string | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| CMPSB    | CMPSB     |                           | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| CMPSW    | CMPSW     |                           | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| CWD      | CWD       |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| DAA      | DAA       |                           | ?     | -  | -  | -  | *  | *  | *  | *  | *  |
| DAS      | DAS       |                           | ?     | -  | -  | -  | *  | *  | *  | *  | *  |
| DEC      | DEC       | Destination               | *     | -  | -  | -  | *  | *  | *  | *  | -  |
| DIV      | DIV       | Source                    | ?     | -  | -  | -  | ?  | ?  | ?  | ?  | ?  |
| ENTER    | ENTER     | Immed16,level             | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| ESC      | ESC       | Ext,opcode,source         | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| HLT      | HLT       |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| IDIV     | IDIV      | source                    | ?     | -  | -  | -  | ?  | ?  | ?  | ?  | ?  |
| IMUL     | IMUL      | Source                    | *     | -  | -  | -  | ?  | ?  | ?  | ?  | *  |
| IMUL     | IMUL      | Dest[,source],immed       | *     | -  | -  | -  | ?  | ?  | ?  | ?  | *  |
| IN       | IN        | Accumulator,port          | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| INC      | INC       | Destination               | *     | -  | -  | -  | *  | *  | *  | *  | -  |
| INS      | INS       | Dest-string,DX            | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| INT      | INT       | Interrupt-type            | -     | -  | 0  | 0  | -  | -  | -  | -  | -  |
| INTO     | INTO      |                           | -     | -  | 0  | 0  | -  | -  | -  | -  | -  |

| mnemonic    | Assembler | format                    | Flags |    |    |    |    |    |    |    |    |
|-------------|-----------|---------------------------|-------|----|----|----|----|----|----|----|----|
|             |           |                           | OF    | DF | IF | TF | SF | ZF | AF | PF | CF |
| IRET        | IRET      |                           | *     | *  | *  | *  | *  | *  | *  | *  | *  |
| JA/JNBE     | JA        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JAE/JNB     | JAE       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JB/JNAE/JC  | JB        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JBE/JNA     | JBE       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JCXZ        | JCXZ      | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JE/JZ       | JE        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JG/JNLE     | JG        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JGE/JNL     | JGE       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JL/JNGE     | JL        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JLE/JNG     | JLE       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JMP         | JMP       | Target                    | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JNC         | JNC       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JNE/JNZ     | JNE       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JNO         | JNO       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JNP/JPO     | JNP       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JNS         | JNS       | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JO          | JO        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JP/JPE      | JP        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| JS          | JS        | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LAHF        | LAHF      |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LDS         | LDS       | Reg16,mem32               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LEA         | LEA       | Reg16,mem16               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LEAVE       | LEAVE     |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LES         | LES       | Reg16,mem32               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LOCK        | LOCK      |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LODS        | LODS      | Source-string             | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LODSB       | LODSB     |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LODSW       | LODSW     |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LOOP        | LOOP      | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LOOPE/LOOPZ | LOOPE     | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| LOOPNE/LOO  | LOOPNE    | Short-label               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| PNZ         |           |                           |       |    |    |    |    |    |    |    |    |
| MOV         | MOV       | Destination, source       | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| MOVS        | MOVS      | Dest-string,source-string | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| MOVSB       | MOVSB     |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| MOVSW       | MOVSW     |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| MUL         | MUL       | Source                    | *     | -  | -  | -  | ?  | ?  | ?  | ?  | *  |
| NEG         | NEG       | Destination               | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| NOP         | NOP       |                           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| NOT         | NOT       | Destination               | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| OR          | OR        | Destination, source       | 0     | -  | -  | -  | *  | *  | ?  | *  | 0  |
| OUT         | OUT       | Port, accumulator         | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| OUTS        | OUTS      | DX, source-string         | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| POP         | POP       | destination               | -     | -  | -  | -  | -  | -  | -  | -  | -  |

| mnemonic    | Assembler   | format              | Flags |    |    |    |    |    |    |    |    |
|-------------|-------------|---------------------|-------|----|----|----|----|----|----|----|----|
|             |             |                     | OF    | DF | IF | TF | SF | ZF | AF | PF | CF |
| POPA        | POPA        |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| POPF        | POPF        |                     | *     | *  | *  | *  | *  | *  | *  | *  | *  |
| PUSH        | PUSH        | Source              | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| PUSH        | PUSH        | Immediate           | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| PUSHA       | PUSHA       |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| PUSHF       | PUSHF       |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| RCL/RCR     | RCL         | Destination,1       | *     | -  | -  | -  | -  | -  | -  | -  | *  |
| RCL/RCR     | RCL         | Destination, CL     | ?     | -  | -  | -  | -  | -  | -  | -  | *  |
| RCL/RCR     | RCL         | Destination, count  | ?     | -  | -  | -  | -  | -  | -  | -  | *  |
| REP         | REP         |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| REPE/REPZ   | REPE        |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| REPNE/REPNZ | REPNE       |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| RET         | [pop-value] |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| ROL/ROR     | ROL         | Destination, 1      | *     | -  | -  | -  | -  | -  | -  | -  | *  |
| ROL/ROR     | ROL         | Destination, CL     | ?     | -  | -  | -  | -  | -  | -  | -  | *  |
| ROL/ROR     | ROL         | Destination, count  | ?     | -  | -  | -  | -  | -  | -  | -  | *  |
| SAHF        | SAHF        |                     | -     | -  | -  | -  | *  | *  | *  | *  | *  |
| SAL/SHL     | SAL         | Destination, 1      | *     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SAL/SHL     | SAL         | Destination, CL     | ?     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SAL/SHL     | SAL         | Destination, count  | ?     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SAR         | SAR         | Destination, 1      | 0     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SAR         | SAR         | Destination, CL     | ?     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SAR         | SAR         | Destination, count  | ?     | -  | -  | -  | *  | *  | ?  | *  | *  |
| SBB         | SBB         | Destination, source | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| SCAS        | SCAS        | Dest-string         | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| SCASB       | SCASB       |                     | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| SCASW       | SCASW       |                     | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| SHR         | SHR         | Destination,1       | *     | -  | -  | -  | 0  | *  | ?  | *  | *  |
| SHR         | SHR         | Destination, CL     | ?     | -  | -  | -  | 0  | *  | ?  | *  | *  |
| SHR         | SHR         | Destination, count  | ?     | -  | -  | -  | 0  | *  | ?  | *  | *  |
| STC         | STC         |                     | -     | -  | -  | -  | -  | -  | -  | -  | 1  |
| STD         | STD         |                     | -     | 1  | -  | -  | -  | -  | -  | -  | -  |
| STI         | STI         |                     | -     | -  | 1  | -  | -  | -  | -  | -  | -  |
| STOS        | STOS        | Dest-string         | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| STOSB       | STOSB       |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| STOSW       | STOSW       |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| SUB         | SUB         | Destination, source | *     | -  | -  | -  | *  | *  | *  | *  | *  |
| TEST        | TEST        | Destination, source | 0     | -  | -  | -  | *  | *  | ?  | *  | 0  |
| WAIT        | WAIT        |                     | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| XCHG        | XCHG        | Destination, source | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| XLAT        | XLAT        | Source-table        | -     | -  | -  | -  | -  | -  | -  | -  | -  |
| XOR         | XOR         | Destination, source | 0     | -  | -  | -  | *  | *  | ?  | *  | 0  |

## ضمیمه شماره ۳

### Instruction times

| Instruction        |                        | Clocks | Bytes |
|--------------------|------------------------|--------|-------|
| AAA                |                        | 3      | 1     |
| AAD                |                        | 14     | 2     |
| AAM                |                        | 16     | 1     |
| AAS                |                        | 3      | 1     |
| ADC                | Register, Register     | 2      | 2     |
| ADC                | Register, memory       | 7*     | 2-4   |
| ADC                | Memory, register       | 7*     | 2-4   |
| ADC                | Register, immediate    | 3      | 3-4   |
| ADC                | Memory, immediate      | 7*     | 3-6   |
| ADC                | Accumulator; immediate | 3      | 2-3   |
| ADD                | Register, register     | 2      | 2     |
| ADD                | Register, memory       | 7*     | 2-4   |
| ADD                | Memory, register       | 7*     | 2-4   |
| ADD                | Register, immediate    | 3      | 3-4   |
| ADD                | Memory, immediate      | 7*     | 3-6   |
| ADD                | Accumulator; immediate | 3      | 2-3   |
| AND                | Register, register     | 2      | 2     |
| AND                | Register, memory       | 7*     | 2-4   |
| AND                | Memory, register       | 7*     | 2-4   |
| AND                | Register, immediate    | 3      | 3-4   |
| AND                | Memory, immediate      | 7*     | 3-6   |
| AND                | Accumulator; immediate | 3      | 2-3   |
| BOUND reg16,source |                        | 13*    | 2     |
| CALL               | Near-proc              | 7+m    | 3     |
| CALL               | Far-proc               | 13+m   | 5     |
| CALL               | Memptr16               | 11+m*  | 2-4   |
| CALL               | Regptr16               | 7+m    | 2     |
| CALL               | Memptr32               | 16+m   | 2-4   |

| Instruction |                                   | Clocks   | Bytes |
|-------------|-----------------------------------|----------|-------|
| CBW         |                                   | 2        | 1     |
| CLC         |                                   | 2        | 1     |
| CLD         |                                   | 2        | 1     |
| CLI         |                                   | 3        | 1     |
| CMC         |                                   | 2        | 1     |
| CMP         | Register, register                | 2        | 2     |
| CMP         | Register, memory                  | 6*       | 2-4   |
| CMP         | Memory, register                  | 7*       | 2-4   |
| CMP         | Register, immediate               | 3        | 3-4   |
| CMP         | Memory, immediate                 | 6*       | 3-6   |
| CMP         | Accumulator; immediate            | 3        | 2-3   |
| CMPS        | Dest-string, source-string        | 8        | 1     |
| CMPS        | (repeat)dest-string,source-string | 5+9(rep) | 1     |
| CWD         |                                   | 2        | 1     |
| DAA/DAS     |                                   | 3        | 1     |
| DEC         | Register                          | 2        | 1-2   |
| DEC         | Memory                            | 7*       | 2-4   |
| DIV         | Reg8                              | 14       | 2     |
| DIV         | Reg16                             | 22       | 2     |
| DIV         | Mem8                              | 17*      | 2-4   |
| DIV         | Mem16                             | 25*      | 2-4   |
| ENTER       | Immed16,0                         | 11       | 4     |
| ENTER       | Immed16,1                         | 15       | 4     |
| ENTER       | Immed16,level                     | 12+4(L)  | 4     |
| ESC         | Immediate, memory                 | 9-20*    | 2-4   |
| ESC         | Immediate, register               | 2        | 2     |
| HLT         |                                   | 2        | 1     |
| IDIV        | Reg8                              | 17       | 2     |
| IDIV        | Reg16                             | 25       | 2     |



| Instruction                                    |                             | Clocks    | Bytes |
|------------------------------------------------|-----------------------------|-----------|-------|
| IDIV                                           | Mem8                        | 20*       | 2-4   |
| IDIV                                           | Mem16                       | 28*       | 2-4   |
| IMUL                                           | Reg8                        | 13        | 2     |
| IMUL                                           | Reg16                       | 21        | 2     |
| IMUL                                           | Mem8                        | 16*       | 2-4   |
| IMUL                                           | Mem16                       | 24*       | 2-4   |
| IMUL                                           | Dest-reg,reg16,immediate    | 21*       | 3-4   |
| IMUL                                           | Dest-reg, memory, immediate | 24*       | 3-4   |
| IN                                             | Accumulator, immed8         | 5         | 2     |
| IN                                             | Accumulator, DX             | 5         | 1     |
| INC                                            | Register                    | 2         | 1-2   |
| INC                                            | Memory                      | 7*        | 2-4   |
| INS                                            | Dest-string, DX             | 5         | 1     |
| INS                                            | (rep) dest-string,DX        | 5+4(rep)  | 1     |
| INT                                            | Immed8                      | 23+m      | 1-2   |
| INTO                                           |                             | 24+m or 3 | 1     |
| IRET                                           |                             | 17+M      | 1     |
| All conditional jump instructions except JCXZ: |                             |           |       |
| Jcc                                            | Short-label                 | 7+m or 3  | 2     |
| JCXZ                                           | Short-label                 | 8+m or 4  | 2     |
| JMP                                            | Short-label                 | 7+m       | 2     |
| JMP                                            | Near-label                  | 7+m       | 3     |
| JMP                                            | Far-label                   | 11+m      | 5     |
| JMP                                            | Memptr16                    | 11+m*     | 2-4   |
| JMP                                            | Regptr16                    | 7+m       | 2     |
| JMP                                            | Memptr32                    | 15+m*     | 2-4   |
| LAHF                                           |                             | 2         | 1     |
| LDS                                            | Reg16, mem32                | 7*        | 2-4   |
| LEA                                            | Reg16,mem16                 | 3*        | 2-4   |

| Instruction   |                                    | Clocks   | Bytes |
|---------------|------------------------------------|----------|-------|
| LEAVE         |                                    | 5        | 1     |
| LES           | Reg16, mem32                       | 7*       | 2-4   |
| LOCK          |                                    | 0        | 1     |
| LODS          | Source-string                      | 5        | 1     |
| LODS          | (repeat)source-string              | 5+4(rep) | 1     |
| LOOP          | Short-label                        | 8+m or 4 | 2     |
| LOOPE/LOOPZ   | Short-label                        | 8+m or 4 | 2     |
| LOOPNE/LOOPNZ | Short-label                        | 8+m or 4 | 2     |
| MOV           | Memory, accumulator                | 3        | 3     |
| MOV           | Accumulator, memory                | 5        | 3     |
| MOV           | Register, register                 | 2        | 2     |
| MOV           | Register, memory                   | 5*       | 2-4   |
| MOV           | Memory, register                   | 3*       | 2-4   |
| MOV           | Register, immediate                | 2        | 2-3   |
| MOV           | Memory, immediate                  | 3*       | 3-6   |
| MOV           | Seg-reg, reg16                     | 2        | 2     |
| MOV           | Seg-reg-mem16                      | 5*       | 2-4   |
| MOV           | Reg16, seg-reg                     | 2        | 2     |
| MOV           | Memory, seg-reg                    | 3*       | 2-4   |
| MOVS          | Dest-string, source-string         | 5        | 1     |
| MOVS          | (repeat)dest-string, source-string | 5+4(rep) | 1     |
| MUL           | Reg8                               | 13       | 2     |
| MUL           | Reg16                              | 21       | 2     |
| MUL           | Mem8                               | 16*      | 2-4   |
| MUL           | Mem16                              | 24*      | 2-4   |
| NEG           | Register                           | 2        | 2     |
| NEG           | Memory                             | 7*       | 2-4   |
| NOP           |                                    | 2        | 1     |
| NOT           | Register                           | 2        | 2     |
| NOT           | Memory                             | 7*       | 2-4   |
| OR            | Register, register                 | 2        | 2     |
| OR            | Register, memory                   | 7*       | 2-4   |

