کتاب الکترونیکی رایگان

طراحی مدارهای واسط کامپیوتر

دکتر جواد راستی

مدرس دانشکده مهندسی دانشگاه اصفهان

0101010101010101010101010101010101010101

به نام خدای مهربان

طراحی
مدارهای واسط کامپیوتری

دکتر جواد راستی
مدرس دانشکده مهندسی دانشگاه اصفهان

سرشناسه	:	راستی، جواد، ۱۳۵۹ –
عنوان و نام پدیدآور	:	طراحی مدارهای واسط کامپیوتری/جواد راستی.
مشخصات نشر	:	اصفهان: پیام علوی، ۱۳۹۰.
مشخصات ظاهری	:	۳۰۴ ص.: مصور، نمودار.
شابک	:	978-964-2564-58-3
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	کتابنامه: ص. ۳۰۳.
موضوع	:	رابط سخت‌افزاری
موضوع	:	ریزپردازنده‌ها
موضوع	:	پورت (سیستم الکترونیکی کامپیوتر)
رده بندی کنگره	:	TK۵/۷۸۷/۴ ر۲ط ۱۳۹۰
رده بندی دیویی	:	۳۹۱۶/۶۲۱
شماره کتابشناسی ملی	:	۲۵۳۹۸۲۸

نام کتاب:	طراحی مدارهای واسط کامپیوتری
مؤلف:	دکتر جواد راستی
طرح روی جلد:	علیرضا محسنی

تقدیم به پدر و مادرم

و

خانواده محبوبم

فهرست

عنوان **صفحه**

مقدمه کتاب الکترونیکی

دقایقی از آغاز سال ۱۳۹۲ سپری شده است؛ پس به حساب تقویمی دو سال و به حساب روزشماری یک سال و نیم از عمر این کتاب می‌گذرد!

آن روزها بعد از چند ترم تدریس درس «مدارهای واسط» و با وجود آن همه کتاب مفصل و دقیق (که به تعدادی از آنها در انتهای کتاب اشاره شده است)، منبعی نیافته بودم که برای تدریس مناسب باشد و مفاهیم طراحی مدارهای واسط و مرتبط کردن دستگاه‌ها با کامپیوتر را به سادگی و روشنی بیان کرده باشد. این بود که (جوانی کرده! و) دست به قلم بردم تا آنچه پیش روی شماست نگاشته شود.

این کتاب یک بار به زیور طبع آراسته و خیلی زود هم نسخه‌های آن نایاب شد. من بودم و تقاضاهای فراوان تجدید چاپ و ضیق وقت و تردید که بکنم یا نکنم که ضربه نهایی را صعود عقاب تیزپرواز قیمت کاغذ زد و یک «نه» بزرگ به سینه تجدید چاپ کوبید!

در لحظه‌ای که این سطور را می‌نویسم، به لطف حضرت دوست ۵ جلد کتاب نگاشته‌ام و سه کتاب نیمه که اگر عمری باقی باشد در آینده نزدیک به بار خواهند نشست. در آنچه چاپ شده کوچک‌ترین نفع مادی نبرده‌ام و به آنچه در راه است نیز طمع مادی ندارم. این کاغذهای سیاه شده را زکات علم می‌دانستم و می‌دانم و بس.

با یک محاسبه سرانگشتی دانستم قیمت کتاب تجدید چاپ شده مدارهای واسط پیش از توان مالی بسیاری از دانشجویان است که آنها را نیز چون خواهران و برادران خود دوست می‌دارم و مشکل آنها، مشکل من نیز هست. این بود که دل به دریا زدم و تصمیم گرفتم نسخه الکترونیکی کتاب را به رایگان منتشر کنم. این «دل به دریا زدن» نیز نه از باب طمع مادی که از بیم کپی بی‌مأخذ و نام و نشان است. یادش بخیر که روزی یکی از همکاران، با آب و تاب فراوان سخن از جزوه بسیار کامل و جامع زبان اسمبلی گفت که «نوشته» و حاضر است در اختیار من قرار دهد. با اشتیاق کامل قبول و پیگیری بسیار کردم تا سرانجام، چشمم به جمال جزوه خودم روشن شد که با شور و شوق تایپ کرده بودم! اسم من با لاک غلط‌گیر! محو و نام آن همکار عزیز زینت‌بخش جزوه شده بود...!

به هر روی، امیدوارم که این بار تجربه بهنجاری باشد تا بقیه مؤلفان عزیز نیز ترغیب شوند و نگاشته‌های مفید خود را در اینترنت به اشتراک بگذارند و خیال‌شان از رعایت حق مؤلف آسوده باشد. صفحه‌آرایی این کتاب را از چاپ اصلی (قطع وزیری) تغییر داده‌ام تا تعداد صفحات و حجم کتاب کمتر و دانلود آن ساده‌تر شود.

