شکیبافر، محسن
اصول کامل راه‌اندازی و کنترل دستگاههای جانبی توسط کامپیوتر/ محسن شکیبافر
تهران: نص، ۱۳۸۴.
۴۳۲ ص. مصور،جدول + یک دیسک فشرده.
ISBN: 964-410-058-1 CD ۴۵۰۰۰ ریال با
فهرستنویسی بر اساس فیپا.
۱. کامپیوترها-- راهنمای آموزشی .الف. عنوان.
۶الف۸۲ ش/QA۷۶/۲۷ ۰۰۴/۰۷
۱۳۸۴
کتابخانه ملی ایران م۸۴-۱۷۳۰۰

موسسه علمی فرهنگی

اصول کامل
راه‌اندازی و کنترل دستگاههای جانبی
توسط کامپیوتر

مولف: محسن شکیبافر
ویراستار: نفیسه صابری
چاپ اول : زمستان ۸۴
ناشر: «نص»
چاپ و صحافی: سازمان چاپ و انتشارات وزارت فرهنگ و ارشاد اسلامی
طراحی، آماده سازی: موسسه علمی فرهنگی «نص» قیمت با CD: ۴۵۰۰ تومان

تهران: میدان انقلاب، خیابان اردیبهشت، بن بست مبین، شماره ۲۳۷
تلفکس: ۶۶۴۱۲۳۸۵ تلفن: ۶۶۹۵۳۸۸۳-۶۶۴۶۵۶۷۴ ص.پ. ۸۶۳-۱۳۱۴۵

شابک: ۱-۰۵۸-۴۱۰-۹۶۴ ISBN:964-410-058-1

مقدمه

علل نگارش کتاب

امروزه فناوری‌های جدید به سمت کامپیوتری کردن سیستمها ، کارگاهها ، ادارات و کارخانه‌ها پیش می‌رود. با توجه به این مطلب و همچنین عدم تسلط کافی دانشجویان و فارغ‌التحصیلان رشته‌های برق به برنامه‌نویسی و عدم تسلط کافی دانشجویان و فارغ‌التحصیلان کامپیوتر به طراحی مدارهای واسط ، تصمیم گرفتیم که کتابی را که اکنون پیش روی شماست، تالیف کنیم.

مثالهای این کتاب برای برآورده ساختن نیازهای دانش‌آموزان ، افراد شاغل در صنایع اوتوماسیون (خودکارسازی) و همچنین دانشجویان و فارغ‌التحصیلان برق و کامپیوتر به چندین زبان برنامه‌نویسی تحت DOS و ویندوز نوشته شده و با ارایه مثال و توضیحات در زمینه برنامه‌نویسی و طراحی سخت‌افزار کمک بسیاری به هر دو گرایش کامپیوتر و برق می‌کند.

چرا از کامپیوتر براي اعمال کنترلي استفاده مي‌شود؟

اخیراً در کنترل دستگاهها، به علت کاهش خطاهای انسانی از مدیریت مستقیم انسان کمتر استفاده می‌شود. در این سیستم‌ها عواملی مانند خستگی، افسردگی، بی‌تجربگی و یا کهولت سن، تاثیر کمتری بر روی نتایج فرآیندها خواهند داشت.

در این سیستمها به جای انسان، از یک کنترلر دیگری استفاده می‌شود، که معمولاً شامل یکی از دسته‌های زیر می‌باشند.

- میکروکنترلرهای سری 8051
- میکروکنترلرهای PIC
- میکروکنترلرهای AVR

- **سیستمهای برنامه‌پذیر منطقی (PLC)**
- **میکروپروسسورها**
- **بوردهای کنترلی مانند MPF1**
- **کامپیوتر**

از میان موارد مذکور، استفاده از کامپیوتر به عنوان کنترلر، برتری‌هایی را به سیستم اضافه می‌کند، که از آن جمله به موارد زیر می‌توان اشاره کرد.

۱. توانایی استفاده از هوش مصنوعی در کامپیوتر
می‌توان پروژه مورد نظر را به یک سیستم خبره یا پردازشگر تصویر یا صوت تبدیل و یا به شبکه‌های عصبی و سیستم یادگیری ماشین نیز مجهز کرد.

۲. توانایی استفاده از شبکه‌های کامپیوتری و اینترنت
با استفاده از امکانات شبکه می‌توان دستگاه‌ها را از کیلومترها دورتر کنترل کرد و یا از عملکرد کارکنان و وضعیت دستگاه‌ها، مطلع شد.

۳. توانایی گزارش‌گیری قوی
تمام کسانی که در صنعت مشغول کار هستند، از اهمیت گزارش و گزارش‌گیری مطلع هستند. بوسیله کامپیوتر می‌توان گزارش دقیق و کاملی را از هر قسمت، به صورت اتوماتیک تهیه کرد. مزیت مهمی که سیستم گزارش‌گیری دارد ، کاهش هزینه‌های نگهداری است.

٤. ظاهر قابل فهم و ساده
یکی از مهمترین ویژگیهای کنترل کننده‌های کامپیوتری، ظاهر ساده برنامه کنترل کننده است. در چنین سیستمی کاربر، نیاز به گذراندن دوره‌های آموزشی پیچیده ندارد، و تمام عناصر مورد نظر دستگاه را با ظاهر گرافیکی، می‌تواند بر روی صفحه نمایش مشاهده کند.

۵. توانایی ذخیره‌سازی و بازیابی اطلاعات در بانک اطلاعاتی
بانکهای اطلاعاتی یکی دیگر از نقاط قوت، استفاده از کامپیوترها هستند. از ویژگی‌های منحصر به فرد بانکهای اطلاعاتی می‌توان به بازیابی و جستجوی بسیار سریع و آسان اطلاعات اشاره کرد.

۶. توانایی در تنظیم ساده پارامترهای دستگاه، متناسب با نیاز، توسط کاربر
در صورت پیش‌بینی این موضوع در نرم‌افزار ، می‌توان این کار را به راحتی با تنظیم پنجره‌های تنظیم (Option) انجام داد.

۷. توانایی ارزیابی اطلاعات سیستم، در قالب نمودارهای گرافیکی
این کار کمک بزرگی در رفع خطا ، عیب‌یابی دستگاه مورد نظر و یا بر طرف کردن شرایط مزاحم می‌کند.

۸. توانایی دریافت و پردازش اطلاعات بسیار پیچیده.
۹. سرعت بسیار بالاتر در پردازش اطلاعات.
۱۰. توانایی ساخت سیستم چند کاربری با چند سطح دسترسی مختلف
سطوح دسترسی‌های مختلف به امنیت سیستم بسیار کمک می‌کند. در این سیستم هر کاربر فقط به اطلاعاتی که از قبل توسط مدیر سیستم مشخص شده ، دسترسی دارد و یا به عبارت بهتر، فقط امکان ویرایش، حذف و افزودن قسمتی از اطلاعات را داراست.

با توجه به نکات فوق قطعاً انتخاب کامپیوتر به عنوان مغز متفکر و کنترلر، سیستم را بهینه خواهد کرد.

مکانیزه کردن و اتوماسیون هر دستگاهی نیاز به دانستن نکات مکانیکی هم دارد، که مورد بحث این کتاب نیست. در این کتاب بیشتر به نکات نرم‌افزاری و سپس به نکات دیجیتالی و الکترونیکی پرداخته خواهد شد. در ضمن تا حد ممکن از ارایه نکات کم استفاده و پیچیده پرهیز شده است.

این کتاب براي چه کساني است؟

این کتاب در سه سطح قابل استفاده است.

۱. مبتدی دانش‌آموزان و یا تمام کسانی که تحصیلات آنها به کامپیوتر و الکترونیک ارتباطی ندارد. فصول مناسب برای این خوانندگان عبارت است از :
- فصل اول (تمام فصل).
- فصل دوم (تمام بخش‌ها به‌جز بخشهای پنجم و ششم).
- فصل سوم (تمام بخش‌ها به‌جز بخشهای هفتم و هشتم).
- فصل چهارم (تمام بخش‌ها به‌جز بخشهای هشتم و سوم).
- فصل هفتم (تمام فصل).
- فصل هشتم (تمام بخش‌ها بجز بخش هفتم).

۲. متوسط دانشجویانی که در ترم‌های پایین برق و کامپیوتر مشغول به تحصیل هستند و یا خوانندگانی که اطلاعات اولیه و یا متوسطی راجع به برق دارند و علاوه بر آن به یکی از زبانهای برنامه‌نویسی تسلط دارند.
- فصل اول (تمام فصل).
- فصل دوم (تمام فصل).
- فصل سوم (تمام فصل).
- فصل چهارم (تمام فصل).
- فصل هفتم (تمام فصل).
- فصل هشتم (تمام فصل).

- فصل نهم (تمام فصل).

۳. پیشرفته دانشجویان و فارغ التحصیلان رشته‌های برق و کامپیوتر و همچنین خوانندگانی که در صنایع اتوماسیون و کنترلی کار می‌کنند. این خوانندگان از تمام فصول این کتاب می‌توانند استفاده کنند.

در ضمن تمام خوانندگان بهتر است، قبل از مطالعه این کتاب به یکی از زبانهای برنامه‌نویسی تسلط نسبی داشته باشند. داشتن معلومات اولیه در مورد الکترونیک و دیجیتال نیز کمک بسیار زیادی به درک و استفاده از مطالب این کتاب خواهد کرد.

علایم استفاده شده در کتاب

در این کتاب، علایم زیر و معانی آن را در نظر داشته باشید.

 : نکته !

 : اخطار و احتیاط !

 : مثال و یا نکته‌ای در مورد مثال.

 : برنامه و یا مثال، که در CD جنبی موجود است.

