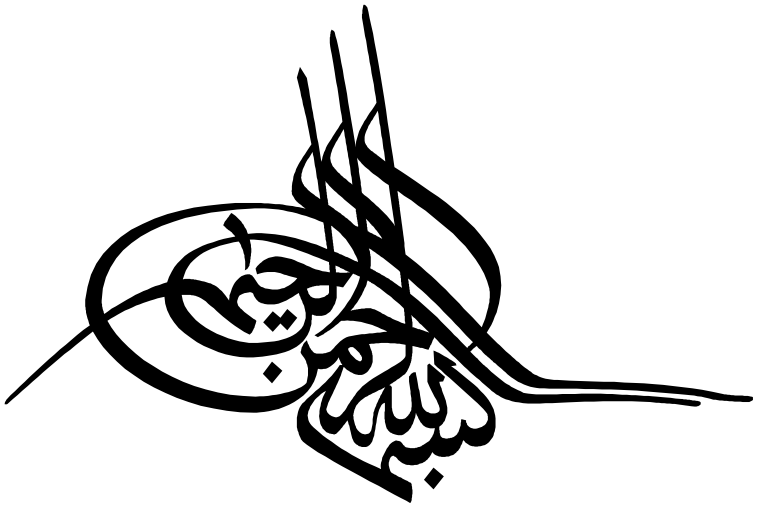
جزوه درس

برنامه سازی سیستم

تهیه کننده

علی چوداری خسروشاهی

ویژه دانشجویان کارشناسی ناپیوسته کامپیوتر – تکنولوژی نرم افزار


بسم الله الرحمن الرحيم

بنام خدا

پیشگفتار

ما تلاش کرده ایم در این جزوه تمام سرفصلهای درس پوشش داده و مطالب مفیدی را در اختیار شما قرار دهیم تا شما با خواندن آن کلیاتی را از درس بدست آورید. در مورد این جزوه یادآوری این نکته مهم، ضروری است که این جزوه جهت کمک به دانشجویان در کاهش یادداشت برداری، ترسیم شکلها، برنامه ها و مثالها در هنگام تدریس بوده است. بنابر این در کنار این جزوه، هر دانشجو مطابق با ذوق و سلیقه خود، جزوه ای دست نویس خواهد داشت که حاوی یادداشت ها به منظور تکمیل و تفهیم این جزوه است.

در اینجا لازم است از تلاش تمام دانشجویانی که اینجانب را در تدوین این جزوه یاری نموده اند کمال تشکر را داشته باشم. همچنین در این جزوه احتمال اشتباه وجود دارد، به این جهت در مطالعه مطالب جزوه دقت کافی را داشته باشید.

از دانشجویان عزیز تقاضا می‌شود در مطالعه مطالب آن دقت کافی را داشته باشند و وجود هرگونه ایراد ، و همچنین نظرات و پیشنهادات خود را به آدرس پست الکترونیکی اینجانب اطلاع دهند.

Akhosroshahi@IAUT.ac.ir
Akhosroshahi@Gmail.com

باتشکر

علی چوداری خسروشاهی

فهرست مطالب

فصل اول ۳

۱- اسمبلی پیشرفته ۳

۱-۱- پارامترهای خط فرمان ۳

۱-۱-۱- بخش PSP در برنامه های COM ۵

۱-۱-۲- بخش PSP در برنامه های EXE ۵

۱-۲- برنامه های مقیم در حافظه ۱۰

۱-۲-۱- گرفتن بردار وقفه ۱۱

۱-۲-۲- تظیم بردار وقفه ۱۱

۱-۲-۳- یک برنامه مقیم در حافظه ۱۱

۱-۳- مجموعه دستورات کامپیوتر های شخصی ۱۳

۱-۳-۱- نشانه گذاری ثبات ۱٤

۱-۳-۲- آدرسدهی بایت وضعیت ۱۵

۱-۳-۳- دستورات دو بایتی ۱۷

۱-۳-٤- دستورات سه بایتی ۱۸

۱-۳-۵- دستورات چهار بایتی ۱۹

۱-۳-٦- چند مثال مختلف ۲۰

۱-٤- تبدیل کد های زبان ماشین به اسمبلی ۲۱

فصل دوم ۲۲

۲- برنامه نویسی سیستمی تحت ویندوز ۲۲

۲-۱- برنامه نویسی نخ ۲۲

۲-۱-۱- متد Sleep ۲۵

۲-۱-۲- ارتباط Thread ها ۲٦

۲-۱-۳- ناحیه بحرانی ۲۹

۲-۱-٤- حل مسئله ناحیه بحرانی با استفاده از Lock ۲۹

۲-۱-۵- حل مسئله ناحیه بحرانی با استفاده از Semaphore ۳۱

۲-۱-٦- پارامتر دار کردن نخ ها ۳۲

۲-۱-۷- نام دار کردن نخ ها ۳٤

۸-۱-۲- نخ پس زمینه ۳٤
۹-۱-۲- متد JOIN ۳۵
۲-۲- توابع API ویندوز ۳۵
۳-۲- دلیل استفاده از توابع API در برنامه‌نویسی ۳۷
۱-۳-۲- فایلهای کتابخانه ای پویا ۳۷
۲-۳-۲- معماری ویندوز ۳۸
۳-۳-۲- استفاده از توابع API در C# ۳۸
٤-۳-۲- تابع MessageBox ۳۹
۵-۳-۲- تابع GetVersionEx ٤۰
٦-۳-۲- تابع GetSystemInfo ٤۲
۷-۳-۲- تابع GlobalMemoryStatus ٤۳
۸-۳-۲- تابع GetLastError ٤٤
۹-۳-۲- تابع Sleep و SleepEx ٤۵
۱۰-۳-۲- تابع GetLocalTime ٤٦
۱۱-۳-۲- تابع SetLocalTime ٤۷
۱۲-۳-۲- تابع CreateDC ٤۸
۱۳-۳-۲- تابع DeleteDC ٤۸
۱٤-۳-۲- تابع Drowtext ٤۹
۱۵-۳-۲- تابع TextOut ۵۰
۱٦-۳-۲- تابع AngleArc ۵۰
۱۷-۳-۲- توابع API رشتهای (String API) ۵۱
۱-۱۷-۳-۲- تابع IsCharAlpha ۵۱
۲-۱۷-۳-۲- تابع IsCharAlphaNumeric ۵۱
۳-۱۷-۳-۲- تابع IsCharUpper و IsCharLower ۵۱
٤-۱۷-۳-۲- تابع Lstrcmp ۵۲
۵-۱۷-۳-۲- تابع Lstrlen ۵۲

ضمیمه A: مجموعه دستورات پردازنده های رده X86 شرکت اینتل ۵۳

فصل اول

۱- اسمبلی پیشرفته

۱-۱- پارامترهای خط فرمان

فایل‌هایی با پسوند exe و com جزء فایل‌های اجرایی سیستم عامل Dos می‌باشند. وارد کردن نام این برنامه ها در خط فرمان و فشار دادن کلید Enter موجب اجرای آنها می شود. بعد از فشار دادن کلید Enter، Loader اقدام به لود کردن برنامه از روی هارد دیسک بر روی RAM خواهد کرد. در حین بارگذار برخی اعمالی نیز انجام می شود، به عنوان مثال مقداردهی بخشPSP می باشد. Loader، PSP را در آفست 00h تا FFh ایجاد می‌کند و برنامه از آفست 100h به بعد قرار خواهد گرفت. PSP حاوی فیلدهایی می‌باشد که در صفحه ۳۹۴ کتاب به آن اشاره شده است.

از جمله اطلاعاتی که در PSP ثبت می‌شود مقدار پارامترهای خط فرمان می‌باشد. در PSP ، آفست 5Ch تا 6Bh مشخص کننده ناحیه پارامتر یک (FCB#1) و آفست 6Ch تا 7Fh ناحیه پارامتر دو (FCB#2) و آفست 80h تا FFh بافر برای DTA پیش‌فرض می باشد. این ناحیه شامل تمام کاراکترهایی می باشد که بعد از نام برنامه وارد کرده‌اید. در اولین بایت DTA تعداد کاراکترهای وارد شده بعد از نام برنامه قرار می‌گیرد. سپس به ترتیب کاراکترهای وارد شده قرار خواهند گرفت.

5CH تا 6BH	ناحیه پارامتر ۱ که FCB#1 نیز گفته می‌شود.
6CH تا 7FH	ناحیه پارامتر 2 که FCB#2 نیز گفته می‌شود.
80H تا FFH	ناحیه بافر برای DTA پیش فرض می‌باشد که کل عبارات وارد شده بعد از نام برنامه در این بخش قرار می‌گیرند

شکل ۱-۱

برای پارامترهای خط فرمان حالت‌های مختلفی وجود دارد:

حالت اول: فرمان بدون عملوند مانند

```
C:\>Test.Com↵
```

وقتی چنین درخواستی داده شود Loader ، FCB#1 ، FCB#2 و DTA را به صورت زیر تنظیم خواهد کرد:

FCB#1	00 20 20 20 20 20 20 20 20 20 20 20
FCB#2	00 20 20 20 20 20 20 20 20 20 20 20
DTA	00 0D .....

در FCB#1 و FCB#2 اولین بایت برابر صفحه می‌باشد که به درایو پیش‌فرض اشاره می‌کند و بایت‌های دوم به بعد برابر 20h می‌باشد که معرف کاراکتر space می‌باشد، چون هیچ پارامتری وارد نشده است. در DTA اولین بایت معرف تعداد کاراکترهای وارد شده است. چون هیچ کاراکتری وارد نشده است این بخش برابر صفر (00) خواهد بود. بایت بعدی برابر 0D خواهد بود که معرف کلید Enter می‌باشد.

حالت دوم: در خط فرمان فقط یک پارامتر وارد کنیم. مانند

C:\> color BY↵

FCB#1 و FCB#2 و DTA را به صورت زیر تنظیم خواهد شد:

		B	Y								
FCB#1:	00	42	59	20	20	20	20	20	20	20	20
FCB#2:	00	20	20	20	20	20	20	20	20	20	20
DTA:	03	20	42	59	0D	…					
			B	Y	↵						

در FCB#1 اولین بایت معرف درایو می باشد که 00 معرف درایو پیشفرض می باشد. هشت بایت بعدی معرف نام فایل می باشد در این مثال دو کاراکتر B (با کد اسکی 42h) و Y (با کد اسکی 59h) استفاده شده است و شش کاراکتر بعدی 20h می باشد که معرف کاراکتر فاصله می باشد. در آخر سه بایت برای نشان دادن پسوند در نظر گرفته شده است، چون در این مثال پسوند نداریم این سه بایت نیز برابر 20h خواهند بود.

چون پارامتر دوم نیز وارد نشده است در FCB#2 در بایت اول عدد 00 و در بایتهای بعدی 20h قرار خواهد گرفت. در DTA نیز بایت اول برابر تعداد کاراکتر های وارد شده خواهد بود در این مثال 03 سپس به ترتیب کاراکترها وارد شده قرار خواهند گرفت. ابتدا کاراکتر فاصله (20h) سپس B (42h) و Y (59h) قرار خواهد گرفت در انتها نیز کاراکتر 0Dh (Enter) قرار خواهد داشت.

برای این حالت مثال دیگری نیز بصورت زیر می تواند مطرح شود.

C:\>Del D:Readme.txt ↵

		R	E	A	D	M	E			T	X	T			
FCB#1:	04	52	45	41	44	4D	45	20	20	54	58	54	…		
FCB#2:	00	20	20	20	20	20	…								
DTA:	0D	20	44	3A	52	65	61	64	6D	65	2E	74	78	74	0D	…
			D	:	R	e	a	d	m	e	.	t	x	t	↵	

در این مثال همچنان که مشاهده می کنید در FSB#1 اولین بایت که نشان دهنده مسیر می باشد برابر 04 می باشد که همان درایو :D را مشخص می کند سپس هشت بایت نشان دهنده نام فایل با حروف بزرگ و سه بایت بعدی نشان دهنده پسوند فایل با حروف بزرگ می باشد. در DTA هم بایت اول نشان دهنده تعداد کاراکترهای وارد شده می باشد که برابر 0Dh (سیزده کاراکتر) می باشد، سپس تک تک کاراکتر های وارد شده به ترتیب عیناً آورده شده است.

نکته: دقت داشته باشید که در FCB#1 و FCB#2 نام فایل و پسوند آن با حروف بزرگ می آیند، اما در DTA کاراکترها به همان شکلی که وارد شده اند آورده می شوند.

حالت سوم: در خط فرمان دو پارامتر وارد کنیم، به عنوان مثال:

C:>Copy A:Filea.asm D:Fileb.asm ↵

		F	I	L	E	A				A	S	M						
FCB#1:	01	46	49	4C	45	41	20	20	20	41	53	4D	…					
		F	I	L	E	B				A	S	M						
FCB#2:	04	46	49	4C	45	42	20	20	20	41	53	4D	…					
DTA:	18	20	41	3A	46	69	6C	65	61	2E	61	73	6D	20	44	3A	46	…
			A	:	F	i	l	e	a	.	a	s	m		D	:	F	…

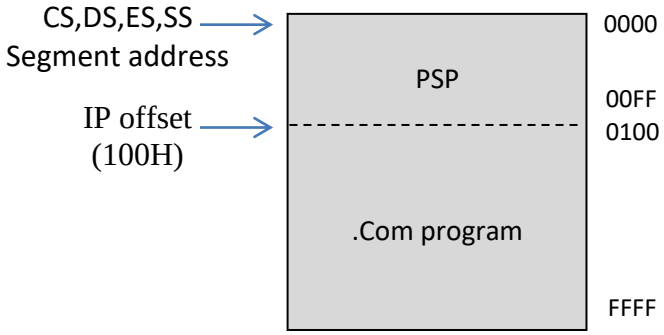
در این مثال همچنان که مشاهده می کنید در FSB#1 چون مسیر انتخابی درایو :A می باشد اولین بایت برابر 01 می باشد سپس هشت بایت نشان دهنده نام فایل با حروف بزرگ و سه بایت بعدی نشان دهنده پسوند فایل با حروف بزرگ می باشد.

در FSB#2 چون مسیر انتخابی درایو :D می باشد اولین بایت برابر 04 می باشد سپس هشت بایت نشان دهنده نام فایل با حروف بزرگ و سه بایت بعدی نشان دهنده پسوند فایل با حروف بزرگ می باشد.

در DTA هم بایت اول نشان دهنده تعداد کاراکترهای وارد شده می باشد که برابر 18h (۲۴ کاراکتر) می باشد، سپس به ترتیب تک تک کاراکتر های وارد شده عیناً آورده شده است.

۱-۱-۱- بخش PSP در برنامه های COM

همچنان که گفته شده در برنامه های COM بخش PSP از آفست 00h تا FFh از سگمنت کد برنامه قرار دارد. می توان ساختار برنامه های COM را بصورت شکل زیر در نظر گرفت.

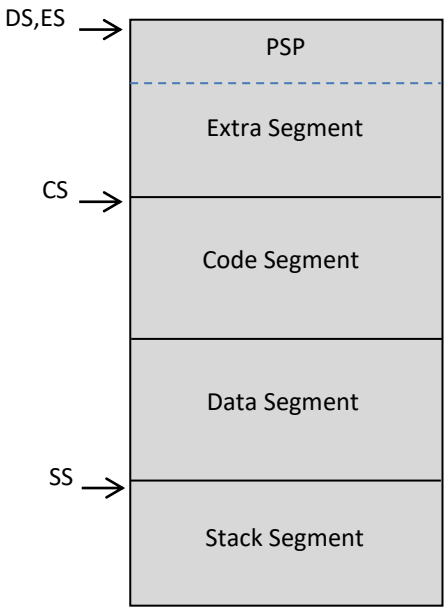


شکل ۱-۲: مقدار دهی یک برنامه COM

۱-۱-۲- بخش PSP در برنامه های EXE

در برنامه‌های EXE پارامترهای خط فمان در سگمنت، ES (اکسترا سگمنت) قرار دارد، همچنان که قبلاً گفته شد بخش PSP در آفست 00h تا FFh قرار دارد همچنان که در شکل زیر نشان داده شده است. پس برای خواندن پارامترهای خط فرمان این بار در برنامه های EXE باید در ES باید به آفست های مورد نظر برویم.

در ES سگمنت PSP از آفست 00h تا ffh قرار می‌گیرد و همانند حالت قبل پارامتر یک در آفست 5Ch ، پارامتر دوم در آفست 6Ch و DTA در آفست 80H قرار دارد.



شکل ۱-۳: مقدار دهی یک برنامه EXE

برنامه ۱-۱ که از نوع برنامه های com برای نحوه چاپ پارامترهای خط فرمان را در خروجی، ایـن برنامـه از نـوع برنامـه های com می باشد.

البته می‌توان با استفاده از تابع 51h از وقفه 21h باز آدرس PSP را بدست آورد این وقفه بعد از اجرا آدرس PSP را روی ثبات BX برمی‌گرداند. قطعه که زیر آدرس PSP را با استفاده از توابع 51h از وقفه 21h بـه دسـت آورده و روی ES قـرار بدهد.

```
MOV   AH,51h
INT   21h
MOV   ES,BX
```

```
          page 60,132
TITLE program comand line
         exit  macro
               mov ax,4c00h
               int 21h
            endm
         print macro arr
               mov ah,09h
               lea dx,arr
               int 21h
            endm
     .model small
     .code
          org 100h
    begin: jmp main
       mis1 db 13, 10, 'you dont input any parametr.','$'
       mis2 db 13, 10, 'your parametr: ','$'
       parameter db 125 dup(20h)
    main proc Near
               mov si,80h
               cmp byte ptr[si],00h
               jz p2
               mov cx,[si]
               and cx,00ffh
               dec cx
               inc si
               inc si
               lea di,parameter
         p1:
               mov al, [si]
               mov [di], al
               inc si
               inc di
               loop p1
               mov byte ptr [di],'$'
               print mis2
               print parameter
               jmp p3
         p2:
               print mis1
         p3:
               exit
    main endp
          end begin
```

برنامه ۱-۱: برنامه com چاپ پارامتر های خط فرمان در خروجی

این برنامه که از نوع برنامه های com می باشد، داده های بخش DTA را خوانده و به ترتیب در خانه های آرایه parameter قرار داده و در انتها کاراکتر $ را به آن اضافه می کند. سپس با استفاده از ماکروی print در خروجی چاپ می کند.

این برنامه همانند برنامه قبلی برای چاپ پارامتر های خط فرمان در خروجی استفاده می شود با این تفاوت که این برنامه از نوع برنامه های exe می باشد.

```
          page 60,132
TITLE program comand line
         exit  macro
               mov ax,4c00h
               int 21h
            endm
         print macro arr
               mov ah,09h
               lea dx,arr
               int 21h
            endm
.model small
     .stack 64
     .data
         mis1 db 13, 10, 'you dont input any parametr .','$'
         mis2 db 13, 10, 'your parametr: ','$'
         parametr db 125 dup(20h)
     .code
     main proc far
          mov ax,@data
          mov DS,ax
          mov Bx,80h
          cmp es:byte ptr[BX],00h
          jz p2
          mov cx,es:[BX]
          and cx,00ffh
          dec cx
          inc bx
          inc bx
          lea di,parametr
p1:
          mov al, es:[BX]
          mov [di], al
          inc bx
          inc di
          loop p1
          mov byte ptr [di],'$'
          print mis2
          print parametr
          jmp p3
p2:
          print mis1
p3:
          exit
     main endp
          end main
```

برنامه ۱-۲: برنامه exe چاپ پارامتر های خط فرمان در خروجی

برنامه ۱-۳ برای تفکیک پارامترهای خط فرمان در برنامه‌های exe استفاده می‌شود. این برنامه پارامتر هایی را از خط فرمان دریافت کرده هر کدام را به تفکیک همراه با پیام Hello در خروجی چاپ می کند.در این برنامه از ثبات BX به عنوان اشاره‌گر در داخل ES استفاده شده است.

```
          page 60,132
TITLE program comand line
         exit  macro
               mov ax,4c00h
               int 21h
             endm
         print macro arr
               mov ah,09h
               lea dx,arr
               int 21h
             endm
          .model small
          .stack 64
          .data
     mis1 db 13,10,'you dont input any parametr .','$'
     mis2 db 13, 10, 'your parametr: ','$'
     mis3 db 13, 10, 'hello: ','$'
     parametr db 30 dup(20h)
          .code
main proc far
     mov ax,@data
     mov DS,ax
     mov Bx,80h
     cmp es:byte ptr[BX],00h
     jz p2
     mov cx,es:[BX]
     and cx,00ffh
     dec cx
     inc bx
     inc bx
     lea di,parametr
p1:
     mov al, es:byte ptr[BX]
     cmp al, ' '
     je p4
     mov [di], al
     inc bx
     inc di
     loop p1
     print mis3
     mov byte ptr [di],'$'
     print parametr
     jmp p3
p4:
     print mis3
     mov byte ptr [di],'$'
     print parametr
     lea di,parametr
     inc bx
     loop p1
     jmp p3
p2:
     print mis1
p3:
     exit
main endp
     end main
```

برنامه ۱-۳: برنامه exe چاپ پارامتر های خط فرمان به تفکیک در خروجی

تمرین: برنامه‌ای بنویسید که پارامترهایی را از خط فرمان پارامتری را دریافت کرده، اگر سوئیچ ?/ وارد شود برنامه باید یک help به کاربر نمایش دهد، اگر A/ وارد شود پارامترهای خط فرمان به حروف بزرگ تبدیل شده و در خروجی چاپ می‌شود و اگر a/ وارد شود پارامترهای وارد شده در خط فرمان کلاً به حروف کوچک تبدیل شده و در خروجی چاپ خواهد شد و اگر هیچ سوئیچی وارد نشود پارامترها عیناً در خروجی چاپ خواهند شد؟

test.exe Test ↵
خروجی: Test
test.exe Test /a
خروجی: test
test.exe Test /A
خروجی: TEST

۲-۱- برنامه های مقیم در حافظه

بعضی از برنامه‌ها طوری طراحی می‌شوند که همیشه روی حافظه مقیم می‌باشند و در مواقعی خاص اجرا می‌شوند. به عنوان مثال با فشار دادن کلید یا در بازه‌های زمانی مشخص براساس تایمر سیستم. اغلب برنامه‌های مقیم در حافظه از نوع برنامه‌های com می باشند اینگونه برنامه‌ها را برنامه‌های خاتمه یافته اما مقیم(TSR) می‌نامند.