این کتاب رایگان است؛ اما خواننده این کتاب از باب منّت به من محبت کند و اولاً اگر این نوشتار را مفید یافت تا می‌تواند در نشر آن بکوشد و ثانیاً اگر وسع مادی دارد به اندازه قیمت کتاب‌های مشابه به نیازمندان کمک کند و اگر نه با آموختن یا بخشیدن چیزی به دیگری، این زنجیر را حلقه‌ای دیگر بزند. اگر هیچکدام هم میسر نبود، با صلواتی یا قرائت سوره‌ای از قرآن کریم، نویسنده حقیر این سطور را در بازپرداخت بدهی‌اش به امام انس و جان، حضرت علی بن موسی الرضا (ع) یاری کند که ولی‌نعمت همه ماست.

ارادتمند – جواد راستی – ۱۳۹۲/۱/۱

مقدمه

کامپیوتر را با کدامیک از کاربردهایش می‌شناسید؟! کاربردهای عمومی؟ بازی؟ پخش موسیقی و ویدئو؟ تایپ و طراحی گزارش؟

یا کاربردهای تخصصی‌تر؟ برنامه‌نویسی وب و پایگاه داده‌ها؟ شبیه‌سازی فرآیندها؟ انجام محاسبات پیچیده؟ یا ...

در واقع کامپیوتر به عنوان مهمترین و قوی‌ترین ابزار کنترلی برنامه‌پذیر، سالیان درازی است که جای خودش را در صنعت، امور شهری و زندگی انسان‌های امروزی باز کرده است. یکی از کاربردهای مهم کامپیوتر، استفاده از آن برای کنترل و نظارت بر دستگاههای دیگر است: نظارت بر محیطی که مشخصات آن (نور، رطوبت، دما، فشار و ...) باید کنترل‌شده باشد، کنترل یک روبات پیچیده مانند یک روبات زیردریایی و ...

هرچند با استفاده از میکروکنترلرهای مدرن و نیز مادربوردهای صنعتی امروزه می‌توان پردازش‌های بسیار قوی و سطح بالا (مانند پردازش تصویر) را بدون نیاز به کامپیوتر انجام داد، اما هنوز در بسیاری از کاربردها کنترل یا نظارت کامپیوتری نقش مهمی را ایفا می‌کند؛ به ویژه وقتی پردازش مورد نظر بسیار قوی و شامل بخش‌های مختلفی مانند ارتباط با شبکه، پردازش ویدئو و یا کار با پایگاه داده‌های مفصل باشد.

«طراحی مدارهای واسط کامپیوتری» یکی از مباحث کاربردی رشته‌های مهندسی سخت‌افزار، مهندسی پزشکی و مهندسی برق می‌باشد که نحوه مرتبط کردن دستگاههای مختلف با کامپیوتر را تشریح می‌کند. دامنه این بحث از طراحی کارت‌های کامپیوتری تا طراحی واسط‌هایی برای ارتباط موازی و سریال با کامپیوتر گسترده است. از دیگر مباحث مرتبط با این حوزه می‌توان به نحوه برقراری ارتباط با چاپگر و مودم، چگونگی کنترل دستگاهها از راه دور به کمک تلفن و شبکه و اینترنت و نیز شبکه‌های کنترلی مانند I^2C و CAN و X10 اشاره کرد. حتی اگر بخواهید به جای کامپیوتر از مادربوردهای صنعتی برای کنترل دستگاه‌های پیچیده استفاده کنید، باید با اصول طراحی مدارهای واسط آشنایی داشته باشید.

در این نوشتار تلاش کرده‌ایم مباحث مرتبط با طراحی مدارهای واسط کامپیوتری به زبان ساده و به صورت پایه‌ای بیان شوند. به همین لحاظ ممکن است بعضی مباحث کمی قدیمی به نظرآیند. مثلاً برای توضیح نحوه طراحی کارتهای کامپیوتری، ناگزیر به یکی از پردازنده‌های قدیمی اینتل (۸۰۸۸) که در اولین کامپیوترهای IBM مورد استفاده قرار گرفت، پرداخته‌ایم. این موضوع برای ورود پایه‌ای به بحث پیچیده‌ای مانند طراحی مدارهای واسط ضروری است و به خواننده کمک می‌کند از اصولی که آموخته برای طراحی مدارهای پیچیده استفاده کند. به علاوه بسیاری از تنظیمات کامپیوترهای اولیه، در کامپیوترهای امروزی نیز به چشم می‌خورد. بعضی از سخت‌افزارها که دیگر در کامپیوترهای شخصی دیده نمی‌شوند (مانند اسلات ISA)، هنوز در دستگاهها و مادربوردهای صنعتی کاربرد زیادی دارند. در کنار مباحث پایه‌ای، تا حدودی به مباحث جدید مرتبط با مدارهای واسط مانند گذرگاه‌های جدید نیز پرداخته شده است. بدیهی است مخاطب علاقه‌مند برای دریافت اطلاعات تخصصی‌تر راجع به هر بحث می‌تواند به مراجع طراحی مدارهای واسط که بسیار متنوع هستند و در پایان این نوشته به برخی از آنها اشاره خواهد شد، مراجعه نماید.

امید است این نوشتار بتواند برای مخاطبین مفید واقع شود. در پایان وظیفه خود می‌دانم یادی از استاد گرانقدر دکتر عباس وفایی عضو هیأت علمی دانشگاه اصفهان بنمایم که این سطور به شوق هم‌قدم و هم‌قلم شدن با ایشان به رشته تحریر درآمد.