ساختار کتاب

این کتاب در ۱۰ فصل، مطالب مربوط به کنترل دستگاههای جانبی را آموزش می‌دهد. دو فصل اول با ارایه یادآوری‌ها و مقدمات سخت‌افزاری و نرم‌افزاری، ذهن خواننده را با اصول مورد نیاز آشنا می‌کند. سه فصل بعدی، طرز کار سه پورت معروف سری، موازی و USB را نشان می‌دهند. این سه فصل شامل نکات سخت‌افزاری و نرم‌افزاری به همراه برنامه‌هایی با زبانهای مختلف برنامه‌نویسی برای استفاده پورتها هستند.

فصل ششم ، تمام اسلاتهای پرکاربرد و نحوه ساخت کارتهای توسعه متناسب با این اسلاتها را معرفی می‌کند. این فصل با ارایه تمام مراحل ساخت، خواننده را با نکات سخت‌افزاری و نرم‌افزاری اسلاتها آشنا می‌کند.

چهار فصل آخر کتاب به ارایه نکات پیشرفته برنامه‌نویسی و طراحی سخت‌افزار مدارهای جانبی می‌پردازد. فصل هفتم نحوه برنامه‌نویسی سخت‌افزاری و سیستمی را در محیط ویندوز و در زبانهای برنامه‌نویسی گوناگون نشان می‌دهد. فصل هشتم یکی از مهمترین فصلهای این کتاب است که نکات طراحی، ساخت و کنترل یک دستگاه جانبی را نشان می‌دهد. فصل نهم با ارایه مقدمه‌ای بر کنترل دستگاههای جانبی از راه دور، به نحوه کنترل از طریق اینترنت می‌پردازد. فصل آخر به روش کنترل دستگاههای جانبی توسط سیستم بلوتوث پرداخته است.

اغلب برنامه‌های کتاب، با هماهنگی سخت‌افزار جانبی کار می‌کنند و به همین علت از اجرای این گونه برنامه‌ها، بدون اتصال با سخت‌افزار پرهیز کنید!
این برنامه‌ها در پاسخ به دستگاه جانبی کار می‌کنند و در صورت عدم وجود آن، برنامه معلق می‌ماند.

در پایان برای تکمیل مباحث کتاب پیوستهای زیر آمده‌اند.

پیوست الف: ساخت فایلهای DLL که نحوه ساخت این فایلها را در Visual C++ نشان می‌دهد.

پیوست ب: ویروسهای بلوتوث که مطالبی در زمینه مقابله و جلوگیری از نفوذ این نوع ویروسها ارایه می‌کند.

پیوست ج: وقفه های بایوس که مرجعی برای استفاده از این وقفه ها به خواننده ارایه شده است.

پیوست د: این پیوست جواب سوالهای آخر فصل را در بر می‌گیرد. همچنین راهنمایی‌هایی مفید برای اجرای پروژه‌های هر فصل دارد.

پیوست **ه**: این پیوست واژه نامه کتاب است.

پیوست و: توابع API لازم در این پیوست هستند. برای مشاهده این پیوست، CD جنبی را ببینید.

پیوست ز: کدهای اسکی و کدهای صفحه کلید در این پیوست آورده شده‌اند. برای مشاهده این پیوست، CD جنبی را ببینید.

CD جنبي كتاب

CD جنبی کتاب حاوی برنامه‌های سودمند، کاتالوگ تراشه‌های استفاده شده در کتاب، مثالهای مفید و مقالات با ارزشی در این زمینه است، که به صورت Autorun اجرا می شود.
برای آشنایی با چگونگی استفاده از CD روی دکمه help کلیک کنید.

سپاسگزاري

در آخر لازم است که از خانم مهندس نفیسه صابری (پژوهشگر مرکز تحقیقات مخابرات ایران)، آقای مهندس احسان ملکیان (عضو هیات علمی دانشگاه تربیت معلم)، آقای مهندس محمدحسین زارع (مدیر انتشارات نص)، آقای دکتر نصرت‌اله علی‌قنبری (عضو هیات علمی دانشگاه صنعتی شیراز)، آقای مهندس محمد مهندسی (عضو هیات علمی دانشگاه شیراز)، آقای مهندس هادی امین‌زاده، آقای علی رمضانی (مسئول فنی انتشارات نص) و تمام دانشجویان فهیم دانشگاه شیراز و دانشگاه صنعتی شیراز تشکر کنم، زیرا بدون همکاری این عزیزان، اتمام کار کتاب میسر نبود.

در ضمن از خوانندگان تقاضا می‌شود، که نظرات و پیشنهادات خود را به آدرس پست الکترونیک مؤلف ارسال کنند.

محسن شکیبافر
m_shakibafar@engineer.com

فهرست مطالب

فصل ۳

فصل ۴

فصل ۵

پیوستها

فصل اول :
اصول نرم‌افزاری

برای ساخت سیستمهایی که توسط کامپیوتر کنترل می‌شوند (PC Base)، داشتن اطلاعات کلی و پایه‌ای در مورد برنامه‌نویسی سیستم لازم است. این اطلاعات شامل نحوه کار با بیتها ، بایتها ، وقفه‌ها ، زبان ماشین ، بافرها و .. می‌باشند. در این فصل، نکات فوق به صورت جامع و به اختصار بررسی خواهند شد.

بخش اول : برنامه‌نویسی سیستم

هدف برنامه‌نویس از نوشتن یک برنامه ، مدیریت و پردازش اطلاعات و یا کنترل سخت‌افزار و دستگاههای جانبی است.

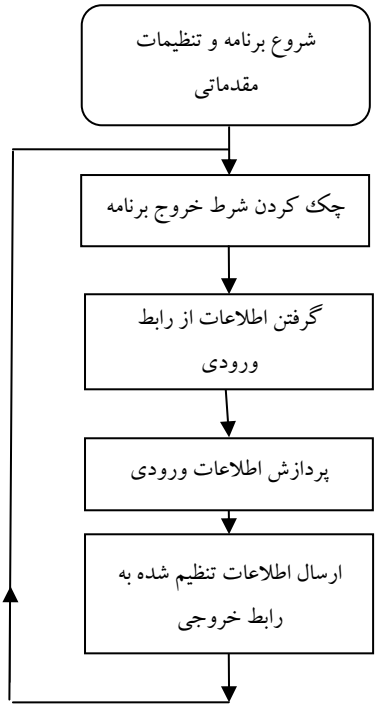
به دسته اول، برنامه‌نویسی کاربردی (Application Program) گفته می‌شود، که متناسب با نیاز خاصی نوشته شده و با امکانات مشخص، در هر کامپیوتری می‌تواند مورد استفاده قرار گیرد.

به دسته دوم برنامه‌نویسی سیستم (System Programming) گفته می‌شود. هدف اصلی در این نوع برنامه‌نویسی، انتقال اطلاعات بین کامپیوتر و سخت‌افزار و یا مدیریت یک دستگاه جانبی (Peripheral) است. مناسب‌ترین وسیله برای اجرای این برنامه‌ها ، کامپیوترهای صنعتی (Industrial Computers) هستند، اما با توجه به پیشرفت روز افزون کامپیوترهای شخصی (Personal Computers) ، این برنامه‌ها را در کامپیوترهای معمولی هم می‌توان اجرا و مدیریت کرد.

ساختار یک برنامه سیستم

همانطور که در شکل ۱-۱ مشخص شده، یک برنامه سیستم از قسمتهای زیر تشکیل شده است.

شکل ۱-۱
ساختار یک برنامه سیستم



۱. **تنظیمات اولیه (Initializations) برای ارتباط با دستگاه جانبی به یک درگاه (Port) و رابط (Interface) نیاز داریم. در بسیاری مواقع این رابطها و حتی خود دستگاه نیاز به ارسال اطلاعاتی برای تنظیمات اولیه دارند. این تنظیمات شامل مواردی مانند سرعت انتقال اطلاعات ، ورودی یا خروجی بودن اطلاعات، خطایابی و .. می‌باشند.**
۲. **چک کردن شرط خروج برنامه نکته بسیار مهم، در برنامه‌نویسی سیستم این است که، برنامه نوشته شده باید در یک حلقه، دائماً در حال اجرا باشد. برای این حلقه حتماً یک شرط خروج بایستی در نظر گرفته شود. بازه زمانی تکرار این حلقه از چند میلی ثانیه تا چند دقیقه متغیر است.**
۳. **دریافت اطلاعات ورودی به وسیله دستورات خاصی که در هر زبان برنامه‌نویسی موجود است، اطلاعات از رابط ورودی (رابط ورودی نیز به دستگاه جانبی متصل است) خوانده شود.**
٤. **پردازش اطلاعات ورودی نرم‌افزار اطلاعاتی را که از ورودی گرفته است، پردازش کرده و تصمیماتش را به صورت پارامترهایی آماده می‌کند.**
۵. **ارسال اطلاعات به رابط خروجی در این قسمت، نرم‌افزار باید پارامترهای تعیین شده را به خروجی ارسال کند.**

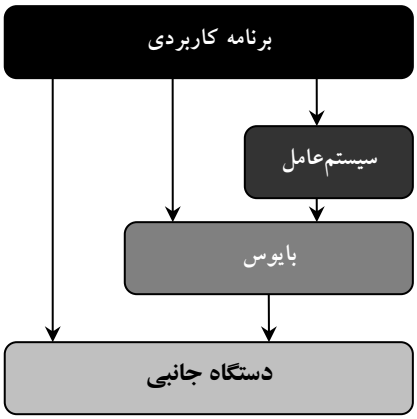
در صورتی که مدت زمان تکرار حلقه، نسبت به تغییرات سیستم متناسب باشد، می‌توان گفت که برنامه در زمان حقیقی کار می‌کند و یا اصطلاحاً بلادرنگ (Real Time) است.