برنامه‌های مقیم در حافظه عمدتاً از دو بخش تشکیل می شوند

- بخش مقیم
- بخش مقیم کننده

بخش مقیم کننده بخشی می باشد که وظیفه مقیم کردن برنامه روی حافظه را بر عهده دارد، و بخش مقیم بخشی می باشد که در مواقع مشخص شده فعال شده و کار خود را انجام می دهد. نوشتن بخش مقیم کننده ساده تر از بخش مقیم می باشد. بخش مقیم کننده به جای خاتمه‌ی معمولی برنامه این کار با استفاده از تابع 31H از INT 21h انجام می‌شود. هنگام فراخوانی این وقفه اندازه‌ی برنامه را باید داخل DX قرار دهیم.

```
MOV  AH , 31H
MOV  DX , Prog-size
INT  21H
```

برای مشخص کردن اندازه برنامه می توان از اختلاف آفست استفاده کرد. زمانی که از این وقفه استفاده شود هنگام اجرا جایی که برنامه باید مقیم شود سیستم بلوک حافظه را رزرو می کند و در برنامه های بعدی در موقعیت بالاتری از حافظه قرار می گیرند. با فراخوانی این وقفه، برنامه کلاً از روی حافظه حذف نمی‌شود بلکه همچنان روی حافظه باقیمانده و موقع مورد نیاز اجرا می‌شود.

بخش مهم یک برنامه مقیم در حافظه فعال کردن آن پس از مقیم نمودن می‌باشد. یک روش معمولی برای این کار تغییر جدول بردار وقفه می‌باشد. .به عنوان مثال اگر کلید خاصی یا ترکیبی از کلیدها فشار داده شود برنامه فعال می شود. برنامه‌ی مقیم در کلیدهای فشار داده شده را متوقف می‌کند و با کلیدهای خاص یا ترکیبی از آنها فعال می‌شود و دیگر کلیدهای فشار داده شده را ارسال می‌دارد.

با این وجود یک برنامه مقیم در حافظه معمولاً (نه همیشه) از بخش‌های زیر تشکیل خواهد شد:

۱. یک بخش که موقعیت‌های جدول بردار وقفه را مجدداً تعریف می‌کند.

۲. یک روال مقداردهی که فقط بار اول اجرای برنامه انجام می‌شود و به موارد زیر عمل خواهد کرد.
   a. جایگزینی آدرس در جدول بردار وقفه با آدرس خودش
   b. مشخص کردن اندازه‌ی بخشی از برنامه که مقیم می‌ماند و
   c. استفاده از یک وقفه که به سیستم می‌گوید اجرای برنامه جاری را خاتمه داده و به بخش خاصی از برنامه در حافظه دسترسی پیدا کند.
۳. یک روال که مقیم باقی می‌ماند و فعال است با برخی از فعالیت‌های خاص مانند صفحه کلید یا زمان سنج فعال خواهد شد.

۱-۲-۱- گرفتن بردار وقفه

برای بدست آوردن آدرس وقفه خاصی از بردار وقفه می‌توان از تابع 35H از INT 21H استفاده کرد. برای این کار باید شماره‌ی وقفه‌ی مورد نظر را داخل ثبات AL قرار خواهیم داد.

```
MOV  AH , 35H
MOV  AL , int# → شماره‌ی وقفه
INT  21H
```

این وقفه بعد از فراخوانی آدرس وقفه‌ی مورد نظر را به صورت ES:BX برگشت خواهد داد.

۲-۲-۱- تظیم بردار وقفه

برای تنظیم یک وقفه‌ی خاص در جدول بردار وقفه با آدرس جدید می‌توان از تابع 25H از INT 21H استفاده کرد. برای این کار شماره‌ی وقفه‌ی مورد نظر را روی ثبات AL قرار داده و آدرس جدید آن را روی ثبات DX قرار می‌دهیم.

```
MOV  AH , 25H
MOV  AL , int#
MOV  DX , newaddr
INT  21H
```

این عملیات آدرس وقفه را با آدرس جدید جایگزین می‌کند.

۳-۲-۱- یک برنامه مقیم در حافظه

برنامه ۱-۴ یک برنامه مقیم در حافظه می باشد. این برنامه کنترل می کند که آیا Num Lock روشن است یا نه، اگر روشن باشد و یکی از کلیدهای ماشین حسابی صفحه کلید فشار داده شده باشد این برنامه صدای بیپ را از بلند گوی کیس کامپیوتر صادر خواهد کرد در غیر این صورت هیچ واکنشی را نشان نخواهد داد.

```
 1  TITLE A24TSTNM (COM) Resident program
 2  BIODATA SEGMENT AT 40H
 3        ORG 17H
 4     KBSTATE DB ?
 5  BIODATA ENDS
 6  ;-------------------------------------
 7  CODESG SEGMENT PARA
 8         ASSUME  CS:CODESG, DS:BIODATA
 9         ORG 100H
10   BEGIN:
11     JMP B10INIT
```

```
12    SAVINT9 DD ?
13    A10TEST:
14        PUSH AX
15        PUSH CX
16        PUSH DS
17
18        MOV AX , BIODATA
19        MOV DS , AX
20        MOV AL , KBSTATE
21        TEST AL , 00100000B
22        JZ  A30EXIT
23
24        IN  AL , 60H
25        CMP AL , 71
26        JL  A30EXIT
27        CMP AL , 83
28        JG  A30EXIT
29
30        MOV AL , 10110110B
31        OUT 43H , AL
32        MOV AX , 1000
33        OUT 42H , AL
34        MOV AL , AH
35        OUT 42H , AL
36        IN  AL , 61H
37        MOV AH , AL
38        OR  AL , 03
39        OUT 61H , AL
40        MOV CX , 9000
41   A20PAUSE:
42        LOOP A20PAUSE
43        MOV AL , AH
44        OUT 61H , AL
45   A30EXIT:
46        POP DS
47        POP CX
48        POP AX
49        JMP CS:SAVINT9
50
51 ;              initilalization routine:
52 ;-----------------------------------------
53
54   B10INIT:
55        CLI
56        MOV AH , 35H
57        MOV AL , 09H
58        INT 21H
59        MOV WORD PTR SAVINT9 , BX
60        MOV WORD PTR SAVINT9+2 , ES
61        MOV AH , 25H
62        MOV AL , 09H
63        MOV DX , OFFSET A10TEST
64        INT 21H
65        MOV AH , 31H
66        MOV DX , OFFSET B10INIT
67        STI
68        INT 21H
69 CODESG ENDS
70        END BEGIN
```

برنامه ۱-۴: یک برنامه مقیم در حافظه

این برنامه برای تشخیص خاموش یا روشن بودن Num Lock از بایت وضعیت صفحه کلید استفاده می کند. این بایت در آدرس 40h:17h قرار دارد. برای دسترسی به این بخش، سگمنت BIODATA تعریف شده است. در این سگمنت KBSTATE مشخص کننده بایت وضعیت صفحه کلید خواهد بود (سطر ۲ تا ۵).

در این برنامه سطر ۵۴ تا ۶۸ بخش مقیم کننده بوده و سطر ۱۴ تا ۴۹ نیز بخش مقیم برنامه می باشد. در بخش مقیم کننده ابتدا با استفاده از دستور CLI وقوع وقفه غیر فعال شده است تا اتمام کار که در انتها دستور STI وقوع وقفه را فعال می کند. این دو دستور با استفاده از ثبات IF این کار را انجام می دهند. در این بخش ابتدا با استفاده از تابع 35h از وقفه شماره 21h آدرس وقفه 09h را بدست آورده (سطرهای ۵۶، ۵۷ و ۵۸) سپس روی متغیر SAVINT9 ذخیره می کند (سطرهای ۵۹ و ۶۰) . سپس با استفاده از تابع 25h وقفه 21h آدرس برنامه را به جای آدرس وقفه 09h جایگذاری می کند (سطرهای ۶۱، ۶۲، ۶۳ و ۶۴). در نهایت برای خروج از برنامه و مقیم کردن آن از تابع 31h دقفه 21h استفاده شده است.

هر بار که کلیدی از صفحه کلید فشار داده می شود، برای پردازش کلید فشار داده شده تابع 09h وقفه 16h اجرا می شود. بعد از مقیم کردن برنامه هر بار که کلیدی از صفحه کلید فشار داده شود به جای این وقفه برنامه ما اجرا خواهد شد.

بخش مقیم نیز در ابتدا اقدام به ذخیره محتویات ثیاتهایی می کند که در طول اجرای برنامه تغییر می کنند. این ثباتها عبارتند از AX، CX و DX. برای ذخیره موقت محتویات این ثباتها از پشته استفاده شده است تا در انتهای اجرای برنامه بتوان محتویات آنها را دوباره بازیابی نمود (سطرهای ۱۴، ۱۵ و ۱۶ برای ذخیره روی پشته و سطرهای ۴۶، ۴۷ و ۴۸ برای بازیابی از پشته). Push کردن ثباتها روی پشته در ابتدای برنامه و pop کردن آنها در انتهای برنامه باعث خواهد شد که مقدار قبلی آنها حفظ شود.

هر بار که برنامه با فشار دادن کلید فعال می شود بایت وضعیت صفحه کلید را برای روشن بودن Num Lock (بیت ششم بایت وضعیت) چک می کند. بایت وضعیت صفحه کلید در سگمنت 40h آفست 17h قرار دارد. در این برنامه BIODATA ، سگمنتی می‌باشد که در 40h قرار دارد و KBSTATE متغیر می‌باشد که در آفست 17h از این سگمنت قرار دارد. برای دسترسی به این متغیر باید آدرس BIODATA را روی DS قرار دهیم. سطر های ۱۸، ۱۹ و ۲۰ باعث می-شوند مقدار KBstate خوانده شده و داخل AL قرار گیرد. در صورتی که بیت ششم بایت وضعیت برابر یک باشد (NumLock روشن باشد) بقیه برنامه برای ایجاد صدا از طریق بلندگوی کیس اجرا می شود.

IN AL,60H بایتی را از Port 60 که حاوی کد کلید فشار داده شده است خوانده و روی ثبات AL قرار می‌دهد. دستورات بعدی این دستور برای تشخیص کلیدهای ماشین حسابی صفحه کلید که کد آن‌ها از 71h تا 83h است استفاده می‌شود. اگر در این بازه قرار نداشته باشد از برنامه خارج خواهیم شد و اگر در این بازه قرار داشته باشد کدهای بعدی باعث ایجاد صدای بیپ از بلندگوی کیس کامپیوتر می‌شود. سطر ۴۹ نیز برای انتقال اجرای برنامه به تابع 09h از وقفه 21h استفاده می شود.

۳-۱- مجموعه دستورات کامپیوتر های شخصی

هر دستور اسمبلی بعد از اسمبل شدن تبدیل به زبان ماشین خواهد شد. در این بخش روش تبدیل کد های زبان اسمبلی به دستورات زبان ماشین برای پردازنده های ۸۰۸۶ اینتل و بالعکس تبدیل کد زبان ماشین به دستور اسمبلی توضیح داده خواهد شد.

سه نوع دستور اسمبلی وجود دارد:

۱. دستوراتی که هیچ عملوندی ندارند. مانند : RET , CLI , STI
۲. دستوراتی که فقط شامل یک عملوند هستند. مانند INC AX

۳. دستوراتی که شامل ۲ عملوند می باشند. مانند MOV AX,BX

ساده ترین روش تبدیل مربوط به دستوراتی می باشد که هیچ عملوندی ندارد. این دستورات همیشه کد زبان ماشین ماشین ثابتی دارند که میتوان از طریق زمیمه این جزوه که شامل کلیه دستورات پردازنده های رده x86 شرکت اینتل می باشد به دست آورد. مانند:

CLC	1111 1000
کد زبان ماشین: F8h	

STI	1111 1011
کد زبان ماشین: FBh	

RET	1100 1011
کد زبان ماشین: CBh	

اما دستوراتی که شامل علوند باشد چه ۱ عملوند یا ۲ عملوند ، این دستورات بسته به عملوند انتخاب شده کدهای زبان ماشین متفاوتی خواهند داشت .

۱-۳-۱- نشانه گذاری ثبات

دستوراتی که دارای عملوند می باشند بسته به نوع عملوند کدهای زبان ماشین مختلفی را خواهند داشت. در برخی از این دستورات ممکن است مراجعه به ثبات را داشته باشیم اینگونه دستورات شامل ۳ بیت reg خواهند بود و همچنین یک بیت w را خواهند داشت w=0 مشخص کننده یک بایت و w=1 مشخص کننده ی یک کلمه (۲ بایت) می باشند.

جدول زیر شماره ی ثبات های مختلف را مشخص میکند.

General, base, and Index Registers			Bits for Segment Registers	
bits	W=0	W=1	000	ES
000	AL	AX/EAX	001	CS
001	CL	CX/ECX	010	SS
010	DL	DX/EDX	011	DS
011	BL	BX/EBX	100	FS
100	AH	SP	101	GS
101	CH	BP		
110	DH	SI		
111	BH	DI		

شکل ۱-۴

برای بدست آوردن کد زبان ماشین هر یک از دستورات می توانید از ضمیمه این جزوه استفاده کنید. در هنگام جستجوی دستور باید دقیقاً نوع عملوند را مشخص نمایید و جستجو را بر اساس آن انجام دهید. دستورات اسمبلی ۳ نوع عملوند خواهند داشت :

۱. عملوند ثباتی یا reg
۲. عملوند بلافصل یا imm (تمام مقادیر ثابت اعم از اعداد یا کاراکترها)
۳. عملوند حافظه ای یا mem (هر نوع ارجاع حافظه اعم از ارجاع مستقیم و چه ارجاع به صورت غیر مستقیم (مانند حالت اشاره گری [SI]) از نوع عملوند حافظه ای خواهند بود).

برای به دست آوردن کد زبان ماشین باید دنبال حالت مورد نظر خود در جدول مجموعه دستورات پردازنده های ۸۰۸۶ بگردیم. سپس بر اساس روالی که توضیح داده خواهد شد می توانیم دستور اسمبلی را به کد زیبان ماشین تبدیل کنیم. به عنوان مثال کد زبان ماشین دستور MOV AH,00:

برای پیدا کردن کد زبان ماشین این دستور در جدول مجموعه دستورات باید دنبال دستور MOV ی می گردیم که عملوند اول آن ثبات ۸ بیت (reg 8) و عملوند دوم آن حافظه ۸ بیت (imm 8) باشد (یعنی حالت MOV reg8 , imm8). بر اساس جدول کد زبان ماشین این دستور به صورت زیر مشخص می گردد

```
MOV   AH,00
MOV   reg8,imm      1011  0     rrr     [imm8]

                    1011  0     100     00000000
                          ↥     ↥↥↥
                          w     reg=AH
```

کد زبان ماشین: B400h

نکته : در این جزوه به جای عبارت rrr که ۳ بیت نشان دهنده ثبات انتخابی می باشد ، از عبارت reg استفاده کردیم . در این مثال چون ثبات انتخابی AH می باشد و ثبات AH ثبات ۱ بایتی می باشد . مقدار w برابر 0 خواهد بود . ۳ بیت reg مشخص کننده ثبات انتخابی خواهد بود. بر اساس جدول قبلی کد ثبات AH=100 خواهد بود . همچنین به جای [imm8] ۸ بیت مشخص کننده مقدار عملوند دوم خواهد بود .

مثال: دستور MOV BX,00 این مثال همانند مثال قبل می باشد با این تفاوت که به جای عملوند های ۸ بیت عملوندهای ۱۶ بیت را داریم.

```
MOV   BX,00
MOV   reg16,imm     1011   1    rrr     [imm16]

                    1011   1    011     00000000 00000000
                           ↥    ↥↥↥
                           W    reg=BX
```

کد زبان ماشین: BB0000h

۲-۳-۱- آدرسدهی بایت وضعیت

ساختار برخی از دستورات شامل بایت وضعیت می باشد در اینگونه دستورات بایت دوم شامل بایت وضعیت خواهد بود این بایت شامل ۳ عنصر زیر خواهد بود :

۱. دوبیت mod که مقدیر 00,01,10 به موقعیت حافظه اشاره می کند و 11 به یک ثبات اشاره می کند.
۲. سه بیت reg که مشخص کننده ثبات می باشد.

۳. سه بیت r/m که به ثبات یا حافظه اشاره می کند. r مشخص کننده ی ثبات و m مشخص کننده ی حافظه می باشد.

با استفاده از بیت‌های Mod می‌توانیم مابین آدرس‌دهی ثباتها و حافظه تفاوت قائل شویم. حالتهای مختلف Mod عبارت است از:

- 00 : در این حالت بیت‌های r/m انتخاب آدرس‌دهی دقیق را بدون بایت آفست ارائه می‌دهد.
- 01 : بیتهای r/m انتخاب آدرس‌دهی دقیق را با یک بایت آفست ارائه می‌دهد.
- 10 : بیتهای r/m انتخاب آدرس‌دهی دقیق را با دو بایت آفست ارائه می‌دهد.
- 11 : r/m مشخص کننده یک ثبات می باشد.

اولین بایت این دستورات شامل بیت d خواهد بود که مشخص کننده ی جهت می باشد (راست به چپ یا چپ به راست) اگر d برابر یک باشد (چپ به راست)، reg مشخص کننده ی عملوند اول و r/m مشخص کننده ی عملوند دوم خواهد بود و اگر d برابر صفر باشد (راست به چپ)، reg مشخص کننده ی عملوند دوم و r/m مشخص کننده ی عملوند اول خواهد بود.

r/m	Mod = 00	Mod=01 or 10	Mod = 11 W =0	Mod = 11 W =1
000	[BX+SI]	DS:[BX+SI+disp]	AL	AX
001	[BX+DI]	DS:[BX+DI+disp]	CL	CX
010	[BP+SI]	SS:[BP+SI+disp]	DL	DX
011	[BP+DI]	SS:[BP+DI+disp]	BL	BX
100	[SI]	DS:[SI+disp]	AH	SP
101	[DI]	DS:[DI+disp]	CH	BP
110	Direct	SS:[BP+disp]	DH	SI
111	[BX]	DS:[BX+disp]	BH	DI

شکل ۱-۵

به عنوان مثال ADD AX,BX و ADD BX,AX

```
ADD   AX,BX
ADD   reg16,reg16   0000   00   x 1    11     reg    r/m

                    0000   00   1 1    11     000    011
                                 ↑ ↑    ↑↑     ↑↑↑    ↑↑↑
                                 d w    mod    reg    r/m
                                               =AX    =BX
```

کد زبان ماشین : 03C3h

ADD	BX,AX						
ADD	reg16,reg16	0000	00	x 1	11	reg	r/m
		0000	00	1 1	11	011	000
				↥ ↥	↥↥	↥↥↥	↥↥↥
				d w	mod	reg	r/m
						=BX	=AX

کد زبان ماشین : 03D8h

در مثالهای بالا بیت Mod برابر با 11 نشان می دهد که r/m مشخص کننده ثبات میباشد و r/m برابـر بـا 011 بـا w=1 نشان می دهد که آن ثبات، ثبات BX می باشد.

MOV	ES,AX						
MOV	segreg,reg16	1000	11	1 0	11	reg	r/m
		1000	11	1 0	11	000	000
				↥	↥↥	↥↥↥	↥↥↥
				d	mod	sreg	r/m
						=ES	=AX

کد زبان ماشین : 8EC0h

MOV	AX,ES						
MOV	reg16,segreg	1000	11	0 0	11	reg	r/m
		1000	11	0 0	11	000	000
				↥	↥↥	↥↥↥	↥↥↥
				d	mod	sreg	r/m
						=ES	=AX

کد زبان ماشین : 8CC0h

همچنان که در این دو مثال مشاهده می کنید تنها تفاوت ایـن دو مثـال در بیـت d مـی باشـد. چـون sreg بایـد ثبـات سگمنت را مشخص کند (ES) و r/m باید ثبات AX را مشخص کند. در مثال اول d=1 مـی باشـد چـون sreg عملونـد اول ES و r/m عملوند دوم AX را مشخص می کنند یعنی جهت چپ به راست ولی در مثـال دوم d=0 مـی باشـد چـون sreg عملوند دوم ES و r/m عملوند اول AX را مشخص می کنند یعنی جهت راست به چپ.

نکته: دقت داشته باشید که در کدها ثباتهای سگمنت با sreg نشان داده شده اند که متفـاوت بـا ثباتهـای معمـولی reg می باشند. برای بدست آوردن کدهای ثباتهای سگمنت به شکل ۱-۴ مراجعه کنید.