نظرات و پیشنهادها و انتقادات خود را به آدرس rasti@eng.ui.ac.ir ارسال فرمایید تا نسخه‌های بعدی این نوشتار جامعتر و کاملتر ارائه گردد.

جواد راستی – شهریور ۱۳۹۰

فصل اول

آشنایی با سیستم‌های مبتنی بر پردازنده

به طور کلی هر سیستم کنترلی دارای سه بخش ورودی، پردازشگر و خروجی است (شکل ۱-۱). چنین سیستمی داده‌ها را از دنیای خارج دریافت می کند (بخش ورودی)، روی این داده‌ها عملیاتی انجام می دهد (بخش پردازش) و بر اساس این پردازشها خروجی لازم را تولید می کند (بخش خروجی). کارخانه پارچه بافی یک مثال ساده برای یک سیستم کنترلی است که نخ بعنوان ماده اولیه وارد آن می شود و پس از انجام یک سری عملیات، پارچه بعنوان خروجی سیستم تولید می‌شود.

شکل ۱-۱-اجزای یک سیستم کنترلی

تصمیم‌گیریهای منطقی، بخش اساسی پردازش در سیستمهای کنترلی هستند. در درس مدارهای منطقی با نحوه طراحی بخش پردازشگر سیستمهای الکترونیکی با استفاده از مدارات منطقی ترکیبی و ترتیبی آشنا شده‌اید. عملکرد پردازنده‌ها از نظر کلی با مدارهای منطقی یکسان است و تنها چگونگی ورود و خروج داده‌ها و نحوه انجام پردازش روی آنهاست که باعث متفاوت شدن ویژگیهای آنها می‌شود. ایده اصلی ایجاد و گسترش پردازنده‌ها، قابلیت برنامه‌پذیری آنهاست. یک سیستم کنترلی مبتنی بر پردازنده دارای تعدادی ورودی و خروجی است که تمام پردازشهایی که قرار است روی ورودیها انجام شود به صورت مجموعه ای از دستورات نرم افزاری که برنامه[1] نام دارد، به سیستم داده می‌شود. تغییر نظر کارفرما، تنها منجر به تغییر برنامه سیستم می‌شود که هزینه آن در مقابل هزینه تغییر کلی سیستم بسیار ناچیز است. به بیان دیگر، پیچیدگی طراحی یک سیستم سخت‌افزاری، به طراحی یک برنامه مبدل می‌شود که بسیار ساده‌تر است. این موضوع اساس کار کنترل‌کننده‌های منطقی برنامه‌پذیر[2] که سیستمهایی مبتنی بر پردازنده (شکل ۱-۲) هستند، می‌باشد. در این سیستمها، به جای طراحی بخش پردازشگر سیستم کنترلی به صورت سخت‌افزار سفارشی که مشکلات ذکر شده را به دنبال دارد، پردازش را به یک برنامه (نرم‌افزار) می سپاریم که هم طراحی و پیاده‌سازی و اشکالزدایی و هم تغییر کارکرد آن نسبت به سخت افزار، بسیار ساده‌تر و کم هزینه‌تر می باشد. وظیفه پردازنده اجرای این برنامه است.

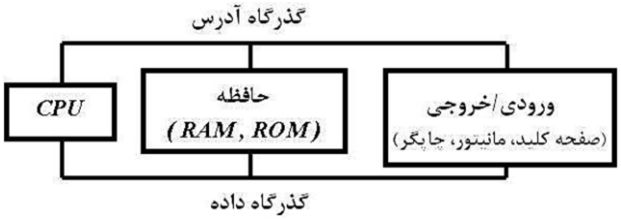
شکل ۱-۲-سیستم‌های منطقی برنامه‌پذیر

اکنون که با ضرورت وجود پردازنده‌ها آشنا شدیم، به بررسی چگونگی عملکرد آنها می‌پردازیم. در ادامه با ساختار کلی یک کامپیوتر به عنوان یک سیستم مبتنی بر پردازنده و سپس با چند مثال با نحوه اجرای برنامه‌ها در پردازنده‌ها آشنا خواهید شد.

[1] Program
[2] Programmable Logic Controller : PLC

یک سیستم پردازنده‌ای چه اجزایی دارد ؟

شما با کامپیوتر به عنوان یک سیستم پردازنده‌ای آشنایی دارید. قطعه CPU داخل کامپیوتر همان پردازنده مورد نظر ماست؛ اما کامپیوتر اجزاء دیگری مانند دیسک سخت، RAM، کارت گرافیکی، صفحه کلید، نمایشگر[1] و ... دارد. تمام این اجزاء را می‌توان در سه بخش اصلی مطابق شکل ۱-۳ (که اجزاء اساسی هر سیستم مبتنی بر پردازنده نیز هستند)، طبقه‌بندی کرد:



شکل ۱-۳- اجزای یک سیستم مبتنی بر پردازنده

۱- *واحد پردازش مرکزی*[2] *(CPU)*

۲- *حافظه*[3]

۳- *وسایل ورودی و خروجی*[4] *(I/O)*

مجموعه دستورات یا برنامه ای که قرار است پردازشهای سیستم ما را انجام دهد، در واحد حافظه ذخیره می شود. کار واحد پردازش مرکزی (که از این به بعد آن را به اختصار پردازنده می نامیم) اجرای خط به خط این برنامه است. کار واحدهای ورودی/خروجی، تبادل اطلاعات با دنیای خارج است.