لایه‌هاي يک برنامه سيستم

همانطور که در شکل ۱-۲ نشان داده شده، برنامه سیستم دارای سه لایه (Layer) است. با توجه به این شکل، برنامه سیستم، خود نوعی از یک برنامه کاربردی است. یکی از مزایای بزرگ ساختار سه لایه‌ای برنامه‌های سیستم، ارتباط غیرمستقیم با سخت‌افزار است. این موضوع باعث استقلال برنامه از نوع کامپیوتر می‌شود. زیرا هماهنگی‌ها بین رابطهای خروجی و برنامه‌ها در هر کامپیوتری، توسط بایوس (Bios) و سیستم‌عامل (Operating System) آن انجام می‌گیرد. یکی از بخش‌های مهم دیگر، برای برنامه سیستم ، انتخاب زبان برنامه‌نویسی است که در ادامه به آن می‌پردازیم.

بخش دوم: زبانهاي برنامه‌نويسي

زبانهای برنامه‌نویسی که معمولاً برای برنامه‌نویسی سیستم مورد استفاده قرار می‌گیرند، به بخشهای زیر تقسیم می‌شوند.



شکل ۱-۲
لایه‌های برنامه سیستم

۱. **تحت DOS**

- خانواده **Basic** : QBasic.
- خانواده **Pascal**: Borland Turbo Pascal.
- خانواده **C** : Turbo C++ و Quick C و Borland C++.
- خانواده **Assembly**: Macro Assembly و Turbo Assembly.
- سایر زبانها: مانند Fortran 77 ، Fortran 90 ، Forth و COBOL که کاربردی در برنامه‌نویسی سیستم ندارند.

۲. **تحت WINDOWS**

- **خانواده Basic**: Visual Basic.
- **خانواده Pascal**: Borland Delphi.
- **خانواده C** : Visual C++ و Borland C++ Builder.
- **خانواده #C** : Visual C# و C# Builder.
- **نرم‌افزارهای دیگر: نرم‌افزارهای تخصصی بسیاری، قابلیت اتصال به سخت‌افزار و مدیریت دستگاههای جانبی را دارند. از مهمترین این نرم‌افزارها می‌توان به** Matlab **و** Labview **اشاره کرد، که قدرت بسیار زیادی برای پردازش و کنترل اطلاعات پیچیده دارند. این نکته را باید در نظر داشت، که نرم‌افزارهای مهم دیگری مانند** AutoCAD **و حتی** Microsoft Access **نیز قابلیت ارسال اطلاعات به پورتها را دارند.**

۳. زبانهاي برنامه‌نويسي ميكروكنترلرها

- **زبانهای میکروکنترلرهای سری 8051** : C51 , PASCAL51 , BASIC51. **از مهمترین کامپایلرهای این زبانها می‌توان به** Keil **و** Franklin **اشاره کرد.**
- **زبانهای سری PIC.**
- **زبانهای سری AVR.**

زبانهای برنامه‌نویسی دسته سوم، با توجه به شباهت بسیار زیاد آنها به زبانهای پرکاربردی مانند Basic **و** C**، دارای مترجمها و شبیه‌سازهایی** (Simulator) **در کامپیوتر هستند. با این حال برای اجرای آنها، باید برنامه مربوطه، در میکروکنترلر مخصوص خود قرار داده شود و اصطلاحاً باید میکروکنترلرها را برنامه‌ریزی** (Program) **کرد. این کار توسط سخت‌افزاری با نام** Programmer **انجام می‌شود.** Programmer**ها، بسته به میکروکنترلرهای خود بسیار متنوع هستند و ممکن است از چند سیم تا چندین قطعه و بوردهای الکترونیکی پیچیده تشکیل شده باشند.**
زبانهای دسته‌های اول و دوم، از لحاظ نزدیکی به زبان ماشین به صورت جدول ۱-۱ تقسیم می‌شوند.

جدول ۱-۱ تقسیم‌بندی زبانهای برنامه‌نویسی بر اساس نزدیکی به زبان ماشین

زبانهای سطح پایین	زبانهای میانه	زبانهای سطح بالا
Assembly	C++ Forth	Basic Pascal Fortran

زبانهای سطح پایین، از لحاظ خوانایی برنامه و برنامه‌نویسی مشکل هستند، در صورتی که زبانهای برنامه‌نویسی سطح بالا از لحاظ خوانایی و برنامه‌نویسی ساده هستند، با این حال قدرت

زبان اسمبلی را در کنترل کامپیوتر ندارند. زبانهای میانه هر دو مزیت را در خود جمع کرده‌اند، از این رو، بهترین زبان برای برنامه‌نویسی سیستم هستند.

بخش سوم: بیتها و بایتها

تعاریف

اعدادی که در طول روز از آنها استفاده می‌شود، در مبنای ۱۰ یا Decimal هستند، اما برای ذخیره و پردازش اطلاعات در کامپیوتر ، باید آنها را به صورت دیجیتالی، یعنی در مبنای دو (Binary) و به صورت صفر و یک درآورد. جدول ۱-۲ تقسیم‌بندی این اعداد را نشان می‌دهد.

جدول ۱-۲ تقسیم‌بندی اطلاعات از نظر تعداد بیتها

نوع	تعداد صفر و یک
Bit	0
Nibble	0000
Byte	0000 0000
Word	0000 0000 0000 0000
Double Word	0000 0000 0000 0000 0000 0000 0000 0000

☜ در منطق **TTL**، کامپیوتر و میکروکنترلرها، یک را با 5Volt + و صفر را با 0Volt مشخص می‌کنند. گاهی اوقات نیز به منطق یک ، High یا لبه بالا و همچنین به منطق صفر، Low یا لبه پایین گفته می‌شود.

مبنای دیگری که در برنامه‌نویسی سیستم، زیاد استفاده می‌شود، مبنای ۱۶ یا هگزادسیمال (Hexadecimal) است، که در جدول ۱-۳ نشان داده شده است. اعدادی که اغلب، در برنامه‌نویسی سیستم بکار می‌روند، به صورت هگزادسیمال هستند، بنابراین همانطور که در زیر مشاهده می‌شود، سازندگان تمام زبانهای برنامه‌نویسی، روشی را برای استفاده از این اعداد طراحی کرده‌اند.

جدول ۱-۳ جدول اعداد هگزادسیمال

Hexadecimal	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10

B	11
C	12
D	13
E	14
F	15

روش نسبت دادن عدد هگزادسیمال F6 به متغیر A، در چند زبان برنامه‌نویسی

```
A=0xF6;          ⇦   C++ Builder, Turbo C++, Visual C++, C#
A=$F6;           ⇦   Delphi, Turbo Pascal
A=&HF6           ⇦   QBasic, Visual Basic
MOV AX,F6H       ⇦   Assembly
```

در زبانهای + + C و #C مبنای هشت یا اکتال (Octal) نیز وجود دارد، که با پیش کد 0 مشخص می‌شود. (به عنوان مثال a = 0215)

عملیات بایتی

برای انجام محاسبات و کنترل روی بیتها و بایتها، از یک سری عملگرهای (Operators) منطقی استفاده می‌شود، که در زیر توضیح داده شده‌اند.

عملگر AND

منطق این عملگر در جدول ۱-٤ نشان شده است.

جدول ۱-٤ جدول منطقی And

AND	1	0
1	1	0
0	0	0

نحوه نمایش عملگر And در زبانهای برنامه‌نویسی مختلف

```
A=0xF6 & 0xA1;          ⇦  C#, C++ Builder, Turbo C++, MATLAB
A:=$F6 and $A1;         ⇦  Delphi, Turbo Pascal
A=&HF6 AND &HA1         ⇦  QBasic, Visual Basic
AND AX, BX              ⇦  Assembly
```

عملگر OR

منطق این عملگر در جدول ۱-۵ نشان شده است.

جدول ۱-۵ جدول منطقی OR

OR	1	0
1	1	1
0	1	0

نحوه نمایش عملگر Or در زبانهای برنامه‌نویسی مختلف

```
A= 0xF6 | 0xA1;          ⇦ C#, C++ Builder, Turbo C++, MATLAB
A:= $F6 or $A1;          ⇦ Delphi, Turbo Pascal
A= &HF6 OR &HA1          ⇦ QBasic, Visual Basic
OR AX, BX                ⇦ Assembly
```

عملگر XOR

جدول ۱–٦ ، منطق این عملگر، که به آن یای انحصاری (Exclusive OR) نیز می‌گویند، را نشان می‌دهد.

جدول ۱-٦ جدول منطقی XOR

XOR	1	0
1	0	1
0	1	0

نحوه نمایش عملگر Xor در زبانهای برنامه‌نویسی مختلف

```
A=xor (10, 12);          ⇦ MATLAB
A:=$F6 xor $A1;          ⇦ Delphi, Turbo Pascal
A=&HF6 xor &HA1          ⇦ QBasic, Visual Basic
A=0xF6 ^ 0xA1            ⇦ C++ Builder, Turbo C++,C#
XOR AX, BX               ⇦ Assembly
```

عملگر NOT

منطق این عملگر در جدول ۱–۷ نشان داده شده است.

جدول ۱-۷ جدول منطقی NOT

NOT	
1	0
0	1

نحوه نمایش عملگر Not در زبانهای برنامه‌نویسی مختلف

```
A=~0xF6;                 ⇦ C#, C++ Builder, Turbo C++, MATLAB
A:= not($F6);            ⇦ Delphi , Turbo Pascal
A= not(&HF6)             ⇦ QBasic, Visual Basic
NOT AX                   ⇦ Assembly
```

عملگر SHIFT

این عملگر به دو صورت انتقال به چپ و یا انتقال به راست استفاده می‌شود. در شیفت به چپ، تمام بیتهای یک بایت به تعداد تعیین شده به چپ منتقل می‌شوند. در شیفت به راست، تمام بیتها به تعداد مشخص شده به سمت راست جابجا می‌شوند.