| Instruction     |                        | Clocks   | Bytes |
|-----------------|------------------------|----------|-------|
| OR              | Memory, register       | 7*       | 2-4   |
| OR              | Register, immediate    | 3        | 3-6   |
| OR              | Memory, immediate      | 7*       | 3-6   |
| OR              | Accumulator; immediate | 3        | 2-3   |
| OUT             | Immed8, accumulator    | 3        | 2     |
| OUT             | DX, accumulator        | 3        | 1     |
| OUTS            | DX,source-string       | 5        | 1     |
| OUTS            | (rep)DX, source-string | 5+4(rep) | 1     |
| POP             | Register               | 5        | 1     |
| POP             | Memory                 | 5*       | 2-4   |
| POPA            |                        | 19       | 1     |
| POPE            |                        | 5        | 1     |
| PUSH            | Register               | 3        | 1     |
| PUSH            | Memory                 | 5*       | 2-4   |
| PUSH            | Immediate              | 3        | 2-3   |
| PUSHA           |                        | 17       | 1     |
| PUSHF           |                        | 3        | 1     |
| RCL/RCR/ROL/ROR | register,1             | 2        | 2     |
| RCL/RCR/ROL/ROR | register, CL           | 5+1/bit  | 2     |
| RCL/RCR/ROL/ROR | memory,1               | 7*       | 2-4   |
| RCL/RCR/ROL/ROR | memory, CL             | 8*+1/bit | 2-4   |
| RCL/RCR/ROL/ROR | reg, count             | 5+1/bit  | 3     |
| RCL/RCR/ROL/ROR | memory, count          | 8*+1/bit | 3-5   |
| REP             |                        | 0        | 1     |
| REPE/REPZ       |                        | 0        | 1     |
| REPNE/REPZ      |                        | 0        | 1     |
| RET             | (near, no pop)         | 11+M     | 1     |
| RET             | (near, pop)            | 11+M     | 3     |
| RET             | (far, no pop)          | 15+M     | 1     |
| RET             | (far, pop)             | 15+M     | 3     |
| SAHF            |                        | 2        | 1     |

| Instruction                   |                        | Clocks   | Bytes |
|-------------------------------|------------------------|----------|-------|
| SAL/SHL/SAR/SHR register, 1   |                        | 2        | 2     |
| SAL/SHL/SAR/SHR register, CL  |                        | 5+1/bit  | 2     |
| SAL/SHL/SAR/SHR memory, 1     |                        | 7*       | 2-4   |
| SAL/SHL/SAR/SHR memory, CL    |                        | 8*+1/bit | 2-4   |
| SAL/SHL/SAR/SHR reg, count    |                        | 5+1/bit  | 3     |
| SAL/SHL/SAR/SHR memory, count |                        | 8*+1/bit | 3-5   |
|                               |                        |          |       |
| SBB                           | Register, register     | 2        | 2     |
| SBB                           | Register, memory       | 7*       | 2-4   |
| SBB                           | Memory, register       | 7*       | 2-4   |
| SBB                           | Register, immediate    | 3        | 3-4   |
| SBB                           | Memory, immediate      | 7*       | 3-6   |
| SBB                           | Accumulator; immediate | 3        | 2-3   |
|                               |                        |          |       |
| SCAS                          | Dest-string            | 7        | 1     |
| SCAS                          | (repeat) dest-string   | 5+8(rep) | 1     |
|                               |                        |          |       |
| STC/STD/STI                   |                        | 2        | 1     |
|                               |                        |          |       |
| STOS                          | Dest-string            | 3        | 1     |
| STOS                          | (repeat) dest-string   | 4+3(rep) | 1     |
|                               |                        |          |       |
| SUB                           | Register, register     | 2        | 2     |
| SUB                           | Register, memory       | 7*       | 2-4   |
| SUB                           | Memory, register       | 7*       | 2-4   |
| SUB                           | Register, immediate    | 3        | 3-4   |
| SUB                           | Memory, immediate      | 7*       | 3-6   |
| SUB                           | Accumulator; immediate | 3        | 2-3   |
|                               |                        |          |       |
| TEST                          | Register, register     | 2        | 2     |
| TEST                          | Register, memory       | 6*       | 2-4   |
| TEST                          | Register, immediate    | 3        | 3-4   |
| TEST                          | Memory, immediate      | 6*       | 3-6   |
| TEST                          | Accumulator; immediate | 3        | 2-3   |
|                               |                        |          |       |
| WAIT                          |                        | 3        | 1     |
|                               |                        |          |       |
| XCHG                          | Accumulator, reg16     | 3        | 1     |
| XCHG                          | Memory, register       | 5*       | 2-4   |
| XCHG                          | Register, register     | 3        | 2     |
|                               |                        |          |       |
| XLAT                          | Source-table           | 5        | 1     |

| Instruction |                        | Clocks | Bytes |
|-------------|------------------------|--------|-------|
| XOR         | Register, register     | 2      | 2     |
| XOR         | Register, memory       | 7*     | 2-4   |
| XOR         | Memory, register       | 7*     | 2-4   |
| XOR         | Register, immediate    | 3      | 3-4   |
| XOR         | Memory, immediate      | 7*     | 3-6   |
| XOR         | Accumulator; immediate | 3      | 2-3   |

## ضمیمه شماره ۴

### کد ماشین دستورالعمل‌ها

| کد  | ثبات ۱۶ بیتی | ثبات ۸ بیتی | ثبات سگمنت |
|-----|--------------|-------------|------------|
| 000 | AX           | AL          | ES         |
| 001 | CX           | CL          | CS         |
| 010 | DX           | DL          | SS         |
| 011 | BX           | BL          | DS         |
| 100 | SP           | AH          |            |
| 101 | BP           | CH          |            |
| 110 | SI           | DH          |            |
| 111 | DI           | BH          |            |

### کدگذاری آدرس موثر

| R/m | Mod=00        | Mod=01 یا mod=10        |
|-----|---------------|-------------------------|
| 000 | [BX+SI]       | [مقدار جابجایی + BX+SI] |
| 001 | [BX+DI]       | [مقدار جابجایی + BX+DI] |
| 010 | [BP+SI]       | [مقدار جابجایی + BP+SI] |
| 011 | [BP+DI]       | [مقدار جابجایی + BP+DI] |
| 100 | [SI]          | [مقدار جابجایی + SI]    |
| 101 | [DI]          | [مقدار جابجایی + DI]    |
| 110 | (حالت مستقیم) | [مقدار جابجایی + BP]    |
| 111 | [BX]          | [مقدار جابجایی + BX]    |

فیلد "mod" در این حالت‌ها تعیین کننده آن است که چند بایت بعنوان مقدار جابجایی وجود دارد. مقدار 00 به این معناست که بایت جابجایی در کد ماشین وجود ندارد. یعنی حالت آدرس دهی غیر مستقیم با ثبات یا آدرس دهی شاخص دار با ثبات پایه بدون مقدار جابجایی اختیاری است. مقدار 10 در فیلد "mod" به این معنی است که دو بایت جابجایی در کد ماشین وجود دارد، این کلمه به مقداری که از ثبات شاخص و یا ثبات مبنا بدست می‌آید اضافه می‌شود. مقدار 01 در فیلد "mod" به این معنی است که یک بایت جابجایی در کد ماشین وجود دارد، این بایت بعنوان یک عدد علامت دار در نظر گرفته می‌شود و قبل از آنکه با مقداری که از ثبات شاخص و یا ثبات مبنا بدست می‌آید جمع گردد به یک کلمه توسعه پیدا می‌کند.

باید توجه داشت که عملوند [BP] در جدول بالا نیامده است. ترکیب منطقی  $r/m=110$  و  $mod=00$  یک جفت ویژه ای است که حالت مستقیم حافظه را مشخص می‌کند. این بدان معناست که هیچ حالت آدرس دهی غیر مستقیم ثبات با ثبات bp وجود ندارد. اسمبلر [BP] را به [BP+0] تبدیل کرده و از حالت  $mod=01$  استفاده کرده و بایت جابجایی را برابر صفر قرار می‌دهد.

کلمه آدرس در جدول بعدی در پراتنز قرار گرفته است زیرا تمام عملوندهای حافظه به بایت‌های اضافی هدف نیاز ندارند. اگر  $r/m=110$  و  $mod=00$  (آدرس دهی مستقیم) باشد، دو بایت اضافی وجود خواهد داشت.

# ضمیمه شماره ۵

## جدول کد اسکی

ASCII Character Sets

| MSD<br>LSD |      | 0<br>000 | 1<br>001 | 2<br>010 | 3<br>011 | 4<br>100 | 5<br>101 | 6<br>110 | 7<br>111 |
|------------|------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0          | 0000 | NUL      | DLE      | SP       | 0        | @        | P        |          | p        |
| 1          | 0001 | SOH      | DC1      | !        | 1        | A        | Q        | a        | q        |
| 2          | 0010 | STX      | DC2      | "        | 2        | B        | R        | b        | r        |
| 3          | 0011 | ETX      | DC3      | #        | 3        | C        | S        | c        | s        |
| 4          | 0100 | EOT      | DC4      | \$       | 4        | D        | T        | d        | t        |
| 5          | 0101 | ENQ      | NAK      | %        | 5        | E        | U        | e        | u        |
| 6          | 0110 | ACK      | SYN      | &        | 6        | F        | V        | f        | v        |
| 7          | 0111 | BEL      | ETB      | ,        | 7        | G        | W        | g        | w        |
| 8          | 1000 | BS       | CAN      | (        | 8        | H        | X        | h        | x        |
| 9          | 1001 | HT       | EM       | )        | 9        | I        | Y        | i        | y        |
| A          | 1010 | LF       | SUB      | *        | :        | J        | Z        | j        | z        |
| B          | 1011 | VT       | ESC      | +        | ;        | K        | [        | k        | {        |
| C          | 1100 | FF       | FS       | ,        | <        | L        | \        | i        |          |
| D          | 1101 | CR       | GS       | -        | =        | M        | ]        | m        | }        |
| E          | 1110 | SO       | RS       | .        | >        | N        | ↑        | n        | ~        |
| F          | 1111 | SI       | US       | /        | ?        | O        | ←        | o        | DEL      |

|     |                        |     |                            |
|-----|------------------------|-----|----------------------------|
| NUL | -Null                  | DLE | -Data Link Escape          |
| SOH | -Start of Heading      | DC  | -Device Control            |
| STX | -Start of Text         | NAK | -Negative Acknowledge      |
| ETX | -End of Text           | SYN | -Synchronous Idle          |
| EOT | -End of Transmission   | ETB | -End of Transmission Block |
| ENQ | -Enquiry               | CAN | -Cancel                    |
| ACK | -Acknowledge           | EM  | -End of Medium             |
| BEL | -Bell                  | SUB | -Substitute                |
| BS  | -Backspace             | ESC | -Escape                    |
| HT  | -Horizontal Tabulation | FS  | -File Separator            |
| LF  | -Line Feed             | GS  | -Group Separator           |
| VT  | -Vertical Tabulation   | RS  | -Record separator          |
| FF  | -Form Feed             | US  | -Unit Separator            |
| CR  | -Carriage Return       | SP  | -Space (Blank)             |
| SO  | -Shift Out             | DEL | -Delete                    |
| SI  | -Shift In              |     |                            |



## ضمیمه شماره ۶

### کد دستورالعملها

| کد عمل | دستورالعملها و عملوندها | ساختار بایتها          |
|--------|-------------------------|------------------------|
| 00     | add mem8, reg8          | mod reg r/m, (address) |
| 01     | add mem 16, reg16       | mod reg r/m, (address) |
| 02     | add reg8, mem8          | mod reg r/m, (address) |
|        | add reg8, reg8          | 11 dest_reg source_reg |
| 03     | add reg16, mem16        | mod reg r/m, (address) |
|        | add reg16, reg16        | 11 dest_reg source_reg |
| 04     | add al, imm8            | immediate byte         |
| 05     | add ax,imm16            | immediate word         |
| 06     | push es                 | (none)                 |
| 07     | pop es                  | (none)                 |
| 08     | or mem8, reg8           | mod reg r/m, (address) |
| 09     | or mem16, reg16         | mod reg r/m, (address) |
| 0a     | or reg8, mem8           | mod reg r/m, (address) |
|        | or reg8, reg8           | 11 dest_reg source_reg |
| 0b     | or reg16, mem16         | mod reg r/m, (address) |
|        | or reg16, reg16         | 11 dest_reg source_reg |
| 0c     | or al, imm8             | immediate byte         |
| 0d     | or ax, imm16            | immediate word         |
| 0e     | push cs                 | (none)                 |
| 0f     |                         |                        |
| 10     | adc mem8, reg8          | mod reg r/m, (address) |
| 11     | adc mem16, reg16        | mod reg r/m, (address) |
| 12     | adc reg8, mem8          | mod reg r/m, (address) |
|        | adc reg8, reg8          | 11 dest_reg source_reg |

|    |                                 |                         |
|----|---------------------------------|-------------------------|
| 13 | adc reg16, mem16                | mod reg r/m, (address)  |
|    | adc reg16, reg16                | 11 dest_reg source_ reg |
| 14 | adc al, imm8                    | immediate byte          |
| 15 | adc ax, imm16                   | immediate word          |
| 16 | push ss                         | (none)                  |
| 17 | pop ss                          | (none)                  |
| 18 | sbb mem8, reg8                  | mod reg r/m, (address)  |
| 19 | sbb mem16, reg16                | mod reg r/m, (address)  |
| 1a | sbb reg8, mem8                  | mod reg r/m, (address)  |
|    | sbb reg8, reg8                  | 11 dest_reg source_ reg |
| 1b | sbb reg16, mem16                | mod reg r/m, (address)  |
|    | sbb reg16, reg16                | 11 dest_reg source_ reg |
| 1c | sbb al, imm8                    | immediate byte          |
| 1d | sbb ax, imm16                   | immediate word          |
| 1e | push ds                         | (none)                  |
| 1f | pop ds                          | (none)                  |
| 20 | and mem8, reg8                  | mod reg r/m, (address)  |
| 21 | and mem16, reg16                | mod reg r/m, (address)  |
| 22 | and reg8, mem8                  | mod reg r/m, (address)  |
|    | and reg8, reg8                  | 11 dest_reg source_ reg |
| 23 | and reg16, mem16                | mod reg r/m, (address)  |
|    | and reg16, reg16                | 11 dest_reg source_ reg |
| 24 | and al, imm8                    | immediate byte          |
| 25 | and ax, imm16                   | immediate word          |
| 26 | es segment override prefix byte |                         |
| 27 | daa                             | (none)                  |
| 28 | sub mem8, reg8                  | mod reg r/m, (address)  |
| 29 | sub mem16, reg16                | mod reg r/m, (address)  |
| 2a | sub reg8, mem8                  | mod reg r/m, (address)  |
|    | sub reg8, reg8                  | 11 dest_reg source_ reg |