از نظر ساختاری می‌توان کامپیوتر را مانند یک انسان در نظرگرفت که داده‌هایی را از حواس خود (ورودی) مانند بویایی و بینایی می‌گیرد، در مغز خود (پردازنده) با توجه به اطلاعاتی که در ذهن (حافظه) دارد، پردازشهایی (شاید ناخودآگاه) را روی آنها انجام می‌دهد و متناسب با آنها عکس‌العمل (خروجی) نشان می‌دهد.

مانند اجزای بدن که از طریق رگها و اعصاب با هم مرتبط هستند، اجزاء داخلی یک کامپیوتر از طریق مجموعه‌ای از سیمها به نام گذرگاه یا باس[5] با یکدیگر در ارتباط می‌باشند.

اکنون به شرح مختصری راجع به بخشهای ذکر شده می‌پردازیم.

کار پردازنده چیست؟

کار پردازنده اجرای مجموعه دستورات یا برنامه‌ای است که در حافظه ذخیره شده است. برای این کار پردازنده باید:

- ✓ دستورات را به ترتیب از حافظه دریافت یا واکشی کند (Fetch)،
- ✓ معنای آن را درک کند (Decode)
- ✓ آن را اجرا کند (Execute).

برای درک سه مرحله فوق، کار یک نوازنده موسیقی را در نظر بگیرید. او برای اجرای یک آهنگ باید سمبل نت موسیقی را از روی یک‌صفحه که نتهای یک آهنگ روی آن نوشته شده‌اند، بخواند (Fetch)، اینکه آن سمبل نشان‌دهنده کدام نت است را درک کند (Decode)، و سپس آن نت را بنوازد (Execute).

---
[1] Monitor
[2] Central Processing Unit : CPU
[3] Memory
[4] Input / Output Devices : I/O
[5] Bus

برنامه چگونه به پردازنده داده می شود؟

اما پردازنده نه سمبلهای موسیقی را می‌فهمد و نه با هیچکدام از زبانهای محاوره‌ای ما آشناست! تنها زبانی که می‌توان به وسیله آن با پردازنده ارتباط برقرار کرد، زبان ارقام دودویی است. در واقع به دلیل سهولت استفاده از اجزاء الکترونیکی مانند لامپ خلأ، رله و ترانزیستور که به صورت دوحالته نیز قابل استفاده هستند، از ابتدا مدارات منطقی و پردازنده‌ها با دستگاه دودویی استاندارد شدند.

کوچکترین واحد اطلاعاتی برای یک پردازنده، یک رقم دودویی یا یک بیت[۱] است که حاوی یک سیگنال منطقی (صفر یا یک) می باشد[۲]. تمام اطلاعات کامپیوتری از مجموعه‌ای از بیتها تشکیل می‌شوند. هر ۸ بیت یک بایت[۳] نامیده می‌شود که یک واحد اطلاعاتی مهم در پردازنده‌هاست. واحد مهم دیگر، Nibble است که از ۴ بیت تشکیل می‌شود.

زبان ماشین[۴]

برای ارتباط با یک پردازنده باید از ارقام دودویی استفاده کنیم. تمام دستورات، ورودیها و خروجیها در این قالب قرار می‌گیرند که آن را زبان ماشین یا زبان صفر و یک می‌نامیم. زبان ماشین، پایینترین سطح گفتگو با پردازنده است.

در ابتدا به دلیل عدم گستردگی کاربرد پردازنده‌ها، از زبان صفر و یک برای برنامه‌نویسی پردازنده‌ها استفاده می‌شد. مثلاً فرض کنید می‌خواهیم با استفاده از یک پردازنده فرضی، دو عدد را بخوانیم و با هم جمع کنیم و نتیجه را نشان دهیم. با مطالعه برگه اطلاعات پردازنده مورد نظر، متوجه می‌شویم که در این پردازنده رشته "10110001" به عنوان دستور خواندن عدد از ورودی در نظر گرفته شده است؛ یعنی پردازنده با دریافت این رشته صفر و یک از حافظه (به عنوان دستور)، یک عدد را از ورودی می خواند. به همین ترتیب دستور جمع دو عدد "00110011" و دستور ارسال نتیجه به خارج "00100010" است. بنابراین برنامه فوق به شکل زیر در می آید :

```
10110001
10110001
00110011
00100010
```

همانطور که می بینید برنامه نویسی و همچنین فهم این برنامه بسیار دشوار است.

پس از گسترده شدن کاربرد پردازنده‌ها در سیستمهای مختلف، نیاز به زبانی که علیرغم نزدیکی کافی به سطح ماشین، دارای دستورات ساده تر و قابل فهم تر باشد به شدت احساس می شد.