نحوه نمایش عملگر شیفت به چپ در زبانهای برنامه‌نویسی مختلف[۱]

[۱] در زبان بیسیک عملگر Shift وجود ندارد.

```
A=B<<4;                    ⇦  C++ Builder, Turbo C++
SHL AX, 4                  ⇦  Assembly
A:=B SHL 4;                ⇦  Pascal
```

نحوه نمایش عملگر شیفت به راست در زبانهای برنامه‌نویسی مختلف

```
A=B>>4;                    ⇦  C++ Builder, Turbo C++
SHR AX, 4                  ⇦  Assembly
A:=B SHR 4;                ⇦  Pascal
```

در انتقال به صورت شیفت، جای بیتهایی که منتقل می‌شوند، را بیت قبلی پر می‌کند و اگر بیتی ماقبل آن نباشد (آخرین بیت)، به جای آن بیت، صفر منظور می‌شود. یعنی مقدار هر بایت، با هشت بار انتقال، صفر می‌شود.

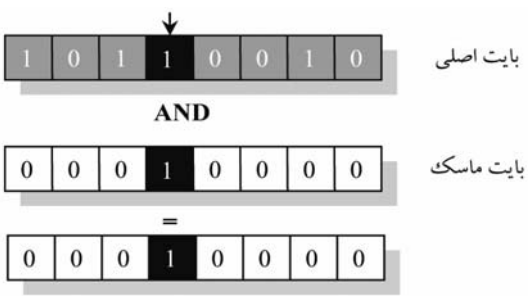
تکنیکهاي عملياتي در سطح بیت

توسط این عملیات، می‌توان اطلاعات دریافتی از دستگاه جانبی را آماده پردازش کرد. این کار در بسیاری از مواقع حیاتی و لازم است، زیرا همیشه داده‌های ورودی اطلاعات اصلی نیستند و ممکن است، چند بیت آنها تغییر کرده باشد. در بعضی موارد نیاز می‌شود که مقدار یک بیت به تنهایی چک شود.

۱. ماسک کردن

پیش از توضیح عملیات ماسک کردن، باید بایت ماسک را شناخت. این بایت بر اساس بیت مورد نظر برای چک کردن (بیتِ چک)، ساخته می‌شود. تمام بیتهای یک بایت ماسک بجز بیت چک، باید با عدد صفر پر شود. شکل ۱-۳ یک نمونه از بایت ماسک را نشان می‌دهد که برای بیت پنجم ساخته شده است.

همانطور که در شکل ۱-۳ مشخص شده، برای چک کردن یکی از بیتهای این بایت (که صفر یا یک است)، از تکنیک ماسک (Mask) کردن استفاده می‌شود. به عنوان مثال یک برنامه‌نویس برای چک کردن بیت پنجم (بیت شماره ٤)، یک بایت ماسک که فقط بیت پنجم آن یک و بقیه صفر است، را ایجاد کرده و آنرا با عدد مورد نظر AND می‌کند. حاصل اگر عددی غیر صفر بود، یعنی آن بیت یک بوده و اگر حاصل صفر بود، بدین معنی است که بیت پنجم صفر بوده است. برنامه ۱-۱ نحوه استفاده از این تکنیک را به وسیله برنامه‌نویسی نشان می‌دهد.



شکل ۱-۳
ماسک کردن یک بایت

برنامه ۱-۱ استفاده از ماسک زبان :Basic

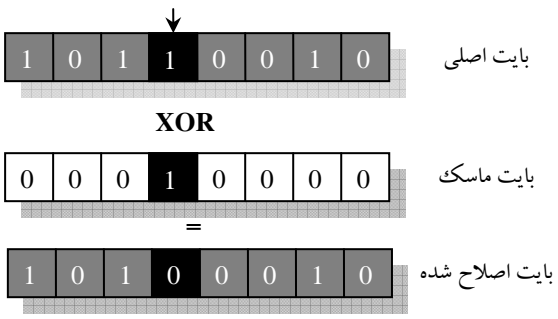
```
INPUT A
B=A AND &h10
IF B<>0 Then   Print "4th Bit is One"
Else
                  Print "4th Bit is Zero"
End IF
End
```

۲. اصلاح بایتها

همانطور که گفته شد، اطلاعاتی که از دستگاه جانبی دریافت می‌شوند، همیشه داده‌های واقعی نیستند. بعضی مواقع پیش می‌آید، که چند بیت از یک بایت Not شده باشند و گاهی دیگر، برخی از بیتها همیشه صفر یا یک هستند، که باید حذف شوند.

به عنوان مثال اطلاعاتی دریافت شده‌اند، که بیت پنجم آن Not شده است. برای اصلاح بایت مورد نظر از عملگر XOR استفاده می‌شود. همانطور که در شکل ۱-٤ مشخص است، با ساخت بایت ماسک مانند قسمت قبل و XOR کردن آن، بیت مورد نظر، Not شده و بقیه دست نخورده باقی می‌مانند. برنامه ۲-۱ نحوه انجام این کار را در زبان C++ نشان می‌دهد.

شکل ۱-٤
اصلاح یک بیت از بایت



برنامه ۱-۲ برنامه اصلاح یک بایت زبان :C++

```
#include<stdio.h>
#include<iostream.h>
main()
{
          int a,b;
          cin>>a;
          b=a^0x10;          // XOR
          cout<<"Result is "<<b;
}
```

نمونه دیگری از اصلاح بایت‌ها زمانی پیش می‌آید که برنامه‌نویس بخواهد، یک بیت خاص از

اطلاعات را همیشه صفر و یا یک نگاه دارد ، بدون آنکه در سایر بیتها تغییری ایجاد شود. برای صفر نگاه داشتن یک بیت، بایت مورد نظر را باید با مکمل بایت ماسک (در قرینه بایت ماسک، بیت مورد نظر صفر و بقیه بیتها یک است)، AND کرد و برای یک نگاهداشتن یک بیت، کافی است که آن بایت، با بایت ماسک OR شود.

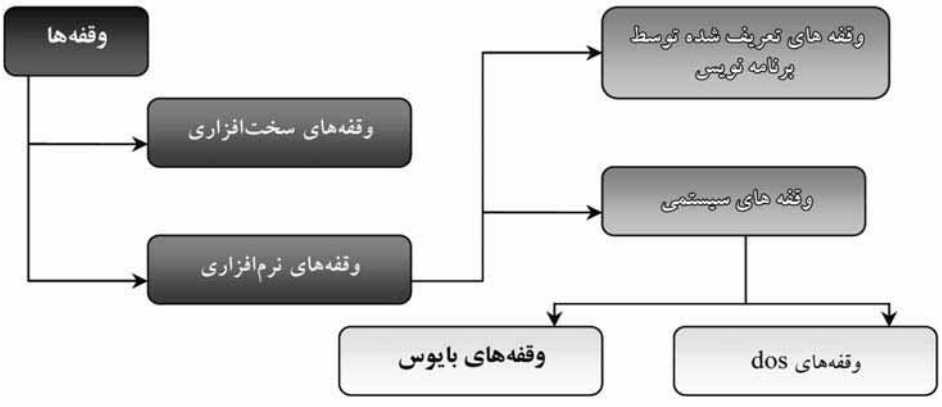
برنامه ۱-۳ مثالی در مورد این نوع اصلاح بایت را نشان می‌دهد. در این مثال، بیت پنجم متغیر a ، باید همیشه یک باقی بماند و عددی هم که در متغیر c است، بیت پنجم آن باید همیشه صفر بماند.

برنامه ۱-۳ برنامه‌ای برای ثابت نگهداشتن برخی از بیتهای یک بایت — زبان C++:

```
#include<iostream.h>
main()
{
          int a,b;
          cin>>a>>c;                    // Input
          b=a|0x10;                     // OR
          cout<<"Result is "<<b;
          b=c&0xEF;                     // AND
          cout<<"Result is "<<b;
 }
```

بخش چهارم: وقفه‌هاي نرم‌افزاري

وقفه‌ها (Interrupt) یکی از مهمترین قسمتهای یک سیستم برای برنامه‌نویسی و کنترل هستند و در کل به دو دسته نرم‌افزاری و سخت‌افزاری تقسیم می‌شوند. شکل ۵-۱ این تقسیم‌بندی را نشان می‌دهد.



شکل ۱-۵ تقسیم‌بندی وقفه‌ها

☜ در این تقسیم‌بندی وقفه‌هایی مانند External Machine check, Restart ,Program و I/O Compulsion دیده نشده‌اند، زیرا در این بخش صرفاً وقفه‌های مورد نیاز در فصلهای بعدی توضیح داده می‌شوند.

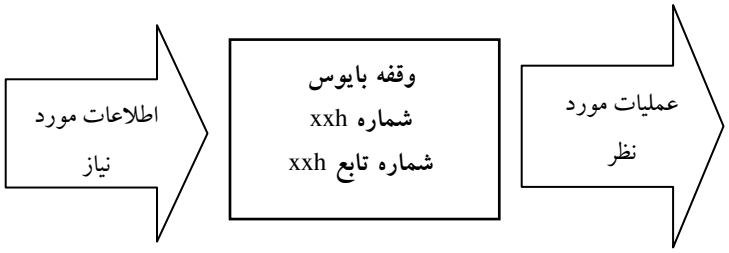
همانطور که در شکل ۱–۵ نشان داده شده است، وقفه‌های نرم‌افزاری به دو دسته سیستمی، که توسط سازندگان کامپیوتر تعریف می‌شوند، و وقفه‌هایی که توسط برنامه‌نویسی تعریف می‌شوند، تقسیم شده‌اند.

وقفه‌های سیستمی نیز به دو دسته بایوس و Dos تقسیم می‌شوند، که وقفه‌های Dos نیز شاخه کوچکی از وقفه‌های بایوس هستند. بنابراین در این فصل فقط وقفه‌های بایوس توضیح داده خواهند شد.

وقفه‌هاي بايوس

در کامپیوتر، برنامه‌های از پیش نوشته شده‌ای وجود دارند. این برنامه‌ها، در یک یا چند تراشه (IC) قرار داده شده‌اند و پایه کاری کل نرم‌افزارهای کامپیوتر را تشکیل می‌دهند و از طریق برنامه‌نویسی می‌توان به آنها دسترسی داشت. این برنامه‌ها ، وقفه‌های بایوس هستند که در برنامه‌نویسی سیستم از آنها استفاده می‌شود.