| کد عمل | دستورالعمل ها و عملوندها        | ساختار بایتها          |
|--------|---------------------------------|------------------------|
| 2b     | sub reg16, mem16                | mod reg r/m, (address) |
|        | sub reg16, reg16                | 11 dest_reg source_reg |
| 2c     | sub al, imm8                    | immediate byte         |
| 2d     | sub ax, imm16                   | immediate word         |
| 2e     | cs segment override prefix byte |                        |
| 2f     | das                             | (none)                 |
| 30     | xor mem8, reg8                  | mod reg r/m, (address) |
| 31     | xor mem16, reg16                | mod reg r/m, (address) |
| 32     | xor reg8, mem8                  | mod reg r/m, (address) |
|        | xor reg8, reg8                  | 11 dest_reg source_reg |
| 33     | xor reg16, mem16                | mod reg r/m, (address) |
|        | xor reg16, reg16                | 11 dest_reg source_reg |
| 34     | xor al, imm8                    | immediate byte         |
| 35     | xor ax, imm16                   | immediate word         |
| 36     | ss segment override prefix byte |                        |
| 37     | aaa                             | (none)                 |
| 38     | cmp mem8, reg8                  | mod reg r/m, (address) |
| 39     | cmp mem16, reg16                | mod reg r/m, (address) |
| 3a     | cmp reg8, mem8                  | mod reg r/m, (address) |
|        | cmp reg8, reg8                  | 11 dest_reg source_reg |
| 3b     | cmp reg16, mem16                | mod reg r/m, (address) |
|        | cmp reg16, reg16                | 11 dest_reg source_reg |
| 3c     | cmp al, imm8                    | immediate byte         |
| 3d     | cmp ax, imm16                   | immediate word         |
| 3e     | ds segment override prefix byte |                        |
| 3f     | aas                             | (none)                 |
| 40     | inc ax                          | (none)                 |
| 41     | inc cx                          | (none)                 |
| 42     | inc dx                          | (none)                 |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها |
|--------|--------------------------|---------------|
| 43     | inc bx                   | (none)        |
| 44     | inc sp                   | (none)        |
| 45     | inc bp                   | (none)        |
| 46     | inc si                   | (none)        |
| 47     | inc di                   | (none)        |
| 48     | dec ax                   | (none)        |
| 49     | dec cx                   | (none)        |
| 4a     | dec dx                   | (none)        |
| 4b     | dec bx                   | (none)        |
| 4c     | dec sp                   | (none)        |
| 4d     | dec bp                   | (none)        |
| 4e     | dec si                   | (none)        |
| 4f     | dec di                   | (none)        |
| 50     | push ax                  | (none)        |
| 51     | push cx                  | (none)        |
| 52     | push dx                  | (none)        |
| 53     | push bx                  | (none)        |
| 54     | push sp                  | (none)        |
| 55     | push bp                  | (none)        |
| 56     | push si                  | (none)        |
| 57     | push di                  | (none)        |
| 58     | pop ax                   | (none)        |
| 59     | pop cx                   | (none)        |
| 5a     | pop dx                   | (none)        |
| 5b     | pop bx                   | (none)        |
| 5c     | pop sp                   | (none)        |
| 5d     | pop bp                   | (none)        |
| 5e     | pop si                   | (none)        |
| 5f     | pop di                   | (none)        |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها                          |
|--------|--------------------------|----------------------------------------|
| 60-6f  |                          |                                        |
| 70     | jo                       | displacement byte                      |
| 71     | jno                      | displacement byte                      |
| 72     | jb/jnae/jc               | displacement byte                      |
| 73     | jnb/jae/jnc              | displacement byte                      |
| 74     | je/jz                    | displacement byte                      |
| 75     | jne/jnz                  | displacement byte                      |
| 76     | jbe/jna                  | displacement byte                      |
| 77     | jnb/ja                   | displacement byte                      |
| 78     | js                       | displacement byte                      |
| 79     | jns                      | displacement byte                      |
| 7a     | jp/jpe                   | displacement byte                      |
| 7b     | jnp/jpo                  | displacement byte                      |
| 7c     | jl/jnge                  | displacement byte                      |
| 7d     | jnl/jng                  | displacement byte                      |
| 7e     | jle/jng                  | displacement byte                      |
| 7f     | jnl/jg                   | displacement byte                      |
| 80     | add mem8, imm8           | mod 000 r/m, (address), immediate byte |
|        | add reg8, imm8           | 11 000 reg, immediate byte             |
|        | or mem8, imm8            | mod 001 r/m, (address), immediate byte |
|        | or reg8, imm8            | 11 001 reg, immediate byte             |
|        | adc mem8, imm8           | mod 010 r/m, (address), immediate byte |
|        | adc reg8, imm8           | 11 010 reg, immediate byte             |
|        | sbb mem8, imm8           | mod 011 r/m, (address), immediate byte |
|        | sbb reg8, imm8           | 11 011 reg, immediate byte             |
|        | and mem8, imm8           | mod 100 r/m, (address), immediate byte |
|        | and reg8, imm8           | 11 100 reg, immediate byte             |
|        | sub mem8, imm8           | mod 101 r/m, (address), immediate byte |
|        | sub reg8, imm8           | 11 101 reg, immediate byte             |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها                          |
|--------|--------------------------|----------------------------------------|
|        | xor mem8, imm8           | mod 110 r/m, (address), immediate byte |
|        | xor reg8, imm8           | 11 110 reg, immediate byte             |
|        | cmp mem8, imm8           | mod 111 r/m, (address), immediate byte |
|        | cmp reg8, imm8           | 11 111 reg, immediate byte             |
| 81     | and mem16,imm16          | mod 000 r/m, (address), immediate word |
|        | and reg16,imm16          | 11 000 reg, immediate word             |
|        | or mem16,imm16           | mod 001 r/m, (address), immediate word |
|        | or reg16,imm16           | 11 001 reg, immediate word             |
|        | adc mem16,imm16          | mod 010 r/m, (address), immediate word |
|        | adc reg16,imm16          | 11 010 reg, immediate word             |
|        | sbb mem16,imm16          | mod 011 r/m, (address), immediate word |
|        | sbb reg16,imm16          | 11 011 reg, immediate word             |
|        | and mem16,imm16          | mod 100 r/m, (address), immediate word |
|        | and reg16,imm16          | 11 100 reg, immediate word             |
|        | sub mem16,imm16          | mod 101 r/m, (address), immediate word |
|        | sub reg16,imm16          | 11 101 reg, immediate word             |
|        | xor mem16,imm16          | mod 110 r/m, (address), immediate word |
|        | xor reg16,imm16          | 11 110 reg, immediate word             |
|        | cmp mem16,imm16          | mod 111 r/m, (address), immediate word |
|        | cmp reg16,imm16          | 11 111 reg, immediate word             |
| 82     |                          |                                        |
| 83     |                          | mod 000 r/m, (address), immediate byte |
|        | and reg16,imm8           | 11 000 reg, immediate byte             |
|        | or mem16, imm8           | mod 001 r/m, (address), immediate byte |
|        | or reg16,imm8            | 11 001 reg, immediate byte             |
|        | adc mem16, imm8          | mod 010 r/m, (address), immediate byte |
|        | adc reg16,imm8           | 11 010 reg, immediate byte             |
|        | sbb mem16, imm8          | mod 011 r/m, (address), immediate byte |
|        | sbb reg16,imm8           | 11 011 reg, immediate byte             |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها                          |
|--------|--------------------------|----------------------------------------|
|        | and mem16, imm8          | mod 100 r/m, (address), immediate byte |
|        | and reg16,imm8           | 11 100 reg, immediate byte             |
|        | sub mem16, imm8          | mod 101 r/m, (address), immediate byte |
|        | sub reg16,imm8           | 11 101 reg, immediate byte             |
|        | xor mem16, imm8          | mod 110 r/m, (address), immediate byte |
|        | xor reg16,imm8           | 11 110 reg, immediate byte             |
|        | cmp mem16, imm8          | mod 111 r/m, (address), immediate byte |
|        | cmp reg16,imm8           | 11 111 reg, immediate byte             |
| 84     | test reg8, mem8          | mod reg r/m, (address)                 |
|        | test reg8, reg8          | 11 dest_reg source_ reg                |
| 85     | test reg16, mem16        | mod reg r/m, (address)                 |
|        | test reg16, reg16        | 11 dest_reg source_ reg                |
| 86     | xchg reg8, mem8          | mod reg r/m, (address)                 |
|        | xchg reg8, reg8          | 11 dest_reg source_ reg                |
| 87     | xchg reg16, mem16        | mod reg r/m, (address)                 |
|        | xchg reg16, reg16        | 11 dest_reg source_ reg                |
| 88     | mov mem8, reg8           | mod reg r/m, (address)                 |
|        | mov mem8, reg8           | 11 dest_reg source_ reg                |
| 89     | mov mem16, reg16         | mod reg r/m, (address)                 |
|        | mov reg16, reg16         | 11 dest_reg source_ reg                |
| 8a     | mov reg8, mem8           | mod reg r/m, (address)                 |
| 8b     | mov reg16, mem16         | 11 dest_reg source_ reg (address)      |
| 8c     | mov mem16, sreg          | mod reg r/m, (address)                 |
|        | mov reg16, sreg          | 11 reg r/m                             |
| 8d     | lea reg16, mem           | mod reg r/m, (address)                 |
| 8e     | mov sreg, mem16          | mod reg r/m, (address)                 |
|        | mov sreg, reg16          | 11 reg r/m                             |
| 8f     | pop mem16                | mod 000 r/m, (address)                 |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها                   |
|--------|--------------------------|---------------------------------|
| 90     | nop                      | (none)                          |
| 91     | xchg ax,cx               | (none)                          |
| 92     | xchg ax,dx               | (none)                          |
| 93     | xchg ax,bx               | (none)                          |
| 94     | xchg ax,sp               | (none)                          |
| 95     | xchg ax,bp               | (none)                          |
| 96     | xchg ax,si               | (none)                          |
| 97     | xchg ax,di               | (none)                          |
| 98     | cbw                      | (none)                          |
| 99     | cwd                      | (none)                          |
| 9a     | call<br>(far direct)     | offset and segment number words |
| 9b     | wait                     | (none)                          |
| 9c     | pushf                    | (none)                          |
| 9d     | popf                     | (none)                          |
| 9e     | sahf                     | (none)                          |
| 9f     | lahf                     | (none)                          |
| A0     | mov al, mem8             | offset word (direct addressing) |
| A1     | mov ax, mem16            | offset word                     |
| A2     | mov mem8, al             | offset word                     |
| A3     | mov mem16, ax            | offset word                     |
| A4     | movsh                    | (none)                          |
| A5     | movsw                    | (none)                          |
| A6     | cmps b                   | (none)                          |
| A7     | cmps w                   | (none)                          |
| A8     | test al, imm8            | immediate byte                  |
| A9     | test ax, imm16           | immediate word                  |
| AA     | stos b                   | (none)                          |
| AB     | stos w                   | (none)                          |



| کد عمل | دستورالعمل ها و عملوندها         | ساختار بایتها                                                        |
|--------|----------------------------------|----------------------------------------------------------------------|
| AC     | lodsb                            | (none)                                                               |
| AD     | lodsw                            | (none)                                                               |
| AE     | scasb                            | (none)                                                               |
| AF     | scasw                            | (none)                                                               |
| B0     | mov al, imm8                     | immediate byte                                                       |
| B1     | mov cl, imm8                     | immediate byte                                                       |
| B2     | mov dl, imm8                     | immediate byte                                                       |
| B3     | mov bl, imm8                     | immediate byte                                                       |
| B4     | mov ah, imm8                     | immediate byte                                                       |
| B5     | mov ch, imm8                     | immediate byte                                                       |
| B6     | mov dh, imm8                     | immediate byte                                                       |
| B7     | mov bh, imm8                     | immediate byte                                                       |
| B8     | mov ax, imm16                    | immediate word                                                       |
| B9     | mov cx, imm16                    | immediate word                                                       |
| BA     | mov dx, imm16                    | immediate word                                                       |
| BB     | mov bx, imm16                    | immediate word                                                       |
| BC     | mov sp, imm16                    | immediate word                                                       |
| BD     | mov bp, imm16                    | immediate word                                                       |
| BE     | mov si, imm16                    | immediate word                                                       |
| BF     | mov di, imm16                    | immediate word                                                       |
| C0, C1 |                                  |                                                                      |
| C2     | ret imm16<br>(near return)       | immediate word                                                       |
| C3     | ret<br>(near return)             | (none)                                                               |
| C4     | les reg16, mem                   | mod reg r/m, (address)                                               |
| C5     | lds reg16, mem                   | mod reg r/m, (address)                                               |
| C6     | mov mem8, imm8<br>mov reg8, imm8 | mod 000 r/m, (address), immediate byte<br>11 000 r/m, immediate byte |

| کد عمل | دستورالعمل ها و عملوندها  | ساختار بایتها                          |
|--------|---------------------------|----------------------------------------|
| C7     | mov mem16, imm16          | mod 000 r/m, (address), immediate word |
|        | mov reg1, imm16           | 11 000 r/m, immediate word             |
| C8,C9  |                           |                                        |
| CA     | ret imm16<br>(far return) | immediate word                         |
| CB     | ret<br>(far return)       | (none)                                 |
| CC     | int 3                     | (none)                                 |
| CD     | int imm8                  | immediate byte                         |
| CE     | into                      | (none)                                 |
| CF     | iret                      | (none)                                 |
| D0     | rol mem8,1                | mod 000 r/m, (address)                 |
|        | rol reg8,1                | 11 000 reg                             |
|        | ror mem8,1                | mod 001 r/m, (address)                 |
|        | ror reg8,1                | 11 001 reg                             |
|        | rcl mem8,1                | mod 010 r/m, (address)                 |
|        | rcl reg8,1                | 11 010 reg                             |
|        | rcr mem8,1                | mod 011 r/m, (address)                 |
|        | rcr reg8,1                | 11 011 reg                             |
|        | shl/sal mem8,1            | mod 100 r/m, (address)                 |
|        | shl/sal reg8,1            | 11 100 reg                             |
|        | shr mem8,1                | mod 101 r/m, (address)                 |
|        | shr reg8,1                | 11 101 reg                             |
|        | sar mem8,1                | mod 111 r/m, (address)                 |
|        | sar reg8,1                | 11 111 reg                             |
| D1     | rol mem16,1               | mod 000 r/m, (address)                 |
|        | rol reg16,1               | 11 000 reg                             |
|        | ror mem16,1               | mod 001 r/m, (address)                 |
|        | ror reg16,1               | 11 001 reg                             |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها            |
|--------|--------------------------|--------------------------|
|        | rcl mem16,1              | mod 010 r/m, (address)   |
|        | rcl reg16,1              | 11 010 reg               |
|        | rcr mem16,1              | mod 011 r/m, (address)   |
|        | rcr reg16,1              | 11 011 reg               |
|        | shl/sal mem16,1          | mod 100 r/m, (address)   |
|        | shl/sal reg16,1          | 11 100 reg               |
|        | shr mem16,1              | mod 101 r/m, (address)   |
|        | shr reg16,1              | 11 101 reg               |
|        | sar mem16,1              | mod 111 r/m, (address)   |
|        | sar reg16,1              | 11 111 reg               |
|        |                          | (110 not used with 8088) |
| D2     | rol mem8,cl              | mod 000 r/m, (address)   |
|        | rol reg8,cl              | 11 000 reg               |
|        | ror mem6,cl              | mod 001 r/m, (address)   |
|        | ror reg8,cl              | 11 001 reg               |
|        | rcl mem8,cl              | mod 010 r/m, (address)   |
|        | rcl reg8,cl              | 11 010 reg               |
|        | rcr mem8,cl              | mod 011 r/m, (address)   |
|        | rcr reg8,cl              | 11 011 reg               |
|        | shl/sal mem8, cl         | mod 100 r/m, (address)   |
|        | shl/sal reg8, cl         | 11 100 reg               |
|        | shr mem8, cl             | mod 101 r/m, (address)   |
|        | shr reg8, cl             | 11 101 reg               |
|        | sar mem8, cl             | mod 111 r/m, (address)   |
|        | sar reg8, cl             | 11 111 reg               |
|        |                          |                          |
|        |                          |                          |
| D3     | rol mem16, cl            | mod 000 r/m, (address)   |
|        | rol reg16, cl            | 11 000 reg               |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها          |
|--------|--------------------------|------------------------|
|        | ror mem16, cl            | mod 001 r/m, (address) |
|        | ror reg16, cl            | 11 001 reg             |
|        | rcl mem16, cl            | mod 010 r/m, (address) |
|        | rcl reg16, cl            | 11 010 reg             |
|        | rcr mem16, cl            | mod 011 r/m, (address) |
|        | rcr reg16, cl            | 11 011 reg             |
|        | shl/sal mem16, cl        | mod 100 r/m, (address) |
|        | shl/sal reg16, cl        | 11 100 reg             |
|        | shr mem16, cl            | mod 101 r/m, (address) |
|        | shr reg16, cl            | 11 101 reg             |
|        | sar mem16, cl            | mod 111 r/m, (address) |
|        | sar reg16, cl            | 11 111 reg             |
| D4     | aam                      | 0a                     |
| D5     | aad                      | 0a                     |
| D6     |                          |                        |
| D7     | xlat                     | (none)                 |
| D8     | esc 0                    | mod 000 r/m, (address) |
| D9     | esc 1                    | mod 001 r/m, (address) |
| DA     | esc 2                    | mod 010 r/m, (address) |
| DB     | esc 3                    | mod 011 r/m, (address) |
| DC     | esc 4                    | mod 100 r/m, (address) |
| DD     | esc 5                    | mod 101 r/m, (address) |
| DE     | esc 6                    | mod 110 r/m, (address) |
| DF     | esc 7                    | mod 111 r/m, (address) |
| E0     | loopnz / loopne          | displacement byte      |
| E1     | loopz/loope              | displacement byte      |
| E2     | loop                     | displacement byte      |
| E3     | jcxz                     | displacement byte      |
| E4     | in al, port              | port byte              |