زبان اسمبلی[۵]

در زبان اسمبلی که به همین هدف عرضه شد، برای هر دستور یک کلمه معادل که شبیه به کلمات انگلیسی است، وجود دارد؛ مثلاً در پردازنده فوق کلمه IN معادل رشته "10110001" در نظر گرفته شده است که با معنای این دستور که خواندن عدد از ورودی (INPUT) است نیز همخوانی دارد. برنامه قبلی با استفاده از معادل اسمبلی دستورات به صورت زیر در می آید:

---

[1] Binary digIT : BIT

[۲] در تراشه‌های با استاندارد TTL، ولتاژ حدود صفر تا ۰/۸ ولت برای بیان صفر منطقی و ولتاژ حدود ۳ تا ۵ ولت برای بیان یک منطقی به کار می رود. صفر و یک منطقی می تواند با دو مقدار متفاوت جریان (مثلاً ۴ و ۲۰ میلی آمپر) نیز نمایش داده شود.

[3] Byte

[4] Machine Language

[5] Assembly Language

```
IN
IN
ADD
OUT
```

برنامه نویسی به زبان اسمبلی، کار را برای برنامه نویس راحت‌تر می کند؛ اما پردازنده تنها زبان صفر و یک را متوجه می شود. نرم افزار اسمبلر[1] کار ترجمه دستورات زبان اسمبلی به دستورات زبان ماشین را بر عهده دارد.

زبان اسمبلی با ایجاد تحول در برنامه نویسی پردازنده‌ها، سالها بعنوان زبان برنامه‌نویسی متداول پردازنده‌ها به کار می رفت. اما از دشواری نسبی برنامه نویسی به زبان اسمبلی که بگذریم، برنامه‌نویس این زبان باید از ساختمان داخلی پردازنده‌ای که برای آن برنامه می نویسد، کاملاً آگاه باشد. بعلاوه برنامه ای که به زبان اسمبلی یک پردازنده نوشته می شود، روی پردازنده دیگری قابل اجرا نیست؛ چون هر برنامه اسمبلی (یا زبان ماشین) با توجه به ساختار داخلی یک پردازنده نوشته می شود که با پردازنده‌های دیگر متفاوت است. به همین دلیل زبان ماشین و زبان اسمبلی را زبانهای وابسته به ماشین[2] می گویند.

زبانهای سطح بالا[3]

دستورات زبانهای برنامه نویسی سطح بالا مانند زبانهای پاسکال، C، فرترن و ... علاوه بر شباهت زیاد به زبان انگلیسی که باعث سادگی برنامه نویسی می شود، به کاربر اجازه می دهد بدون نیاز به اطلاع از ساختار داخلی پردازنده، برنامه‌های خود را بنویسد؛ مثلاً دستور پاک کردن صفحه نمایش کامپیوتر در زبانهای پاسکال و C ، دستور clrscr (Clear Screen) است؛ اما برای انجام همین کار به زبان اسمبلی کامپیوتر این دستورات باید نوشته شوند :

```
MOV     AH,6H
MOV     AL,25
MOV     CX,0
MOV     DH,24
MOV     DL,79
MOV     BH,14H
INT     10H
```

این مثال به خوبی تفاوت برنامه نویسی به زبان سطح بالا و زبان اسمبلی را نشان می‌دهد.

چون پردازنده تنها دستورات زبان ماشین را درک می کند، از نرم افزار مترجم (کامپایلر)[4] برای تبدیل دستورات زبان سطح بالا به سطوح پایین‌تر استفاده می شود.

ذکر این نکته ضروریست که زبان اسمبلی هنوز در مواردی مانند وقتی که حجم و زمان اجرای برنامه‌ها مهم است، مورد استفاده قرار می گیرد، به طوری که حتی گاهی بخشی از یک برنامه سطح بالا را با استفاده از دستورات اسمبلی می نویسیم. این کار برای کاهش حجم کد ماشین نهایی و یا زمان اجرای آن و نیز استفاده مؤثر از سخت‌افزار انجام می‌شود.

---

[1] Assembler
[2] Machine Dependent
[3] High Level Languages : HLL
[4] Compiler

اجزاء داخلی پردازنده‌ها

پردازنده برای انجام سه وظیفه اصلی خود یعنی واکشی، رمزگشایی و اجرای دستورات، به امکاناتی مجهز است که در زیر به مهمترین آنها می‌پردازیم :

۱- ثبات‌ها[1]**:** ثبات‌ها، حافظه‌های کوچک و سریعی هستند که داخل پردازنده قرار دارند و برای ذخیره موقت داده‌ها و دستکاری آنها به کار می‌روند. این ثباتها بسته به نوع پردازنده می‌توانند ۸ بیتی، ۱۶بیتی، ۳۲ بیتی، ۶۴ بیتی و ... باشند. هرچه تعداد و اندازه ثباتها بیشتر باشد، کارایی پردازنده بالاتر است.

پردازنده‌ها دارای ثباتهای متنوعی هستند که در زیر چند نوع از آنها را معرفی می‌کنیم:

ثبات انباره[2]**:** یک ثبات همه‌منظوره[3] است که در انواع دستورات حسابی و منطقی و انتقال داده‌ها بعنوان برگه کاری موقت به‌کار می‌رود.