نحوه اجرای یک وقفه نرم‌افزاری به این صورت است که در هنگام اجرای یک برنامه، هنگام رسیدن به دستورالعمل وقفه ، کامپایلر آدرس جاری اجرای برنامه اصلی را در رجیسترهایی (Register) با نام CS (Code Segment) و IP (Instruction Pointer) ذخیره می‌کند، سپس با توجه به شماره وقفه، آدرس آنرا در جدولی با نام جدول بردار وقفه (Interrupt Vector) پیدا می‌کند (شکل ۱–۷). در این هنگام برنامه‌های موجود در این آدرس را اجرا کرده و سپس به آدرسهای CS و IP بازمی‌گردد. علت نام‌گذاری این سیستم، به عنوان وقفه به همین علت است، زیرا در اجرای برنامه اصلی به علل سخت‌افزاری و یا نرم‌افزاری تاخیری ایجاد شده و برنامه مربوط به وقفه را اجرا می‌کند.



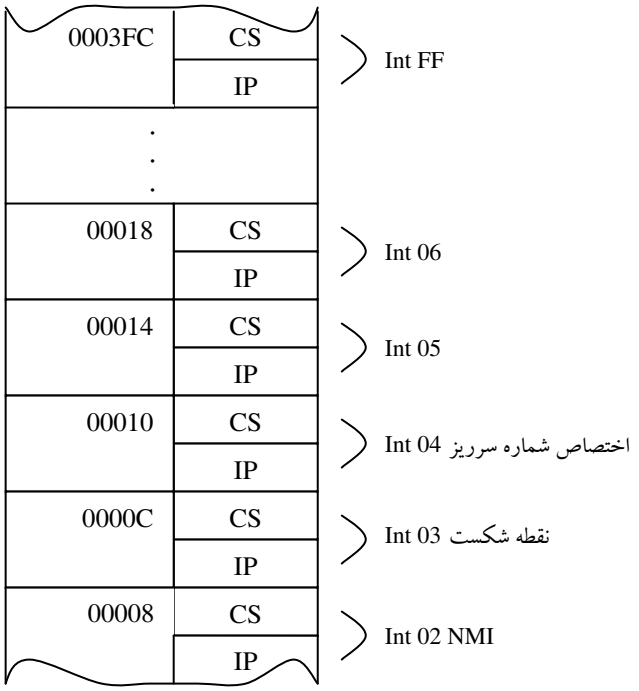
شکل ۱-٦ بلاک یک وقفه بایوس

همانطور که در شکل ۱-٦ مشاهده می‌شود، برای اجرای یک وقفه، دانستن اطلاعاتی در مورد آن لازم است. این اطلاعات به صورت جدولی در کتابهای مرجع یافت می‌شوند که تعدادی از این جدولها در ضمیمه این کتاب آورده شده‌اند. علاوه بر آن، برای استفاده از یک وقفه باید با موارد زیر آشنا بود.

۱. شماره وقفه

هر وقفه دارای یک شماره است که طبق یک فرمول خاصی به آدرس معینی در بایوس اشاره می‌کند. در اجرای یک برنامه به محض آنکه کامپیوتر به یک وقفه بایوس برخورد کند، اجرای

شکل ۱-۷
جدول بردار وقفه



برنامه را متوقف کرده و پردازنده به آدرس تعیین شده رفته و برنامه‌های موجود در آن را اجرا می‌کند. در نتیجه عملیات مورد نظر آن وقفه اجرا می‌شود. هر شماره وقفه، شامل چندین تابع مربوط به یک واحد خاص می‌باشد. به عنوان مثال وقفه شماره 10h و تمام توابع آن، مربوط به کنترل صفحه نمایش می‌شوند. در جدول ۱-۸ تعدادی از وقفه‌های بایوس و به همراه شماره آنها نشان داده شده است.

جدول ۱-۸ وقفه‌های بایوس

شماره وقفه	عمل وقفه
5h	فراهم نمودن امکان چاپ برای نمایش
10h	ورودی خروجی صفحه نمایش
11h	تشخیص دستگاه جانبی
12h	تعیین میزان حافظه اصلی کامپیوتر
13h	ورودی - خروجی دیسک
14h	ورودی خروجی پورت سری
15h	ورودی- خروجی کاست
16h	ورودی- خروجی صفحه کلید
17h	ورودی- خروجی چاپگر
18h	اجرای Rom Basic
19h	اجرای بار کننده خودکار(Reset)
1Ah	زمان و تاریخ
21h , 20h	وقفه‌های Dos

۲. **شماره تابع**

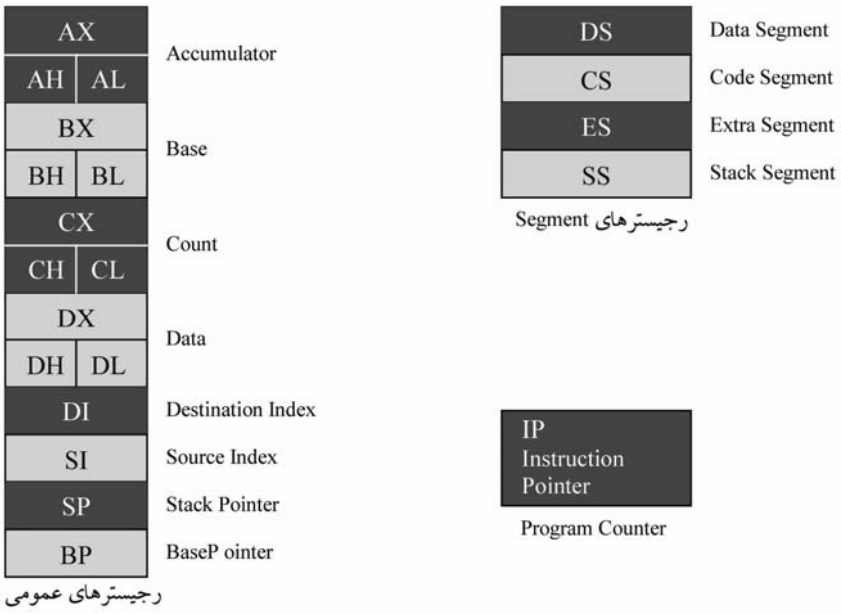
هر تابعی در وقفه‌ها، دارای یک شماره خاص است و کار معینی را انجام می‌دهد. به عنوان مثال تابع شماره 0 از وقفه شماره 10h برای تنظیم مد گرافیکی صفحه نمایش در نظر گرفته شده است.

۳. **اطلاعات مورد نظر که توسط رجیسترها منتقل می‌شوند**

برای آنکه عملیات مورد نظر از تابع دلخواه انتخاب شود، باید اطلاعات ، به یک سری متغیرهای خاص به نام رجیستر وارد شوند. به عنوان مثال برای تنظیم مد گرافیکی باید شماره مد را در رجیستر AL قرار داد.

رجیسترها

دسته‌بندی رجیسترهای کامپیوترهای خانواده 8086 ، در شکل ۱–۸ نشان داده شده‌اند. سری رجیسترهایی که برای اجرای وقفه‌ها بسیار مهم هستند، رجیسترهای عمومی و یا Common Register نامیده می‌شوند. این رجیسترها شامل رجیسترهای دو بایتی AX,BX,CX,DX و همچنین رجیسترهای یک بایتی AL,AH,BL,BH,CL,CH,DL,DH می‌باشند.



شکل ۱-۸ رجیسترهای 8086

☜ در وقفه‌های بایوس معمولاً شماره تابع وقفه، در رجیستر ah و یا ax قرار می‌گیرد.

این رجیسترها در حقیقت، متغیرهای زبان اسمبلی هستند و به علت اینکه بایوس نیز به این زبان نوشته شده، برای استفاده از وقفه‌های بایوس باید از آنها استفاده کرد. طراحان سایر زبانهای برنامه‌نویسی با ساختارهایی این رجیسترها را شبیه‌سازی کرده‌اند، تا از آنها بتوان در زبان برنامه‌نویسی مورد نظر استفاده کرد، که یکی از استفاده‌های مهم آنها، اجرای وقفه‌های بایوس است. در ادامه، به این ساختارها، در زبانهای برنامه‌نویسی مختلف پرداخته شده است.

وقفه‌ها در خانواده C

در این زبان، از واحدها (Unions) برای تعریف وقفه‌ها استفاده می‌شود. این واحدها در سرفایل (header file) Dos.h به صورت زیر تعریف شده‌اند.

```
union REGS{
        union WORDREGS x;
        union BYTEREGS h;
};
union BYTEREGS{
                unsigned char al,bl,ah,bh;
                unsigned char cl,ch,dl,dh;
};
union WORDREGS{
                unsigned int ax,bx,cx,dx;
                unsigned int si,di,cflag,flags;
};
union REGS{
        union WORDREGS x;
        union BYTEREGS h;
};
union BYTEREGS{
                unsigned char al,bl,ah,bh;
                unsigned char cl,ch,dl,dh;
};
union WORDREGS{
                unsigned int ax,bx,cx,dx;
                unsigned int si,di,cflag,flags;
};
```

```
struct SREGS{
   unsigned int  es;
   unsigned int  cs;
   unsigned int  ss;
unsigned int  ds
};
```

پس برای تعریف یک رجیستر باید به صورت برنامه ۱-٤ عمل کرد.