۳-۳-۱- دستورات دو بایتی

مثالی از دستورات دو بایتی در مثالهای قبلی آمده است نمونه ای از آنها دستور زیر می باشد.

```
OR    CL,DH
OR    reg8,reg8     0000  10  x 0   11    reg   r/m

                    0000  10  1 0   11    001   110
                              ↑ ↑   ↑↑    ↑↑↑   ↑↑↑
                              d w   mod   reg   r/m
                                          =CL   =DH
```

کد زبان ماشین : 0ACEh

همچنان که قبلاً گفته شده d=1 یعنی جهت چپ به راست، w=0 یعنی اندازه یک بایت، mod=11 یعنی r/m مشخص کننده ثبات می باشد و reg=011 با w=0 یعنی ثبات CL و r/m=110 با mod=11 و w=0 یعنی ثبات DH همچنان که در شکل ۱-۵ آمده است.

```
MUL   BL
MUL   reg8     1111  011  0    11    100   r/m

               1111  011  0    11    100   011
                          ↑    ↑↑    ↑↑↑   ↑↑↑
                          w    mod   reg   r/m
                                           =BL
```

کد زبان ماشین : F6 E3h

دستور MUL فقط شامل یک عملوند می باشد. در این مثال BL به عنوان عملوند انتخاب شده است. r/m=011 با mod=11 و w=0 یعنی ثبات BL. در اینجا reg=100 هیچ مفهومی ندارد و همیشه در دستور MUL این مقدار را دارد.

نکته: اگر دستوری داشته باشیم که در آن دستور مقدار ثابتی استفاده شده باشد (imm) یا آفستی وجود داشته باشد (Disp) و یا متغیر استفاده شده باشد، باید به انتهای آن کد زبان ماشین آن دستور مقدار ثابت، آفست و یا آدرس متغیر اضافه گردد در مثالهای بعدی چنین حالتهایی را خواهیم دید.

۴-۳-۱- دستورات سه بایتی

دستور MOV زیر سه بایت کد زبان ماشین تولید می کند.

توجه: در مثالهای زیر فرض شده است آدرس متغیر Temp برابر 000Ah می باشد.

```
MOV   Temp,AX
MOV   Disp,AX     1010  00  1  1   [Disp]

                  1010  00  1  1   0000 0000 0000 1010
                                ↑   ↑        آدرس متغیر Temp
                                d   w
```

کد زبان ماشین: A3 000Ah

d=1 یعنی جهت چپ به راست و w=1 یعنی یک کلمه (دو بایت).

اگر به این مثال و بخش ضمیمه دقت کنید، در برخی از دستورات مانند دستور MOV، SUB، ADD و ... که یکی از عملوندها ثباتهای AL ، AX یا EAX و عملوند دیگر حافظه مستقیم می باشد به عنوان یک استثناء محسوب می شود. و در بخش ضمیمه نباید برای این ثباتها دنبال حالت reg بگردیم. به عنوان مثال روش بدست آوردن کد زبان ماشین MOV AL,Temp با MOV BL,Temp مثل هم نیست.

```
MOV   AL,Temp
MOV   AL,Disp     1010    00    0 0  [Disp]

                  1010    00    0 0  0000 0000 0000 1010
                                 ↥ ↥      آدرس متغیر Temp
                                 d w
```

کد زبان ماشین: A0 000Ah

```
MOV   BL,Temp
MOV   Reg8,mem    1000    10    1 0  mod    reg      r/m

                  1000    10    1 0  00     011      110      0000 0000
                                                              0000 1010
                                 ↥ ↥  ↥↥    ↥↥↥      ↥↥↥      آدرس متغیر Temp
                                 d w  mod   reg=      r/m=Temp
                                           BL
```

کد زبان ماشین : 8A 1E 000Ah

۵-۳-۱- دستورات چهار بایتی

دستور ۴ بایتی زیر AL را در یک موقعیت حافظه ضرب می کند.

```
MUL   Temp
MUL   mem8     1111    011   0    mod    100    r/m

               1111    011   0    00     100    110     0000 0000 0000 1010
                              ↥    ↥↥     ↥↥↥    ↥↥↥     آدرس متغیر Temp
                              w    mod    reg    r/m
```

کد زبان ماشین : F6 26 000Ah

در این دستور عملوند دستور MUL از نوع حافظه ای مستقیم می باشد، در چنین حالتی بر اساس شکل ۱-۵ mod=00 r/m مقدار 110 را خواهد داشت و آدرس متغیر در انتهای کد دستور آورده می شود.

```
LEA   DX , Temp
LEA   Reg , mem   1000    1101    mod    reg    r/m

                  1000    1101    00     010    110     0000 0000 0000 1010
                                  ↥↥     ↥↥↥    ↥↥↥     آدرس متغیر Temp
                                  mod    reg=   r/m
                                         DX
```

کد زبان ماشین : 8D 16 000Ah

دراین مثال نیز همانند مثال قبل بر اساس شکل ۱-۵ mod=00 و r/m=110 نشان دهنده عملوند مستقیم حافظه می باشد و آدرس عملوند حافظه در انتهای کد دستور آورده می شود.

۶-۳-۱- چند مثال مختلف

در مثالهایی که گفته شد یکی از موارد ذکر شده mod می باشد که دو بیته می باشد و می تواند مقادیر 00، 01، 10 و 11 را داشته باشد. همچنان که در بخش ۰ گفته شده حالتهای مختلف mod عبارت است از:

- 00 : بیت‌های r/m انتخاب آدرس‌دهی دقیق را ارائه می‌دهد بدون بایت آفست.
- 01 : بیت‌های r/m انتخاب آدرس‌دهی دقیق را ارائه می‌دهد با یک بایت آفست.
- 10 : بیت‌های r/m انتخاب آدرس‌دهی دقیق را ارائه می‌دهد با دو بایت آفست.
- 11 : r/m یک بیت ثبات را مشخص می‌کند.

مقدار mod=00 یعنی r/m مشخص کننده آدرس دقیق بدون فیلد آفست می باشد این مقدار زمانی استفاده می شود که هنگام ارجاع به حافظه هیچ آفستی نداشته باشیم مانند متغیر (Direct) یا اشاره به حافظه (مثل [BX]). Mod=01 مشخص کننده آدرس دقیق با یک بایت آفست می باشد در این حالت ارجاع به حافظه دارای آفستی می باشد که این آفست عددی مابین 128- تا 127+ می باشد (مانند [BX+10] یا [BX-10]) و mod=10 آدرس دقیق با دو بایت آفست می باشد در این حالت ارجاع به حافظه دارای آفستی می باشد که این آفست عددی مابین 128- تا 127+ نمی باشد (مانند [BX+200] یا [BX-200])

برای اینکه درک بهتری از حالتهای مختلف mod داشته باشید مثالهای زیر آورده شده است.

SUB	BX,[BX+SI]						
SUB	reg16,mem16	0010	10	1 1	mod	reg	r/m
		0010	10	1 1	00	011	000
				↥ ↥	↥↥	↥↥↥	↥↥↥
				d w	mod	reg= BX	r/m

کد زبان ماشین : 2B 18h

در این مثال عملوند دوم از نوع عملوند حافظه ای می باشد، در این حالت بر اساس شکل ۱-۵ mod=00 و r/m=000 خواهد بود.

SUB	[SI+100],BX							
SUB	mem16,reg16	0010	10	0 1	mod	reg	r/m	
		0010	10	0 1	01	011	100	0110 0100
				↥ ↥	↥↥	↥↥↥	↥↥↥	مقدارد ۱۰۰ در
				d w	mod	reg =BX	r/m	مبنای ۲

کد زبان ماشین : 29 5C 64h

در این مثال عملوند اول از نوع عملوند حافظه ای می باشد که شامل آفست ۱۰۰ می باشد. آفست یک عدد علامت دار می باشد در این مثال چون عدد ۱۰۰ (برابر با 64h) روی یک بایت قابل ذخیره سازی می باشد به همین دلیل mod=01 قرار داده شده است. و برای حالت [SI+disp] ، r/m=100 خواهد بود.

توجه داشته باشید که چون بصورت پیشفرض SI و DI با DS کار می کنند بر این اساس [DI] معادل DS:[DI] می باشد و همچنان که در مثال قبل قابل مشاهده است [SI+disp] معادل DS:[SI+disp] می باشد. بقیه حالتها هم بر این اساس خواهد بود، در چنین مواردی مشخص کردن یا نکردن DS اهمیت ندارد.

```
SUB    [SI-200],BX
SUB    mem16,reg16    0010   10   0 1    mod    reg    r/m

                      0010   10   0 1    10     011    100    1111 1111 0011 1000
                                  ↑ ↑    ↑↑     ↑↑↑    ↑↑↑    مقدارد 200- در مبنای ۲
                                  d w    mod    reg    r/m
                                                =BX
```

کد زبان ماشین : 29 9C FF38h

در این مثال عملوند اول از نوع عملوند حافظه ای می باشد که شامل آفست 200- می باشد. آفست یک عدد علامت دار می باشد در این مثال چون عدد 200- (برابر با FF38h) روی دو بایت قابل ذخیره سازی می باشد به همین دلیل mod=10 قرار داده شده است. و برای حالت [SI+disp] ، r/m=100 خواهد بود.

```
SUB    BX,CX
SUB    reg16,reg16    0010   10   x 1    11     reg    r/m

                      0010   10   1 1    11     011    001
                                  ↑ ↑    ↑↑     ↑↑↑    ↑↑↑
                                  d w    mod    reg    r/m
                                                =BX    =CX
```

کد زبان ماشین : 2B D9h

در این مثال چون r/m مشخص کننده تبات می باشد به این دلیل mod=11 قرار داده شده است.

۴-۱- تبدیل کد های زبان ماشین به اسمبلی

برای تبدیل ابتدا کد زبان ماشین را تبدیل به مبنای ۲ خواهیم کرد. سپس در جدول مجموعه دستورات دنبال دستوری می گردیم که ۶ بیت اول آن با دستور مورد نظر ما یکسان باشد. سپس می توانیم بر اساس دستور مشخص شده و جداولی که قبلا توضیح داده شده اقدام به تبدیل کد زبان ماشین به دستور اسمبلی معادل خواهیم کرد .

فصل دوم

۲- برنامه نویسی سیستمی تحت ویندوز

۲-۱- برنامه نویسی نخ[1]

در روی سیستم عامل تعاریف زیادی وجود دارد که از جمله آنها می توانیم به مفهوم Thread و یا Process اشاره کرد. Process یا فرآیند برنامه در حال اجرا می باشد. هر برنامه بعد از اجرا تبدیل به Process خواهد شد. واحد تخصیص منابع در سیستم عامل فرایند می باشد. یعنی زمانیکه قرار باشد حافظه یا منابع دیگری از سیستم تخصیص یابد این تخصیص برای فرآیند انجام خواهد شد نه نخ. بر اساس طراحی سیستم عامل ممکن است سیستم عامل های تک فرآینده مثل DOS ، و یا چند فرآینده مثل Windows داشته باشیم.

نکته: در اکثر زبان برنامه نویسی تحت ویندوز می توان برنامه نویسی نخ را انجام داد اما ما برای مثالهای این بخش از زبان C# استفاده کرده ایم.

در صورتی که در یک برنامه از Thread ها استفاده نشود و بخواهیم توابعی را فراخوانی کنیم، اجرای این توابع به صورت پشت سر هم خواهد بود. یعنی تا زمانیکه اجرای تابع اول تمام نشده به هیچ وجه تابع دوم فراخوانی نمی شود. به عنوان مثال می توان برنامه ۲-۱ را در نظر گرفت. در این برنامه دو تابع PrintX و PrintY تعریف شده است . این دو تابع هنگام فراخوانی در خروجی ۱۰۰ کاراکتر X یا ۱۰۰ کاراکتر Y چاپ می کند. بدون استفاده از نخ ترتیب فراخوانی مشخص کننده خروجی خواهد بود. یعنی در صورتی که اول تابع PrintX و سپس تابع PrintY فراخوانی می شود و باعث می شود ابتدا ۱۰۰ کاراکتر X و سپس ۱۰۰ کاراکتر Y چاپ شود.

برای برنامه نویسی نخ در C# ابتدا باید عبارت Using System.Threading را به ابتدای برنامه اضافه می کنیم. سپس می توان از امکانات برنامه نویسی نخ داخل برنامه استفاده کرد. C# برای هر برنامه در ابتدا به صورت خودکار یک نخ را به عنوان نخ اصلی ایجاد می کند. این نخ اصلی ایجاد کننده بقیه نخ ها خواهد بود. برای مثال قبل، می توانیم نخی را تعریف کنیم که این نخ فراخوانی کننده تابع PrintX باشد . برای این منظور Thread را به شکل زیر تعریف خواهیم کرد :

```
Thread FirestThread = new Thread(PrintX);
```

زمانی که Thread جدیدی ایجاد می شود سیستم عامل باری هر Thread به صورت مجزا بلوک کنترل وضعیت Thread ، پشته Thread و متغیر های محلی Thread را ایجاد می کند که هر یک از این موارد مجزا از بقیه Thread ها خواهد بود. برای شروع اجرای هر Thread باید تابع Start آن Thread را فراخوانی کرد :

```
FirestThread.Start();
```

[1] Thread Programming

برای مثال قبل مـی تـوانیم دو Thread ایجـاد کنـیم کـه FirestThread فراخـوانی تـابع PrintX و SecondThread فراخوانی تابع PrintY باشد. بعد از اجرای این برنامه خواهید دید که تعداد چاپ های X و Y ثابت می باشد اما ترتیـب آنهـا در هر اجرا متفاوت است. برنامه ۲-۲ چنین برنامه ای را نشان می دهد.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ThreadSample1
{
    class Program
    {
        static void Main(string[] args)
        {
            PrintY();
            PrintX();

            Console.ReadKey();
        }
        static void PrintX()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('X');
            }
        }
        static void PrintY()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('Y');
            }
        }
    }
}
```

برنامه ۲-۱: چاپ X و Y بدون استفاده از نخ

خروجی

```
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample2
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread FirestThread = new Thread(PrintX);
            Thread SecondThread = new Thread(PrintY);
            FirestThread.Start();
            SecondThread.Start();
            Console.ReadKey();
        }
        static void PrintX()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('X');
                //Thread.Sleep(1);
            }
        }
        static void PrintY()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('Y');
                //Thread.Sleep(1);
            }
        }
    }
}
```

برنامه ۲-۲: چاپ X و Y با استفاده از نخ

خروجی

```
XXXXXXXYYXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

زمانیکه نخ ایجاد شد، سیستم عامل اقدام به زمانبندی نخ خواهد کرد و برای اجرای هر نخ CPU تخصیص داده خواهد شد و نهایتا باعث خواهد شد در زمانیکه نخ اول اجرا می شود نخ دوم اجرا نشود و برعکس زمانی که Thread های دیگر اجرا می شوند FirestThread اجرا نخواهد شد. این زمانبندی کاملا به عهده سیستم عامل خواهد بود. به عنوان مثال ممکن است در زمان اجرای FirestThread ده عدد X و در زمان اجرای SecondThread دوازده عدد Y چاپ شود. تعداد این چاپ ها در هر بار ممکن است متفاوت باشد.

با استفاده از Thread ها می توان هر تابعی را به صورت Thread فراخوانی نمود. هنگام ایجاد یک Thread جدید نام تابع را برای فراخوانی به Thread اعلام می کنیم. به عنوان مثال:

```
Thread SampleThread = new Thread(GO);
```

در این مثال SampleThread فراخوانی کننده تابع Go می باشد. تابعی که در این حالت فراخوانی می شود نباید پارامتر ورودی داشته باشد و همچنین هیچ مقداری را نمی تواند برگشت دهد. بهترین روش برای ایجاد Thread استفاده از ThreadStart می باشد. ThreadStart ابتدا یک نماینده[1] به تابع مورد نظر ایجاد کرده و سپس از طریق آن می توانیم Thread را ایجاد کنیم. مثال زیر نمونه ای از ایجاد Thread با استفاده از ThreadStart را نشان می دهد:

```
ThreadStart TS = new ThreadStart(GO);
Thread SampleThread = new Thread(TS);
```

۱-۱-۲- متد Sleep

فراخوانی متد Thread.Sleep نخ را برای مدت زمان معینی متوقف می کند. با فراخوانی این متد سیستم عامل نخ مربوطه را متوقف می کند و CPU از آن نخ گرفته می شود و به نخ دیگری داده خواهد شد و تا زمانی که این این مدت تعیین تمام نشود دوباره زمانبندی نمی شود. پارامتر ورودی این متد عدد صحیحی می باشد که مشخص کننده مدت زمان توقف بر اساس میلی ثانیه می باشد. این عدد می تواند برابر صفر باشد یعنی Thread.Sleep(0) تنها تاثیر این فراخوانی این است که CPU از نخ مربوطه گرفته شده تا به نخهای دیگر شانس اجرا داده می شود.

در برنامه ۲-۲ برای اینکه بخواهیم تعداد چاپ های پشت سر هم X یا Y را کاهش دهیم، باید در نخهای مربوطه بعد از هر چاپ X یا Y متد Sleep را فراخوانی کنیم. این کار در برنامه ۳-۲ انجام شده است.

در زمان sleep نخ FirestThread ممکن است زمان به نخ SecondThread داده شود و این کار باعث شود که Y چاپ شود و بر عکس در زمان sleep نخ SecondThread ممکن است زمان به نخ FirestThread داده شود و این کار باعث شود که X چاپ شود. شاید تصور کنید که X و Y بصورت یک در میان چاپ خواهند شد اما اگر به خروجی برنامه دقت کنید خواهید دید که یک در میان چاپ شدند اما کاملاً هم یک در میان نیست بعضی مواقع ما دو X یا دو Y پشت سر هم را نیز داریم.

خروجی اجرای برنامه ۲-۳

```
XYXYYXXYYXXYXYXYXYXYXYXYXYXYXYYXXYYXXYXYXXYYXXYXYXYXYYXYXYXXYYXYXXYXYXYXYXYXYXY
XYXYXYXYYXXYXYXYXYYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYX
YXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYX
```

[1] Delegate

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample3
{
    class Program
    {
        static void Main(string[] args)
        {
            ThreadStart PrintXref = new ThreadStart(PrintX);
            ThreadStart PrintYref = new ThreadStart(PrintY);
            Thread FirestThread = new Thread(PrintXref);
            Thread SecondThread = new Thread(PrintYref);
            FirestThread.Start();
            SecondThread.Start();

            Console.ReadKey();
        }
        static void PrintX()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('X');
                Thread.Sleep(10);
            }
        }
        static void PrintY()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.Write('Y');
                Thread.Sleep(10);
            }
        }
    }
}
```

برنامه ۲-۳: استفاده از متد Sleep

۲-۱-۲- ارتباط Thread ها

برنامه های قبلی که برای چاپ X و Y در خروجی استفاده استفاده می شد، Thread های برنامه هیچ ارتباطی با یکدیگر ندارند. برای ایجاد ارتباط بین Thread ها یک روش ارتباط غیر مستقیم می باشد. ارتباط غیر مستقیم می تواند از طریق متغیر های استاتیک سراسری ایجاد شود. در این روش هیچ ارتباط مستقیمی ما بین Thread ها وجود ندارد و هر Thread از وجود Thread دیگر بی خبر است.

به عنوان مثال اگر بخواهیم Thread ی داشته باشیم که چاپ X و Y را کنترل کرده و مشخص کند که FirestThread و SecondThread چه مقدار X و Y را چاپ کنند، می توانیم از متغیر سراسری N و Done برای ایجاد ارتباط مابین نخها استفاده کنیم. متغیر N مشخص کننده تعداد چاپهای X و Y می باشد و متغیر Done که از نوع bool می باشد مشخص می کند که آیا تعداد چاپ از کاربر گرفته شده یا نه. پیاده سازی این حالت در برنامه ۲-۴ نشان داده شده است.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample4
{
    class Program
    {
        private static bool done = false;
        private static int n;
        static void Main(string[] args)
        {
            ThreadStart GetNref = new ThreadStart(GetN);
            ThreadStart PrintXref = new ThreadStart(PrintX);
            ThreadStart PrintYref = new ThreadStart(PrintY);
            Thread CoorThread = new Thread(GetNref);
            Thread FirestThread = new Thread(PrintXref);
            Thread SecondThread = new Thread(PrintYref);
            CoorThread.Start();
            FirestThread.Start();
            SecondThread.Start();
            Console.ReadKey();
        }

        static void GetN()
        {
            n=Convert.ToInt32(Console.ReadLine());
            done = true;
        }

        static void PrintX()
        {
            while (!done) ;
            for (int i = 0; i < n; i++)
            {
                Console.Write('X');
                Thread.Sleep(50);
            }
        }

        static void PrintY()
        {
            while (!done) ;
            for (int i = 0; i < n; i++)
            {
                Console.Write('Y');
                Thread.Sleep(50);
            }
        }
    }
}
```

برنامه ۲-۴: ارتباط مابین نخ ها

خروجی

```
30
XYYXYXYXYXYXYXYXYXYXYXYXYXYXYXXYYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYX
```