ثبات حالت (پرچم)[4]**:** بیتهای این ثبات بیانگر حالت پردازنده بعد از اجرای دستورات است. مثلاً یکی از بیتهای این ثبات، پرچم صفر[5] است که اگر نتیجه یک عمل منطقی یا حسابی برابرصفر باشد، مقدار آن برابر یک می‌شود؛ یعنی مقدارZF بعد از انجام یک عملیات منطقی یا حسابی، بیانگر صفر بودن یا صفر نبودن نتیجه عملیات است.

از بیتهای ثبات پرچم که در اکثر پردازنده‌ها وجود دارند می‌توان به پرچم صفر، پرچم سرریز[6]، پرچم علامت[7] و پرچم رقم نقلی[8] اشاره کرد. کاربرد اصلی ثبات پرچم در تصمیم‌گیریهای لازم بعد از انجام یک عمل داخلی پردازنده است.

ثبات شمارنده برنامه[9] **:** چون دستورات یک برنامه باید به ترتیب اجرا شوند، پردازنده باید به طریقی بداند دستور بعدی که باید اجرا کند کدام است. کار ثبات شمارنده برنامه، نگهداری آدرس دستور بعدی است که قرار است توسط پردازنده اجرا شود. با اجرای هر دستور، پردازنده به طور خودکار یک واحد به این ثبات اضافه می‌کند تا به دستور بعدی اشاره کند. با کمک این ثبات و گذرگاههای داده و آدرس، پردازنده دستورات را از حافظه دریافت می‌کند.

گفتیم که پردازنده، دستوراتی را که از حافظه دریافت می کند، اجرا می نماید. این دستورات چگونه اجرا می شوند؟

۲- واحد محاسبه و منطق[10]**:** این واحد پردازنده که اصطلاحاً به آن ALU گفته می‌شود، قسمتی از پردازنده است که مسؤول انجام اعمال ریاضی مانند جمع، تفریق، ضرب و تقسیم و اعمال منطقی مانند AND ، OR و NOT می‌باشد.

واحد ALU تنها یک حسابگر است و برای عملکرد درست باید کنترل شود.

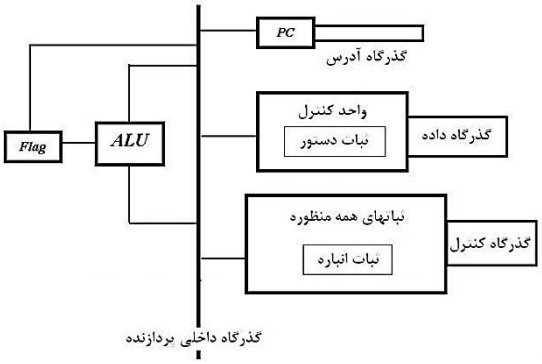
۳ - واحد کنترل[11]**:** کار این بخش، کنترل تمام فعالیتهای پردازنده است.

---

[1] Registers
[2] Accumulator Register
[3] ثباتهای همه منظوره (General Purpose Registers) ثباتهایی هستند که برای انجام هر عملیاتی داخل پردازنده می توانند مورد استفاده قرار گیرند.
[4] Flag Register
[5] Zero Flag : ZF
[6] Oveflow Flag : OF
[7] Sign Flag : SF
[8] Carry Flag : CF
[9] Program Counter : PC یا Instruction Pointer : IP
[10] Arithmetic & Logic Unit : ALU
[11] Control Unit : CU

یکی از مهمترین قسمتهای واحد کنترل، ثبات رمزگشای دستور[1] است. می‌توان این قسمت را به‌عنوان یک فرهنگ لغت تصور کرد که معنای هر دستور و مراحلی را که پردازنده باید برای اجرای آن در پیش بگیرد، مشخص می‌کند. دستورات پس از واکشی از حافظه وارد این ثبات می‌شوند تا پس از رمزگشایی اجرا شوند.

بلوک دیاگرام داخلی پردازنده را در شکل ۱-۴ می‌بینید.



شکل ۱-۴-بلوک دیاگرام داخلی پردازنده

سرعت پردازنده به چه معناست؟

در آخر به این نکته توجه کنید که پردازنده از نظر عملیاتی، مانند یک مدار ترتیبی منطقی همزمان (سنکرون) عمل می‌کند و تمام اعمال آن با یک موج مربعی که پالس ساعت نام دارد، هماهنگ می‌شود. فرکانس پالس ساعت، یکی از معیارهای کارایی پردازنده و بیانگر سرعت آن خواهد بود. معیار مشابه دیگر به صورت MIPS[2] یعنی تعداد دستورات قابل اجرا توسط پردازنده بر حسب میلیون دستور در ثانیه، بیان می‌شود.

برای بحث دقیقتر در مورد چگونگی اجرای یک دستورالعمل در پردازنده، به مراجع معماری کامپیوتر مراجعه کنید.