برنامه ۱-٤ استفاده از وقفه‌های بایوس — زبان :C++

```
#include<dos.h>
#include<stdio.h>
       main()
       {
              union REGS in,out;          // Define the Register
              in.h.ah=0;
              in.h.al=1;                  // Define the Graphic Mode
              int86(0x10,&in,&out);       //  Run the Interrupt
        }
```

اجرای وقفه ، به وسیله یکی از توابع زیر میسر می‌باشد، که فرم کلی آنها نشان داده شده است.

int86(شماره وقفه,&رجیسترهای ورودی, &رجیسترهای خروجی);

int86x(شماره وقفه,&رجیسترهای ورودی, &رجیسترهای خروجی, &SREGS);

intdos(&رجیسترهای ورودی, &رجیسترهای خروجی);

intdosx(&رجیسترهای ورودی, &رجیسترهای خروجی, &SREGS);

توابع intdos و intdosx **برای اجرای وقفه‌های** Dos **و توابع** int86 و int86x **برای اجرای توابع بایوس استفاده می‌شوند. در توابع مربوط به وقفه‌های** Dos **نیازی به شماره وقفه نیست، زیرا این وقفه‌ها همگی دارای یک شماره هستند. در توابع** intdosx **و** int86x **، علاوه بر رجیسترهای معمولی، از رجیسترهای** ss, es, cs, ds **نیز می‌توان استفاده کرد.**

وقفه‌ها در زبانهاي خانواده پاسکال

در این زبانهای برنامه‌نویسی رجیسترها به صورت یک رکورد مانند زیر تعریف شده‌اند.

```
Type   Registers =Record
                Case integer of
                       0 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Word);
                       1 : (AL,AH,BL,BH,CL,CH,DL,DH : Byte);
                End;
```

برنامه ۱-۵ نحوه استفاده از وقفه‌ها را در زبان پاسکال نشان می‌دهد.

برنامه ۱-۵ استفاده از وقفه‌های بایوس — زبان: Pascal

```
Program registers_;
Uses dos;
Var
        Regs : Registers;
Begin
              Regs.ah:=0;
              Regs.al:=1;                          { Define the Graphic Mode }
              Intr($10,regs);                      {  Run the Interrupt }
End.
```

همانطور که در برنامه ۱-۵ مشخص شد، برای اجرای وقفه از تابع Intr استفاده مـی‌شـود، کـه فرم کلی آن به صورت زیر است.

Intr(شماره وقفه , رجیسترها);

در زبانهای Turbo C++ و Turbo Pascal مقادیر رجیسـترها را در پنجـره register در منـوی Window می‌توان مشاهده کرد.

وقفه‌ها در زبانهاي خانواده بيسيک

در این زبانها رجیسترها به صورت زیر تعریف می‌شوند.

```
TYPE RegTypeX
           ax as INTEGER
           bx as INTEGER
           cx as INTEGER
           dx as INTEGER
           bp as INTEGER
           si as INTEGER
           di as INTEGER
           flags as INTEGER
End Type
```

برنامه ۱-٦ نحوه استفاده از وقفه‌ها را در زبان بیسیک مشخص می‌کند.

برنامه ۱-٦ استفاده از وقفه‌های بایوس — زبان :Basic

```
' Interrupt Test in Basic
DIM RegsX as RegTypeX
CLS
RegsX.AX=&H1B00
CALL interruptx(&H21,RegsX,RegsX)
PRINT "Media ID=";RegsX.BX     ' Run the Interrupt
END
```

وقفه‌ها در زبان برنامه‌نویسی بیسیک توسط تابع interruptx اجرا می‌شوند، که در زیر، فرم کلی آن نشان داده شده است.

CALL interruptx(**رجیسترهای خروجی , رجیسترهای ورودی ,شماره وقفه**)

وقفه‌ها در زبان اسمبلي

برنامه ۱-۷ استفاده از وقفه‌های بایوس زبان :Assembly

```
; Interrupt Test in Assembly
Mov Ah,1h
Mov al,0h
INT 10h
```

همانطور که در برنامه ۱-۷ نشان داده شده، در این زبان، از دستور INT برای اجرای وقفه استفاده می‌شود، که فرم کلی این دستور به صورت زیر است.

INT شماره وقفه

بخش پنجم: استفاده از اسمبلي، در زبانهاي ديگر

به علت خوانایی بیشتر و پشتیبانی راحت‌تر پروژه ، در برنامه‌های سیستم از زبانهای سطح بالا استفاده می‌شود و هر جا که نیاز به کنترل بیشتری باشد، دستورات اسمبلی را بکار می‌برند. در این بخش روش استفاده از دستورات اسمبلی، در زبانهای مختلف توضیح داده شده است.

استفاده از اسمبلي در زبانهاي خانواده پاسکال

نحوه استفاده از این دستورات در برنامه ۱-۸ نشان داده شده است.

برنامه ۱-۸ استفاده از اسمبلی در زبانهای دیگر زبان Pascal:

```
Program asmb;
Begin
        ASM
                        { ASSEMBLY CODE HERE }
                        MOV AX,10
                        MOV BX,10
                        INT 10h
        END;
End.
```

همانطور که در این برنامه مشخص شده، دستورات اسمبلی باید بین ASM و END نوشته شود.

استفاده از اسمبلي در خانواده C

در برنامه ۱-۹ نحوه استفاده از دستورات اسمبلی در زبان C++ نشان داده شده است.

برنامه ۱-۹ استفاده از اسمبلی در زبانهای دیگر زبان :C++

```
#include<dos.h>
#include<stdio.h>
    main()
    {
                    // Assembly Code Here
              asm   Mov ax,1
              asm   Mov bx,1
              asm   Int 10h

              asm{
                   // Assembly Code Here
                Mov ax,1
                Mov bx,1
```

```
                Int 10h
            } ;
}
```

همانطور که در برنامه ۱-۹ نشان داده شده ، دستورات اسمبلی را به دو صورت، در این زبان می‌توان استفاده کرد. در ضمن بوسیله Turbo C++ ، برنامه نوشته شده به زبان C++ را می‌توان به اسمبلی تبدیل کرد . این کار به وسیله فایل TCC.EXE[1] و به صورت زیر انجام می‌شود.

```
C:\TC\BIN> TCC -S     filename.cpp
```

در صورتی که تنظیمات محیط Turbo C++ درست انجام شده باشد، پس از اجرای دستور فوق، فایل filename.asm به وجود خواهد آمد، که حاوی کلیه دستورات اسمبلی برنامه filename.cpp است.

انتقال مقادیر متغیرها در زبان اسمبلی

گاهی، لازم است که مقدار متغیری را در یکی از زبانهای برنامه‌نویسی به رجیستری در زبان اسمبلی انتقال داد و یا بالعکس، برای انجام اینکار، باید مانند زیر عمل شود.

```
MOV AX, WORD PTR [variable]
MOV WORD PTR [variable], AX
```

دستور اول مقدار متغیر variable را به رجیستر ax در اسمبلی منتقل می‌کند و دستور دوم مقدار رجیستر ax را به متغیر variable انتساب می‌دهد. برنامه ۱-۱۰ مثالی را در این زمینه نشان می‌دهد.

برنامه ۱-۱۰ انتقال مقادیر متغیرهای زبان اسمبلی زبان C++:

```
#include<stdio.h>
main()
{
   int a;
   asm MOV ax,19
   asm MOV word ptr [a],ax
   printf("%d",a);
}
```

با توجه به انتساب مقادیر در این برنامه، خروجی عدد ۱۹ را نشان می‌دهد.

بخش ششم: کنترل صفحه کلید بوسیله برنامه

برای آنکه بتوان به وسیله نرم‌افزار، سخت‌افزار را کنترل کرد. باید برنامه‌ها و دستورات آنها در یک حلقه دائماً در حال اجرا باشند. برای کنترل این حلقه‌ها در زبانهای تحت ویندوز از تایمرها استفاده می‌شود، اما در زبانهای تحت Dos این حلقه‌ها بهتر است با صفحه کلید کنترل شوند. در این بخش، کار با صفحه کلید در زبانهای تحت Dos توضیح داده شده است.

قبل از آنکه به نحوه کنترل کلیدهای صفحه کلید در زبانهای مختلف برنامه‌نویسی بپردازیم،

[1] این فایل، حاوی مترجم زبان C++ است.

باید اطلاعاتی راجع به کدهای اسکی (ASCII[1]) و کدهای صفحه کلید و کاراکترها (Character) داشت.

کاراکترهاي اسکي

کلیدهای یک صفحه کلید را می‌توان به دو دسته تقسیم کرد. یک دسته کلیدهایی که با فشردن آن کاراکتری در مانیتور چاپ می‌شود (مانند کلیدهای الفبایی) و دسته دیگر، کلیدهایی که با فشردن آنها عملیاتی انجام می‌شود (مانند کلید F1). کلیدهای دسته اول جزیی از مجموعه کاراکترهای اسکی (ASCII Character) بشمار می‌آیند. این مجموعه ۲۵۶تایی شامل کاراکترهای الفبایی، عددی و علایم می‌باشند. هر کدام از این کاراکترها دارای یک کد خاص هستند که کد اسکی نامیده می‌شوند. جدول کدهای اسکی در ضمیمه کتاب نشان داده شده است.

☜ کاراکتر هر کدام از کدهای اسکی را می‌توان به وسیله قسمت عددی صفحه کلید(Numeric Keys) دید. این کار با گرفتن کلید Alt و تایپ همزمان کد اسکی روی قسمت عددی انجام می‌شود.

کدهاي صفحه کلید

همانطور که در قسمت فوق بیان شد، سری دیگری از کلیدها وجود دارند که دارای کد اسکی نیستند. برای کنترل این کلیدها از کدهایی با نام Scan Code استفاده می‌شود. Scan Codeها دارای دو دسته عادی و توسعه یافته هستند. کدهای دسته عادی، مانند کدهای اسکی دارای یک شماره معمولی هستند، ولی کدهای دسته توسعه یافته به وسیله پیش کد صفر مشخص می‌شوند که به محض فشردن یک کلید توسعه یافته، ابتدا پیش کد آن ارسال و بعد کد اصلی آن فرستاده می‌شود. بنابراین باید در دو مرحله کد کلیدهای توسعه یافته را دریافت و کنترل کرد.

استفاده از صفحه کلید در زبانهاي خانواده بیسیک

برنامه ۱–۱۱ نمونه‌ای از کاربرد کدهای صفحه کلید را در بیسیک نشان می‌دهد.