| کد عمل | دستورالعمل ها و عملوندها              | ساختار بایتها                          |
|--------|---------------------------------------|----------------------------------------|
| E5     | in al, port                           | port byte                              |
| E6     | out al, port                          | port byte                              |
| E7     | out ax, port                          | port byte                              |
| E8     | call<br>(near relative)               | displacement word                      |
| E9     | jmp<br>(intra-segment relative)       | displacement word                      |
| EA     | jmp<br>(inter-segment direct)         | offset and segment number words        |
| AB     | jmp<br>(inter-segment relative short) | displacement byte                      |
| EC     | in al, dx                             | (none)                                 |
| ED     | in ax, dx                             | (none)                                 |
| EE     | out al, dx                            | (none)                                 |
| EF     | out ax, dx                            | (none)                                 |
| F0     | lock prefix byte                      |                                        |
| F1     |                                       |                                        |
| F2     | repne /repnz prefix byte              |                                        |
| F3     | rep/repe/repz prefix byte             |                                        |
| F4     | hlt                                   | (none)                                 |
| F5     | cmc                                   | (none)                                 |
| F6     | test mem8, imm8                       | mod 000 r/m, (address), immediate byte |
|        | test reg8, imm8                       | 11 000 reg, immediate byte             |
|        | not mem8                              | mod 010 r/m, (address)                 |
|        | not reg8                              | 11 010 reg                             |
|        | neg mem8                              | mod 011 r/m, (address)                 |
|        | neg reg8                              | 11 011 reg                             |
|        | mul mem8                              | mod 100 r/m, (address)                 |
|        | mul reg8                              | 11 100 reg                             |
|        | imul mem8                             | mod 101 r/m, (address)                 |
|        | imul reg8                             | 11 101 reg                             |

| کد عمل | دستورالعمل ها و عملوندها | ساختار بایتها                          |
|--------|--------------------------|----------------------------------------|
|        | div mem8                 | mod 110 r/m, (address)                 |
|        | div reg8                 | 11 110 reg                             |
|        | idiv mem8                | mod 111 r/m, (address)                 |
|        | idiv reg8                | 11 111 reg                             |
| F7     | test mem16, imm16        | mod 000 r/m, (address), immediate word |
|        | test reg16, imm16        | 11 000 reg, immediate word             |
|        | not mem16                | mod 010 r/m, (address)                 |
|        | not reg16                | 11 010 reg                             |
|        | neg mem16                | mod 011 r/m, (address)                 |
|        | neg reg16                | 11 011 reg                             |
|        | mul mem16                | mod 100 r/m, (address)                 |
|        | mul reg16                | 11 100 reg                             |
|        | imul mem16               | mod 101 r/m, (address)                 |
|        | imul reg16               | 11 101 reg                             |
|        | div mem16                | mod 110 r/m, (address)                 |
|        | div reg16                | 11 110 reg                             |
|        | idiv mem16               | mod 111 r/m, (address)                 |
|        | idiv reg 16              | 11 111 reg                             |
| F8     | clc                      | (none)                                 |
| F9     | stc                      | (none)                                 |
| FA     | cli                      | (none)                                 |
| FB     | sti                      | (none)                                 |
| FC     | cld                      | (none)                                 |
| FD     | std                      | (none)                                 |
| FE     | inc mem8                 | mod 000 r/m, (address)                 |
|        | inc reg8                 | 11 000 reg                             |
|        | dec mem8                 | mod 001 r/m, (address)                 |
|        | dec reg8                 | 11 001 reg                             |
| FF     | inc mem16                | mod 000 r/m, (address)                 |

| ساختار بایتها          | دستورالعمل ها و عملوندها         | کد عمل |
|------------------------|----------------------------------|--------|
| mod 001 r/m, (address) | dec mem16                        |        |
| mod 010 r/m, (address) | call<br>(far indirect)           |        |
| mod 011 r/m, (address) | call<br>(near indirect)          |        |
| mod 100 r/m, (address) | jmp<br>(intra –segment indirect) |        |
| mod 101 r/m, (address) | jmp<br>(inter-segment indirect)  |        |
| mod 110 r/m, (address) | push mem16                       |        |

## سؤالات چهار گزینه‌ای

۱- اگر  $AX=0A21$  و  $BX=0010$  باشد وضعیت فلگها در اثر اجرای دستورالعمل

$CMP\ AX,BX$  چیست؟

|             |        |        |        |
|-------------|--------|--------|--------|
| الف. $CF=0$ | $OF=0$ | $SF=0$ | $ZF=0$ |
| ب. $CF=1$   | $OF=0$ | $SF=1$ | $ZF=1$ |
| ج. $CF=1$   | $OF=1$ | $SF=0$ | $ZF=0$ |
| د. $CF=1$   | $OF=1$ | $SF=1$ | $ZF=1$ |

۲- کدام یک از دستورالعملهای ذیل روی  $CF$  بی تأثیر هستند؟

الف.  $SAR$

ب.  $ROR$

ج.  $CMC$

د.  $MOVS$



۳- کدام یک از دستورالعملهای ذیل روی ثبات نشانه‌ها تاثیر می‌گذارد؟

الف. POP

ب. IN

ج. OR

د. PUSHF

۴- کدامیک از دستورات ذیل غیر مجاز است؟

الف. MUL BX

ب. INC NUM

ج. NOP

د. SHR AX,3

۵- کدامیک از دستورات ذیل مجاز نیست؟

الف. CMP 10,BX

ب. MOV TEMP,COUNT

ج. MOV DS,ES

د. CBW

۶- دستورالعملهای DEC و INC روی کدامیک از نشانه‌های ذیل بی‌تأثیر

هستند؟

الف. CF

ب. OF

ج. SF

د. ZF

۷- بر اثر جمع کدام یک از زوج اعداد ذیل، سرریزی رخ خواهد داد (اعداد در

مبنای ۱۶ هستند)؟

الف. 0A07,0FD3

ب. 0206,FFB0

ج. FFE7,FFF6

د. 483F,745A

۸- در دستورالعمل MOV AX,SEG DATA مقدار SEG DATA چه زمانی

معین می گردد (DATA نام سگمنت داده است)؟

الف. زمان اسمبل کردن

ب. زمان LINK کردن

ج. زمان اجرای واقعی برنامه

د. هیچکدام

۹- دستور تعریف نوع ساختار ذیل چند بایت را تخصیص می دهد؟

S1 STRUCT

F1 DW ?

F2 DB 10 DUP(?)

S1 ENDS

الف. ۱۲      ب. ۱۴      ج. صفر      د. ۱۵

۱۰- آدرس پنج رقمی 1F558 معادل کدامیک از آدرسهای ذیل است؟

الف. 1055:1018

ب. 1E00:5581

ج. 1E55:1008

د. 18A3:5B28

۱۱- اگر  $CX=01A2, AX=0075$  باشد بعد از اجرای دستورالعمل `SUB AX,CX`

محتوای `SF` و `AX` کدام یک از موارد ذیل است؟

الف. `SF=1 AX=FED3`

ب. `SF=0 AX=FED3`

ج. `SF=1 AX=0127`

د. `SF=0 AX=0127`

۱۲- پس از اجرای قطعه کد ذیل محتوای `AX` چه خواهد بود؟

```
STC
MOV AX,10
MOV BX,4
ADC AX,BX
```

الف. ۱۶

ب. ۱۵

ج. ۱۰

د. ۵

۱۳- کدامیک از دستورات ذیل باقیمانده تقسیم محتوای `DX` بر ۸ را محاسبه

می کند؟

الف. `AND DX,0007`

ب. `OR DX,0007`

ج. `AND DX,0008`

د. `OR DX,0008`

۱۴- روال `PROC1` از نوع `FAR` دو پارامتر `N1` و `N2` را از طریق پشته و به

صورت ذیل دریافت می کند.

```
PUSH N1
PUSH N2
CALL PROC1
```

در لحظه ورود به روال PROC1 آدرس N2 کدامیک از موارد ذیل است؟

الف. SP+2

ب. SP- 2

ج. SP+4

د. SP- 4

۱۵- قطعه کد ذیل چه عبارتی را محاسبه خواهد کرد؟

```
MOV AX,X
AND AX,AX
NEG AX
ADD AX,Y
SUB AX,Z
INC AX
```

الف.  $2X+Y-Z$

ب.  $2(X+Y+Z)+1$

ج.  $(-2X)+Y-Z+1$

د.  $2(X+Y-Z)$

۱۶- اگر  $AX=FFFF$  باشد بعد از اجرای قطعه کد ذیل محتوای AX چیست؟

```
NEG AX
ROR AX,1
MOV CL,3
SAR AX,CL
```

الف. F000

ب. 1000

ج. 0000

د. FFFO

۱۷- بعد از اجرای قطعه کد ذیل محتوای AX چیست؟

```
MOV CX,10
MOV AX,5
FOR: DEC AX
CMP AX,0
LOOPNE FOR
```

الف. صفر

ب. ۱

ج. -۱

د. -۲

۱۸- کدام دستورالعمل مجاز است؟

الف. IMUL BL

ب. XCHG TEMP, AX

ج. DIV 10,VALUE

د. CLD

۱۹- کدامیک از دستورالعملهای ذیل روی ثبات نشانه‌ها تاثیر می‌گذارند؟

الف. MOV

ب. XCHG

ج. STOS

د. MUL

۲۰- در دستورالعمل MOV AX, SEG DATA حالت عملوندهای AX, SEG

DATA به ترتیب کدامیک از موارد ذیل است (DATA نام سگمنت است)؟

الف. بی‌واسطه، ثبات

ب. حافظه مستقیم، ثبات

ج. حافظه غیر مستقیم از نوع دارای مبنا، ثبات

د. بلاواسطه، حافظه مستقیم

۲۱- کدام یک از موارد ذیل پیاده سازی طرح ذیل است؟

IF (AX<=10 OR BX=100) THEN  
AX=AX+1,

الف. CMP AX,10

JLE T1

.CMP. BX,100

JE T1

T1: INC AX

ب. CMP AX,10

JLE T1

CMP BX,100

JE T1

JMP T2

T1: INC AX

T2:

ج. CMP AX,10

JNE T1

CMP BX,100

JNLE T1

INC AX

T1:

د. هیچکدام

۲۲- کدامیک از موارد ذیل صحیح نیست؟

الف. نتیجه شیفت ریاضی و منطقی به طرف چپ یکسان هستند.

ب. فلگ سرریزی برای شیفت چند بیتی تعریف نشده است.

ج. عمل شیفت به طرف چپ روی فلگ PF بی تاثیر است.

د. در عمل شیفت چند بیتی به چپ ، بیتهایی که از طرف چپ بیرون می روند به

دور ریخته می شوند، مگر آخرین بیت که در فلگ CF ذخیره می گردد.

۲۳- قطعه کد ذیل چه کاری انجام می دهد. (فرض می کنیم ثباتهای لازم دیگر

تنظیم شده اند.)

```
MOV AL,'*'
MOV CX,10
MOV DI,OFFSET STR
CID
REP STOSB
```

الف. کاراکتر '\*' را در رشته STR جستجو می کند.

ب. تا وقتی به کاراکتر '\*' نرسیده است پویش را تا ۱۰ کاراکتر اول STR ادامه می دهد.

ج. کاراکتر '\*' را در ۱۰ کاراکتر اول STR کپی می کند.

د. هیچکدام

۲۴- اگر اجرای دستورهای ADD، سه پالس و اجرای MOV، ۴ پالس و LOOP،

۸ پالس زمانی مصرف کنند، در این صورت اجرای قطعه کد ذیل چند پالس

زمانی نیاز دارد؟

```
MOV CX,5
B:ADD AX,2
LOOP B
```

الف. ۱۵      ب. ۵۹      ج. ۵۵      د. ۴۹

۲۵- کدامیک از موارد ذیل مجاز است؟

الف. MOV TEMP,COUNT

ب. MUL 2

ج. PUSHF AX

د. CBW

۲۶- کدامیک از دستورات ذیل روی فلگها تاثیر می گذارد؟

الف. MOV

ب. INC

ج. STOS

د. LOOP

۲۷- پس از اجرای قطعه کد ذیل CX حاوی چه مقداری خواهد بود (temp و

num1 توسط دستور DW در سگمنت DATA تعریف شده اند)؟

```

MOV CX,0
MOV BX,10
MOV AX,123
LOOP1: CWD
 DIV BX
 MOV NUM1,DX
 MOV TEMP,AX
 MOV AX,CX
 MUL BX
 ADD AX,NUM1
 MOV CX,AX
 MOV AX,TEMP
 CMP AX,0
 JNZ LOOP1

```

الف. ۴۴۴      ب. ۴۵۶      ج. ۱۲۳      د. ۳۲۱

۲۸- کدامیک از دستورات زیر مجاز است؟

الف. CMP BH,'\*'

ب. OUT 05A4H,AL

ج. MOV DS,CS

د. MOV IP,AX



۲۹- کدامیک از موارد ذیل پیاده سازی دستور شرطی ذیل است؟

IF (NOT (A>=10 OR BX<=1) ) THEN  
CX=CX+1

الف.

```
CMP A,10
JGE EXIT
CMP B,1
JLE EXIT
JMP M
EXIT: INC CX
M:
```

ب.

```
CMP A,10
JGE EXIT
CMP B,1
JLE EXIT
EXIT: INC CX
```

ج.

```
CMP A,10
JNL EXIT
CMP B,1
JNG EXIT
INC CX
EXIT:
```

د. هیچکدام

۳۰- پس از اجرای دستورالعملهای ذیل محتوای BX کدامیک از موارد ذیل

است؟

```
MOV CL,4
MOV AX,FFFF
CLC
RCR AX,CL
```

الف. 0FFF

ب. FFF0

ج. F000

د. 000F

۳۱- ماکروی ذیل را در نظر بگیرید.

```
MAC1 MACRO
 CMP AX,0
 JE TEST
 ADD AX,AX
TEST: ADD AX,AX
 ENDM
```

پس از اجرای قطعه کد ذیل محتوای AX چه خواهد بود؟

MOV AX,3

MAC1

MAC1

الف. ۱۲

ب. ۹

ج. ۱۵

د. اسمبلر پیغام خطا می دهد.

۳۲- کدامیک از دستورات ذیل غیر مجاز است؟

الف. CMPSW

ب. IRET

ج. LEA BX,AX

د. LOOPNE FOR1

۳۳- با توجه به تعریف ماکروی ذیل کدامیک از فراخوانی های ذیل باعث بروز

خطا خواهد شد.

```
MAC2 MACRO N1,N2,N3,N4
 MOV AX,N1
 IFNB<N2>
 ADD AX,N2
 ENDIF
 IFNB<N3>
 ADD AX,N3
```

```
ENDIF
IFNB<N4>
ADD AX,N4
ENDIF
ENDM
```

الف. MAC1 BX, CX,DX,1

ب. MAC1 BX,CX,100

ج. MAC1 BX,DX

د. هیچکدام

۳۴- دستور REPE در کدام یک از شرایط ذیل تکرار را ادامه می دهد؟

الف. CX=0 AND ZF=0

ب. CX<>0 OR ZF=0

ج. CX<>0 AND ZF=1

د. CX=0 OR ZF=1

۳۵- اگر در ۲۰ کاراکتر اول رشته STRING حداقل یک کاراکتر S وجود داشته

باشد، پس از اجرای قطعه کد ذیل DI حاوی چیست؟ (فرض کنید ثباتهای لازم

دیگر به درستی تنظیم شده است)

```
MOV AL,'S'
LEA DI, STRING
D
REPNE SCASB
DEC DI
```

الف. DI حاوی افسس اولین وقوع کاراکتر S در رشته STRING است.