نقش حافظه در سیستمهای کامپیوتری

اطلاعاتی که باید توسط پردازنده پردازش شود، اعم از دستورات و بعضی داده‌های ورودی/خروجی، باید در حافظه ذخیره شود. در سیستمهای پردازنده‌ای معمولاً از دو نوع حافظه استفاده می‌شود :

حافظه فقط خواندنی(*ROM*)[3]: اطلاعات این حافظه با قطع برق از میان نمی‌رود. بنابراین معمولاً (همانطور که از نام آن برمی‌آید) برای ذخیره اطلاعات دایمی و ثابت به کار می‌رود. مثلاً برنامه‌ای که حافظه کامپیوتر را تست می‌کند و حتماً اجرای آن را هنگام روشن کردن کامپیوتر خود دیده‌اید، باید پس از هر بار روشن شدن کامپیوتر اجرا شود؛ بنابراین باید در ROM کامپیوتر ذخیره شود[4]. معمولاً برای نوشتن در حافظه ROM به دستگاه مخصوصی به نام برنامه‌ریز[5] نیاز است.

حافظه خواندنی و نوشتنی (*RAM*[6] یا *RWM*[7]): اطلاعات این نوع حافظه با قطع برق از میان می‌رود. به علاوه برای نوشتن در آن نیازی به دستگاه برنامه‌ریز نیست. بنابراین برای ذخیره داده‌های موقت مانند متغیرهای یک برنامه یا در

---

[1] Instruction (Decoder) Register : IR
[2] Milion Instruction Per Second : MIPS
[3] Read Only Memory : ROM
[4] به برنامه فوق که عملیات ابتدایی سیستم از جمله تست حافظه کامپیوتر را انجام می‌دهد، اصطلاحاً بایوس (Basic Input Output System: BIOS) گفته می‌شود.
[5] Programmer
[6] Random Access Memory : RAM
[7] Read/Write Memory

سطوح بالاتر برای نگهداری سیستم عامل یا نرم‌افزارهای مختلف مانند انواع بازیها یا برنامه‌های کاربردی دیگر به کار می‌رود.

اطلاعات ROM بر خلاف RAM دایمی است و با قطع برق از بین نمی‌رود؛ به همین دلیل، گاهی ROM را حافظه غیرفرّار[1] هم می‌نامند.

در بخشهای بعدی با عملکرد و نحوه نوشتن و خواندن اطلاعات از حافظه بیشتر آشنا خواهیم شد.

نقش وسایل ورودی/خروجی در سیستمهای کامپیوتری

در حقیقت پردازنده‌ها برای رفع نیازهای کاربران طراحی شده‌اند؛ به این منظور باید داده‌هایی به پردازنده وارد شده و پس از انجام عملیات مورد نظر، نتایج از پردازنده خارج شوند. مجموعه‌ای شامل پردازنده، گذرگاهها و وسایل ورودی/خروجی، این وظیفه را در سیستمهای کامپیوتری به عهده دارند.

وسایل ورودی می‌توانند بسیار ساده مانند یک کلید دوحالته و یا پیچیده مانند صفحه کلید یا ماوس باشند. به طور مشابه، وسایل خروجی نیز انواع ساده مانند دیود نوری تا انواع پیچیده مانند نمایشگر را دربرمی‌گیرند. در ادامه این فصل با چگونگی استفاده از وسایل ورودی/خروجی ساده در سیستمهای پردازنده‌ای آشنا می‌شوید.

گذرگاهها در سیستمهای کامپیوتری

در هر کامپیوتر به طور معمول سه نوع گذرگاه وجود دارد : گذرگاه داده[2]، گذرگاه آدرس[3] و گذرگاه کنترلی[4].

تمام اطلاعاتی که باید در یک سیستم کامپیوتری جابجا شوند، از گذرگاه داده عبور می‌کنند. گذرگاه آدرس مشخص می‌کند که گذرگاه داده در هر لحظه باید در اختیار چه وسیله‌ای باشد و گذرگاه کنترلی این ارتباط را نظم می‌بخشد.

اهمیت گذرگاهها به حدی است که یکی از مهمترین معیارهای کارایی پردازنده‌ها، پهنای (تعداد خطوط) گذرگاه آدرس و گذرگاه داده آنها و تنوع سیگنالهای گذرگاه کنترلی آنهاست. کمی بیشتر با این گذرگاهها آشنا می‌شویم :

گذرگاه داده: همانطور که گفته شد، گذرگاه داده محل عبور تمام اطلاعاتی است که باید داخل سیستم جابجا شوند؛ بعنوان مثال برای انتقال دستورات از حافظه به پردازنده، برای خواندن اطلاعات از ورودی و نیز برای ارسال اطلاعات به خروجی باید از گذرگاه داده استفاده شود.

پهنای گذرگاه داده (یعنی اینکه گذرگاه دادهٔ یک پردازنده چند بیتی است) تا آنجا مهم است که یکی از مهمترین معیارهای طبقه‌بندی پردازنده‌ها به شمار می‌رود که ابتدایی‌ترین نوع آن ۴ بیتی (مانند پردازنده‌های ۴۰۰۴ و ۴۰۴۰) بود و سپس انواع کاملتر ۸ بیتی (Z80، ۶۸۰۰، ۸۰۸۵)، ۱۶ بیتی (۸۰۸۶، ۸۰۸۸، ۸۰۲۸۶) و ۳۲ بیتی (۸۰۳۸۶، ۸۰۴۸۶، پنتیوم) به بازار آمد. وقتی گفته می‌شود پردازنده ۸۰۸۵، هشت بیتی است یعنی می تواند هر بار (در هر سیکل کاری) ۸ بیت را پردازش کند؛ به بیان دیگر ورود و خروج اطلاعات در آن باید در قالبهای ۸ بیتی صورت گیرد.