برنامه ۱-۱۱ کنترل صفحه کلید — زبان Basic:

```
CLS
While inkey$(1)=""
     Print "No KEY pressed!"
Wend
End
```

برای کنترل صفحه کلید به توابعی نیاز است که کلید فشرده شده در صفحه کلید را برگردانند، و همچنین به تابع دیگری احتیاج است که به وسیله آنها بتوان فشردن یک کلید توسط کاربر را

[1] American Standard Code for Information Interchange

متوجه شد. در زبان بیسیک هر دو کار با $inkey **انجام‌پذیر است. برای بدست آوردن کد کلید نیز می‌توان از تابع** ASC **استفاده کرد.**

مثال:

```
If asc(inkey$(1))=13        Then          Print "Enter Key"
```

استفاده از صفحه کلید در زبانهاي خانواده پاسکال

برنامه ۱-۱۲ نحوه استفاده از صفحه کلید را در زبان پاسکال نشان می‌دهد. همانطور که در این برنامه نشان داده شده، به وسیله تابع keypressed **می‌توان چک کرد که آیا کلیدی فشرده شده است یا نه، (در صورتی که کلیدی فشرده شود مقدار** true **و در غیر این صورت مقدار** false **را برمی‌گرداند) و توسط تابع** readkey **می‌توان کد آن کلید را دریافت کرد. این دو تابع در یونیت** CRT (Unit) **تعریف شده‌اند.**

;

برنامه ۱-۱۲ کنترل صفحه کلید زبان

Pascal:

```
Program keys;
Uses CRT;
Var
     Ch:Char;
Begin
   While not(keypressed) do
Begin
        Writeln(' No key pressed!');
End;
Ch:=readkey;
Writeln(ch);
End.
```

برای بدست آوردن کد یک کلید می‌توان از عملگر "#" استفاده کرد. به عنوان مثال، #13 **معادل کلید** Enter **است.**

مثال :

```
If   Readkey=#13       Then WriteLn('Enter Key')
```

استفاده از صفحه کلید در زبانهاي خانواده C++

برنامه ۱-۱۳ نمونه‌ای از کاربرد کدهای صفحه کلید را در زبان C++ **نشان می‌دهد.**

به وسیله تابع kbhit **می‌توان فشرده شدن یک کلید را بررسی کرد، (در صورتی که کلیدی فشرده شود مقدار غیر صفر و در غیر این صورت مقدار صفر را برمی‌گرداند) و سپس توسط تابع** getkey **آن می‌توان مقدار آنرا دریافت کرد. برای استفاده از این توابع باید سرفایلهای** dos.h **و** conio.h **را در اول برنامه تعریف کرد.**

برنامه ۱-۱۳ کنترل صفحه کلید زبان

C++:

```
#include<stdio.h>
```

```
#include<conio.h>
#include<dos.h>
  main()
  {
        char ch;
        while(!kbhit())
              printf(" No key pressed!");
       ch=getch();
       putchar(ch);
  }
```

در زبان C++ برای بدست آوردن کد یک کلید نیاز به هیچ تابعی نیست. به عنوان مثال خود عدد 13 معادل کلید Enter است.

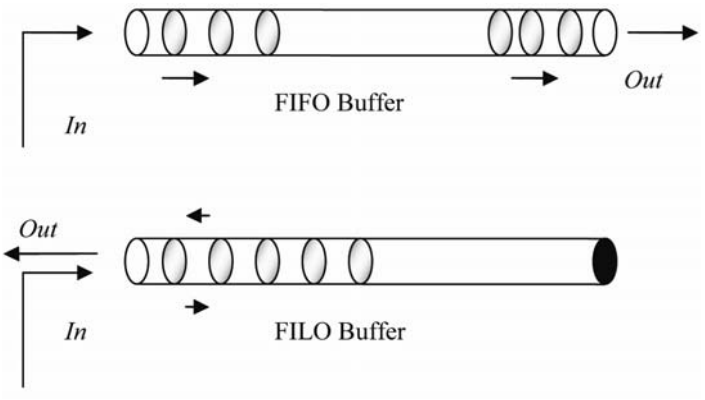
مثال :

```
if (getch () =13)       puts("Enter Key");
```

بخش هفتم: بافرها (Buffer)

بافرها برای جابجایی اطلاعات استفاده می‌شوند و سرعت انتقال را تا حدودی افزایش می‌دهند. به عنوان مثال برنامه‌نویسی قصد دارد، ٦٤ بیت اطلاعات را انتقال دهد، بدترین روش آن است که این بیتها به صورتی تکی انتقال داده شوند. بهتر آن است، ۳۲ بیت را در فضایی ذخیره کرد و سپس ۳۲ بیت را با هم انتقال داد. به فضایی که از این بیتها به صورت موقت نگهداری می‌کند، بافر گفته می‌شود.

در حقیقت بافر، فضای نگهداری اطلاعات به صورت موقت است و مهمترین کاربردی که می‌تواند داشته باشد، افزایش سرعت انتقال اطلاعات است، زیرا حجم اطلاعات انتقالی را بسیار افزایش می‌دهد و در حالی که اطلاعات در حال ارسال است ، یک بافر دیگر در حال پر شدن است و به محض آماده شدن به مقصد ارسال می‌شود.



شکل ۱-۹
ساختار دو بافر FILO و FIFO

از انواع پر استفاده بافرها می‌توان به FIFO (صف) و FILO (پشته) اشاره کرد، که شکل ۱-۹ تفاوت این دو بافر را نشان می‌دهد.

بافر FIFO (First In First Out) بدین صورت عمل می‌کند که اطلاعات وارد شده به این بافر در هنگام دریافت، از اوّل دریافت می‌شوند. یعنی اولین بیتی که ارسال شده ، در هنگام دریافت، اولین بیتی است که دریافت می‌شود.

بافر FILO (First In Last Out) بدین صورت عمل می‌کند که اطلاعاتی، که در ابتدا وارد بافر شده‌اند، آخرین اطلاعاتی هستند که خارج می‌شوند. برای درک بهتر این بافر به سینی‌هایی توجه کنید، که روی هم قرار می‌گیرند. سینی که آخر همه گذاشته می‌شود ، اولین سینی است که برداشته می‌شود.

بهترین زبان، برای تعریف مستقیم بافر زبان C++ است و این تعریف به صورت زیر انجام می‌شود.

```
void far buffer;
```

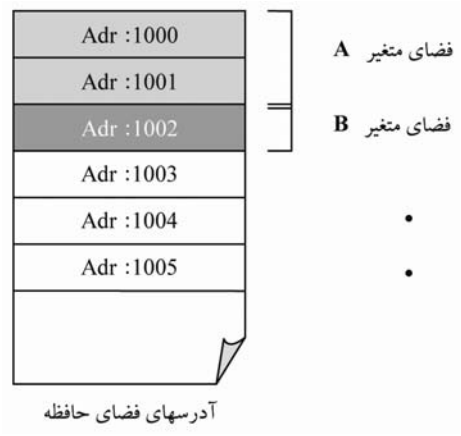
بخش هشتم : آدرسهاي حافظه و اشاره‌گرها

اختصاص فضا براي متغيرها

در ابتدا به نحوه اختصاص فضا برای متغیرها پرداخته خواهد شد. به عنوان مثال دو متغیر A و B در خطوط زیر تعریف شده‌اند. کامپایلرها برای متغیرهای نوع Integer دو بایت و برای متغیرهای نوع Byte یک بایت را در نظر می‌گیرند.

شکل ۱-۱۰ نحوه اختصاص فضا برای این دو متغیر را در حافظه (Memory) نشان می‌دهد. همانطور که در این شکل نشان داده شده، به هر متغیر بسته به نوع آن از یک بایت تا چندین بایت اختصاص داده می‌شود. هر فضای حافظه نیز دارای یک آدرس است و در نتیجه هر متغیر نیز دارای یک آدرس حافظه می‌شود، که این آدرس‌ها مربوط به اولین فضای حافظه آن متغیر هستند. بنابراین آدرس متغیر A عدد 1000 و آدرس متغیر B عدد 1002 است (آدرسها به صورت هگزادسیمال هستند).

شکل ۱-۱۰
فضای حافظه



```
A: Integer;
B: Byte;
```

در برنامه‌نویسی به این آدرسها، اشاره‌گر (Pointer) گفته می‌شود.
اشاره‌گرها، کاربردهای بسیار مهمی در برنامه‌نویسی پیشرفته دارند. در ادامه، نحوه بکـارگیری این آدرسها در زبانهای برنامه‌نویسی مختلف نشان داده شده است.

اشاره‌گرها در زبان C++

برنامه ۱-۱٤ نحوه بدست آوردن آدرسها را در زبان C++ نشان می‌دهد. در این برنامه، اپراتور %P در تابع printf، آدرس و اشاره‌گر هر متغیر و تابعی(حتی تابع Main) را چاپ می‌کند، امـا کـار بـا چاپ آدرسها تمام نمی‌شود، زیرا برای استفاده مفیدتر از اشاره‌گرها باید بتوان آنها تعریف کرد. در زیر نحوه تعریف یک اشاره‌گر نشان داده شده است.

```
int *a;
float *f;
char *ch;
```

برنامه ۱-۱٤ استفاده از اشاره‌گرها زبان :C++

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>

  main()
  {
        char ch;
        int a;

        printf(" Pointers   A=%p  Ch=%p \n",a,ch);
        printf(" Pointer of Main Function is %p ",main);

  }
```

برای تعریف هر اشاره‌گر، باید مانند یک متغیر معمولی عمل کرد، با این تفاوت که قبل از نـام آن باید از علامت * استفاده شود. پس از این تعریف، از متغیر اشاره‌گر، می‌توان بـه عنـوان یـک متغیر معمولی نیز استفاده کرد.