ب. DI حاوی تعداد کاراکتر S در رشته STRING است.

ج. DI حاوی افسس کاراکتر قبل از اولین وقوع کاراکتر S در رشته STRING است.

د. هیچکدام

۳۶- محتوای AL چه عددی باشد تا بعد از اجرای دستورات ذیل مقدار AL صفر شود؟

XOR AL,0FH

الف. 00

ب. FF

ج. 0F

د. F0

۳۷- اگر AX=FF15 و VALUE=0003 باشد بعد از اجرای دستور IMUL VALUE فلگهای CF و OF کدامیک از مقادیر ذیل را دارند؟

الف. CF=0 ، OF=1

ب. CF=1 ، OF=0

ج. CF=1 ، OF=1

د. CF=0 ، OF=0

۳۸- با توجه به دستور ذیل چه عددی در مبنای ۱۶ در BYTE1 ذخیره می شود.  
BYTE1 DB - 128

الف. 70

ب. 80

ج. FF

د. 08

۳۹- بعد از اجرای دستورالعملهای ذیل DX حاوی چه عددی خواهد بود (در مبنای ۱۶)؟

MOV BX,1  
MOV AX,- 1  
MUL BX

الف. DX تغییری نمی کند و مقدار قبلی خود را دارد.

ب. 1111

ج. 0000

د. FFFF

۴۰- آدرس پنج رقمی (در مبنای ۱۶) شروع سگمنت شماره  $(0010)_{16}$  کدامیک

از موارد ذیل است؟

الف. 01010

ب. 00100

ج. 0011F

د. 00111

۴۱- مکمل پایه  $(342)_5$  کدامیک از موارد ذیل است؟

الف. 113

ب. 213

ج. 102

د. 103

۴۲- کدامیک از اسامی ذیل مجاز نیست؟

الف. TEMP\$

ب. TEMP. NUM

ج. TEMP%

د. TEMP4

۴۳- دستورالعمل ذیل چند بایت تخصیص می دهد؟

B DB 2(5 DUP('\*'),4 DUP('0'))

الف. 120

ب. 11

ج. 40

د. 18

۴۴- با توجه به قطعه کد ذیل در D چه عددی ذخیره می‌شود؟

```
DATA SEGMENT
A DB 5DUP(9)
B DB 6DUP(8)
C DB OFFSET B
```

الف. 0      ب. 6      ج. 5      د. 11

۴۵- اگر AX حاوی عدد ده باشد پس از اجرای دستورات ذیل محتوای AX

کدامیک از موارد ذیل است؟

```
NEG AX
ADD AX,AX
NEG AX
DEC AX
```

الف. 18      ب. 19      ج. 20      د. 21

۴۶- پس از اجرای دستورات ذیل BX حاوی چه مقداری است؟

```
MOV CX,5
MOV BX,16
LOOP: DEC BX
 CMP BX,0
 LOOPNE FOR
```

الف. 12      ب. 15      ج. 0      د. 11

۴۷- کدام یک از دستورات ذیل مجاز است؟

الف. cbw

ب. cwd ax

ج. cmp 10,ax

د. cld bx

۴۸- قطعه کد ذیل پیاده سازی کدامیک از موارد ذیل است؟

```
cmp temp,10
Jng e
Cmp ali,8
Jne e
Dec count
E:
```

الف. if (temp > 10 and ali=8)

count=count-1

ب. if (temp < 10 or ali=8)

count = count-1

ج. if (temp > 10 or ali <> 8)

count = count-1

د. if (temp < 10 and ali <> 8)

count = count-1

۴۹- فرض کنید سگمنت داده حاوی دو دستور ذیل باشد:

```
source DB "Summe"
dest DB "Summi"
```

اگر آدرس شروع Source 0000 و آدرس شروع dest 0005 باشد مقادیر ثباتهای SI و

DI بعد از اجرای قطعه کد ذیل چه خواهد بود؟

```
Lea Si,source
Lea Di,dest
Cld
Mov Cx,6
Repne Cmpsb
```

الف. DI=0005, SI=0000

ب. DI=0009, SI=0004

ج. DI=000A, SI=0005

د. DI=0006, SI=0001

۵۰- اگر  $AX=A9D7$  بعد از اجرای  $rol AX,1$  مقدار  $AX$  کدامیک از موارد ذیل

است؟

الف. 54EB

ب. 53AF

ج. A9D7

د. B9D6

۵۱- دستورالعمل ذیل چند باید حافظه را تخصیص می دهد؟

temp DB 20Dup(2Dup('\*'),3Dup('1'))

الف. 25 ب. 120 ج. 100 د. 20

۵۲- کدامیک از دستورات ذیل مجاز است؟

الف. `mov ds,12`

ب. `mov cs,ds`

ج. `mov IP,12`

د. `mov ax, OFFSET TE`

۵۳- کدامیک از موارد ذیل مجاز است؟

الف. `DW 12*10`

ب. `DW "ABC"`

ج. `mov [bx],2`

د. `push 2`

۵۴- مکمل 2 عدد 00001010 کدامیک از موارد ذیل است؟

الف. 11110110 ب. 11110101

ج. 00000110 د. 01101111



۵۵- کدامیک از دستورالعملهای ذیل غیرمجاز است؟

الف. mul bx

ب. add 1,ax

ج. inc num

د. inc ax

۵۶- کدامیک از دستورالعملهای ذیل CF را تغییر نمی دهد؟

الف. cmc

ب. stc

ج. clc

د. Jc

۵۷- اگر dh=F5 باشد بعد از اجرای دستور neg dh محتوای dh چیست؟

الف. oB

ب. F4

ج. 5F

د. F5

۵۸- اگر SP=0100 باشد بعد از اجرای قطعه کد ذیل محتوای SP چیست؟

push ax

push bx

push cx

الف. 0106

ب. 0104

ج. 00FA

د. 00FC

۵۹- دستور rep در کدامیک از حالات ذیل تکرار را ادامه می‌دهد؟

الف.  $CX \neq 0$

ب.  $ZF \neq 0$

ج.  $ZF = 0$

د.  $CX = 0$

۶۰- اگر  $AX = A9D7$  و  $CL = 04$  باشد بعد از اجرای دستور  $SAR AX, CL$  مقدار  $AX$

چيست؟

الف.  $FA9D$

ب.  $0F9D$

ج.  $9D7F$

د.  $9D70$

۶۱- اگر  $AX = FE01$  و  $BL = FF$  باشد بعد از اجرای دستور  $DIV BL$  مقدار  $AX$

کدامیک از موارد ذیل است؟

الف.  $EF01$

ب.  $00FF$

ج.  $FE01$

د.  $E0E0$

۶۲- اگر  $AL = 6E$  و  $CH = 0A$  باشد بعد از اجرای دستور  $IMUL CH$  مقادیر  $AX$  و  $CF$

و  $OF$  کدامیک از موارد ذیل است؟

الف.  $AX = 044c, CF = 1, OF = 1$

ب.  $AX = 044c, CF = 0, OF = 0$

ج.  $AX = 046E, CF = 0, OF = 0$

د.  $AX = 046E, CF = 1, OF = 1$

۶۳- حلقه ذیل چند بار اجرا می شود؟

```
mov cx,- 1
for:
...
loop for
```

الف. 65535

ب. 65536

ج. 0

د. 1

۶۴- کدامیک از اعداد ذیل نمی تواند به عنوان عملوند DW استفاده شوند؟

الف. 32768 -

ب. 132768

ج. 65535

د. 32000

۶۵- کدامیک از دستورات ذیل غیرمجاز است؟

الف. lea ax,array

ب. lea ax,bx

ج. mov ax,[bx]

د. mov ax,SEG data

۶۶- اگر بخواهیم عدد FF را در مبنای 16 در خانه ای از حافظه ذخیره کنیم از

کدامیک از عملوندهای ذیل نمی توان به عنوان عملوند DB استفاده کرد؟

الف. 255

ب. 1 -

ج. 11111111b

د. 88Q

۶۷- کدامیک از موارد ذیل صحیح نیست؟

- الف. کد ماکرو در محل فراخوانی ماکرو کپی می‌شود.  
 ب. استفاده از ماکرو به جای روال معمولاً باعث طولانی شدن کد هدف می‌شود.  
 ج. استفاده از ماکرو به جای روال معمولاً باعث اجرای سریعتر برنامه می‌شود.  
 د. عمل جایگزینی دستور فراخوانی ماکرو با کد ماکرو در زمان اجرا انجام می‌شود.

۶۸- دستورالعمل ذیل چند بایت حافظه را تخصیص می‌دهد؟

ABLE, DB 10DUP(3DUP('A'), 9 DUP ('B'))

الف. ۲۲

ب. ۱۲۰

ج. ۲۷۰

د. ۳۹

۶۹- دستورالعمل ذیل چند بایت حافظه را تخصیص می‌دهد؟

DW 100H DUP(?)

الف. 100

ب. 1024

ج. 512

د. 200

۷۰- کدام یک از نامهای ذیل مجاز نیست؟

الف. TEMP. 2

ب. ?TEMP

ج. TEMP?

د. TEMP4

۷۱- با توجه به کد ذیل، آفست شروع دستور NUM DB 12 چیست؟

|       |         |    |
|-------|---------|----|
| DATA  | SEGMENT |    |
| TOTAL | DW      | 23 |
| SUM   | DB      | 8  |
| NUM   | DB      | 12 |
| DATA  | ENDS    |    |

الف. 0004

ب. 0003

ج. 0002

د. 0001

۷۲- کوچکترین و بزرگترین عملوندهای عددی دستور DB کدام یک از موارد

ذیل است؟

الف. 128,128 -

ب. 127,128 -

ج. 128,255 -

د. 255, -255

۷۳- مکمل ۲، عدد ۱۱۰، کدام یک از موارد ذیل است؟

الف. 1001

ب. 1010

ج. 1110

د. 0111

۷۴- در مورد استفاده از دستورالعمل INC AX و دستورالعمل ADD AX,1 کدام

مورد غلط است؟

الف. سرعت اجرای INC AX از ADD AX,1 بیشتر است.

ب. حافظه مصرفی INC AX از ADD AX کمتر است.

ج. نتیجه (مقدار AX) اجرای این دو دستورالعمل یکسان است.

د. هیچکدام

۷۵- در مورد قطعه کد ذیل کدام یک از موارد ذیل صحیح است؟

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

الف. این قطعه کد دارای خطا بوده و ترجمه نمی‌شود

ب. این کد هیچ تغییری در مقادیر AX و BX ایجاد نمی‌کند

ج. مقدار AX و BX را تعویض می‌کند.

د. مقدار AX و BX را دو برابر می‌کند.

۷۶- قطعه کد ذیل پیاده سازی کدام یک از موارد ذیل است؟

```
CMP SUM,200
JGE A
CMP A,VER,5
JNE B
A: MOV COUNT,100
B:
```

الف. IF (SUM>=200 OR AVER=5)

COUNT=100

ب. IF (SUM>=200 AND AVER=5)

COUNT=100

ج. IF(SUM>=200 OF AVER< >5)

COUNT=100

د. IF (SUM<=200 AND AVER< > 5)

COUNT=100

۷۷- کدام یک از دستورات ذیل مجاز است؟

ب. CMP 100,AX

الف. PUSHF AX

د. هیچکدام

ج. MOV [BX],0

۷۸- کدام یک از دستورالعملهای ذیل روی فلگها موثرند؟

الف. JCXZ

ب. PUSHF

ج. CMC

د. XCHG

۷۹- پیشوند تکرار REPZ تحت کدام یک از شرایط ذیل تکرار را ادامه

می دهد؟

الف.  $zf=1$  یا  $CX=0$

ب.  $zf=0$  یا  $CX=0$

ج.  $zf=1$  و  $CX > 0$

د.  $zf=0$  و  $CX > 0$

۸۰- دستورالعمل XOR CL,11111111 معادل کدام دستورالعمل ذیل است؟

الف. NOT CL

ب. OR CL,CL

ج. AND CL,00000000

د. TEST CL 11111111

۸۱- کدام یک از موارد ذیل صحیح نیست:

الف. روالهای NEAR در همان سگمنت کدی که فراخوانی می شود، تعریف می گردند.

ب. روالهای FAR در سگمنتی جدا از سگمنت کد فراخواننده روال قرار دارد.

ج. دستور CALL برای فراخوانی روالهای FAR ثباتهای CS و IP و برای روال

NEAR فقط IP را در پشته ذخیره می کند.

د. وقتی مقداری در پشته قرار می گیرد SP افزایش می یابد.

۸۲- قطعه برنامه ذیل را در نظر بگیرد.

```
MOV AX,1
MOV CX,10
P:
INC AX
LOOP P
```

پس از اجرای قطعه برنامه مقدار AX چیست؟

- الف. ۹
- ب. ۱۱
- ج. ۱۰
- د. ۶۵۵۳۶

۸۳- اگر AX=ffff باشد بعد از اجرای دستور AX NEG محتوای AX کدام یک از مقادیر ذیل (در مبنای ده) است.

- الف. ۱
- ب. -۱
- ج. ۰
- د. ۶۵۵۳۶

۸۴- اگر AX=0032 و DX=0000 و CX=000B باشد بعد از اجرای دستور IDIV CX مقادیر DX و AX کدام یک از موارد ذیل است؟

- الف. DX=0006 AX=0004
- ب. DX=000B AX=0006
- ج. DX=0006 AX=000B
- د. DX=0004 AX=0006



۸۵- اگر  $AX=ff50$  و  $CX=0023$  و  $CF=1$  باشد آنگاه بعد از اجرای دستورالعمل

$ADC\ AX,CX$  مقادیر  $AX$  و  $CF$  کدامیک از مقادیر ذیل است؟

الف.  $CF=0\ AX=FF74$

ب.  $CF=1\ AX=FF73$

ج.  $CF=0\ AX=FF73$

د.  $CF=1\ AX=FF74$

۸۶- تعریف ذیل چند بایت حافظه را تخصیص می دهد؟

```
P STRUC
C DW ?
C2 DW ?
C3 DB 10DUP(?)
P ENDS
S1 P 50DUP(<>)
```

الف. ۷۰۰

ب. ۱۴

ج. ۶۴

د. ۵۰۰

۸۷- اگر اختلاف عددی کد اسکی حروف کوچک و بزرگ ۳۲ باشد آنگاه برای

تبدیل حروف کوچک به حروف بزرگ و بالعکس از کدام دستورالعمل ذیل

می توان استفاده نمود (فرض کنید حرف مورد نظر در  $CL$  باشد).

الف.  $XOR\ CL,00100000$

ب.  $OR\ CL,00100000$

ج.  $NOT\ CL$

د.  $AND\ 00100000$

۸۸- اگر  $DX=D0$  باشد بعد از اجرای دستور  $ROL DX,1$  مقدار  $DX$  چه

خواهد بود؟

الف.  $A0AC$  ب.  $6828$

ج.  $A0AD$  د.  $ADD0$

۸۹- قطعه کد زیر به چه معنی است.

```

MOV CX,20
MOV BX,6
FOR: :
DEC BX
CMP BX,0
LOOPN1: FOR
```

الف. یک حلقه با ۲۰ بار اجرا

ب. یک حلقه با ۶ بار اجرا

ج. یک حلقه با ۲۶ بار اجرا

د. یک حلقه با ۱۳ بار اجرا

۹۰- دستورات زیر معادل کدام گزینه است.

```

CMP AX,10
JE L1
CMP BX,20
JNE LJ
ADD AX
LE ADD BX
```

الف.  $IF(AX<>10 \text{ AND } BX=20)$

$AX=AX+1$

$ELSE \ BX=BX+1$

ب.  $IF(AX<>10 \text{ AND } BX=20)$

$AX=AX+1$

$ELSE \ BX=BX+1$

ج.  $IF (AX=10 \text{ AND } BX<>20)$

$AX=AX+1$

$ELSE \ BX=BX+1$

د. هیچکدام

۹۱- کدام یک از دستورات زیر مجاز است.