برنامه های قبلی مشکلی که دارند این است که در اجراهای مختلف برنامه خروجی های متفاوتی ایجاد می شد و چاپهای X و Y بصورت کاملاً یک در میان نیست، حتی استفاده از متد Sleep نیز باعث نشد که چاپهای X و Y بصورت کاملاً یک در میان شود. برای حل این مشکل می توان از متغیر های سراسری برای ایجاد هماهنگی ما بین Thread ها استفاده کرد. برنامه ۵-۲ از متغیر Flag که از نوع bool می باشد برای ایجاد هماهنگی ما بین FirestThread و SecondThread استفاده می کند. در این برنامه زمانی که Flag=True باشد در خروجی X و زمانی که Flag=False باشد در خروجی Y چاپ خواهد شد. همچنین بعد از چاپ X برای اینکه Y چاپ شود Flag=true خواهد شد و برعکس بعد از چاپ Y برای اینکه X چاپ شود Flag=false خواهد شد.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample2
{
    class Program
    {
        private static bool b = false;
        static void Main(string[] args)
        {
            Thread FirestThread = new Thread(PrintX);
            Thread SecondThread = new Thread(PrintY);
            FirestThread.Start();
            SecondThread.Start();
            Console.ReadKey();
        }
        static void PrintX()
        {
            for (int i = 0; i < 100; i++)
            {
                while (b) ;
                Console.Write('X');
                b = true;
            }
        }
        static void PrintY()
        {
            for (int i = 0; i < 100; i++)
            {
                while (!b) ;
                Console.Write('Y');
                b = false;
            }
        }
    }
}
```

برنامه ۲-۵: چاپ یک در میان X و Y

خروجی

```
XYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXY
XYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXY
XYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXYXY
```

۳-۱-۲- ناحیه بحرانی[1]

یکی از منابع ایجاد خطا در برنامه ها استفاده از متغیر های مشترک ما بین نخ ها می باشد. به منابع مشترک در سیستم عامل ناحیه بحرانی گفته می شود. به این مفهوم Thread safity گفته می شود. فرض کنید تابع GetPrint را به صورت شکل زیر داشته باشیم :

```
GetPrint()
   {
       chin=getch();
       chout=chin;
       putch(chout);
   }
```

فرض کنید دو نخ T1 و T2 را داشته باشیم که هرکدام از این Thread ها اجرا کننده تابع GetPrint باشند و همچنین فرض کتید که در T1 کاراکتر a وارد می شود و در T2 کاراکتر b وارد می شود . با این حساب باید در خروجی عبارت ab و یا عبارت ba چاپ شود. اما ممکن است در خروجی شرایطی پیش بیاید که bb چاپ شود که خروجی صحیحی نمی باشد.

مثال زیر ترتیبی از اجرای T1 و T2 را نشان می دهد که باعث تولید خروجی bb خواهد شد:

T1	T2
Chin=getch(); ➔	Chin=getch(); Chout=chin; Putch(chout); ➔
Chout=chin; Putch(chout);	

۴-۱-۲- حل مسئله ناحیه بحرانی با استفاده از Lock

زمانیکه منابع مشترکی مابین نخ ها وجود داشته باشد عاملی خواهد بود که باعث بروز ایراداتی در برنامه ها می شود و برنامه در اجرا های مختلف مکن است خروجی های متفاوتی تولید کند. نمونه ای از این مشکلات در برنامه ۵-۲ نشان داده شده است.

این برنامه منطقاً باید در خروجی یک بار عبارت Done را چاپ کند اما به خاطر مشکلی که توضیح داده شد، ممکن است در خروجی دو بار عبارت Done چاپ شود، این مشکل به این خاطر است که همزمان دو نخ متفاوت به متغیر یکسانی دسترسی پیدا می کنند. برای حل مشکل باید دسترسی به ناحیه مشترک، کنترل شده انجام شود. روش های مختلفی برای این کار وجود دارد، یکی از روشها استفاده از Semaphor ها و یا استفاده از Lock می باشد. استفاده از Lock باعث می شود تا در یک لحظه از بین نخ ها فقط یک نخ داخل بدنه Lock قرار داشته باشد.

تا زمانی که یکی از نخ ها داخل بدنه Lock قرار دارد به هیچ نخ دیگری اجازه ورود به داخل این بدنه داده نمی شود و نخ یا منتظر می ماند یا مسدود[2]، تا زمانیکه نخ قبلی از بدنه Lock خارج نشده. معمولاً کدهایی که برای دسترسی به

[1] Critical Section
[2] block

متغیرهای مشترک استفاده می شود داخل این بدنه قرار داده می شوند. در برنامه ۲-۶برنامه ۲-۶ با اسـتفاده از Lock سـعی شده مشکل برنامه ۲-۷ شود. در این مثال از بین Thread هایی که تابع Go را فراخوانی می کنند فقـط یـک Thread مـی تواند وارد Locker شود. برای استفاده از Lock باید متغیری از نوع Object تعریف کنیم تا عمل قفل بر اسـاس آن Object انجام شود.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample5
{
    class Program
    {
        static bool Done=false;
        static void Main(string[] args)
        {
            ThreadStart TS = new ThreadStart(Go);
            Thread SampleThread = new Thread(TS);
            SampleThread.Start();

            Go();

            Console.ReadKey();
        }

        static void Go()
        {
            if (!Done)
            {
                Thread.Sleep(1);
                Done = true;
                Console.WriteLine("Done");
            }
        }
    }
}
```

برنامه ۲-۶: مشکل استفاده از متغیر های مشترک مابین نخ ها

خروجی

```
Done
Done
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample6
{
    class Program
    {
        static bool Done = false;
        static object Locker = new object();
        static void Main(string[] args)
        {
            ThreadStart TS = new ThreadStart(Go);
            Thread SampleThread = new Thread(TS);
            SampleThread.Start();
            Go();

            Console.ReadKey();
        }

        static void Go()
        {
            lock (Locker)
            {
                if (!Done)
                {
                    Thread.Sleep(1);
                    Done = true;
                    Console.WriteLine("Done");
                }
            }
        }
    }
}
```

برنامه ۲-۷: حل مسئله ناحیه بحرانی با استفاده از Lock

خروجی

```
Done
```

۵-۱-۲- حل مسئله ناحیه بحرانی با استفاده از Semaphore

یکی از قابلیت های سی شارپ این است که به ما امکان می دهد بتوانیم از سمافورها برای حل مسئله ناحیه بحرانی استفاده نماییم. سمافور تفاوتی که با Lock دارد این است که Lock فقط به یک نخ اجازه می دهد که داخل بدنه Lock قرار داشته باشد اما شما برای سمافور می توانید مشخص نمایید که چه تعداد نخ می توانند داخل بدنه سمافور قرار داشته باشند و همچنین یکی دیگر از مزایای آن این است که بدنه آن با { و } مشخص نمی شود و می تواند داخل توابه متعدد

قرار بگیرد که در درس سیستم عامل توضیح داده می شود. شکل تعریف سمافور بصورت زیر می باشد که بصورت سراسری تعریف خواهد شد.

```
static Semaphore SampleSemaphore = new Semaphore(initialCount, maximumCount);
```

در این تعریف maximumCount مشخص کننده حداکثر تعداد نخهایی می باشد که می توانند داخل بدنه سمافور قرار داشته باشند و initialCount نیز مقدار اولیه این نخ ها را نشان میدهد که در ابتدا می توانند داخل بدنه سمافور وارد شوند. سمافور دارای دو تابع می باشد که بدنه آنرا مسخص می کنند. شروع بدنه سمافور با تابع WaitOne خواهد بود که کنترل می کند که آیا نخ وارد بدنه شود یا نه اگر تعداد نخهای داخل بدنه کافی باشد اجازه ورود نخ جدید را نمی دهد تا این که یکی از نخهای داخل بدنه سمافور تابع Release را فراخوانی کند.

برنامه ۲-۸ نمونه ای از استفاره از سفافور ها را نشان می دهد.

```
static Thread[] t = new Thread[5];
static Semaphore SampleSemaphore = new Semaphore(2, 2);
static void Main(string[] args)
{
   for (int j = 0; j < 5; j++)
   {
        t[j] = new Thread(DoSomething);
        t[j].Start();
   }
   Console.Read();
}
static void DoSomething()
{
   Console.WriteLine("waiting");
   SampleSemaphore.WaitOne();
   Console.WriteLine("{0} begins!");
   Thread.Sleep(1000);
   Console.WriteLine("{0} releasing...");
   SampleSemaphore.Release();
}
```

برنامه ۲-۸: نمونه مثالی از استفاده سمافورها

در این مثال سمافور SampleSemaphore با مقدار اولیه ۲ و حد اکثر مقدار ۲ ایجاد شده یعنی مابین دو تابع WateOne و Release از این سمافور حداکثر دو نخ می توانند قرار داشته باشند. اگر به کد برنامه نگاه کنید چاپ عبارت begins و عبارت releasing مابین این دو تابع قرار دارند، پس در خروجی ما حداکثر دو تا از این عبارتها را کنار هم خواهیم داشت، تا یکی از releasing ها چاپ نشود، begins نخ دیگر چاپ نخواهد شد.

۶-۱-۲- پارامتر دار کردن نخ ها

هنگام استفاده از ThreadStart تابع ورودی نمی تواند شامل پارامتر ورودی باشد. اگر تابعی که می خواهید به عنوان نخ فراخوانی شود شامل پارامتر ورودی باشد، به جای استفاده از ThreadStart باید از ParameterizedThreadStart استفاده کرد. در برنامه ۲-۷ دو تابع PrintX و PrintY شامل یک پارامتر ورودی می باشد که این پارامتر ورودی مشخص کننده تعداد چاپ های X و Y در خروجی می باشد. نماینده این توابع باید از نوع ParameterizedThreadStart باشد .

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample7
{
    class Program
    {
        static void Main(string[] args)
        {
            int m = 100;
            int n = 120;
            ParameterizedThreadStart PrintXref =
                                new ParameterizedThreadStart(PrintX);
            ParameterizedThreadStart PrintYref =
                                new ParameterizedThreadStart(PrintY);
            Thread FirestThread = new Thread(PrintXref);
            Thread SecondThread = new Thread(PrintYref);
            FirestThread.Start(m);
            SecondThread.Start(n);

            Console.ReadKey();
        }
        static void PrintX(object k)
        {
            for (int i = 0; i < (int)k; i++)
            {
                Console.Write('X');
                Thread.Sleep(50);
            }
        }
        static void PrintY(object p)
        {
            for (int i = 0; i < (int)p; i++)
            {
                Console.Write('Y');
                Thread.Sleep(50);
            }
        }
    }
}
```

برنامه ۲-۹: نخ های پارامتر دار

خروجی

```
XYXYYXYXYXYXXYXYYXXYXYXYYXXYYXYXXYXYYXYXXYYXXYYXXYXYYXYXYXXYXYXYYXXYXYYXXYXYX
YXXYYXXYYXYXYXXYYXXYYXXYXYYXXYYXYXYXXYXYYXYXYXXYXYYXXYYXXYYXXYYXYXXYYXXYYXXYX
XYYXXYYXXYXYYXXYYXYXXYYXXYYXYXXYXYYXXYXYYYYYYYYYYYYYYYYYYYYY
```

۷-۱-۲- نام دار کردن نخ ها

برای شناسایی نخ ها در برنامه می توان برای آنها نامی را اختصاص داد. این کار با استفاده از خصوصیت Name نخ انجام می شود. برنامه ۲-۱۰ نمونه ای از نام گذاری نخ را نشان می دهد. در این مثال دو نخ وجود دارد که یکی از آنها با نام Main نام گذاری شده است و Thread دیگر با نام SampleThread نامگذاری شده است و هر یک از نخها نام خود را در خروجی چاپ می کنند.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample8
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread.CurrentThread.Name = "Main";
            Thread SampleThread = new Thread(TestFunction);
            SampleThread.Name = "SampleThread";
            SampleThread.Start();
            TestFunction();

            Console.ReadKey();
        }
        static void TestFunction()
        {
            Console.WriteLine("Hello From "+Thread.CurrentThread.Name);
        }
    }
}
```

برنامه ۲-۱۰: نام دار کردن نخ ها

خروجی

```
Hello From Main
Hello From SampleThread
```

۸-۱-۲- نخ پس زمینه

می توان نخ را به عنوان نخ پس زمینه انتخاب کرد. اگر نخی به عنوان نخ پس زمینه انتخاب شود، هنگام اتمام کار برنامه نخ نیز خاتمه خواهد یافت چه کار نخ تمام شده باشد چه تمام نشده باشد. در صورتی که نخ به عنوان پس زمینه انتخاب نشود برنامه تا پایان نخ منتظر می ماند.

در برنامه ۲-۱۱ اگر هنگام اجرای برنامه پارامتری وارد شود، Worker Thread به عنوان نخ پس زمینه انتخاب خواهد شد و در غیر اینصورت یعنی زمانیکه پارامتری وارد نشود، این عمل اتفاق نمی افتد. قاعدتا اگر پارامتری وارد نشود، برنامه تا اتمام کار Worker Thread منتظر خواهد ماند این نخ نیز کلیدی از کاربر خواهد گرفت در نتیجه اگر هنگام اجرای برنامه

پارامتری وارد نشود برنامه پس از گرفتن کلید از کاربر خاتمه خواهد یافت. اگر پارامتری وارد شود Worker Thread به عنوان پس زمینه تعریف شده و با اتمام برنامه Worker Thread نیز به اجبار به تمام می رسد و دیگر برنامه منتظر فشار دادن کلید نمی ماند.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadSample9
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread worker = new Thread(delegate() { Console.ReadKey(); });
            if(args.Length>0)
                worker.IsBackground=true;
            worker.Start();
        }
    }
}
```

برنامه ۲-۱۱: نخ پس زمینه

۹-۱-۲- متد JOIN

از این متد می توان برای ملحق کردن نخها به هم استفاده کرد. زمانیکه یک نخ متد Join نخ دیگری را فراخوانی می کند، متوقف شده و منتظر می ماند تا زمانیکه کار آن نع تمام نشده اجرا نخواهد شد. به محض اتمام کار نخی که متد Join آن فراخوانی شده، اجرای نخ مربوطه از سطر بعد Join ادامه می یابد.

در برنامه ۲-۱۲ چهار نخ وجود دارد. نخ اصلی بعد از Start کردن CoorThread تابع Join آن را فراخوانی می کند. به همین دلیل نخ اصلی تا زمانیکه CoorThread تمام نشده اجرا نخواهد شد. همچنین بعد از Start شدن FirstThread و SecondThread متد Join آنها توسط نخ اصلی برنامه فراخوانی می شود به همین دلیل تا اتمام کار آنها متوقف می شود. به همین دلیل تا زمانی که کار این نخ ها تمام نشده پیام انتهایی برنامه چاپ نخواهد شد.

۲-۲- توابع API ویندوز

واژه API مخفف شده سه کلمه Application Programming Interface می باشد که یک رابط نرم افزار است که در برنامه های دیگر استفاده می گردد. در سیستم مدل Dos با مفهوم Interrupt روبرو بودیم، Interrupt ها کدهی از پیش نوشته شده ای می باشند که کاربر از آنها استفاده می‌کند. در سیستم عامل ویندوز به جای Interrupt با توابع API روبرو هستیم. سیستم عامل ویندوز بیش از هزار تابع API دارد و هر برنامه ای تحت ویندوز کارهای خود را با استفاده از فراخوانی توابع API انجام می‌دهد. این توابع API مابین تمام برنامه ها مشترک خواهد بود و همچنین تمام زبانهای برنامه سازی می توانند از آنها استفاده کنند. یک API رابط نرم افزاری سیستم های عامل می باشد. API یک سیستم عامل یکی از دلایل بنیادی و اساسی برای وجود تفاوت و عدم هماهنگی سیستمهای عامل با هم می باشند. برای مثال یک قطعه در یک

سیستمی بر پایه مکینتاش نمی تواند در ویندوز اجرا شود (البته بدون وجود شبیه ساز آن) زیـرا سیسـتم عامـل مکینتـاش و ویندوز API های متفاوتی دارند.

```
using System.Threading;

namespace ThreadSample10
{
    class Program
    {
        private static int n;
        static void Main(string[] args)
        {
            ThreadStart GetNref = new ThreadStart(GetN);
            ThreadStart PrintXref = new ThreadStart(PrintX);
            ThreadStart PrintYref = new ThreadStart(PrintY);
            Thread CoorThread = new Thread(GetNref);
            Thread FirestThread = new Thread(PrintXref);
            Thread SecondThread = new Thread(PrintYref);

            CoorThread.Start();
            CoorThread.Join();
            Console.WriteLine("CoorThread operation has been finished.");
            FirestThread.Start();
            SecondThread.Start();

            FirestThread.Join();
            SecondThread.Join();

            Console.WriteLine();
            Console.WriteLine("FirestThread & SecondThread operation has been
finished.");

            Console.ReadKey();
        }
        static void GetN()
        {
            n = Convert.ToInt32(Console.ReadLine());
        }
        static void PrintX()
        {
            for (int i = 0; i < n; i++)
            {
                Console.Write('X');
                Thread.Sleep(50);
            }
        }
        static void PrintY()
        {
            for (int i = 0; i < n; i++)
            {
                Console.Write('Y');
                Thread.Sleep(50);
            }
        }
    }
}
```

برنامه ۲-۱۲: فراخوانی متد join

۳-۲- دلیل استفاده از توابع API در برنامه‌نویسی

دلایل استفاده از توابع API در زبانهای مختلف برنامه‌نویسی می‌تواند این باشد که:

۱. توابع API به دلیل آنکه در فایلهای dll هر نسخه از سیستم عامل ویندوز قرار دارند و در هر ویندزی پیشتیبانی می شوند پس نیازی به ارائه آن فایل dll در نسخه برنامه نمی‌باشد و درنتیجه حجم نسخه کم می‌شود و درضمن سندیت برنامه نیز بیشتر می‌شود و می‌توان گفت که شما از منابع ویندوز به نحو احسن استفاده کرده‌اید.

۲. نسخه‌های ویندوز به طور مداوم تغییر می کند ولی به دلیل آنکه سازندگان همیشه حالتی را در نظر می‌گیرند که نسخه‌های قبلی را نیز پشتیبانی کند. درنتیجه اگر شما برنامه‌ای را به کمک توابع API بنویسید با تغییر نسخه ویندوز نیازی به تغییر جدی در توابع API نمی‌باشد.

۳. بیشتر زبانهای برنامه‌نویسی (به‌خصوص زبانهای تحت ویندوز) که خود به صورت پنهان از توابع API استفاده می‌کنند، ممکن است به علت محدودیتهایی نتوانند تمام امکانات توابع را در اختیار قرا دهند. شما با دسترسی مستقیم به توابع می‌توانید از حداکثر قابلیتهای تابع استفاده کنید.

۴. در بعضی از زبانهای برنامه‌نویسی برای آنکه بتوان یک حالت را بوجود آورد و یا کار مشخصی را انجام داد، باید کدهای زیادی بنویسیم و یا در زمان خطاگیری مدت زیادی را صرف کنیم و به طور حتم کاربر استفاده‌کننده از برنامه شما نیز باید زمان بیشتری را برای گرفتن جواب صرف کند. این موارد ذکر شده هر کدام به نوبه خود می‌توانند از محبوبیت، قدرتمند و خوانابودن برنامه بکاهند. ولی توابع API به دلیل آنکه روتین شده و از قبل نوشته شده می‌باشند پس فقط کافیست تابع را فراخوانی کنیم و به آن ورودی دهیم و خروجی مورد نظر خود را دریافت کنیم.

۵. بیشتر توابع کارهایی را انجام می‌دهند که زبانهای برنامه‌نویسی قادر به انجام آن نمی‌باشند. به عنوان مثال به تابع SetlateradwindowAttributes مراجعه کنید که باعث می‌شود یک پنجره (فرم و یا کنترلهای داخل آن) و با یک رنگ مشخص در آنها به مقدار دلخواه به حالت شفاف و Transparent تبدیل شوند. و یا توابع دیگر مانند StretchBlt , TransparentBlt , LokworkStation , TimGetTim , HShutDownDialog و ...

۶. و چندین علت دیگر....

توابع API ویندوز توابع داخلی ویندوز هستند که اکثر زبانهای برنامه سازی تحت ویندوز با عملیاتی می تواند از آنها استفاده کند با استفاده از روتین های API هرکاری که در ویندوز قابل اجرا باشد در آن زبان برنامه سازی نیز قابل اجرا می گردد. تمام توابع API ویندوز در درون DLL ها قرار دارند. DLL مخفف عبارت Dynamic Link Library می باشد.

۱-۳-۲- فایلهای کتابخانه ای پویا

فایلهای کتابخانه‌ای می‌توانند ***ایستا*** باشند مانند Header فایلهای زبان C و می‌توانند ***پویا*** باشند مانند فایلهای DLL.

هنگام فراخوانی کردن توابعی از فایل کتابخانه‌ای ایستا کد مربوط به این توابع هنگام کامپایل به برنامه کامپایل شده اضافه خواهد شد، در نتیجه بعد از کامپایل کردن برنامه هنگام اجرا اگر فایل کتابخانه‌ای مربوطه وجود نداشته باشد مشکلی ایجاد نخواهد شد. اما زمانی که از فایلهای کتابخانه‌ای پویا استفاده شود هنگام کامپایل کردن کد مربوط به این توابع به

برنامه کامپایل شده اضافه نمی‌شود بلکه هنگام اجرا تابع مربوطه از فایل Dll بر روی حافظه بارگذاری شده و اجـرا مـی‌شـود. پس در نتیجه اگر هنگام اجرا، فایل Dll مربوطه وجود نداشته باشد برنامه اجرا نخواهد شد.

برخی از مزایای فایلهای Dll عبارتند از:

۱- کاهش حجم برنامه‌های تولید شده

۲- قابلیت استفاده مجدد از توابع نوشته شده

۳- کاهش حجم برنامه‌نویسی

۴- برخی از کارها بدون استفاده از توابع API سیستمی که داخل فایلهای Dll ویندوز قرار دارند امکان‌پذیر نمی‌باشـد و ...

اغلب DLL های ویندوز در دایرکتوری windows یا \windows\system32\ قرار گرفته است. مهمتـرین فایلهـای Dll که اکثـر توابـع API ویـنـدوز در آنهـا قـرار دارنـد عبارتنـد از: Kernel32.Dll، Advapi32.dll، User32.dll، Ole32.dll ، Winmm.dll، Winspool.dll، Gdi32.dll، Mpr.dll، Netapi32.dll، Sheu32.dll، Rasapi32.dll، Version.dll، Wsock32.dll، Comdlg32.dll و ...

۲-۳-۲- معماری ویندوز

معماری ویندوز براساس پیـام‌هـا مـی‌باشـد، پیـام‌هـا هشـدارهایی مـی‌باشـند ه براسـاس Event هـا تولیـد شـده و بـه Application یا برنامه مربوطه ارسال می‌شود، مانند حرکت ماوس، کلیک کردن و ... هر پیام درواقع رکوردی مـی‌باشـد کـه شامل اطلاعاتی راجع به نوع event و پارامترهای احتمالی event می‌باشد. از جمله مواردی که در پیام‌ها وجود دارد عبارتند از:

۱. hwnd : دستگیره پنجره‌ای که پیام به مقصد آن ارسال می‌شود.
۲. message : مقدار ثابت که پیام را نشان می‌دهد.
۳. wparam : شامل اطلاعات اضافی پیام می‌باشد.
۴. IParam : این فیلد نیز مانند wParam شامل اطلاعات اضافی پیام می‌باشد.

درواقع در سیستم پیام‌رسانی ویندوز اتفاقات زیر روی می‌دهد:

۱. رخ دادن event در سیستم
۲. انتقال event به یک پیام و قرار دادن آن در صف پیام
۳. واکنش پیام توسط برنامه از صف
۴. انتقال پیام توسط برنامه به رویّه‌ی پنجره مورد نظر در برنامه‌ی کاربردی
۵. انجام عمل از سوی رویّه‌ی پنجره در پاسخ به پیام

۲-۳-۳- استفاده از توابع API در C#

برای استفاده از توابع API در C# اولین کاری که باید انجام دهیم این است که عبارت