```
*a=12;
*f=3.141592;
*ch='A';
```

اما نام متغیر بدون علامت * حاوی یک آدرس است. این آدرس به صورت هگزادسـیمال و از نوع long unsigned **است.**

آدرس یک متغیر معمولی را نیز می‌توان با استفاده از عملگر & بدست آورد. به عنوان مثال :

```
int a;
int *p;
p=&a;
```

در این مثال آدرس متغیر معمولی a در اشاره‌گر p قرار می‌گیرد.

نکته بسیار مهم در زبان C++ این است که آرایه‌ها و رشته‌ها نیز در این زبان چیزی به جز چند اشاره‌گر نیستند.

به عنوان مثال :

```
int a[25];
a[0]=12;
printf("%d,%d",a[0],*a);
```

بعد از اجرای این تکه برنامه، مشخص می‌شود که مقدار a[0] با a* برابر است. علت این امر این است که در C++، اسم هر آرایه، اشاره‌گر به اولین عضو آن آرایه است. پس با در نظر گرفتن این مطلب می‌توان نتیجه گرفت که *(a+2) برابر a[1] است. استفاده از عدد ۲ به این علت است، که آرایه از نوع int است، پس هر عضو آن با عضو بعدی ۲ بایت اختلاف دارد.

با توجه به مطالب فوق، این مطلب روشن می‌شود، که چرا در اجرای وقفه‌ها به جای متغیرهای in و out از &in و &out استفاده می‌شود.

اشاره‌گرها در زبان پاسکال

برای تعریف یک اشاره‌گر در زبان پاسکال از علامت ^ استفاده می‌شود.

```
A:^Integer;
F:^Real;
CH:^Char;
```

به عنوان مثال در قطعه برنامه فوق متغیر A از نوع اشاره‌گر به عدد صحیح تعریف شده است. بعد از این تعریف، از متغیر ^A می‌توان به عنوان یک متغیر صحیح معمولی استفاده کرد. برای قرار دادن یک آدرس در اشاره‌گر ، باید از عملگر @ استفاده کرد. برنامه ۱-۱۵ نحوه استفاده از اشاره‌گرها را در زبان پاسکال نشان می‌دهد.

برنامه ۱-۱۵ استفاده از اشاره‌گرها — زبان :Pascal

```
Program Pointer;
Var
   A,B: Integer;
   P:^Integer;
Begin
   A:=15;
   P:=@A;
   B:=P^;
   WriteLn('A=',a,' B=',b);
End;
```

در این برنامه، مقدار A و B با هم برابر می‌شوند.

بخش نهم: بایوس و سیستم‌عامل

بایوس تنها بخشی است که می‌تواند تمام گرداننده‌هایی (Drivers) را که در یک سیستم به عنوان واسط سخت‌افزار سیستم و سیستم‌عامل کار می‌کنند، شرح دهد. بایوس، در حقیقت نرم‌افزار را به سخت‌افزار متصل می‌نماید. قسمتی از بایوس روی تراشه ROM (Read Only Memory) مادربرد

(Motherboard) و قسمتی دیگر بر روی تراشه کارتهای وفق دهنده قرار دارد که Firmware (یعنی میانه‌افزار) نامیده می‌شود . یک PC می‌تواند شامل لایه‌هایی (نرم‌افزاری و سخت‌افزاری) باشد، که واسط بین یکدیگرند. در اکثر اوقات می‌توان یک کامپیوتر را به چهار لایه تقسیم کرد که هر کدام از لایه‌ها به زیر مجموعه‌هایی کوچکتر تقسیم می‌شوند. هدف از این نوع طراحی این است که سیستم‌عامل‌ها و نرم‌افزارهای مختلف بر روی سخت‌افزارهای مختلف اجرا شوند (حالت مستقل از سخت‌افزار). بدین طریق دو ماشین با دو پردازنده مختلف، حافظه‌های ذخیره‌سازی متفاوت و دو نوع واحد گرافیکی و غیره ... ، یک نرم‌افزار را می‌توانند اجرا کنند.

در معماری این لایه‌ها برنامه‌های کاربردی با سیستم‌عامل از طریق API (Application Program Interface) ارتباط برقرار می‌کنند. API بر اساس سیستم‌عامل مورد استفاده ، مجموعه توابع و دستورالعملهایی که برای یک بسته نرم‌افزاری ارایه می‌دهد ، متغیر می‌باشد. به طور مثال یک نرم‌افزار می‌تواند از سیستم‌عامل برای ذخیره و بازیابی اطلاعات استفاده کند و خود نرم‌افزار مجبور نیست که این اعمال را انجام دهد. نرم‌افزارها به گونه‌ای، طراحی شده‌اند که می‌توان آنها را بر روی سیستمهای دیگر نصب و اجرا کرد و این امر، به دلیل مجزا شدن سخت‌افزار از نرم‌افزار است و در این سیستم‌ها، نرم‌افزار از سیستم‌عامل برای دستیابی به سخت‌افزار سیستم استفاده می‌کند، سپس سیستم‌عامل از طریق واسطها به لایه‌های بایوس دست می‌یابد .

بایوس شامل نرم‌افزارهای گرداننده‌ای است که بین سخت‌افزار و سیستم‌عامل ارتباط برقرار می‌کنند. سیستم‌عامل هیچگاه نمی‌تواند، مستقیماً به سخت‌افزار دستیابی پیدا کند ، و مجبور است، از طریق برنامه‌های گرداننده‌ای که به این کار تخصیص یافته‌اند، ارتباط را برقرار کند. یکی از وظایف تولیدکنندگان قطعات سخت‌افزاری آن است که گرداننده‌ای برای قطعات تولیدی خود ارایه دهند و چون گرداننده‌ها باید بین سخت‌افزار و نرم‌افزار عمل نمایند، باید گرداننده‌های هر سیستم‌عامل مجزا تولید شوند . بنابراین کارخانه سازنده قطعات باید گرداننده‌های مختلفی ارایه دهد تا قطعه مورد نظر بتواند، بر روی سیستم‌عامل‌های متداول کار کند. چون لایه‌های API همانند یک سیستم‌عامل به نظر می‌رسند، مهم نیست که با چه سخت‌افزاری کار می‌کنند ، همچنین می‌توان سیستم‌عاملها را بر روی هر کامپیوتری و با هر نوع مشخصات سخت‌افزاری نصب و استفاده کرد. برای مثال می‌توان ویندوز ۹۸ را بر روی دو سیستم متفاوت با پردازنده ،هارد دیسک، و کارت گرافیکی و ... که متفاوت از یکدیگرند نصب و اجرا کرد، اما بر روی هر دو سیستم همان کارهای خود را انجام می‌دهد، زیرا که گردانندهها همان عملکرد پایه را انجام می‌دهند و مهم نیست که بر روی چه سخت‌افزاری، کار می‌کنند.

معماري سخت‌افزار و نرم‌افزار بایوس

البته بایوس، شامل گردانندههای مختلفی است که رابط بین سخت‌افزار و سیستم‌عامل هستند. یعنی بایوس نرم‌افزاری است که تمام آن از روی دیسک بارگذاری نمی‌شود ، بلکه قسمتی از آن ، قبلاً

بر روی تراشه‌های موجود در سیستم یا برروی کارتهای وفق دهنده نصب شده‌اند.
بایوس در سیستم به سه صورت وجود دارد.

۱. Rom bios ،نصب شده بر روی مادربرد.
۲. بایوس نصب شده بر روی کارتهای وفق دهنده (همانند کارت ویدیویی) .
۳. بایوس بارگذاری شده از دیسک (گرداننده‌ها).

چون بایوس مادربرد، مقدمات لازم را برای گرداننده‌ها و نرم‌افزارهای مورد نیاز فراهم می‌کند ، اکثراً به صورت سخت‌افزاری که شامل یک تراشه ROM می‌باشد موجود است.

سالهای پیش هنگامی که سیستم‌عامل داس بر روی سیستم اجرا می‌شد، خود به تنهایی کافی بود و گرداننده‌ای نیاز نداشت. بایوس مادربرد به طور عادی شامل گرداننده‌هایی برای یک سیستم پایه همانند صفحه کلید، فلاپی درایو، هارد دیسک ، پورتهای سریال و موازی و ... است. به جای اینکه برای دستگاههای جدید لازم باشد، که بایوس مادربرد را ارتقاء داد، یک نسخه از گرداننده آن بر روی سیستم‌عامل خود نصب می‌شود تا سیستم‌عامل پیکربندی لازم را در هنگام بوت شدن برای استفاده از آن دستگاه انجام دهد. برای مثال می‌توان CD ROM، Scanner، Printer، گرداننده‌های PC CARD را نام برد. چون این دستگاهها لازم نیست که در هنگام راه‌اندازی فعال باشند ، سیستم ابتدا از هارد دیسک راه‌اندازی می‌شود و سپس گرداننده‌های آنرا بارگذاری می‌کند. البته بعضی از دستگاهها لازم است که در طول راه‌اندازی سیستم‌عامل فعال باشند.

سوالات فصل ۱

۱. PC Base به چه نوع سیستمهایی گفته می‌شود؟
۲. انواع برنامه‌ها و خواص آنها را توضیح دهید.
۳. یک برنامه سیستم از چه قسمتهایی تشکیل شده است؟

۴. یک برنامه بلادرنگ، چه نوع برنامه‌ای است؟

۵. برای چک کردن بیتهای زیر چه نوع ماسکهایی پیشنهاد می‌کنید؟ (متغیر اصلی A است)

الف) بیت پنجم یک باشد.

ب) بیت ششم صفر باشد.

پ) بیت پنجم و ششم یک باشد.

ت) بیت پنجم یا ششم یک باشد.

۶. توسط برنامه‌نویسی متغیر A را به صورت زیر اصلاح کنید.

الف) تمام چهار بیت پایین Not شوند.

ب) بیت یکم همیشه یک و بیت دوم همیشه صفر باشد.

۷. وقفه‌های بایوس را شرح دهید.

۸. انواع بافرها را توضیح دهید.

۹. خروجی برنامه زیر چیست؟

```
int a[5];
int *b;

a[1]=20;
b=a;
cout<<*(b+2);
```