الف. `shr ax,4`

ب. `sar bx,3`

ج. `imul 3`

د. `pushf`

۹۲- انجام کدام جمع زیر منجر به سرریز (overflow) خواهد شد.

الف. `0A07+01d3`

ب. `0206+FFB0`

ج. `483F+645A`

د. هیچکدام

۹۳- دستورالعمل `repne` تحت کدام یک از شرایط زیر تکرار را ادامه می‌دهد؟

الف. `cx<>0 or zf<>0`

ب. `cx<>0 or zf=0`

ج. `cx<>0 and zf=0`

د. `cx=0 and zf<>0`

۹۴- اگر `ax=0F` و `bx=0E` باشد بعد از اجرای دستورالعمل زیر فلگها به چه

صورت خواهد شد؟

`cmp bx,ax`

الف. `of=0, cf=1, sf=1`

ب. `of=0, cf=0, sf=0`

ج. `of=1, cf=1, sf=1`

د. `of=0, cf=1, sf=0`

۹۵- پس از اجرای دستورات زیر مقدار ax (در مبنای ۱۶) چیست؟

```
mov ax, 100h
stc
adc ax,ax
```

الف. 200

ب. 201

ج. 202

د. 199

۹۶- مقدار نهایی ax پس از اجرای کد زیر چیست؟

```
mov ax,x
mov bx,y
add bx,z
add bx,bx
ncg ax
add ax,bx
```

الف.  $2(z+y) - x$

ب.  $2(x+y - x)$

ج.  $2y+z - x$

د. هیچکدام

۹۷- برای اینکه فقط بیت سوم ax را یک کنیم از کدام یک از دستورات

می توانیم استفاده کنیم؟

الف. `or ax,4`

ب. `and ax,4`

ج. `xor ax,4`

د. `not ax,4`

۹۸- بعد از اجرای کد زیر مقدار CX چیست؟

```
mov cx,10
ll:
:
loop ll
```

الف. 10

ب. 0

ج. 1

د. -۱

۹۹- کدامیک از مجموعه دستورات زیر صحیح نیست؟

الف. in ax, 0ffh

ب. in ax, 07ch

ج. mov cx, 0717h

in ax, cx

د. mov dx, 07ch

in ax, dx

۱۰۰- پس از اجرای قطعه کد زیر مقدار AX چیست؟

```
MOV AX,1
MOV CL,4
SHL AX,CL
XOR AX,AX
```

الف. 000F

ب. FFF0

ج. 0000

د. FFFF

۱۰۱- اگر SP=00F0 باشد بعد از اجرای قطعه کد زیر مقدار SP چیست؟

```
PUSH AX
PUSH BX
PUSH CX
```

الف. 00EA

ب. 00F9

ج. 00F6

د. هیچکدام

۱۰۲- کدام مورد غلط است؟

الف. شیفت ریاضی و منطقی به چپ یکسان است.

ب. فلگ سرریزی برای شیفت چند بیتی تعریف نشده است.

ج. شیفت ریاضی و منطقی به راست یکسان هستند.

د. هیچکدام

۱۰۳- بعد از اجرای قطعه کد زیر مقدار AX چقدر است؟

```
SEGMENT DATA
DB A "ALI",13,10
DB B "REZA"
ENDS
CODE
MOV AX, Offest B
```

الف. 6

ب. 4

ج. 3

د. 10

۱۰۴- اگر si=0000 و di=0005 باشد مقادیر si,di بعد از اجرای قطعه کد زیر

چیست؟

```
Source Db "book"
Dest Db "book"
Lea Si,source
Lea Di,dest
Mov cx,5
Repne cmpsb
```

الف. SI=0000 , di=0005

ب. SI=0001 , di=0006

ج. SI=0004 , di=0009

د. هیچکدام

۱۰۵- اگر کد دستور add چهار بایت و mov دو بایت باشد اسمبلر برای

دستورات زیر چند بایت در نظر می گیرد؟

```
M1 Macro N1,n2
 Mov Ax,n1
 Add Ax,n2
 Endm
M1 Cx,bx
M1 Dx,cx
```

الف. 4

ب. 6

ج. 10

د. 20

۱۰۶- کدامیک از دستورالمعلهای ذیل روی فلگها اثر دارند؟

الف. CALL

ب. MOV

ج. INC

د. LOOP

۱۰۷- کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. SHL DL,5

ب. MOV CS,300

ج. MOV IP,3600

د. کلیه موارد بالا

۱۰۸- دستور تکرار زیر چند بار اجرا می‌گردد؟

```
XOR AX,AX
MOV CX,AX
LOOP1:
-
-
-
-
LOOP LOOP1
```

الف. اصلاً اجرا نمی‌شود

ب. یکبار اجرا می‌شود

ج. 65536 دفعه اجرا می‌شود

د. 32767 دفعه اجرا می‌شود

۱۰۹- با توجه به دستورالعملهای ذیل چند بایت از حافظه اشغال می‌شود؟

```
X DB 'P LEASE YWAIT'
Y DW 4DUP(?)
Z DD 35000,42000
```

الف. 20

ب. 27

ج. 7

د. 30



۱۱۰- قطعه برنامه زیر چه کاری را انجام می دهد؟

```
MOV AL, 'a'
AND AL, 0DFH
```

الف. حرف 'A' را به 'a' تبدیل می کند

ب. حرف 'a' را به 'A' تبدیل می کند

ج. محتوای AL را تغییر نمی دهد

د. هیچکدام

۱۱۱- کدام گزینه صحیح می باشد؟

الف. CLD مقدار DF را صفر می کند

ب. دستورالعمل LAHF بایت دارای ارزش کمتر رجیستر فلگ را در AH قرار می دهد

ج. دستور X و bx و LEA آدرس X را در رجیستر BX قرار می دهد

د. کلیه موارد بالا

۱۱۲- در مورد دستورالعمل CBW کدام گزینه صحیح می باشد؟

الف. روی هیچ فلگی اثر ندارد

ب. برای تبدیل محتوی یک بایت که در رجیستر AL قرار دارد بیک Word استفاده می شود

ج. الف و ب

د. در محتوی رجیستر AL را دو برابر می نماید

۱۱۳- در مورد دستورالعمل CMPS کدام گزینه صحیح می باشد؟

- الف. برنامه مقایسه دو مقدار استفاده می گردد
- ب. برای مقایسه دو رشته استفاده می گردد
- ج. استفاده از چنین دستوری مجاز نمی باشد
- د. هیچکدام

۱۱۴- برای مکمل نمودن بیت های شماره 7,5,2 رجیستر AL از چه دستورالعملی

استفاده می شود؟

- الف. XOR AL, 0A4H
- ب. AND AL, 0A4H
- ج. NOT AL
- د. NEG AL

۱۱۵- برای جابه جا نمودن دو مقدار حافظه X و Y از نوع WORD از کدام

دستورالعمل می توان استفاده نمود؟

- الف. MOV X,Y
- ب. XCHG X,Y
- ج. LEA X,Y
- د. هیچکدام

۱۱۶- در مورد آدرس شروع segmentها کدام گزینه صحیح می باشد؟

- الف. از هر آدرس دلخواهی در حافظه می توانند شروع شوند
- ب. بایستی قابل تقسیم بر 16 باشد
- ج. بایستی قابل تقسیم بر 8 باشد
- د. بایستی فرد باشد

۱۱۷- پس از اجرای عمل زیر کدام گزینه صحیح می باشد؟

A2B4+

88F3

| ZF | SF | CF | OF |      |
|----|----|----|----|------|
| 0  | 0  | 1  | 1  | الف. |
| 0  | 1  | 1  | 0  | ب.   |
| 0  | 0  | 1  | 1  | ج.   |
| 1  | 1  | 1  | 1  | د.   |

۱۱۸- دستورالعمل Z, LEA BX, معادل کدام دستورالعمل می باشد؟

الف. MOV BX,Z

ب. MOV BX OFFSET Z

ج. MOV BX OFFSET,Z

د. هیچکدام

۱۱۹- کدام گزینه صحیح است؟

الف. دستورالعمل CMP مانند دستورالعمل SUB عمل نموده ولی نتیجه در جایی

ذخیره نمی شود

ب. دستورالعمل CMP دقیقاً مانند دستورالعمل SUB عمل می نماید

ج. دستورالعمل CMP روی فلگ های AF و CF اثر ندارند

د. استفاده از دستور CMP AX,AX مجاز نیست

۱۲۰- اگر  $AX=300$ ,  $BL=-5$ ,  $AL=4$  کدامیک از دستورالعملهای زیر مجاز

نیست؟

الف.  $MUL\ BL$

ب.  $MUL\ AL,BL$

ج. الف و ب

د.  $IDIV\ BL$

۱۲۱- کدامیک از دستورالعملهای زیر باعث تغییر مقدار فلگ CF می شود؟

الف.  $CLC$

ب.  $CMC$

ج.  $STC$

د. کلیه موارد

۱۲۲- استفاده از کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف.  $PUSH\ CS$

ب.  $MOV\ IP,100$

ج.  $POP\ AL$

د. کلیه موارد

۱۲۳- کدامیک از دستورالعملهای ذیل روی فلگ ها بی اثرند؟

الف.  $PUSH,POP$

ب.  $PUSHF$

ج. الف و ب

د.  $POPF$

۱۲۴- بعد از دستورالعمل 20- , CMP AL از کدامیک از دستورالعملهای زیر

نمی توان استفاده نمود؟

الف. JL LAB1

ب. JNGE LAB1

ج. JB LAB1

د. JLE LAB1

۱۲۵- در دستورالعمل LAB1 JBE کنترل به LAB1 منتقل می شود اگر

الف.  $CF=0$  باشد

ب.  $ZF=0$  باشد

ج.  $CF=1$  و  $ZF=0$  باشد

د.  $CF=0$  و  $ZF=0$  باشد

۱۲۶- کدام گزینه غلط می باشد؟

الف. در ماکروها دستورالعملهایی که با ; شروع می شوند در برنامه لیست نمی شود.

ب. در فراخوانی ماکروها می توان بعضی از پارامترها را ذکر نکرد

ج. دستورالعمل STOS مقدار بعضی از فلگها را تغییر می دهد

د. دستورالعمل LODS روی هیچ فلگی اثر ندارد

۱۲۷- اگر  $CL=3$  و  $DL=8DH$  و  $CF=1$  باشد. پس از اجرای دستورالعمل CL

و DL مقدار ثبات DL برابر است با؟

الف. 11110001B

ب. 00010001B

ج. 10001000B

د. هیچکدام

۱۲۸- دستور تکرار زیر چند بار اجرا می‌گردد؟

```
MOV CX,10
LOOP1: -
 -
 -
 -
 MOV CX,5
 -
 -
 -
 LOOP LOOP1
```

الف. 10 بار

ب. بی‌نهایت بار

ج. 5 بار

د. 50 بار

۱۲۹- دستورالعملهای ذیل چه کاری را انجام می‌دهند؟

```
MOV AH, 01H
INT 21H
```

الف. منتظر می‌ماند که کلیدی از صفحه کلید فشار داده شود

ب. کارکتری از صفحه کلید داده می‌شود در AL قرار می‌دهد

ج. الف و ب

د. کارکتری از صفحه کلید داده می‌شود روی صفحه مانیتور نمایش می‌دهد

۱۳۰- کدام گزینه غلط است؟

الف. انتقال اطلاعات از حافظه به ثبات امکان پذیر است.

ب. انتقال اطلاعات از ثبات به حافظه امکان پذیر است.

ج. انتقال اطلاعات از ثبات به رجیستر امکان پذیر می‌باشد

د. انتقال اطلاعات از حافظه به حافظه مستقیماً امکان پذیر است

۱۳۱- دستورالعمل تکرار زیر را در نظر بگیرید:

```
Loop1: MOV CX,10
 :
 Loop Loop1
```

الف. بی نهایت بار اجرا می شود

ب. 5 بار اجرا می شود

ج. اصلاً اجرا نمی شود

د. ۰ بار اجرا می شود

۱۳۲- کدامیک از دستورالعملهای ذیل مجاز است؟

الف. MOV AX,BL

ب. MOV CS,100

ج. MOV DS,CS

د. MOV AX,[BX]

۱۳۳- دستورالعملهای ذیل را در نظر بگیرید:

```
CMP AL,- 5
```

```
JA LAB1
```

الف. استفاده از این دو دستورالعمل بترتیب داده شده صحیح است.

ب. JA بایستی به JG تغییر کند.

ج. مقادیر منفی را نمی توان در دستور CMP استفاده نمود.

د. JA بایستی به JL تغییر نماید.

۱۳۴- دستورالعمل تکرار زیر چند مرتبه اجرا می شود؟

```
MOV CX, 0
Loop1: ⋮
 Loop Loop1
```

الف. صفر مرتبه

ب. یکبار

ج. ۶۵۵۳۶ دفعه

د. دو مرتبه

۱۳۵- کدام گزینه غلط است؟

الف. دستورالعمل CBW روی هیچ فلگی اثر ندارد.

ب. دستورالعمل CWD روی هیچ فلگی اثر ندارد.

ج. در دستورالعمل CBW عملوند بایستی در ثبات AL قرار گیرد.

د. دستورالعمل CBW باعث تبدیل یک WORD به DOUBLE WORD می شود.

۱۳۶- کدام گزینه غلط می باشد؟

الف. برای مکمل نمودن تعدادی بیت در یک ثبات از دستورالعمل OR استفاده می گردد.

ب. برای صفر نمودن تعدادی بیت در یک ثبات از دستورالعمل AND استفاده می شود.

ج. دستورالعمل TEST نتیجه را در جایی ذخیره نمی کند.

د. دستورالعمل NOT روی هیچ فلگی اثر ندارد.

۱۳۷- دستورالعمل LEA BX, TAB+5 معادل

الف. MOV BX, OFFSET TAB+5

ب. MOV BX, TAB+5

ج. MOV TAB+5, BX

د. هیچکدام



۱۳۸- اگر محتوی AX برابر با  $101010000111001B$  باشد و محتوی BX برابر با  $100010101101010B$  باشد پس از اجرای دستور ADD AX,BX کدام گزینه صحیح است؟

الف.  $OF=1, SF=1, AF=1, PF=1$

ب.  $OF=1, SF=1, AF=0, PF=0$

ج.  $OF=1, SF=0, AF=1, PF=1$

د.  $OF=1, SF=0, AF=1, PF=1$

۱۳۹- کدام گزینه صحیح است؟

الف. دستور CALL روی هیچ فلگی اثر ندارد.

ب. دستور LODS روی هیچ فلگی اثر ندارد.

ج. دستور SUB AX,AX محتوی CF را صفر می کند

د. هر سه مورد

۱۴۰- برای مکمل نمودن بیت های شماره فرد ثبات AL از چه دستوری استفاده

می گردد؟

الف. XOR AL,AAH

ب. XOR AL,0AAH

ج. OR AL, 0AAH

د. AND AL, 0AAH

۱۴۱- کدامیک از دستورالعملهای ذیل مجاز نیست؟

الف. PUSHF AX

ب. POP AL

ج. PUSH AL

د. کلیه موارد

۱۴۲- دستورالعمل LAB JCXZ را در نظر بگیرید.