```
using System.Runtime.InteropServices;
```

را به ابتدای برنامه اضافه می‌کنیم، سپس تابع API مورد نظر از فایل Dll مربوطه داخل برنامه باید import شود به منظور Import کردن از دستور DllImport استفاده می‌کنیم. به عنوان مثال دستور زیر تابع charupper را از فایل user32.Dll داخل برنامه Import می‌کند، این تابع یک ورودی از نوع string و یک خروجی از نوع string خواهد داشت:

```
[DllImport("user32.dll")]
public static extern string CharUpper(string m);
```

میتوان از تابع API در C# با نام اصلی آن استفاده کرد یا با نام تغییر یافته. در این متال تابع CharUpper در برنامه با نام اصلی خود شناخته شده و استفاده خواهد شد. اما می توان نام آنرا در برنامه تغییر داد و استفاده کرد به عنوان مثال:

```
[DllImport("user32.dll", EntryPoint = "CharUpper")]
public static extern string ChUpper(string m);
```

در این مثال تابع CharUpper در برنامه با نام ChUpper شناخته و استفاده خواهد شد.

نکته: باید در هنگام استفاده از DllImport به نحوه نگارش نام تابع دقت داشته باشید. باید به همان شکلی تعریف شود که داخل فایل DLL تعریف شده است.

۴-۳-۲- تابع MessageBox

این تابع که در فایل user32.dll قرار دارد برای نمایش پیام در صفحه نمایش استفاده می شود. شکل کلی تعریف این تابع در C# بصورت زیر می باشد.

```
[DllImport("user32.dll", EntryPoint = "MessageBox")]
private static extern int ShowMessage(
    int hWnd,
    string text,
    string caption,
    uint type);
```

هر یک از پارامترهای این تابع عبارتند از:

- hWnd: دستگیره ای برای پنجره ایجاد کننده پیام می باشد که باید ایجاد شود. اگر برابر null قرار داده شود جعبه پیام پنجره مالک نخواهد داشت.
- text: مشخص کننده متن پنجره پیام می باشد.
- caption: مشخص کننده عنوان پنجره پیام می باشد.
- type: مشخص کننده نوع پنجره پیام می باشد.

قطعه کد زیر نمونه ای از فراخوانی این تابع را نشان می دهد.

```
string caption = "Visual C# 2015 - MessageBox Sample";
string text = "!!!  This is a sample message   !!!";
ShowMessage(0, text, caption, 0);
```

برنامه ۲-۱۳: نمونه ای از کاربرد MessageBox

برنامه ۲-۱۴ نمونه ای دیگر از فراخوانی این تابع را نمایش می دهد. در این قطعه کد نوع پنجره پیام که عددی در بازه ۰ تا ۷ می باشد توسط numericUpDown از کاربر گرفته شده و پس از نمایش پنجره پیام نوع کلید فشار داده شده توسط کاربر را تشخیص می دهد این تشخیص از طریق عددی که توسط تابع برگشت داده می شود تشخیص داده می شود.

۵-۳-۲- تابع GetVersionEx

این تابع در فایل kernel32.dll قرار داشته و برای بدست آوردن اطلاعات نسخه سیستم عامل ویندوز استفاده می شود.

برای استفاده از این تابع باید ساختمان OS_VERSION_INFO به شکل زیر تعریف شود.

```
public struct OS_VERSION_INFO
{
    public Int32 dwOSVersionInfoSize;
    public Int32 dwMajorVersion;
    public Int32 dwMinorVersion;
    public Int32 dwBuildNumber;
    public Int32 dwPlatformId;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 128)]
    public String szCSDVersion;
}
```

اطلاعات نسخه سیستم عامل از طریق این ساختمان برگشت داده خواهد شد. شکل کلی تعریف این تابع بصورت زیر خواهد بود.

```
[DllImport("kernel32")]
static extern bool GetVersionEx(ref OS_VERSION_INFO lpVersionInfo);
```

ورودی این تابع متغیری از نوع ساختمان OS_VERSION_INFO بوده که اطلاعات نسخه سیستم عامل ویندوز از طریق آن برگشت داده خواهد شد. بعد از فراخوانی این تابع به عنوان مثال مولفه dwMajorVersion از این ساختمان شماره نسخه اصلی ویندوز، مولفه dwMinorVersion شماره نسخه فرعی و مولفه dwBuildNumber شماره تولید سیستم عامل را مشخص خواهد کرد. این تابع همچنین یک مقدار bool را برگشت خواهد داد اگر این مقدار برابر True باشد یعنی تابع توانسته به درستی نسخه سیستم عامل را تشخیص دهد و اگر درست تشخیص نداده باشد مقدار False را برگشت خواهد داد. برنامه ۲-۱۵ نمونه ای از فراخوانی این تابع را نشان می دهد.

```
string caption = "Visual C# 2015 - MessageBox Sample";
string text = textBox1.Text;
uint type = Convert.ToUInt32(numericUpDown1.Value);
int k = ShowMessage(0, text, caption, type);

string s = "Returned Value is " + k + "\n and Pressed key is ";
switch(k)
{
   case 0:
   case 1:
       s += "OK";
       break;
   case 2:
       s += "CANCEL";
       break;
   case 3:
       s += "ABORT";
       break;
   case 4:
       s += "RETRY";
       break;
   case 5:
       s += "IGNORE";
       break;
   case 6:
       s += "YES";
       break;
   case 7:
       s += "NO";
       break;
   case 10:
       s += "TRY AGAIN";
       break;
   case 11:
       s += "CONTINUE";
       break;
}
ShowMessage(0, s, "", 0);
```

برنامه ۲-۱۴: نمونه ای دیگر از کاربرد MessageBox

```
const int MB = 1048576;//MEGABYTE

string osName = "Unknown";
OS_VERSION_INFO info = new OS_VERSION_INFO();
info.dwOSVersionInfoSize = Marshal.SizeOf(info);

if (!GetVersionEx(ref info))
    ShowMessage(0, "GetVersion failed" , "Error", 0);

switch (info.dwMajorVersion + "." + info.dwMinorVersion)
{
    case "10.0":
        osName = "Win 10";
        break;
    case "6.2":
        osName = "Win 8";
        break;
    case "6.1":
        osName = "Win 7";
        break;
    case "6.0":
        osName = "Win Vista";
        break;
    case "5.2":
        osName = "Win 2003";
        break;
    case "5.1":
        osName = "Win XP";
        break;
    case "5.0":
        osName = "Win 2000";
        break;
    case "4.0":
        osName = "Win NT 4.0";
        break;
}

label8.Text = osName;
label9.Text = Convert.ToString(info.dwMajorVersion)+'.'+
              Convert.ToString(info.dwMinorVersion)+'.'+
              Convert.ToString(info.dwBuildNumber);
```

برنامه ۲-۱۵: نمونه ای از کاربرد تابع GetVersionEx

۶-۳-۲- تابع GetSystemInfo

این تابع در فایل kernel32.dll قرار داشته و برای بدست آوردن اطلاعات سیستم استفاده می شود. برای استفاده از این تابع باید ساختمان SYSTEM_INFO به شکل زیر تعریف شود.

```
public struct SYSTEM_INFO
{
    public uint dwOemId;
    public uint dwPageSize;
    public uint lpMinimumApplicationAddress;
    public uint lpMaximumApplicationAddress;
    public uint dwActiveProcessorMask;
    public uint dwNumberOfProcessors;
    public uint dwProcessorType;
    public uint dwAllocationGranularity;
    public uint dwProcessorLevel;
    public uint dwProcessorRevision;
}
```

هر یک از مولفه های این ساختمان اطلاعاتی از سیستم را در اختیار ما قرار می دهد. به عنوان مثال dwNumberOfProcessors تعداد پردازنده های سیستم را مشخص می کند، lpMinimumApplicationAddress حداقل آدرس برنامه ها را مشخص می کند و ... . شکل تعریف این تابع بصورت زیر می باشد.

```
[DllImport("kernel32")]
static extern void GetSystemInfo(ref SYSTEM_INFO pSI);
```

اطلاعات سیستم از طریق پارامتر ورودی این تابع برگشت داده خواهد شد.

```
const int MB = 1048576;//MEGABYTE
SYSTEM_INFO pSI = new SYSTEM_INFO();
GetSystemInfo(ref pSI);

label10.Text = Convert.ToString(pSI.dwNumberOfProcessors);

label11.Text = Convert.ToString(pSI.lpMinimumApplicationAddress / MB)
+ " to " + Convert.ToString(pSI.lpMaximumApplicationAddress / MB);
```

برنامه ۲-۱۶: نمونه ای از استفاده تابع GetSystemInfo

۷-۳-۲- تابع GlobalMemoryStatus

این تابع در فایل kernel32.dll قرار داشته و برای بدست آوردن اطلاعات وضعیت حافظه سیستم استفاده می شود. برای استفاده از این تابع باید ساختمان MEMORY_STATUS به شکل زیر تعریف شود.

```
public struct MEMORY_STATUS
{
    public uint dwLength;
    public uint dwMemoryLoad;
    public uint dwTotalPhys;
    public uint dwAvailPhys;
    public uint dwTotalPageFile;
    public uint dwAvailPageFile;
    public uint dwTotalVirtual;
    public uint dwAvailVirtual;
}
```

هر یک از مولفه های این ساختمان اطلاعاتی از حافظه سیستم را در اختیار ما قرار می دهد. به عنوان مثال dwTotalPhys مجموع حافظه فیزیکی سیستم را بر حسب بایت مشخص می کند، dwTotalVirtual مجموع حافظه مجازی سیستم را مشخص می کند و ... . شکل تعریف این تابع بصورت زیر می باشد.

```
[DllImport("kernel32")]
static extern void GlobalMemoryStatus(ref MEMORY_STATUS buf);
```

اطلاعات سیستم از طریق پارامتر ورودی این تابع برگشت داده خواهد شد. برنامه ۲-۱۷ نمونه ای از استفاده این تابع را نشان می دهد.

```
const int MB = 1048576;//MEGABYTE
MEMORY_STATUS memSt = new MEMORY_STATUS();
GlobalMemoryStatus(ref memSt);

label12.Text = Convert.ToString(memSt.dwAvailPhys / MB) + " : " +
Convert.ToString(memSt.dwTotalPhys / MB)+" (available:total)";

label13.Text = Convert.ToString(memSt.dwAvailVirtual / MB) + " : " +
Convert.ToString(memSt.dwTotalVirtual / MB) + " (available:total)";
```

برنامه ۲-۱۷: نمونه ای از استفاده تابع GetSystemInfo

۸-۳-۲- تابع GetLastError

این تابع که در فایل kernel32.dll قرار دارد برای بدست آوردن کد آخرین ایراد اتفاق افتاده در سیستم استفاه می شود. شکل کلی تعریف این تابع بصورت زیر می باشد.

```
[DllImport("kernel32")]
static extern int GetLastError();
```

این تابع پارامتر ورودی نداشته و پس از اجرا عددی را برگشت خواهد داد که نشان دهنده کد ایراد اتفاق افتاده در سیستم می باشد. برنامه ۲-۱۸ نمونه ای از فراخوانی این تابع را نمایش می دهد. در این برنامه یک ایراد عمدی در برنامه

ایجاد می شود تا کد آن توسط تابع GetLastError برگشت داده شود. برای مشخص کردن نوع ایراد اتفاق افتاده در سیستم از طریق کد آن باید به سایت ماکروسافت مراجعه نمایید.

```
int errorCode;
string caption = "Visual C# 2015";
string text = "Hello, world!";
ShowMessage(10, text, caption, 0);

errorCode = GetLastError();
label15.Text = Convert.ToString(errorCode);
```

برنامه ۲-۱۸: نمونه برنامه استفاده از تابع GetLastError

۹-۳-۲- تابع Sleep و SleepEx

این توابع که در فایل Kernel32.dll قرار دارند برای ایجاد وقفه در اجرای برنامه استفاده می شوند. شکل کلی تعریف این توابع بصورت زیر خواهد بود.

```
[DllImport("Kernel32.dll")]
public static extern void Sleep(Int16 T);

[DllImport("Kernel32.dll")]
public static extern Int16 SleepEx(Int16 T,bool b);
```

در این تعریف تابع Sleep یک عدد صحیح را به عنوان پارامتر دریافت کرده که این عدد مشخص کننده میزان تاخیر بر اساس میلی ثانیه می باشد. تابع Sleep هیچ مقداری را برگشت نمی دهد. تابع SleepEx دارای دو پارامتر ورودی می باشد پارامتر اول مشخص کننده میزان تاخیر بر اساس میلی ثانیه می باشد و پارامتر دوم یک bool می باشد. اگر این پارامتر برابر False باشد در مدت زمان تاخیر اجازه عملیات I/O داده نخواهد شد. اما اگر برابر True قرار داده شود اجازه I/O در مدت تاخیر (ReadFileEx یا WriteFileEx) داده خواهد شد.

این تابع یک عدد صحیح را به عنوان نتیجه برگشت خواهد داد. این عدد مشخص کننده میزان تاخیر باقی مانده از این تابع می باشد. اگر برابر صفر باشد یعنی تاخیر بصورت کامل انجام شده و باقی ماندهای ندارد. اما هر عدد غیر صفر مشخص کننده این می باشد که تاخیر کامل نشده است. برنامه ۲-۱۹ و برنامه ۲-۲۰ نمونه ای از فراخوانی این توابع را نشان می دهد.

```
int i;
for (i = 0; i <= 100; i++)
{
      Sleep(Convert.ToInt16(numericUpDown1.Text));
      progressBar1.Value = i;
}
```

برنامه ۲-۱۹: نمونه کد فراخوانی تابع Sleep

```
int i;
for (i = 0; i <= 100; i++)
{
      SleepEx(Convert.ToInt16(numericUpDown1.Text), true);
      progressBar1.Value = i;
}
```

برنامه ۲-۲۰: نمونه کد فراخوانی تابع SleepEx

۱۰-۳-۲- تابع GetLocalTime

این تابع که در فایل Kernel32.dll قرار دارد برای گرفتن تاریخ و ساعت سیستم استفاده می شود. برای استفاده از این تابع باید ساختمانی به شکل زیر تعریف شود.

```
    public struct LPSYSTEMTIME
    {
        public ushort wYear;
        public ushort wMonth;
        public ushort wDayOfWeek;
        public ushort wDay;
        public ushort wHour;
        public ushort wMinute;
        public ushort wSecond;
        public ushort wMilliseconds;

        public void FromDateTime(DateTime time)
        {
            wYear = (ushort)time.Year;
            wMonth = (ushort)time.Month;
            wDayOfWeek = (ushort)time.DayOfWeek;
            wDay = (ushort)time.Day;
            wHour = (ushort)time.Hour;
            wMinute = (ushort)time.Minute;
            wSecond = (ushort)time.Second;
            wMilliseconds = (ushort)time.Millisecond;
        }
    };
```

در این ساختمان هر یک از مولفه ها مشخص کننده سال، ماه، روز هفته، روز، ساعت، دقیقه، ثانیه و میلی ثانیه می باشند. همچنین در این ساختمان تابعی به نام FromDateTime برای تبدیل نوع داده ای DateTime به مولفه های این ساختمان قرار داده شده است. شکل کلی تعریف این ساختمان بصورت زیر می باشد.

```
[DllImport("Kernel32.dll")]
public static extern void GetLocalTime(ref LPSYSTEMTIME ST);
```

این تابع متغیری را از نوع ساختمان LPSYSTEMTIME را به عنوان پارامتر دریافت کرده و تاریخ و ساعت سیستم را از طریق آن برگشت میدهد. برنامه ۲-۲۱ نمونه ای از فراخوانی این تابع را نشان می دهد.

```
LPSYSTEMTIME S_T = new LPSYSTEMTIME();
GetLocalTime(ref S_T);
textBox1.Text = Convert.ToString(S_T.wYear);
textBox2.Text = Convert.ToString(S_T.wMonth);
textBox3.Text = Convert.ToString(S_T.wDayOfWeek);
textBox4.Text = Convert.ToString(S_T.wDay);
textBox5.Text = Convert.ToString(S_T.wHour);
textBox6.Text = Convert.ToString(S_T.wMinute);
textBox7.Text = Convert.ToString(S_T.wSecond);
textBox8.Text = Convert.ToString(S_T.wMilliseconds);
```

برنامه ۲-۲۱: برنامه نمونه برای فراخوانی تابع GetLocalTime

۱۱-۳-۲- تابع SetLocalTime

این تابع که در فایل Kernel32.dll قرار دارد برای تغییر تاریخ و ساعت سیستم استفاده می شود. برای استفاده از این تابع باید ساختمان LPSYSTEMTIME همانند بخش قبلی (بخش ۱۰-۳-۲-) به تعریف شود. شکل کلی تعریف این تابع بصورت زیر می باشد.

```
[DllImport("Kernel32.dll")]
public static extern bool SetLocalTime(ref LPSYSTEMTIME Time);
```

این تابع متغیری از نوع ساختمان LPSYSTEMTIME را به عنوان پارامتر دریافت کرده که مشخص کننده تاریخ و ساعت جدید سیستم می باشد. این تابع یک مقدار bool را به عنوان نتیجه برگشت میدهد که مشخص کننده این است که آیا تغییر تاریخ و ساعت به موفقیت انجام شده است (True) یا نه (False).

نکته: باید دقت داشته باشید که روز هفته توسط این تابع قابل تغییر نمی باشد.

نکته: باید دقت داشته باشید در روی ویندوزهای ۸ به بعد زمانی تاریخ و ساعت قابل تغییر می باشند که برنامه بصورت Run as administrator اجرا شده باشد. در غیر این صورت خطای کد ۱۳۱۴ اتفاق خواهد افتاد.

```
LPSYSTEMTIME S_T = new LPSYSTEMTIME();
S_T.wYear = Convert.ToUInt16(textBox1.Text);
S_T.wMonth = Convert.ToUInt16(textBox2.Text);
S_T.wDay = Convert.ToUInt16(textBox4.Text);
S_T.wHour = Convert.ToUInt16(textBox5.Text);
S_T.wMinute = Convert.ToUInt16(textBox6.Text);
S_T.wSecond = Convert.ToUInt16(textBox7.Text);
S_T.wMilliseconds = Convert.ToUInt16(textBox8.Text);
if (SetLocalTime(ref S_T)) label11.Text = "OK";
else  label11.Text = "Error "+Convert.ToString(GetLastError());
```

برنامه ۲-۲۲: نمونه فراخوانی تابع SetLocalTime

```
DateTime t = DateTime.Now.AddDays(7);

LPSYSTEMTIME st = new LPSYSTEMTIME();
st.FromDateTime(t);

if (SetLocalTime(ref st)) label11.Text = "OK";
else label11.Text = "Error " + Convert.ToString(GetLastError());
```

برنامه ۲-۲۳: نمونه ای از فراخوانی تابع SetLocalTime که موجب می شود ۷ روز به تاریخ جاری سیستم افزوده شود.

۱۲-۳-۲- تابع CreateDC

این تابع که در فایل gdi32.dll قرار دارد برای ایجاد دستگره به صفحه نمایش استفاده می شود. شکل کلی این تابع بصورت زیر می باشد.

```
[DllImport("gdi32.dll")]
static extern IntPtr CreateDC(string lpszDriver, string lpszDevice,
string lpszOutput, IntPtr lpInitData);
```

این تابع شامل ۴ پارامتر ورودی می باشد و همچنین مقداری از نوع IntPtr را به عنوان نتیجه برگشت خواهد داد. این مقدار برگشت داده شده دسگیره ای به صفحه نمایش می باشد که از طریق آن می توان تغییراتی را در صفحه ایجاد کرد. پارامتر اول که از نوع string می‌باشد مشخص کننده‌ی نام درایوری می‌باشد که می‌خواهید پرده‌ی DC روی آن ایجاد شود. این گزینه معمولاً برابر است با عبارت Display اگر پارامتر اول برابر Display باشد پارامترهای بعدی برابر Null خواهند بود. اگر این تابع کار خود را به درستی انجام دهد خروجی آن دستگیره (Handle) ای به پرده زمینه‌ی ایجاد شده خواهد بود. برای ایجاد دستگیره به صفحع نمایش پارامتر های این تابع باید به شکل زیر تنظیم شوند.

```
CreateDC("DISPLAY", "", "", IntPtr.Zero)
```

برنامه زیر نمونه ای از فراخوانی این تابع را نمایش میدهد.

```
DC = CreateDC("DISPLAY", "", "", IntPtr.Zero);
if (DC == IntPtr.Zero) label1.Text = "Error";
else label1.Text = Convert.ToString(DC);
```

برنامه ۲-۲۴: نمونه فراخوانی تابع CreateDC

از نتیجه اجرای این تابع می توان برای بقیه توابع وستکاری کننده صفحه نمایش استفاده کرد.

۱۳-۳-۲- تابع DeleteDC

این تابع که در فایل gdi32.dll قرار دارد برای حذف دستگیره ایجاد شده به صفحه نمایش استفاده می شود. بعد از حذف دستگیره دیگر صفحه نمایش با آن دستگیره قابل تغییر نخواهد بود و باید دوباره تابع CreateDC فراخوانی شود. شکل کلی تعریف این تابع بصورت زیر می باشد.

```
[DllImport("gdi32.dll", ExactSpelling = true, SetLastError = true)]
static extern bool DeleteDC(IntPtr hdc);
```

این تابع یک مقدار از نوع IntPtr را به عنوان پارامتر دریافت کرده که همان دستگیره ای می باشد که توسط تابع CreateDC ایجاد شده بود. این تابع یک مقدار bool را به عنوان نتیجه برگشت خواهد داد که مشخص می کند آیا دستگیره با موفقیت حذف شده است (True) یا نه (False).

۱۴-۳-۲- تابع Drowtext

این تابع که در فایل User32.dll قرار دارد برای رسم متن بر روی پرده‌ی DC ایجاد شده استفاده می‌شود و شکل کلی این تابع به صورت زیر می‌باشد:

```
[DllImport("user32.dll", CharSet = CharSet.Unicode)]
static extern int DrawText(IntPtr hdc, string lpStr, int nCount, ref
Rect lpRect, int wFormat);
```

پارامتر اول مشخص کننده‌ی دستگیره‌ی پرده زمینه‌ی DC ایجاد شده می‌باشد. پارامتر دوم مشخص کننده‌ی متنی می‌باشد که باید نمایش داده شود. پارامتر سوم مشخص کننده‌ی طول رشته‌ای می‌باشد که باید نمایش داده شود، عدد ۱- مشخص کننده‌ی کل رشته می‌باشد. پارامتر چهارم مشخص کننده‌ی ابعاد محدوده‌ای می‌باشد که متن باید داخل آن نمایش داده شود و پارامتر پنجم مشخص کننده‌ی فرمت متن برای نمایش می‌باشد.

برای مشخص کردن فرمت ثابتهایی را به صورت زیر تعریف کرده‌ایم:

```
private const int DT_CENTER = 0x1;
private const int DT_VCENTER = 0x4;
private const int DT_SINGLELINE = 0x20;
```

به غیر از این حالتها، حالتهای دیگری نیز وجود دارد که میتوانید از اینترنت لیست آنها پیدا کنید. همچنین برای مشخص کردن محدوده چاپ متن باید ساختمانی به شکل زیر تعریف شود.

```
private struct Rect
{
     public int Left, Top, Right, Bottom;
     public Rect(int L, int T, int B, int R)
     {
        this.Left = L;
        this.Top = T;
        this.Bottom = B;
        this.Right = R;
     }
}
```

قطعه کد زیر نمونه مثالی از کاربرد این تابع را نشان می‌دهد:

```
Rect bounds = new Rect(200,200,1,1);
int flags = DT_CENTER | DT_VCENTER | DT_SINGLELINE;
DrawText(DC, textBox1.Text, -1,ref bounds, flags);
```

۱۵-۳-۲- تابع TextOut

این تابع که در فایل gdi32.dll قرار دارد برای چاپ متن در صفحه نمایش استفاده می شود. شکل کلی تعریف این تابع بصورت زیر می باشد:

```
[DllImport("gdi32.dll", CharSet = CharSet.Auto)]
static extern bool TextOut(IntPtr hdc, int nXStart, int
nYStart, string lpString, int cbString);
```

این تابع دارای پنج پارامتر ورودی می باشد. پارامتر اول از نوع IntPtr مشخص کننده دستگیره پرده ایجاد شده در صفحه نمایش می باشد. پارامتر دوم و سوم مشخص کننده موقعیت X و Y در صفحه نمایش برای چاپ متن می باشند. پارامتر چهارم مشخص کننده متنی می باشد که باید چاپ شود. پارامتر پنجم مشخص کننده تعداد کاراکتر های چاپی می باشد. همچنین این تابع یک bool را به عنوان نتیجه برگشت می دهد که نشان دهنده آن است که آیا چاپ موفق بوده (True) یا نه (False). قطعه کد زیر نمونه ای از فراخوانی تابع را نشان می دهد.

```
TextOut(DC, 100, 100, textBox1.Text, textBox1.Text.Length);
```

۱۶-۳-۲- تابع AngleArc

این تابع که در فایل gdi32.dll قرار دارد برای رسم کمان در صفحه نمایش استفاده می شود. شکل کلی تعریف این تابع بصورت زیر می باشد.

```
[DllImport("gdi32.dll")]
static extern bool AngleArc(IntPtr hdc, int X, int Y, uint dwRadius,
          float eStartAngle, float eSweepAngle);
```

این تابع دارای شش پارامتر ورودی می باشد. پارامتر اول از نوع IntPtr مشخص کننده دستگیره پرده ایجاد شده در صفحه نمایش می باشد. پارامتر دوم و سوم مشخص کننده موقعیت X و Y در صفحه نمایش کمان می باشند. پارامتر چهارم مشخص کننده شعاع کمان می باشد. پارامنر پنجم مشخص کننده نقطه شروع کمان و پارامتر پنجم مشخص کننده نقطه پایانی کمان می باشد. همچنین این تابع یک bool را به عنوان نتیجه برگشت می دهد که نشان دهنده آن است که آیا نمایش کمان موفق آمیز بوده (True) یا نه (False). قطعه کد زیر نمونه ای از فراخوانی تابع را نشان می دهد.

```
for (int i = 0; i < 200; i = i + 5)
{
   AngleArc(DC, 200+i, 130+i, 80, 30, 190);
}
```

۱۷-۳-۲- توابع API رشته‌ای (String API)

در این فصل توابع API ای که برای کارکردن با رشته‌ها طراحی شده‌اند بررسی خواهند شد. رشته‌هایی که توابع API از آنها استفاده می‌کنند رشته‌های ختم به Null می‌باشند. در دلفی این رشته‌ها با نام Pchar شناخته می‌شوند. رشته‌های معمولی (string) را می‌توانیم با استفاده از تابع Pchar به رشته‌های ختم به Null تعریف کنیم، به عنوان مثال Pcahr(s) رشته‌ی s را به رشته‌ی ختم به Null تبدیل خواهد کرد.

۱-۱۷-۳-۲- تابع IsCharAlpha

این تابع که در فایل user32.dll قرار دارد یک کاراکتر را به عنوان پارامتر دریافت کرده و تشخیص می‌دهد که این کاراکتر جزو حروف الفبایی می‌باشد یا نه. این تابع را هم می‌توان برای حروف زبان انگلیسی و هم برای حروف زبان فارسی استفاده کرد. شکل تعریف این تابع در C# به صورت زیر می‌باشد:

```
[DllImport("user32.dll")]
public static extern Boolean IsCharAlpha(char m);
```

۲-۱۷-۳-۲- تابع IsCharAlphaNumeric

این تابع که در فایل user32.dll قرار دارد یک کاراکتر را به عنوان پارامتر دریافت کرده و تشخیص می‌دهد که این کاراکتر جزو حروف الفبایی و یا ارقام می‌باشد یا نه. شکل تعریف این تابع در C# به صورت زیر می‌باشد:

```
[DllImport("user32.dll")]
public static extern Boolean IsCharAlphaNumeric(char m);
```

۳-۱۷-۳-۲- تابع IsCharUpper و IsCharLower

این توابع که در فایل user32.dll قرار دارند جهت تشخیص حروف کوچک و بزرگ زبان انگلیسی استفاده می‌شوند. شکل کلی توابع به صورت زیر می‌باشد:

```
[DllImport("user32.dll")]
public static extern Boolean IsCharUpper(char m);

[DllImport("user32.dll")]
public static extern Boolean IsCharLower(char m);
```

۴-۱۷-۳-۲- تابع Lstrcmp

این تابع که در فایل kernel32.dll قرار دارد دو رشته را به عنوان پارامتر دریافت کرده آنها را با یکدیگر مقایسه می‌کند در صورتی که رشته‌ی اول از رشته‌ی دوم کوچکتر باشد منفی ۱ ، برابر باشد . و در صورتی که رشته‌ی اول از رشته دوم بزرگتر باشد ۱ را برگشت می‌دهد. شکل کلی این تابع به صورت زیر می‌باشد:

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
static extern int lstrcmp(string lpString1, string lpString2);
```

نمونه مثال زیر کاربردی از این تابع را نشان می‌دهد:

```
int i = lstrcmp(textbox2.Text, textbox3.Text);
label6.Text = Convert.Tostring(i) +"-";
switch(i)
{
       case -1;
         label6.Text = label6.Text +"s1 is Less than s2";
         break;
       case 0:
         label6.Text = label6.Text +"s1 is Equal to s2";
         break;
       case 1:
         label6.Text = label6.Text +"s1 is Greater than s2";
         break;
}
```

۵-۱۷-۳-۲- تابع Lstrlen

این تابع که در فایل kernel32.dll قرار دارد یک رشته را به عنوان پارامتر دریافت کرده و طول رشته را به عنوان نتیجه برگشت می‌دهد. شکل کلی این تابع به صورت زیر می‌باشد:

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
static extern int lstrlen(string lpString);
```

ضمیمه A: مجموعه دستورات پردازنده های رده x86 شرکت اینتل

Instruction	Encoding (bin)[b]
adc reg32, reg32	0110 0110 0001 00x1 [11-reg-r/m]
adc reg8, mem8	0001 0010 [mod-reg-r/m]
adc reg16, mem16	0001 0011 [mod-reg-r/m]
adc reg32, mem32	0110 0110 0001 0011 [mod-reg-r/m]
adc mem8, reg8	0001 0000 [mod-reg-r/m]
adc mem16, reg16	0001 0001 [mod-reg-r/m]
adc mem32, reg32	0110 0110 0001 0001 [mod-reg-r/m]
adc reg8, imm8	1000 00x0 [11-010-r/m] [imm]
adc reg16, imm16	1000 00s0 [11-010-r/m] [imm]
adc reg32, imm32	0110 0110 1000 00s0 [11-010-r/m] [imm]
adc mem8, imm8	1000 00x0 [mod-010-r/m] [imm]
adc mem16, imm16	1000 00s1 [mod-010-r/m] [imm]
adc mem32, imm32	0110 0110 1000 00s1 [mod-010-r/m] [imm]
adc al, imm	0001 0100 [imm]
adc ax, imm	0001 0101 [imm]
adc eax, imm	0110 0110 0001 0101 [imm]
add reg8, reg8	0000 00x0 [11-reg-r/m]
add reg16, reg16	0000 00x1 [11-reg-r/m]
add reg32, reg32	0110 0110 0000 00x1 [11-reg-r/m]

Instruction	Encoding (bin)[b]
add reg8, mem8	0000 0010 [mod-reg-r/m]
add reg16, mem16	0000 0011 [mod-reg-r/m]
add reg32, mem32	0110 0110 0000 0011 [mod-reg-r/m]
add mem8, reg8	0000 0000 [mod-reg-r/m]
add mem16, reg16	0000 0001 [mod-reg-r/m]
add mem32, reg32	0110 0110 0000 0001 [mod-reg-r/m]
add reg8, imm8	1000 00x0 [11-000-r/m] [imm]
add reg16, imm16	1000 00s0 [11-000-r/m] [imm]
add reg32, imm32	0110 0110 1000 00s0 [11-000-r/m] [imm]
add mem8, imm8	1000 00x0 [mod-000-r/m] [imm]
add mem16, imm16	1000 00s1 [mod-000-r/m] [imm]
add mem32, imm32	0110 0110 1000 00s1 [mod-000-r/m] [imm]
add al, imm	0000 0100 [imm]
add ax, imm	0000 0101 [imm]
add eax, imm	0110 0110 0000 0101 [imm]
and reg8, reg8	0010 00x0 [11-reg-r/m]
and reg16, reg16	0010 00x1 [11-reg-r/m]
and reg32, reg32	0110 0110 0010 00x1 [11-reg-/rm]
and reg8, mem8	0010 0010 [mod-reg-r/m]

Instruction	Encoding (bin)[b]
and reg16, mem16	0010 0011 [mod-reg-r/m]
and reg32, mem32	0110 0110 0010 0011 [mod-reg-r/m]
and mem8, reg8	0010 0000 [mod-reg-r/m]
and mem16, reg16	0010 0001 [mod-reg-r/m]
and mem32, reg32	0110 0110 0010 0001 [mod-reg-r/m]
and reg8, imm8	1000 00x0 [11-100-r/m] [imm]
and reg16, imm16	1000 00s1 [11-100-r/m] [imm]
and reg32, imm32	0110 0110 1000 00s1 [11-100-r/m] [imm]
and mem8, imm8	1000 00x0 [mod-100-r/m] [imm]
and mem16, imm16	1000 00s1 [mod-100-r/m] [imm]
and mem32, imm32	0110 0110 1000 00s1 [mod-100-r/m] [imm]
and al, imm	0010 0100 [imm]
and ax, imm	0010 0101 [imm]
and eax, imm	0110 0110 0010 0101 [imm]
bound reg16, mem32	0110 0010 [mod-reg-r/m]
bound reg32, mem64	0110 0110 0110 0010 [mod-reg-r/m]
bsf reg16, reg16	0000 1111 1011 1100 [11-reg-r/m]

Instruction	Encoding (bin)[b]
bsf reg32, reg32	0110 0110 0000 1111 1011 1100 [11-reg-r/m]
bsf reg16, mem16	0000 1111 1011 1100 [mod-reg-r/m]
bsf reg32, mem32	0110 0110 0000 1111 1011 1100 [mod-reg-r/m]
bsr reg16, reg16	0000 1111 1011 1101 [11-reg-r/m]
bsr reg32, reg32	0110 0110 0000 1111 1011 1101 [11-reg-r/m]
bsr reg16, mem16	0000 1111 1011 1101 [mod-reg-r/m]
bsr reg32, mem32	0110 0110 0000 1111 1011 1101 [mod-reg-r/m]
bswap reg32	0000 1111 11001rrr
bt reg16, reg16	0000 1111 1010 0011 [11-reg-r/m]
bt reg32, reg32	0110 0110 0000 1111 1010 0011 [11-reg-r/m]
bt mem16, reg16	0000 1111 1010 0011 [mod-reg-r/m]
bt mem32, reg32	0110 0110 0000 1111 1010 0011 [mod-reg-r/m]
bt reg16, imm	0000 1111 1011 1010 [11-100-r/m] [imm8]
bt reg32, imm	0110 0110 0000 1111 1011 1010 [11-100-r/m] [imm8]
bt mem16, imm	0000 1111 1011 1010 [mod-100-r/m]

Instruction	Encoding (bin)[b]
bt mem32, imm	0110 0110 0000 1111 1011 1010 [mod-100-r/m]
btc reg16, reg16	0000 1111 1011 1011 [11-reg-r/m]
btc reg32, reg32	0110 0110 0000 1111 1011 1011 [11-reg-r/m]
btc mem16, reg16	0000 1111 1011 1011 [mod-reg-r/m]
btc mem32, reg32	0110 0110 0000 1111 1011 1011 [mod-reg-r/m]
btc reg16, imm	0000 1111 1011 1010 [11-111-r/m] [imm8]
btc reg32, imm	0110 0110 0000 1111 1011 1010 [11-111-r/m] [imm8]
btc mem16, imm	0000 1111 1011 1010 [mod-111-r/m] [imm8]
btc mem32, imm	0110 0110 0000 1111 1011 1010 [mod-111-r/m] [imm8]
btr reg16, reg16	0000 1111 1011 0011 [11-reg-r/m]
btr reg32, reg32	0110 0110 0000 1111 1011 0011 [11-reg-r/m]
btr mem16, reg16	0000 1111 1011 0011 [mod-reg-r/m]
btr mem32, reg32	0110 0110 0000 1111 1011 0011 [mod-reg-r/m]

Instruction	Encoding (bin)[b]
btr reg16, imm	0000 1111 1011 1010 [11-110-r/m] [imm8]
btr reg32, imm	0110 0110 0000 1111 1011 1010 [11-110-r/m] [imm8]
btr mem16, imm	0000 1111 1011 1010 [mod-110-r/m] [imm8]
btr mem32, imm	0110 0110 0000 1111 1011 1010 [mod-110-r/m] [imm8]
bts reg16, reg16	0000 1111 1010 1011 [11-reg-r/m]
bts reg32, reg32	0110 0110 0000 1111 1010 1011 [11-reg-r/m]
bts mem16, reg16	0000 1111 1010 1011 [mod-reg-r/m]
bts mem32, reg32	0110 0110 0000 1111 1010 1011 [mod-reg-r/m]
bts reg16, imm	0000 1111 1011 1010 [11-101-r/m] [imm8]
bts reg32, imm	0110 0110 0000 1111 1011 1010 [11-101-r/m] [imm8]
bts mem16, imm	0000 1111 1011 1010 [mod-101-r/m] [imm8]
bts mem32, imm	0110 0110 0000 1111 1011 1010 [mod-101-r/m] [imm8]
call near	1110 1000 [disp16]

Instruction	Encoding (bin)[b]
call far	1001 1010 [offset] [segment]
call reg16	1111 1111 [11-010-r/m]
call mem16	1111 1111 [mod-010-r/m]
call mem32	1111 1111 [mod-011-r/m]
cbw	1001 1000
cdq	0110 0110 1001 1001
clc	1111 1000
cld	1111 1100
cli	1111 1010
cmc	1111 0101
cmp reg8, reg8	0011 10x0 [11-reg-r/m]
cmp reg16, reg16	0011 10x1 [11-reg-r/m]
cmp reg32, reg32	0110 0110 0011 10x1 [11-reg-/rm]
cmp reg8, mem8	0011 1010 [mod-reg-r/m]
cmp reg16, mem16	0011 1011 [mod-reg-r/m]
cmp reg32, mem32	0110 0110 0011 1011 [mod-reg-r/m]
cmp mem8, reg8	0011 1000 [mod-reg-r/m]
cmp mem16, reg16	0011 1001 [mod-reg-r/m]
cmp mem32, reg32	0110 0110 0011 1001 [mod-reg-r/m]
cmp reg8, imm8	1000 00x0 [11-111-r/m] [imm]
cmp reg16, imm16	1000 00s0 [11-111-r/m] [imm]
cmp reg32, imm32	0110 0110 1000 00s0 [11-111-r/m] [imm]
cmp mem8, imm8	1000 00x0 [mod-111-r/m] [imm]

Instruction	Encoding (bin)[b]
cmp mem16, imm16	1000 00s1 [mod-111-r/m] [imm]
cmp mem32, imm32	0110 0110 1000 00s1 [mod-111-r/m] [imm]
cmp al, imm	0011 1100 [imm]
cmp ax, imm	0011 1101 [imm]
cmp eax, imm	0110 0110 0011 1101 [imm]
cmpsb	1010 0110
cmpsw	1010 0111
cmpsd	0110 0110 1010 0111
repe cmpsb	1111 0011 1010 0110
repne cmpsb	1111 0010 1010 0110
repe cmpsw	1111 0011 1010 0111
repne cmpsw	1111 0010 1010 0111
repe cmpsd	0110 0110 1111 0011 1010 0111
repne cmpsd	0110 0110 1111 0010 1010 0111
cmpxchg reg8, reg8	0000 1111 1011 0000 [11-reg-r/m] Note: r/m is first register operand.
cmpxchg reg16, reg16	0000 1111 1011 0001 [11-reg-r/m]
cmpxchg reg32, reg32	0110 0110 0000 1111 1011 0001 [11-reg-r/m]
cmpxchg mem8, reg8	0000 1111 1011 0000 [mod-reg-r/m]

Instruction	Encoding (bin)[b]
cmpxchg mem16, reg16	0000 1111 1011 0001 [mod-reg-r/m]
cmpxchg mem32, reg32	0110 0110 0000 1111 1011 0001 [mod-reg-r/m]
cmpxchg8b mem64	0000 1111 1100 0111 [mod-001-r/m]
cpuid	0000 1111 1010 0010
cwd	1001 1001
cwde	0110 0110 1001 1000
daa	0010 0111
das	0010 1111
dec reg8	1111 1110 [11-001-r/m]
dec reg16	0100 1rrr
dec reg16 (alternate encoding)	1111 1111 [11-001-r/m]
dec reg32	0110 0110 0100 1rrr
dec reg32 (alternate encoding)	0110 0110 1111 1111 [11-001-r/m]
dec mem8	1111 1110 [mod-001-r/m]
dec mem16	1111 1111 [mod-001-r/m]
dec mem32	0110 0110 1111 1111 [mod-001-r/m]
div reg8	1111 0110 [11-110-r/m]
div reg16	1111 0111 [11-110-r/m]
div reg32	0110 0110 1111 0111 [11-110-r/m]
div mem8	1111 0110 [mod-110-r/m]
div mem16	1111 0111 [mod-110-r/m]
div mem32	0110 0110 1111 0111 [mod-110-r/m]

Instruction	Encoding (bin)[b]
enter local, 0	1100 1000 [locals-imm16] 0000 0000
enter local, 1	1100 1000 [locals-imm16] 0000 0001
enter local, lex	1100 1000 [locals:imm16] [lex:imm8]
hlt	1111 0100
idiv reg8	1111 0110 [11-111-r/m]
idiv reg16	1111 0111 [11-111-r/m]
idiv reg32	0110 0110 1111 0111 [11-111-r/m]
idiv mem8	1111 0110 [mod-111-r/m]
idiv mem16	1111 0111 [mod-111-r/m] [disp]
idiv mem32	0110 0110 1111 0111 [mod-111-r/m]
imul reg8	1111 0110 [11-101-r/m]
imul reg16	1111 0111 [11-101-r/m]
imul reg32	0110 0110 1111 0111 [11-101-r/m]
imul mem8	1111 0110 [mod-101-r/m]
imul mem16	1111 0111 [mod-101-r/m]
imul mem32	0110 0110 1111 0111 [mod-101-r/m]
imul reg16, reg16, imm8 imul reg16, imm8 (Second form assumes reg and r/m are the same, instruction sign extends eight bit immediate operand to 16 bits)	0110 1011 [11-reg-r/m] [imm8] (1st reg operand is specified by reg field, 2nd reg operand is specified by r/m field)
imul reg16, reg16, imm imul reg16, imm	0110 1001 [11-reg-r/m] [imm16]
imul reg32, reg32, imm8 imul reg32, imm8	0110 0110 0110 1011 [11-reg-r/m] [imm8]
imul reg32, reg32, imm imul reg32, imm	0110 0110 0110 1001 [11-reg-r/m] [imm32]
imul reg16,mem16,imm8	0110 1011 [11-reg-r/m] [imm8]
imul reg16,mem16,imm	0110 1001 [11-reg-r/m] [imm16]
imul reg32, mem32, imm8	0110 0110 0110 1011 [11-reg-r/m] [imm8]
imul reg32, mem32, imm	0110 0110 0110 1001 [11-reg-r/m] [imm32]
imul reg16, reg16	0000 1111 1010 1111 [11-reg-r/m] (reg is dest operand)
imul reg32, reg32	0110 0110 0000 1111 1010 1111 [11-reg-r/m] (reg is dest operand)
imul reg16, mem16	0000 1111 1010 1111 [mod-reg-r/m]
imul reg32, mem32	0110 0110 0000 1111 1010 1111 [mod-reg-r/m]
in al, port	1110 0100 [port8]
in ax, port	1110 0101 [port8]
in eax, port	0110 0110 1110 0101 [port8]
in al, dx	1110 1100
in ax, dx	1110 1101
in eax, dx	0110 0110 1110 1101
inc reg8	1111 1110 [11-000-r/m]
inc reg16	0100 0rrr
inc reg16 (alternate encoding)	1111 1111 [11-000-r/m]
inc reg32	0110 0110 0100 0rrr
inc reg32 (alternate encoding)	0110 0110 1111 1111 [11-000-r/m]
inc mem8	1111 1110 [mod-000-r/m]
inc mem16	1111 1110 [mod-000-r/m] [disp]
inc mem32	0110 0110 1111 1110 [mod-000-r/m]
insb	1010 1010
insw	1010 1011
insd	0110 0110 1010 1011
rep insb	1111 0010 1010 1010
rep insw	1111 0010 1010 1011
rep insd	0110 0110 1111 0010 1010 1011
int nn	1100 1101 [imm8]
int 03	1100 1100
into	1100 1110
iret	1100 1111
iretd	0110 0110 1100 1111
ja short	0111 0111 [disp8]
ja near	0000 1111 1000 0111 [disp16]
jae short	0111 0011 [disp8]
jae near	0000 1111 1000 0011 [disp16]
jb short	0111 0010 [disp8]

Instruction	Encoding (bin)[b]
jb near	0000 1111 1000 0010 [disp16]
jbe short	0111 0110 [disp8]
jbe near	0000 1111 1000 0110 [disp16]
jc short	0111 0010 [disp8]
jc near	0000 1111 1000 0010 [disp16]
je short	0111 0100 [disp8]
je near	0000 1111 1000 0100 [disp16]
jg short	0111 1111 [disp8]
jg near	0000 1111 1000 1111 [disp16]
jge short	0111 1101 [disp8]
jge near	0000 1111 1000 1101 [disp16]
jl short	0111 1100 [disp8]
jl near	0000 1111 1000 1100 [disp16]
jle short	0111 1110 [disp8]
jle near	0000 1111 1000 1110 [disp16]
jna short	0111 0110 [disp8]
jna near	0000 1111 1000 0110 [disp16]
jnae short	0111 0010 [disp8]
jnae near	0000 1111 1000 0010 [disp16]
jnb short	0111 0011 [disp8]

Instruction	Encoding (bin)[b]
jnb near	0000 1111 1000 0011 [disp16]
jnbe short	0111 0111 [disp8]
jnbe near	0000 1111 1000 0111 [disp16]
jnc short	0111 0011 [disp8]
jnc near	0000 1111 1000 0011 [disp16]
jne short	0111 0101 [disp8]
jne near	0000 1111 1000 0101 [disp16]
jng short	0111 1110 [disp8]
jng near	0000 1111 1000 1110 [disp16]
jnge short	0111 1100 [disp8]
jnge near	0000 1111 1000 1100 [disp16]
jnl short	0111 1101 [disp8]
jnl near	0000 1111 1000 1101 [disp16]
jnle short	0111 1111 [disp8]
jnle near	0000 1111 1000 1111 [disp16]
jno short	0111 0001 [disp8]
jno near	0000 1111 1000 0001 [disp16]
jnp short	0111 1011 [disp8]
jnp near	0000 1111 1000 1011 [disp16]
jns short	0111 1001 [disp8]

Instruction	Encoding (bin)[b]
jns near	0000 1111 1000 1001 [disp16]
jnz short	0111 0101 [disp8]
jnz near	0000 1111 1000 0101 [disp16]
jo short	0111 0000 [disp8]
jo near	0000 1111 1000 0000 [disp16]
jp short	0111 1010 [disp8]
jp near	0000 1111 1000 1010 [disp16]
jpe short	0111 1010 [disp8]
jpe near	0000 1111 1000 1010 [disp16]
jpo short	0111 1011 [disp8]
jpo near	0000 1111 1000 1011 [disp16]
js short	0111 1000 [disp8]
js near	0000 1111 1000 1000 [disp16]
jz short	0111 0100 [disp8]
jz near	0000 1111 1000 0100 [disp16]
jcxz short	1110 0011 [disp8]
jecxz short	0110 0110 1110 0011 [disp8]
jmp short	1110 1011 [disp8]
jmp near	1110 1001 [disp16]
jmp reg16	1111 1111 [11-100-r/m]

Instruction	Encoding (bin)[b]
jmp mem16	1111 1111 [mod-100-r/m]
jmp far	1110 1010 [offset16] [segment16]
jmp mem32	1111 1111 [mod-101-r/m]
lahf	1001 1111
lds reg, mem32	1100 0101 [mod-reg-r/m]
lea reg, mem	1000 1101 [mod-101-r/m]
leave	1100 1001
les reg, mem32	1100 0100 [mod-reg-r/m]
lfs reg, mem32	0000 1111 1011 0100 [mod-reg-r/m]
lgs reg, mem32	0000 1111 1011 0101 [mod-reg-r/m]
lodsb	1010 1100
lodsw	1010 1101
loadsd	0110 0110 1010 1101
loop short	1110 0010 [disp8]
loope short loopz short	1110 0001 [disp8]
loopne short loopnz short	1110 0000 [disp8]
lss reg, mem32	0000 1111 1011 0010 [mod-reg-r/m]
mov reg8, reg8	1000 1000 [11-reg-r/m] (r/m specifies destination reg)
mov reg8, reg8 (alternate encoding)	1000 1010 [11-reg-r/m] (reg specifies destination reg)
mov reg16, reg16	1000 1001 [11-reg-r/m] (r/m specifies destination reg)
mov reg16, reg16 (alternate encoding)	1000 1011 [11-reg-r/m] (reg specifies destination reg)
mov reg32, reg32	0110 0110 1000 1001 [11-reg-r/m] (r/m specifies destination reg)
mov reg32, reg32 (alternate encoding)	0110 0110 1000 1011 [11-reg-r/m] (reg specifies destination reg)
mov mem, reg8	1000 1000 [mod-reg-r/m]
mov reg8, mem	1000 1010 [mod-reg-r/m]
mov mem, reg16	1000 1001 [mod-reg-r/m]
mov reg16, mem	1000 1011 [mod-reg-r/m]
mov mem, reg32	0110 0110 1000 1001 [mod-reg-r/m]
mov reg16, mem	0110 0110 1000 1011 [mod-reg-r/m]
mov reg8, imm	1011 0rrr [imm8]
mov reg8, imm (alternate encoding)	1100 0110 [11-000-r/m] [imm8]
mov reg16, imm	1011 1rrr [imm16]
mov reg16, imm (alternate encoding)	1100 0111 [11-000-r/m] [imm16]
mov reg32, imm	0110 0110 1011 1rrr [imm32]
mov reg32, imm (alternate encoding)	0110 0110 1100 0111 [11-000-r/m] [imm32]
mov mem8, imm	1100 0110 [mod-000-r/m] [imm8]
mov mem16, imm	1100 0111 [mod-000-r/m] [imm16]
mov mem32, imm	1100 0111 [mod-000-r/m] [imm32]
mov al, disp	1010 0000 [disp]
mov ax, disp	1010 0001 [disp]
mov eax, disp	0110 0110 1010 0001 [disp]
mov disp, al	1010 0010 [disp]
mov disp, ax	1010 0011 [disp]
mov disp, eax	0110 0110 1010 0011 [disp]
mov segreg, reg16	1000 1110 [11-sreg-r/m]
mov segreg, mem	1000 1110 [mod-reg-r/m]
mov reg16, segreg	1000 1100 [11-sreg-r/m]
mov mem, segreg	1000 1100 [mod-reg-r/m]
movsb	1010 0100
movsw	1010 0101
movsd	0110 0110 1010 0101
rep movsb	1111 0010 1010 0100
rep movsw	1111 0010 1010 0101
rep movsd	0110 0110 1111 0010 1010 0101
movsx reg16, reg8	0000 1111 1011 1110 [11-reg-r/m] (dest is reg operand)
movsx reg32, reg8	0110 0110 0000 1111 1011 1110 [11-reg-r/m]
movsx reg32, reg16	0110 0110 0000 1111 1011 1111 [11-reg-r/m]
movsx reg16, mem8	0000 1111 1011 1110 [mod-reg-r/m]

Instruction	Encoding (bin)[b]
movsx reg32, mem8	0110 0110 0000 1111 1011 1110 [mod-reg-r/m]
movsx reg32, mem16	0110 0110 0000 1111 1011 1111 [mod-reg-r/m]
movzx reg16, reg8	0000 1111 1011 0110 [11-reg-r/m] (dest is reg operand)
movzx reg32, reg8	0110 0110 0000 1111 1011 0110 [11-reg-r/m]
movzx reg32, reg16	0110 0110 0000 1111 1011 0111 [11-reg-r/m]
movzx reg16, mem8	0000 1111 1011 0110 [mod-reg-r/m]
movzx reg32, mem8	0110 0110 0000 1111 1011 0110 [mod-reg-r/m]
movzx reg32, mem16	0110 0110 0000 1111 1011 0111 [mod-reg-r/m]
mul reg8	1111 0110 [11-100-r/m]
mul reg16	1111 0111 [11-100-r/m]
mul reg32	0110 0110 1111 0111 [11-100-r/m]
mul mem8	1111 0110 [mod-100-r/m]
mul mem16	1111 0111 [mod-100-r/m]
mul mem32	0110 0110 1111 0111 [mod-100-r/m]
neg reg8	1111 0110 [11-011-r/m]
neg reg16	1111 0111 [11-011-r/m]

Instruction	Encoding (bin)[b]
neg reg32	0110 0110 1111 0111 [11-011-r/m]
neg mem8	1111 0110 [mod-011-r/m]
neg mem16	1111 0111 [mod-011-r/m]
neg mem32	0110 0110 1111 0111 [mod-011-r/m]
nop (same as xchg ax, ax)	1001 0000
not reg8	1111 0110 [11-010-r/m]
not reg16	1111 0111 [11-010-r/m]
not reg32	0110 0110 1111 0111 [11-010-r/m]
not mem8	1111 0110 [mod-010-r/m]
not mem16	1111 0111 [mod-010-r/m]
not mem32	0110 0110 1111 0111 [mod-010-r/m]
or reg8, reg8	0000 10x0 [11-reg-r/m]
or reg16, reg16	0000 10x1 [11-reg-r/m]
or reg32, reg32	0110 0110 0000 10x1 [11-reg-r/m]
or reg8, mem8	0000 1010 [mod-reg-r/m]
or reg16, mem16	0000 1011 [mod-reg-r/m]
or reg32, mem32	0110 0110 0000 1011 [mod-reg-r/m]
or mem8, reg8	0000 1000 [mod-reg-r/m]
or mem16, reg16	0000 1001 [mod-reg-r/m]
or mem32, reg32	0110 0110 0000 1001 [mod-reg-r/m]
or reg8, imm8	1000 00x0 [11-001-r/m] [imm]

Instruction	Encoding (bin)[b]
or reg16, imm16	1000 00s0 [11-001-r/m] [imm]
or reg32, imm32	0110 0110 1000 00s0 [11-001-r/m] [imm]
or mem8, imm8	1000 00x0 [mod-001-r/m] [imm]
or mem16, imm16	1000 00s1 [mod-001-r/m] [imm]
or mem32, imm32	0110 0110 1000 00s1 [mod-001-r/m] [imm]
or al, imm	0000 1100 [imm]
or ax, imm	0000 10101 [imm]
or eax, imm	0110 0110 0000 1101 [imm]
out port, al	1110 0110 [port8]
out port, ax	1110 0111 [port8]
out port, eax	0110 0110 1110 0111 [port8]
out dx, al	1110 1110
out dx, ax	1110 1111
out dx, eax	0110 0110 1110 1111
outsb	1010 1010
outsw	1010 1011
outsd	0110 0110 1010 1011
rep outsb	1111 0010 1010 1010
rep outsw	1111 0010 1010 1011
rep outsd	0110 0110 1111 0010 1010 1011
pop reg16	0101 1rrr
pop reg16 (alternate encoding)	1000 1111 [11-000-r/m]

Instruction	Encoding (bin)[b]
pop reg32	0110 0110 0101 1rrr
pop reg32 (alternate encoding)	0110 0110 1000 1111 [11-000-r/m]
pop mem16	1000 1111 [mod-000-r/m]
pop mem32	1000 1111 [mod-000-r/m]
pop es	0000 0111
pop ss	0001 0111
pop ds	0001 1111
pop fs	0000 1111 1010 0001
pop gs	0000 1111 1010 1001
popa	0110 0001
popad	0110 0110 0110 0001
popf	1001 1101
popfd	0110 0110 1001 1101
push reg16	0101 0rrr