- الف. کنترل به LAB منتقل می شود اگر  $CX=0$  باشد.
- ب. کنترل به LAB منتقل می شود اگر  $CF=0$  باشد.
- ج. کنترل به LAB منتقل می شود اگر  $DF=0$  باشد.
- د. کنترل به LAB منتقل می شود اگر CX معادل صفر نباشد.

۱۴۳- کدامیک از دستورالعملهای ذیل مجاز نیست؟

- الف. NEG AL
- ب. NOT AL
- ج. PUSH AL
- د. هیچکدام

۱۴۴- در دستور JA LAB

- الف. کنترل به LAB منتقل می گردد اگر  $CF=0$  باشد.
- ب. کنترل به LAB منتقل می گردد اگر  $CF=1$  باشد.
- ج. کنترل به LAB منتقل می گردد اگر  $CF=0, ZF=0$  باشد.
- د. کنترل به LAB منتقل می گردد اگر  $ZF=1$  باشد.

۱۴۵- کدام گزینه غلط می باشد؟

- الف. دستورهای پرش روی هیچ فلگی اثر ندارد
- ب. دستور TEST روی هیچ فلگی اثر ندارد.
- ج. دستور MOV روی هیچ فلگی اثر ندارد.
- د. دستور LEA روی هیچ فلگی اثر ندارد.

۱۴۶- در مورد جمع دو مقدار از نوع WORD کدام گزینه غلط می باشد؟

- الف. اگر مقدار MSB نتیجه برابر با یک شود SF برابر یک می شود.
- ب. اگر تعداد بیت های یک در 8 بیت اول زوج باشد ZF برابر صفر می شود.
- ج. اگر نتیجه جمع صفر شود ZF برابر با صفر می شود.
- د. ب و ج

۱۴۷- کدام گزینه غلط می باشد؟

- الف. دستور XCHG روی هیچ فلگی اثر ندارد
- ب. استفاده از دستور XCHG باعث مبادله متحوی X,Y می گردد. (X,Y از نوع WORD می باشد.)
- ج. دستور AL NOT مکمل 1 مقدار AL را می دهد.
- د. دستور AL NEG مکمل 2 مقدار AL را می دهد.

۱۴۸- در مورد دستورالعمل INC کدام گزینه غلط می باشد؟

- الف. روی CF اثر دارد.
- ب. روی CF اثر ندارد
- ج. روی SF اثر دارد
- د. روی ZF اثر دارد

۱۴۹- کدام دستورالعمل مجاز نیست؟

- الف. SUB 100,AL
- ب. IMUL 20
- ج. DIV 100
- د. کلیه موارد

### ۱۵۰- در دستورالعمل MUL BL

- الف. اگر OF و CF برابر با یک شوند نتیجه حاصل ضرب در یک بایت جا نمی شود.
- ب. دستورالعمل MUL فقط روی OF و CF اثر دارند.
- ج. در دستورالعمل MUL عملوند نبایستی مقدار ثابتی باشد.
- د. کلیه موارد بالا

### ۱۵۱- کدامیک از دستورالعملهای ذیل مجاز نیست؟

- الف. SHL OPR, CL
- ب. ROL OPR, 2
- ج. SAL OPR, CL
- د. STD

### ۱۵۲- کدام گزینه صحیح است؟

- الف. SCAS برای پویش یک رشته جهت وجود یا عدم وجود یک عنصر رشته ای معین بکار می رود.
- ب. دستورالعمل MOVS بر روی هیچ فلگی اثر نمی گذارد.
- ج. دستورالعمل CMPS برای مقایسه محتوی دو رشته بکار می رود.
- د. کلیه موارد بالا

### ۱۵۳- کدام گزینه صحیح نیست؟

- الف. پارامترها در دستور MACRO، نمادهای معمولی هستند که بوسیله علامت کاما از یکدیگر جدا می شوند.
- ب. تعریف یک MACRO شبیه تعریف یک روال در یک زبان سطح بالا می باشد.
- ج. یک MACRO می تواند در هر جای برنامه اسمبلی تعریف شود.
- د. توضیحاتی که با :: در یک MACRO شروع می شود هرگز لیست نمی شود.

۱۵۴- دستورالعملهای ذیل را در نظر بگیرید. با این دستورالعملها چند بایت حافظه تخصیص می یابد؟

X DB 10 DUP(?)  
Y DB 'WAIT', 13, 20  
Z DW 4DUP(?)

الف. 23

ب. 20

ج. 24

د. 22

۱۵۵- چرا از زبان اسمبلی استفاده می کنیم؟ (دو پردازنده های معمولی مثل 68060, Pentium, 8086 و ...)

- الف. بدلیل سرعت بیشتر نسبت به زبان سطح بالا
- ب. بعلت قابلیت زیاد در کنترل مستقیم روی پردازنده
- ج. بدلیل حجم کوچکتر برنامه نسبت به زبان سطح بالا
- د. هر سه مورد

۱۵۶- ثباتهای همه منظوره کدامند؟

الف. SI, DI, SP, IP, BP

ب. CS, SS, DS, ES

ج. DX, CX, BX, AX

د. CS, IP, Flags

۱۵۷- از ثابت Flags چه استفاده‌ای می‌شود؟ (پرچم‌ها=Flags)

- الف. برای انتقال اطلاعات از حافظه به داخل CPU استفاده می‌شود.
- ب. برای انتقال اطلاعات از CPU به حافظه استفاده می‌شود.
- ج. برای تصمیم‌گیری‌ها از آن استفاده نمی‌شود.
- د. هیچکدام

۱۵۸- پشته یا Stack چیست؟

- الف. قسمتی از حافظه جانبی که به منظور ذخیره سازی اطلاعات از آن استفاده می‌شود.
- ب. قسمتی از حافظه اصلی که به منظور ذخیره سازی بازیابی اطلاعات خصوصاً در هنگام Int, Call استفاده می‌شود.
- ج. قسمتی از دیسک سخت که در مقابل دستیابی cpu های دیگر محافظت شده است.
- د. قسمتی از حافظه کمکی که در مقابل "خواند شدن" محافظت شده است.

۱۵۹- از پشته یا Stack اطلاعات به چه صورت ذخیره می‌شوند؟

- الف. LIFO (last in first out و آخرین ورود اولین خروج)
- ب. FIFO (First in first out و اولین ورود اولین خروج)
- ج. هم LIFO و هم FIFO
- د. یا LIFO یا FIFO

۱۶۰- چگونه از دو ثابت 16 بیتی، می‌توان یک آدرس 20 بیتی ساخت؟

- الف. امکان ندارد.
- ب. یکی از ثابت‌ها را در عدد 16 ضرب کنیم و آن یکی دیگر را با آن جمع کنیم.
- ج. یکی از ثابت‌ها را چهار بیت به سمت چپ Shift دهیم و ثابت دیگر را با آن جمع کنیم.
- د. موارد ب و ج

۱۶۱- ثبات ES برای یک "ثبات قطعه داده اضافی" مفید واقع شده است. اگر چنین است پس چرا بیش از یک ثبات اضافی (مثلاً FS, GS و ...) نداشته باشیم؟

- الف. بیش از یکی مورد نیاز مرکز واقع نخواهد شد.
- ب. تکنولوژی زمان ساخت 8086 محدودیت داشته است ... اینکه از سیستم 80386 آنرا می بینیم (یعنی ثباتهای اضافی علاوه بر FS)
- ج. اصولاً ثبات ES بعنوان یک ثبات داده اضافی استفاده نمی شود.
- د. ممکن است این کار عملی نباشد، اما تاکنون مشاهده شده است (یعنی استفاده از ثباتهای قطعه علاوه بر FS)

#### ۱۶۲- تفاوت فایل COM, EXE چیست؟

- الف. فایل EXE محدودیت اندازه ندارد ولی فایل COM حداکثر به اندازه یک قطعه می باشد.
- ب. فایل COM از آدرس 100H شروع می شود ولی فایل EXE چنین نیست.
- ج. اصولاً این دو در واقع یکی هستند و هر دو فایل های قابل اجرا نیستند و تفاوتی ندارند.
- د. موارد الف و ب

#### ۱۶۳- تفاوت Call و INT چیست؟

- الف. بوسیله Call می توان سیستم را وادار ساخت تا CS, IP را در پشته قرار دهد ولی بوسیله INT می توان این کار را نمود علاوه بر اینکه Floy را نیز ذخیره ساخت.
- ب. Call برای صدا زدن یک رویه بکار می رود و INT وجود ندارد.
- ج. تفاوتی ندارند و ارقام یکسان برای یک دستور می باشند.
- د. در Call فقط IP ذخیره می شود (در رشته) ولی در INT هم CS و هم IP ذخیره می شود.

۱۶۴- مجموعه سه دستورالعمل زیر چه عملی را انجام می دهد؟

```
XOR AX,BX
XOR BX,AX
XOR AX,BX
```

الف. محتوای ثبت AX را با BX عوض می کند.

ب. محتوای ثابت AX را به  $(AX \oplus BX)$  تبدیل می کند.

ج. محتوای ثابت BX را به  $(BX \oplus AX)$  تبدیل می کند.

د. هیچکدام.

۱۶۵- در 8086 CPU دو دستور وجود دارد که ظاهراً عیناً یک کار می کنند.

چرا این دو دستورالعمل در کنار یکدیگر وجود دارند؟ (در صورتیکه وجود یکی از آنها کافی به نظر می رسد):

```
INC DX,ADD AX,1
```

الف. بدلیل اینکه شاید برنامه نویس یکی از این دو فرم را نتواند بخاطر بسپارد .

ب. دستور ADD دو بایتی است در صورتیکه دستور INC DX یک بایتی است.

ج. دستور ADD یک بایتی است، در صورتیکه دستور INC DX دو بایتی است.

د. این دو دستور دقیقاً معادل می باشند و یک معادل باینری کاملاً یکسان دارند

166- زیر چند بار اجرا می شود: loop دستور

```
MOV CX,0
Lob1: Loop Lob1
```

الف. بی نهایت

ب. اصلاً اجرا نمی شود

ج. FFFF بار بعلاوه 1

د. FFFF بار

۱۶۷- هدف از FAT چه می باشد؟

الف. اختصاص فضای دیسک برای فایلها



ب. اختصاص فضای دیسک برای I/O

ج. مقیم ساختن برنامه‌ها

د. ایجاد توابع DOS

۱۶۸- هدف از ایجاد ماکرو چیست؟

الف. سادگی و کم کردن تعداد دستورالعمل

ب. ایجاد قابلیت خوانائی زیادتر

ج. موارد الف و ب

د. سرعت بخشیدن به اجرا برنامه

## واژه نامه

|                      |                          |
|----------------------|--------------------------|
| Abacus               | چرتکه                    |
| Abort                | متوقف کردن-ناقص تمام شدن |
| Abstract             | مجرد - انتزاع            |
| Access               | دسترسی                   |
| Access time          | زمان دسترسی              |
| Accounting           | حسابداری                 |
| Accumulator          | آکومولاتور- انباره- مخزن |
| Action               | عمل                      |
| Action cycle         | دوره یا سیکل عمل         |
| Action rate          | میزان عمل                |
| Active               | فعال                     |
| Actual               | واقعی                    |
| Actual address       | آدرس واقعی               |
| Actual decimal point | نقطه اعشار واقعی         |
| Add                  | جمع کردن                 |
| Addendum             | ضمیمه                    |
| Adder                | جمع کننده                |

|                     |                |
|---------------------|----------------|
| Addition            | جمع            |
| Additional          | اضافی          |
| Address             | آدرس - نشانی   |
| Addressee           | گیرنده - مخاطب |
| Addressing system   | سیستم آدرس دهی |
| Adjacent            | مجاور - نزدیک  |
| Adjective           | صفت            |
| Alarm               | سیگنال - آژیر  |
| Alarm display       | نمایش سیگنال   |
| Algebra             | جبر            |
| Algorithm           | الگوریتم       |
| Alphabet            | حروف الفباء    |
| Ambiguity           | ابهام          |
| Analog              | قیاسی          |
| Analysis            | آنالیز - تحلیل |
| And gate            | مدار 'و'       |
| And operator        | عملگر 'و'      |
| Application         | کاربرد         |
| Application program | برنامه کاربردی |

|                         |                                 |
|-------------------------|---------------------------------|
| Applicant               | متقاضی                          |
| Approach                | نزدیکی-تمایل-دسترسی             |
| Arbitrary               | اختیاری-دلخواه                  |
| Arithmetic              | (علم) حساب                      |
| Arithmetical            | حسابی                           |
| Arithmetic register     | ثبات محاسباتی                   |
| Arithmetic section      | قسمت محاسباتی                   |
| Arithmetic unit         | واحد محاسباتی                   |
| Arm                     | بازو                            |
| Array                   | آرایه                           |
| Artificial intelligence | هوش مصنوعی                      |
| Artificial language     | زبان مصنوعی                     |
| Ascending               | صعودی                           |
| Ascending sort          | مرتب نمودن صعودی                |
| ASCII                   | کد آمریکائی برای مبادله اطلاعات |
| Assemble                | مونتاژ-مونتاژ کردن              |
| Audio                   | شنوائی                          |
| Audit                   | رسیدگی-ممیزی                    |
| Automatic               | خودکار-اتوماتیک                 |

|                      |                      |
|----------------------|----------------------|
| Auxiliary            | کمکی                 |
| Auxiliary operations | عملیات کمکی          |
| Available            | موجود                |
| Background           | زمینه                |
| Back up              | پشتیبانی             |
| Back up system       | سیستم پشتیبانی       |
| Base data            | داده مبنا            |
| Base number          | عدد مبنا             |
| Base register        | ثبات مبنا یا شاخص    |
| Begin                | شروع کردن - شروع     |
| Bi Conditional       | دو شرطی              |
| Bidirectional        | دو جهتی              |
| Binary               | دودویی               |
| Binary code          | کد دودویی            |
| Binary digits        | ارقام دودویی         |
| Binary half adder    | نیم جمع کننده دودویی |
| Binary logic         | منطق دودویی          |
| Binary numbers       | اعداد دودویی         |
| Binary notation      | نمایش دودویی         |

|                  |                             |
|------------------|-----------------------------|
| Binary operation | عملیات دودویی               |
| Binary variable  | متغیر دودویی                |
| Bit              | رقم 0 یا 1                  |
| Bit pattern      | الگوی بیتی                  |
| Blank            | خالی - فاصله                |
| Block            | بلوک                        |
| Block entry      | ورودی بلوک                  |
| Boolean          | بول                         |
| Boolean algebra  | جبر بول                     |
| Branch           | شاخه                        |
| Branching        | شاخه شاخه کردن - منشعب کردن |
| Bubble sort      | مرتب کردن حبابی             |
| Buffer           | بافر - قسمتی از حافظه اصلی  |
| Button           | دکمه                        |
| Cable            | کابل                        |
| Cache memory     | حافظه نهان                  |
| Calculate        | محاسبه کردن                 |
| Calculator       | ماشین حساب                  |
| Capacity         | ظرفیت                       |

|                       |                       |
|-----------------------|-----------------------|
| Cell                  | سلول                  |
| Center                | مرکز                  |
| Central               | مرکزی                 |
| Central processor     | پردازشگر مرکزی        |
| Chain                 | زنجیر                 |
| Character recognition | شناخت کارکتر          |
| Check bit             | بیت کنترل             |
| Check digit           | رقم کنترلی            |
| Chip                  | تراشه                 |
| Circuit               | مدار                  |
| Clear                 | پاک کردن              |
| Column                | ستون                  |
| Combination           | ترکیب                 |
| Command               | فرمان                 |
| Comment               | تفسیر-ملاحظات - نظریه |
| Communications        | ارتباطات              |
| Compare               | مقایسه کردن           |
| Comparing unit        | واحد مقایسه کننده     |
| Comparison            | مقایسه                |

|                    |                |
|--------------------|----------------|
| Compatible         | سازگار کردن    |
| Compatiblity       | سازگاری        |
| Compile            | ترجمه کردن     |
| Compiler           | مترجم          |
| Compiling phase    | فاز ترجمه      |
| Complement         | مکمل           |
| Component          | مؤلفه          |
| Compound condition | شرط ترکیبی     |
| Compress           | فشرده کردن     |
| Compute            | محاسبه کردن    |
| Computer           | محاسبه کننده   |
| Computer network   | شبکه کامپیوتری |
| Condensed          | متراکم - خلاصه |
| Connect time       | زمان اتصال     |
| Content            | محتوی          |
| Context            | زمینه          |
| Continuous         | پیوسته         |
| Control cycle      | حلقه کنترل     |
| Conversion         | تبدیل          |



|                   |              |
|-------------------|--------------|
| Counter           | شمارنده      |
| Critical          | بحرانی       |
| Critical path     | مسیر بحرانی  |
| CRT               | صفحه نمایش   |
| Cylinder          | سیلندر       |
| Data              | داده         |
| Data base         | پایگاه داده  |
| Data description  | توصیف داده   |
| Date error        | خطای داده    |
| Data location     | محل داده     |
| Data management   | مدیریت داده  |
| Data manipulation | دستکاری داده |
| Data preparation  | تدارک داده   |
| Data processing   | پردازش داده  |
| Data rules        | قواعد داده   |
| Data transfer     | انتقال داده  |
| Deciding          | تصمیم گیری   |
| Decimal           | اعشاری       |
| Decimal digit     | رقم اعشاری   |

|                          |                    |
|--------------------------|--------------------|
| Decimal number           | عدد اعشاری         |
| Decimal notation         | نمایش اعشاری       |
| Decimal numbering system | سیستم اعداد اعشاری |
| Decimal point            | نقطه اعشار         |
| Decision                 | تصمیم گیری         |
| Decision box             | جعبه تصمیم گیری    |
| Decision logic           | منطق تصمیم گیری    |
| Decision – making system | سیستم تصمیم گیری   |
| Decision mechanism       | مکانیزم تصمیم گیری |
| Decision rules           | قواعد تصمیم گیری   |
| Define                   | تعریف کردن         |
| Definition               | تعریف              |
| Delay                    | تأخیر              |
| Delay time               | زمان تأخیر         |
| Delete                   | حذف                |
| Density                  | چگالی – دانسیته    |
| Dependent variables      | متغیرهای وابسته    |
| Descending               | نزولی              |
| Descending sort          | مرتب سازی نزولی    |

|                           |                 |
|---------------------------|-----------------|
| Design                    | طرح             |
| Device                    | وسيله - دستگاه  |
| Diagnosis                 | تشخيص           |
| Diagram                   | نمودار          |
| Difference                | اختلاف          |
| Digital clock             | ساعت دیجیتال    |
| Disk                      | دیسک            |
| Disk operating system DOS | سیستم عامل دیسک |
| Display                   | نمایش           |
| Display unit              | واحد نمایش      |
| Distance                  | فاصله           |
| Distribute                | توزیع کردن      |
| Division                  | تقسیم           |
| Document                  | مدرک - سند      |
| Documentation             | مستندات         |
| Double length             | طول مضاعف       |
| Double precision          | دقت مضاعف       |
| Dynamic                   | پویا            |
| Dynamic memory            | حافظه پویا      |

|                   |                      |
|-------------------|----------------------|
| Effective         | مؤثر                 |
| Effective speed   | سرعت مؤثر            |
| Effective address | آدرس مؤثر            |
| Efficiency        | کارایی               |
| End of file       | انتهای فایل          |
| Epitome           | رئوس مطالب-خلاصه     |
| Equality          | کیفیت                |
| Equipment         | تجهیزات              |
| Erase             | از بین بردن-پاک کردن |
| Eraser            | پاک کننده-پاک کن     |
| Error             | خطا                  |
| Error code        | رمز خطا              |
| Error detection   | تشخیص خطا            |
| Error message     | پیغام خطا            |
| Error rate        | نرخ خطا              |
| Exchange          | مبادله               |
| Execute           | اجرا کردن            |
| Execution         | اجرا                 |
| Execution cycle   | سیکل اجرا            |

|                        |                           |
|------------------------|---------------------------|
| Exit                   | خروج                      |
| Exit point             | نقطه خروج                 |
| Expression             | عبارت                     |
| Extension              | بسط - توسعه - تمدید       |
| External               | خارجی                     |
| External interrupt     | وقفه خارجی                |
| External labels        | برچسب‌های خارجی           |
| Facility               | تسهیلات                   |
| Factor                 | فاکتور - عامل - ضریب      |
| Failure                | شکست - خرابی              |
| Fixed                  | ثابت                      |
| Fixed length           | طول ثابت                  |
| Flag                   | فلگ - پرچم                |
| Floating point         | نقطه اعشار شناور          |
| Floating point numbers | اعداد با نقطه اعشار شناور |
| Flowchart              | فلوچارت - نمودار          |
| Folder                 | پوشه                      |
| Format                 | قالب - شکل                |
| Gate                   | دروازه مدار               |

|             |             |
|-------------|-------------|
| General     | عمومی       |
| Generator   | ایجاد کننده |
| Global      | سراسری      |
| Half        | نصف-نیم     |
| Halt        | متوقف کردن  |
| Hardware    | سخت افزار   |
| Heading     | تیتر-عنوان  |
| Help        | کمک         |
| High speed  | سرعت بالا   |
| High order  | مرتبه بالا  |
| Identifier  | شناسه       |
| Idle time   | زمان بیکاری |
| Ignore      | اغماض کردن  |
| Image       | تصویر       |
| Implicit    | ضمنی        |
| Inactive    | غیرفعال     |
| Inclusive   | دربرگیرنده  |
| Increment   | افزایش      |
| Independent | مستقل       |

|                              |                        |
|------------------------------|------------------------|
| Index                        | اندیس-شاخص             |
| Indexed address              | آدرس شاخص دار          |
| Indicate                     | نشان دادن - تعیین کردن |
| Indirect                     | غیر مستقیم             |
| Indirect address             | آدرس غیر مستقیم        |
| Indirect reference address   | آدرس ارجاع غیر مستقیم  |
| Information                  | اطلاعات                |
| Information retrieval        | بازیابی اطلاعات        |
| Input                        | ورودی                  |
| Input data                   | داده ورودی             |
| Instruction                  | دستورالعمل             |
| Instruction address          | آدرس دستورالعمل        |
| Instruction address register | ثبات آدرس دستورالعمل   |
| Integer                      | صحیح                   |
| Integrated system            | سیستم مجتمع            |
| Interface                    | واسطه-میانجی           |
| Internal                     | داخلی                  |
| Internal code                | کد یا رمز داخلی        |
| Internal sort                | مرتب سازی داخلی        |

|                   |                                                 |
|-------------------|-------------------------------------------------|
| Interrupt         | وقفه                                            |
| Interrupt system  | سیستم وقفه                                      |
| Iteration         | تکرار                                           |
| Iterative         | تکراری                                          |
| Key               | کلید                                            |
| Keyboard          | صفحه کلید                                       |
| Label             | برچسب                                           |
| Language          | زبان                                            |
| Language compiler | مترجم زبان                                      |
| Level             | سطح                                             |
| Library           | کتابخانه-مرکز اسناد                             |
| Light             | نور                                             |
| Limit             | محدود کردن                                      |
| Limited           | محدود                                           |
| Link              | متصل کردن-اتصال                                 |
| List              | فهرست                                           |
| Load              | بارکردن                                         |
| Loader            | برنامه سیستم عامل برای بار کردن برنامه در حافظه |
| Location          | محل - جا                                        |
| Logical           | منطقی                                           |
| Logical operation | عملیات منطقی                                    |
| Logical operator  | عملگر منطقی                                     |
| Logical shift     | شیفت منطقی                                      |
| Loop              | حلقه-حلقه تکرار                                 |



|                         |                      |
|-------------------------|----------------------|
| Low                     | پائین                |
| Low order               | مرتبه پائین          |
| Low order digit         | رقم مرتبه پائین      |
| LSD                     | بیت باکمترین ارزش    |
| Magnetic                | مغناطیسی             |
| Magnetic disk           | دیسک مغناطیسی        |
| Main                    | اصلی                 |
| Maintenance             | تعمیر و نگهداری      |
| Major                   | اصلی                 |
| Major key               | کلید اصلی            |
| Malfunction             | خرابی-از کار افتادگی |
| Match                   | تطبیق-جور            |
| Matching error          | خطای تطبیق           |
| Mathematical            | ریاضی                |
| Matrix                  | ماتریس               |
| Mechanism               | مکانیزم-روش          |
| Medium                  | وسیله برای ضبط داده  |
| Memory                  | حافظه                |
| Memory address register | ثبات آدرس حافظه      |
| Memory cycle            | سیکل یا دوره حافظه   |
| Memory dump             | تخلیه حافظه          |
| Memory exchange         | مبادله حافظه         |
| Memory hierarchy        | سلسله مراتب حافظه    |
| Memory location         | محل حافظه            |

|                   |                                |
|-------------------|--------------------------------|
| Memory protection | حفاظت حافظه                    |
| Memory register   | ثبات حافظه                     |
| Merge             | ادغام                          |
| Message           | پیغام                          |
| Message exchange  | مبادله پیغام                   |
| Method            | روش                            |
| Microsecond       | میلیونیوم ثانیه-میکروثانیه     |
| Minimum           | کوچکترین                       |
| Mistake           | اشتباه                         |
| Mode              | روش-طرز-طریقه                  |
| Module            | ماژول-برنامه                   |
| Monitor           | صفحه نمایش                     |
| Multiply          | ضرب کردن                       |
| Multiplication    | ضرب                            |
| Multiprocessor    | کامپیوتر دارای چند ریزپردازنده |
| Negative          | منفی                           |
| Notation          | نمایش                          |
| Number            | عدد                            |
| Numeric           | عددی                           |
| Operand           | عملوند                         |
| Operand address   | آدرس عملوند                    |
| Operating system  | سیستم اجرایی                   |
| Operation         | عمل                            |
| Operational unit  | واحد اجرایی                    |

|                      |                           |
|----------------------|---------------------------|
| Operator             | اپراتور-عملگر             |
| Operator error       | خطای اپراتور              |
| Option               | گزینه                     |
| Order                | ترتیب                     |
| Out of range         | خارج از دامنه             |
| Output               | خروجی                     |
| Output equipment     | تجهیزات خروجی             |
| Output unit          | واحد خروجی                |
| Overflow             | سرریزی-سرریز شدن          |
| Overflow check       | بررسی سرریزی              |
| Page                 | صفحه                      |
| Page address         | آدرس صفحه                 |
| Page heading         | عنوان صفحه                |
| Paging               | صفحه به صفحه کردن         |
| Parallel             | موازی                     |
| Parallel operation   | عملیات موازی              |
| Parity bit           | بیت توازن                 |
| Path                 | مسیر                      |
| Pattern              | الگو                      |
| Pattern recognition  | تشخیص الگو                |
| Perform              | انجام دادن-عمل کردن       |
| Peripheral           | جانبی                     |
| Peripheral equipment | وسایل جانبی-تجهیزات جانبی |
| Permanent            | دائم                      |

|                     |                         |
|---------------------|-------------------------|
| Phase               | فاز-مرحله               |
| Point               | نقطه                    |
| Pointer             | اشاره گر                |
| Position            | موقعیت - جا             |
| Precision           | دقت                     |
| Predefined          | از قبل تعریف شده        |
| Primary             | اصلی-اولیه              |
| Printer             | چاپگر                   |
| Priority            | اولویت                  |
| Priority interrupt  | وقفه اولویت دار         |
| Problem             | مسأله                   |
| Procedural          | رویه ای                 |
| Procedure           | روال-رویه               |
| Process             | فرآیند                  |
| Processor           | پردازشگر                |
| Production          | تولید-محصول             |
| Program             | برنامه                  |
| Program compilation | ترجمه برنامه            |
| Program error       | خطای برنامه             |
| Programmer          | برنامه نویس             |
| Programming         | برنامه نویسی            |
| Program segment     | قطعه یا قسمتی از برنامه |
| Quantity            | مقدار                   |
| Queue               | صف                      |

|                  |                       |
|------------------|-----------------------|
| Ram              | حافظه اصلی کامپیوتر   |
| Random           | تصادفی                |
| Random access    | دسترسی تصادفی         |
| Random number    | اعداد تصادفی          |
| Rapid memory     | حافظه سریع            |
| Rate             | نرخ                   |
| Ratio            | نسبت                  |
| Raw data         | داده‌های خام یا اولیه |
| Read             | خواندن                |
| Read time        | زمان خواندن           |
| Real number      | اعداد حقیقی           |
| Receive          | دریافت کردن           |
| Receiver         | دریافت کننده          |
| Record           | رکورد                 |
| Record length    | طول رکورد             |
| Register         | ثبات-رجیستر           |
| Register address | آدرس ثبات             |
| Relative         | نسبی                  |
| Relative code    | کد نسبی               |
| Repeat           | تکرار کردن            |
| Repeat counter   | شاخص تکرار            |
| Repeater         | تکرار کننده           |
| Report           | گزارش کردن-گزارش      |
| Retrieve         | بازیابی               |

|                       |                                                     |
|-----------------------|-----------------------------------------------------|
| Return                | برگشت                                               |
| Rules                 | قواعد                                               |
| Run                   | اجرا کردن برنامه-اجرای برنامه                       |
| Sample                | نمونه                                               |
| Scan                  | با دقت نگاه کردن-بررسی کردن                         |
| Screen                | صفحه نمایش                                          |
| Section               | بخش                                                 |
| Segment               | قطعه-قسمت                                           |
| Segmentation          | قسمت کردن-قطعه قطعه کردن                            |
| Separator             | جدا کننده                                           |
| Sequence              | دنباله-ترتیب                                        |
| Sequential            | متوالی                                              |
| Sequential processing | پردازش متوالی                                       |
| Serial access         | دسترسی سری                                          |
| Set                   | قراردادن-تعیین کردن-تنظیم کردن -میزان کردن - مجموعه |
| Shared storage        | حافظه اشتراکی                                       |
| Shift                 | جابجایی                                             |
| Sign                  | علامت                                               |
| Signed                | علامتدار                                            |
| Significance          | باهمیت                                              |
| Software              | نرم افزار                                           |
| Solution              | راه حل                                              |
| Sort                  | مرتب نمودن                                          |
| Sort bubble           | مرتب کردن به روش حبابی                              |

|                  |                    |
|------------------|--------------------|
| Space            | فضا                |
| Stack            | پشته               |
| Standard         | استاندارد          |
| Status           | وضعیت              |
| Step             | قدم-مرحله-گام      |
| Storage          | حافظه              |
| Storage block    | بلوک حافظه         |
| String           | رشته               |
| Symbol           | نماد               |
| Symbolic address | آدرس نمادی         |
| Symbol table     | جدول نمادی         |
| System           | سیستم - دستگاه     |
| Table            | جدول               |
| Table lookup     | جستجوی جدول        |
| Tape             | نوار مغناطیسی      |
| Temporary        | موقت               |
| Terminal         | ترمینال-نقطه نهائی |
| Test data        | داده جهت تست       |
| Time             | زمان               |
| Time scale       | مقیاس زمان         |
| Transmission     | انتقال             |
| Unconditional    | بدون شرط           |
| Unit             | واحد-قسمت          |
| User             | کاربر              |

|                 |                 |
|-----------------|-----------------|
| Variable        | متغیر           |
| Variable length | بطول متغیر      |
| Verify          | بررسی کردن      |
| Virtual         | مجازی           |
| Waiting state   | وضعیت انتظار    |
| Warning         | اخطار           |
| Word            | کلمه-شانزده بیت |
| Write           | نوشتن           |
| Zero            | صفر             |
| Zero divide     | تقسیم بر صفر    |