push reg16 (alternate encoding)	1111 1111 [11-110-r/m]
push reg32	0110 0110 0101 0rrr
push reg32 (alternate encoding)	0110 0110 1111 1111 [11-110-r/m]
push mem16	1111 1111 [mod-110-r/m]
push mem32	1111 1111 [mod-110-r/m]
push cs	0000 1110
push ds	0001 1110
push es	0000 0110
push ss	0001 0110
push fs	0000 1111 1010 0000
push gs	0000 1111 1010 1000
push imm8->16	0110 1000 [imm8] (sign extends value to 16 bits)
push imm16	0110 1010 [imm16]

Instruction	Encoding (bin)[b]
push imm32	0110 0110 0110 1010 [imm32]
pusha	0110 0000
pushad	0110 0110 0110 0000
pushf	1001 1100
pushfd	0110 0110 1001 1100
rcl reg8, 1	1101 0000 [11-010-r/m]
rcl reg16, 1	1101 0001 [11-010-r/m]
rcl reg32, 1	0110 0110 1101 0001 [11-010-r/m]
rcl mem8, 1	1101 0000 [mod-010-r/m]
rcl mem16, 1	1101 0001 [mod-010-r/m]
rcl mem32, 1	0110 0110 1101 0001 [mod-010-r/m]
rcl reg8, cl	1101 0010 [11-010-r/m]
rcl reg16, cl	1101 0011 [11-010-r/m]
rcl reg32, cl	0110 0110 1101 0011 [11-010-r/m]
rcl mem8, cl	1101 0010 [mod-010-r/m]
rcl mem16, cl	1101 0011 [mod-010-r/m]
rcl mem32, cl	0110 0110 1101 0011 [mod-010-r/m]
rcl reg8, imm8	1100 0000 [11-010-r/m] [imm8]
rcl reg16, imm8	1100 0001 [11-010-r/m] [imm8]
rcl reg32, imm8	0110 0110 1100 0001 [11-010-r/m] [imm8]
rcl mem8, imm8	1100 0000 [mod-010-r/m] [imm8]

Instruction	Encoding (bin)[b]
rcl mem16, imm8	1100 0001 [mod-010-r/m] [imm8]
rcl mem32, imm8	0110 0110 1100 0001 [mod-010-r/m] [imm8]
rcr reg8, 1	1101 0000 [11-011-r/m]
rcr reg16, 1	1101 0001 [11-011-r/m]
rcr reg32, 1	0110 0110 1101 0001 [11-011-r/m]
rcr mem8, 1	1101 0000 [mod-011-r/m]
rcr mem16, 1	1101 0001 [mod-011-r/m]
rcr mem32, 1	0110 0110 1101 0001 [mod-011-r/m]
rcr reg8, cl	1101 0010 [11-011-r/m]
rcr reg16, cl	1101 0011 [11-011-r/m]
rcr reg32, cl	0110 0110 1101 0011 [11-011-r/m]
rcr mem8, cl	1101 0010 [mod-011-r/m]
rcr mem16, cl	1101 0011 [mod-011-r/m]
rcr mem32, cl	0110 0110 1101 0011 [mod-011-r/m]
rcr reg8, imm8	1100 0000 [11-011-r/m] [imm8]
rcr reg16, imm8	1100 0001 [11-011-r/m] [imm8]
rcr reg32, imm8	0110 0110 1100 0001 [11-011-r/m] [imm8]
rcr mem8, imm8	1100 0000 [mod-011-r/m] [imm8]
rcr mem16, imm8	1100 0001 [mod-011-r/m] [imm8]

Instruction	Encoding (bin)[b]
rcr mem32, imm8	0110 0110 1100 0001 [mod-011-r/m] [imm8]
ret retn	1100 0011
ret imm16 retn imm16	1100 0010 [imm16]
ret retf	1100 1011
ret imm16 retf imm16	1100 1010 [imm16]
rol reg8, 1	1101 0000 [11-000-r/m]
rol reg16, 1	1101 0001 [11-000-r/m]
rol reg32, 1	0110 0110 1101 0001 [11-000-r/m]
rol mem8, 1	1101 0000 [mod-000-r/m]
rol mem16, 1	1101 0001 [mod-000-r/m]
rol mem32, 1	0110 0110 1101 0001 [mod-000-r/m]
rol reg8, cl	1101 0010 [11-000-r/m]
rol reg16, cl	1101 0011 [11-000-r/m]
rol reg32, cl	0110 0110 1101 0011 [11-000-r/m]
rol mem8, cl	1101 0010 [mod-000-r/m]
rol mem16, cl	1101 0011 [mod-000-r/m]
rol mem32, cl	0110 0110 1101 0011 [mod-000-r/m]
rol reg8, imm8	1100 0000 [11-000-r/m] [imm8]
rol reg16, imm8	1100 0001 [11-000-r/m] [imm8]
rol reg32, imm8	0110 0110 1100 0001 [11-000-r/m] [imm8]

Instruction	Encoding (bin)[b]
rol mem8, imm8	1100 0000 [mod-000-r/m] [imm8]
rol mem16, imm8	1100 0001 [mod-000-r/m] [imm8]
rol mem32, imm8	0110 0110 1100 0001 [mod-000-r/m] [imm8]
ror reg8, 1	1101 0000 [11-001-r/m]
ror reg16, 1	1101 0001 [11-001-r/m]
ror reg32, 1	0110 0110 1101 0001 [11-001-r/m]
ror mem8, 1	1101 0000 [mod-001-r/m]
ror mem16, 1	1101 0001 [mod-001-r/m]
ror mem32, 1	0110 0110 1101 0001 [mod-001-r/m]
ror reg8, cl	1101 0010 [11-001-r/m]
ror reg16, cl	1101 0011 [11-001-r/m]
ror reg32, cl	0110 0110 1101 0011 [11-001-r/m]
ror mem8, cl	1101 0010 [mod-001-r/m]
ror mem16, cl	1101 0011 [mod-001-r/m]
ror mem32, cl	0110 0110 1101 0011 [mod-001-r/m]
ror reg8, imm8	1100 0000 [11-001-r/m] [imm8]
ror reg16, imm8	1100 0001 [11-001-r/m] [imm8]
ror reg32, imm8	0110 0110 1100 0001 [11-001-r/m] [imm8]
ror mem8, imm8	1100 0000 [mod-001-r/m] [imm8]

Instruction	Encoding (bin)[b]
ror mem16, imm8	1100 0001 [mod-001-r/m] [imm8]
ror mem32, imm8	0110 0110 1100 0001 [mod-001-r/m] [imm8]
sahf	1001 1110
sal reg8, 1 (Same instruction as shl)	1101 0000 [11-100-r/m]
sal reg16, 1	1101 0001 [11-100-r/m]
sal reg32, 1	0110 0110 1101 0001 [11-100-r/m]
sal mem8, 1	1101 0000 [mod-100-r/m]
sal mem16, 1	1101 0001 [mod-100-r/m]
sal mem32, 1	0110 0110 1101 0001 [mod-100-r/m]
sal reg8, cl	1101 0010 [11-100-r/m]
sal reg16, cl	1101 0011 [11-100-r/m]
sal reg32, cl	0110 0110 1101 0011 [11-100-r/m]
sal mem8, cl	1101 0010 [mod-100-r/m]
sal mem16, cl	1101 0011 [mod-100-r/m]
sal mem32, cl	0110 0110 1101 0011 [mod-100-r/m]
sal reg8, imm8	1100 0000 [11-100-r/m] [imm8]
sal reg16, imm8	1100 0001 [11-100-r/m] [imm8]
sal reg32, imm8	0110 0110 1100 0001 [11-100-r/m] [imm8]
sal mem8, imm8	1100 0000 [mod-100-r/m] [imm8]

Instruction	Encoding (bin)[b]
sal mem16, imm8	1100 0001 [mod-100-r/m] [imm8]
sal mem32, imm8	0110 0110 1100 0001 [mod-100-r/m] [imm8]
sar reg8, 1	1101 0000 [11-111-r/m]
sar reg16, 1	1101 0001 [11-111-r/m]
sar reg32, 1	0110 0110 1101 0001 [11-111-r/m]
sar mem8, 1	1101 0000 [mod-111-r/m]
sar mem16, 1	1101 0001 [mod-111-r/m]
sar mem32, 1	0110 0110 1101 0001 [mod-111-r/m]
sar reg8, cl	1101 0010 [11-111-r/m]
sar reg16, cl	1101 0011 [11-111-r/m]
sar reg32, cl	0110 0110 1101 0011 [11-111-r/m]
sar mem8, cl	1101 0010 [mod-111-r/m]
sar mem16, cl	1101 0011 [mod-111-r/m]
sar mem32, cl	0110 0110 1101 0011 [mod-111-r/m]
sar reg8, imm8	1100 0000 [11-111-r/m] [imm8]
sar reg16, imm8	1100 0001 [11-111-r/m] [imm8]
sar reg32, imm8	0110 0110 1100 0001 [11-111-r/m] [imm8]
sar mem8, imm8	1100 0000 [mod-111-r/m] [imm8]

Instruction	Encoding (bin)[b]
sar mem16, imm8	1100 0001 [mod-111-r/m] [imm8]
sar mem32, imm8	0110 0110 1100 0001 [mod-111-r/m] [imm8]
sbb reg8, reg8	0001 10x0 [11-reg-r/m]
sbb reg16, reg16	0001 10x1 [11-reg-r/m]
sbb reg32, reg32	0110 0110 0001 10x1 [11-reg-r/m]
sbb reg8, mem8	0001 1010 [mod-reg-r/m]
sbb reg16, mem16	0001 1011 [mod-reg-r/m]
sbb reg32, mem32	0110 0110 0001 1011 [mod-reg-r/m]
sbb mem8, reg8	0001 1000 [mod-reg-r/m]
sbb mem16, reg16	0001 1001 [mod-reg-r/m]
sbb mem32, reg32	0110 0110 0001 1001 [mod-reg-r/m]
sbb reg8, imm8	1000 00x0 [11-011-r/m] [imm]
sbb reg16, imm16	1000 00s1 [11-011-r/m] [imm]
sbb reg32, imm32	0110 0110 1000 00s1 [11-011-r/m] [imm]
sbb mem8, imm8	1000 00x0 [mod-011-r/m] [imm]
sbb mem16, imm16	1000 00s1 [mod-011-r/m] [imm]
sbb mem32, imm32	0110 0110 1000 00s1 [mod-011-r/m] [imm]
sbb al, imm	0001 1100 [imm]

Instruction	Encoding (bin)[b]
sbb ax, imm	0001 1101 [imm]
sbb eax, imm	0110 0110 0001 1101 [imm]
scasb	1010 0100
scasw	1010 0101
scasd	0110 0110 1010 0101
rep scasb	1111 0010 1010 0100
rep scasw	1111 0010 1010 0101
rep scasd	0110 0110 1111 0010 1010 0101
seta reg8	0000 1111 1001 0111 [11-000-r/m][e]
seta mem8	0000 1111 1001 0011 [mod-000-r/m]
setae reg8	0000 1111 1001 0011 [11-000-r/m]
setae mem8	0000 1111 1001 0011 [mod-000-r/m]
setb reg8	0000 1111 1001 0010 [11-000-r/m]
setb mem8	0000 1111 1001 0010 [mod-000-r/m]
setbe reg8	0000 1111 1001 0110 [11-000-r/m]
setbe mem8	0000 1111 1001 0110 [mod-000-r/m]
setc reg8	0000 1111 1001 0010 [11-000-r/m]
setc mem8	0000 1111 1001 0010 [mod-000-r/m]
sete reg8	0000 1111 1001 0100 [11-000-r/m]

Instruction	Encoding (bin)[b]
sete mem8	0000 1111 1001 0100 [mod-000-r/m]
setg reg8	0000 1111 1001 1111 [11-000-r/m]
setg mem8	0000 1111 1001 1111 [mod-000-r/m]
setge reg8	0000 1111 1001 1101 [11-000-r/m]
setge mem8	0000 1111 1001 1101 [mod-000-r/m]
setl reg8	0000 1111 1001 1100 [11-000-r/m]
setl mem8	0000 1111 1001 1100 [mod-000-r/m]
setle reg8	0000 1111 1001 1110 [11-000-r/m]
setle mem8	0000 1111 1001 1110 [mod-000-r/m]
setna reg8	0000 1111 1001 0110 [11-000-r/m]
setna mem8	0000 1111 1001 0110 [mod-000-r/m]
setnae reg8	0000 1111 1001 0010 [11-000-r/m]
setnae mem8	0000 1111 1001 0010 [mod-000-r/m]
setnb reg8	0000 1111 1001 0011 [11-000-r/m]
setnb mem8	0000 1111 1001 0011 [mod-000-r/m]
setnbe reg8	0000 1111 1001 0111 [11-000-r/m]
setnbe mem8	0000 1111 1001 0111 [mod-000-r/m]
setnc reg8	0000 1111 1001 0011 [11-000-r/m]
setnc mem8	0000 1111 1001 0011 [mod-000-r/m]
setne reg8	0000 1111 1001 0101 [11-000-r/m]
setne mem8	0000 1111 1001 0101 [mod-000-r/m]
setng reg8	0000 1111 1001 1110 [11-000-r/m]
setng mem8	0000 1111 1001 1110 [mod-000-r/m]
setnge reg8	0000 1111 1001 1100 [11-000-r/m]
setnge mem8	0000 1111 1001 1100 [mod-000-r/m]
setnl reg8	0000 1111 1001 1101 [11-000-r/m]
setnl mem8	0000 1111 1001 1101 [mod-000-r/m]
setnle reg8	0000 1111 1001 1111 [11-000-r/m]
setnle mem8	0000 1111 1001 1111 [mod-000-r/m]
setno reg8	0000 1111 1001 0001 [11-000-r/m]
setno mem8	0000 1111 1001 0001 [mod-000-r/m]
setnp reg8	0000 1111 1001 1011 [11-000-r/m]
setnp mem8	0000 1111 1001 1011 [mod-000-r/m]
setns reg8	0000 1111 1001 1001 [11-000-r/m]
setns mem8	0000 1111 1001 1001 [mod-000-r/m]
setnz reg8	0000 1111 1001 0101 [11-000-r/m]
setnz mem8	0000 1111 1001 0101 [mod-000-r/m]
seto reg8	0000 1111 1001 0000 [11-000-r/m]
seto mem8	0000 1111 1001 0000 [mod-000-r/m]
setp reg8	0000 1111 1001 1010 [11-000-r/m]
setp mem8	0000 1111 1001 1010 [mod-000-r/m]
setpe reg8	0000 1111 1001 1010 [11-000-r/m]
setpe mem8	0000 1111 1001 1010 [mod-000-r/m]
setpo reg8	0000 1111 1001 1011 [11-000-r/m]
setpo mem8	0000 1111 1001 1011 [mod-000-r/m]
sets reg8	0000 1111 1001 1000 [11-000-r/m]
sets mem8	0000 1111 1001 1000 [mod-000-r/m]
setz reg8	0000 1111 1001 0100 [11-000-r/m]
setz mem8	0000 1111 1001 0100 [mod-000-r/m]
shl reg8, 1	1101 0000 [11-100-r/m]
shl reg16, 1	1101 0001 [11-100-r/m]
shl reg32, 1	0110 0110 1101 0001 [11-100-r/m]

Instruction	Encoding (bin)[b]
shl mem8, 1	1101 0000 [mod-100-r/m]
shl mem16, 1	1101 0001 [mod-100-r/m]
shl mem32, 1	0110 0110 1101 0001 [mod-100-r/m]
shl reg8, cl	1101 0010 [11-100-r/m]
shl reg16, cl	1101 0011 [11-100-r/m]
shl reg32, cl	0110 0110 1101 0011 [11-100-r/m]
shl mem8, cl	1101 0010 [mod-100-r/m]
shl mem16, cl	1101 0011 [mod-100-r/m]
shl mem32, cl	0110 0110 1101 0011 [mod-100-r/m]
shl reg8, imm8	1100 0000 [11-100-r/m] [imm8]
shl reg16, imm8	1100 0001 [11-100-r/m] [imm8]
shl reg32, imm8	0110 0110 1100 0001 [11-100-r/m] [imm8]
shl mem8, imm8	1100 0000 [mod-100-r/m] [imm8]
shl mem16, imm8	1100 0001 [mod-100-r/m] [imm8]
shl mem32, imm8	0110 0110 1100 0001 [mod-100-r/m] [imm8]
shld reg16, reg16, imm8 r/m is 1st operand, reg is second operand.	0000 1111 1010 0100 [11-reg-r/m] [imm8]
shld reg32, reg32, imm8 r/m is 1st operand, reg is second operand.	0110 0110 0000 1111 1010 0100 [11-reg-r/m] [imm8]

Instruction	Encoding (bin)[b]
shld mem16, reg16, imm8	0000 1111 1010 0100 [mod-reg-r/m] [imm8]
shld mem32, reg32, imm8	0110 0110 0000 1111 1010 0100 [mod-reg-r/m] [imm8]
shld reg16, reg16, cl r/m is 1st operand, reg is second operand.	0000 1111 1010 0101 [11-reg-r/m]
shld reg32, reg32, cl r/m is 1st operand, reg is second operand.	0110 0110 0000 1111 1010 0101 [11-reg-r/m]
shld mem16, reg16, cl	0000 1111 1010 0101 [mod-reg-r/m]
shld mem32, reg32, cl	0110 0110 0000 1111 1010 0101 [mod-reg-r/m]
shr reg8, 1	1101 0000 [11-101-r/m]
shr reg16, 1	1101 0001 [11-101-r/m]
shr reg32, 1	0110 0110 1101 0001 [11-101-r/m]
shr mem8, 1	1101 0000 [mod-101-r/m]
shr mem16, 1	1101 0001 [mod-101-r/m]
shr mem32, 1	0110 0110 1101 0001 [mod-101-r/m]
shr reg8, cl	1101 0010 [11-101-r/m]
shr reg16, cl	1101 0011 [11-101-r/m]
shr reg32, cl	0110 0110 1101 0011 [11-101-r/m]
shr mem8, cl	1101 0010 [mod-101-r/m]
shr mem16, cl	1101 0011 [mod-101-r/m]
shr mem32, cl	0110 0110 1101 0011 [mod-101-r/m]

Instruction	Encoding (bin)[b]
shr reg8, imm8	1100 0000 [11-101-r/m] [imm8]
shr reg16, imm8	1100 0001 [11-101-r/m] [imm8]
shr reg32, imm8	0110 0110 1100 0001 [11-101-r/m] [imm8]
shr mem8, imm8	1100 0000 [mod-101-r/m] [imm8]
shr mem16, imm8	1100 0001 [mod-101-r/m] [imm8]
shr mem32, imm8	0110 0110 1100 0001 [mod-101-r/m] [imm8]
shrd reg16, reg16, imm8 r/m is 1st operand, reg is second operand.	0000 1111 1010 1100 [11-reg-r/m] [imm8]
shrd reg32, reg32, imm8 r/m is 1st operand, reg is second operand.	0110 0110 0000 1111 1010 1100 [11-reg-r/m] [imm8]
shrd mem16, reg16, imm8	0000 1111 1010 1100 [mod-reg-r/m] [imm8]
shrd mem32, reg32, imm8	0110 0110 0000 1111 1010 1100 [mod-reg-r/m] [imm8]
shrd reg16, reg16, cl r/m is 1st operand, reg is second operand.	0000 1111 1010 1101 [11-reg-r/m]
shrd reg32, reg32, cl r/m is 1st operand, reg is second operand.	0110 0110 0000 1111 1010 1101 [11-reg-r/m]
shrd mem16, reg16, cl	0000 1111 1010 1101 [disp]
shld mem32, reg32, cl	0110 0110 0000 1111 1010 1101 [mod-reg-r/m]

Instruction	Encoding (bin)[b]
stc	1111 1001
std	1111 1101
sti	1111 1011
stosb	1010 1010
stosw	1010 1011
stosd	0110 0110 1010 1011
rep stosb	1111 0010 1010 1010
rep stosw	1111 0010 1010 1011
rep stosd	0110 0110 1111 0010 1010 1011
sub reg8, reg8	0010 10x0 [11-reg-r/m]
sub reg16, reg16	0010 10x1 [11-reg-r/m]
sub reg32, reg32	0110 0110 0010 10x1 [11-reg-r/m]
sub reg8, mem8	0010 1010 [mod-reg-r/m]
sub reg16, mem16	0010 1011 [mod-reg-r/m]
sub reg32, mem32	0110 0110 0010 1011 [mod-reg-r/m]
sub mem8, reg8	0010 1000 [mod-reg-r/m]
sub mem16, reg16	0010 1001 [mod-reg-r/m]
sub mem32, reg32	0110 0110 0010 1001 [mod-reg-r/m]
sub reg8, imm8	1000 00x0 [11-101-r/m] [imm]
sub reg16, imm16	1000 00s1 [11-101-r/m] [imm]
sub reg32, imm32	0110 0110 1000 00s1 [11-101-r/m] [imm]
sub mem8, imm8	1000 00x0 [mod-101-r/m] [imm]
sub mem16, imm16	1000 00s1 [mod-101-r/m] [imm]
sub mem32, imm32	0110 0110 1000 00s1 [mod-101-r/m] [imm]
sub al, imm	0010 1100 [imm]
sub ax, imm	0010 1101 [imm]
sub eax, imm	0110 0110 0010 1101 [imm]
test reg8, reg8	1000 0100 [11-reg-r/m]
test reg16, reg16	1000 0101 [11-reg-r/m]
test reg32, reg32	0110 0110 1000 0101 [11-reg-r/m]
test reg8, mem8	1000 0110 [mod-reg-r/m]
test reg16, mem16	1000 0111 [mod-reg-r/m]
test reg32, mem32	0110 0110 1000 0111 [mod-reg-r/m]
test reg8, imm8	1111 0110 [11-000-r/m] [imm]
test reg16, imm16	1111 0111 [11-000-r/m] [imm]
test reg32, imm32	0110 0110 1111 0111 [11-000-r/m] [imm]
test mem8, imm8	1111 0110 [mod-000-r/m] [imm]
test mem16, imm16	1111 0111 [mod-000-r/m] [imm]
test mem32, imm32	0110 0110 1111 0111 [mod-000-r/m] [imm]
test al, imm	1010 1000 [imm]
test ax, imm	1010 1001 [imm]
test eax, imm	0110 0110 1010 1001 [imm]
xadd reg8, reg8 r/m is first operand, reg is second operand.	0000 1111 1100 0000 [11-reg-r/m]
xadd reg16, reg16	0000 1111 1100 0001 [11-reg-r/m]
xadd reg32, reg32	0110 0110 0000 1111 1100 0001 [11-reg-r/m]
xadd mem8, reg8	0000 1111 1100 0000 [mod-reg-r/m]
xadd mem16, reg16	0000 1111 1100 0001 [mod-reg-r/m]
xadd mem32, reg32	0110 0110 0000 1111 1100 0001 [mod-reg-r/m]
xchg reg8, reg8	1000 0110 [11-reg-r/m]
xchg reg16, reg16	1000 0111 [11-reg-r/m]
xchg reg32, reg32	0110 0110 1000 0111 [11-reg-r/m]
xchg mem8, reg8[f]	1000 0110 [11-reg-r/m]
xchg mem16, reg16	1000 0111 [11-reg-r/m]
xchg mem32, reg32	0110 0110 1000 0111 [11-reg-r/m]
xchg ax, reg16	1001 0rrr
xchg ax, reg32	0110 0110 1001 0rrr
xlat	1101 0111
xor reg8, reg8	0011 00x0 [11-reg-r/m]
xor reg16, reg16	0011 00x1 [11-reg-r/m]

Instruction	Encoding (bin)[b]
xor reg32, reg32	0110 0110 0011 00x1 [11-reg-r/m]
xor reg8, mem8	0011 0010 [mod-reg-r/m]
xor reg16, mem16	0011 0011 [mod-reg-r/m]
xor reg32, mem32	0110 0110 0011 0011 [mod-reg-r/m]
xor mem8, reg8	0011 0000 [mod-reg-r/m]
xor mem16, reg16	0011 0001 [mod-reg-r/m]
xor mem32, reg32	0110 0110 0011 0001 [mod-reg-r/m]
xor reg8, imm8	1000 00x0 [11-110-r/m] [imm]
xor reg16, imm16	1000 00s1 [11-110-r/m] [imm]
xor reg32, imm32	0110 0110 1000 00s1 [11-110-r/m] [imm]
xor mem8, imm8	1000 00x0 [mod-110-r/m] [imm]
xor mem16, imm16	1000 00s1 [mod-110-r/m] [imm]
xor mem32, imm32	0110 0110 1000 00s1 [mod-110-r/m] [imm]
xor al, imm	0011 0100 [imm]
xor ax, imm	0011 0101 [imm]
xor eax, imm	0110 0110 0011 0101 [imm]

Instruction	Encoding (bin)[b]
aaa	0011 0111
aad	1101 0101 0000 1010
aam	1101 0100 0000 1010
aas	0011 1111
adc reg8, reg8	0001 00x0 [11-reg-r/m]
adc reg16, reg16	0001 00x1 [11-reg-r/m]

Key to special bits in encodings:

x: Don't care. Can be zero or one.

s: Sign extension bit for immediate operands. If zero, immediate operand is 16 or 32 bits depending on destination operand size. If s bit is one, then the immediate operand is eight bits and the CPU sign extends to 16 or 32 bits, as appropriate.

rrr: Same as reg field in [mod-reg-r/m] byte.

Other Notes:

[disp] This field can be zero, one, two, or four bytes long as required by the instruction.

[imm] This field is one byte long if the operand is an eight bit operand or if the *s* bit in the instruction opcode is one. It is two or four bytes long if the *s* bit contains zero and the destination operand is 16 or 32 bits, respectively.

[mod-reg-r/m]: Instructions that have a mod-reg-r/m byte may have a scaled index byte (sib) and a zero, one, two, or four byte displacement. See Appendix E for details concerning the encoding of this portion of the instruction.

reg,reg Many instructions allow two operands using a [mod-reg-r/m] byte. A single *direction* bit in the opcode determines whether the instruction treats the *reg* operand as the destination or the mod-r/m operand as the destination (e.g., mov reg,mem vs. mov mem,reg). Such instructions also allow two register operands. It turns out there are two encodings for each such reg-reg instruction. That is, you can encode an instruction like mov ax, bx with ax encoded in the reg field and bx encoded in the mod-r/m field, or you can encode it with bx encoded in the reg field and ax encoded in the mod-r/m field. Such instructions always have an *x* bit in the opcode. If the *x* bit is zero, the destination is the register specified by the mod-r/m field. If the *x* bit is one, the destination is the register specified by the reg field. Other types of instructions support multiple encodings for similar reasons.