---

[1] Non Volatile Memory : NV-Memory

[2] Data Bus
[3] Address Bus
[4] Control Bus

واضح است که قدرت پردازش یک پردازنده به طور مستقیم با اندازه گذرگاه داده آن در ارتباط است؛ به عنوان مثال، یک پردازنده ۸ بیتی در هر سیکل کاری ۸ بیت را می‌تواند پردازش کند؛ اما یک پردازنده ۱۶ بیتی در همین مدت زمان می‌تواند ۱۶ بیت را مورد پردازش قرار دهد، به تعبیری سرعت پردازش ۲ برابر حالت قبل است.

گذرگاه داده یک گذرگاه دوجهته[1] است؛ یعنی داده‌ها از طریق آن هم می‌توانند به پردازنده وارد و هم از آن خارج شوند.

در بعضی پردازنده‌ها، پهنای گذرگاه داده خارجی (پینهای داده پردازنده) با پهنای گذرگاه داده داخلی (مسیر انتقال داده‌ها درون پردازنده) متفاوت است؛ مثلاً پردازنده ۸۰۸۸ دارای گذرگاه داده داخلی ۱۶ بیتی و گذرگاه داده داخلی ۸ بیتی است، در حالی که گذرگاههای داده داخلی و خارجی پردازنده هر دو ۱۶ بیتی‌اند.

گذرگاه آدرس: همانطور که گفته شد، گذرگاه داده محل مشترک عبور اطلاعات سیستم است. پرسشی که بلافاصله به ذهن می آید این است که در هر لحظه چه کسی حق استفاده از این گذرگاه مشترک را دارد ؟

برای حل این مشکل، هر پردازنده دارای تعدادی پین آدرس است که با گذاشتن یک شماره روی آن، هویت وسیله ای را می تواند از گذرگاه داده استفاده کند را معلوم می کند. به مجموعه این پینها گذرگاه آدرس پردازنده گفته می‌شود.

نحوه استفاده از گذرگاه آدرس به این صورت است که طراح سیستم به همه واحدهایی که می خواهند از گذرگاه داده استفاده کنند (خانه‌های حافظه و تمام ورودیها و خروجیها)، یک شماره (آدرس) می دهد. در هنگام عملکرد سیستم، شماره یا آدرسی که پردازنده روی گذرگاه آدرسش قرار می دهد نشان دهنده شماره واحدی است که حق استفاده از گذرگاه داده را دارد. مثلاً اگر پردازنده‌ای ۸ خط آدرس (۸ پین آدرس) داشته باشد، با قرار گرفتن عدد 00110010 روی این پینها توسط پردازنده، به کل سیستم اعلام می شود که در این لحظه واحدی به شماره 00110010 (که ممکن است یک خانه حافظه، یک ورودی یا یک خروجی باشد) حق استفاده از گذرگاه داده (خواندن یا نوشتن) را دارد. این شماره در هنگام طراحی سیستم توسط طراح به این واحد داده شده است.

از آنجا که گذرگاه آدرس برای اعلان شماره واحدی که می تواند از گذرگاه داده استفاده کند به کار می‌رود، هر چه تعداد خطوط آن بیشتر باشد، پردازنده می‌تواند مقدار بیشتری حافظه یا تعداد بیشتری ورودی/خروجی را شناخته و از آنها استفاده کند. به‌عبارت دیگر، تعداد خطوط آدرس یک پردازنده، معیاری از تعداد واحدهایی است که می‌تواند آدرس‌دهی کند.

این تعداد واحدها همیشه توانی از ۲ است؛ به این ترتیب که یک پردازنده با n خط آدرس، می‌تواند به ۲^n واحد مختلف آدرس بدهد[2]. مثلاً پردازنده Z80، ۱۶ خط آدرس دارد، این ۱۶ خط یا ۱۶ بیت می‌توانند $۲^{۱۶}$ یا ۶۵۵۳۶ حالت (آدرس دودویی) را بسازند، بنابراین Z80 می‌تواند تا ۶۵۵۳۶ واحد مختلف (حافظه یا ورودی/خروجی) (که هر کدام حاوی یک بایت می‌باشد) را آدرس‌دهی کند. اصطلاحاً گفته می‌شود که فضای آدرس‌دهی Z80 برابر ۶۵۵۳۶ بایت یا ۶۴ کیلوبایت است. توجه کنید که در اندازه‌گیری ظرفیت حافظه و فضای آدرس‌دهی، معنای کلمه کیلو، با سایر علوم

[1] Bi-directional

[2] چون یک عدد دودویی با n رقم، می‌تواند ۲^n مقدار مختلف داشته باشد

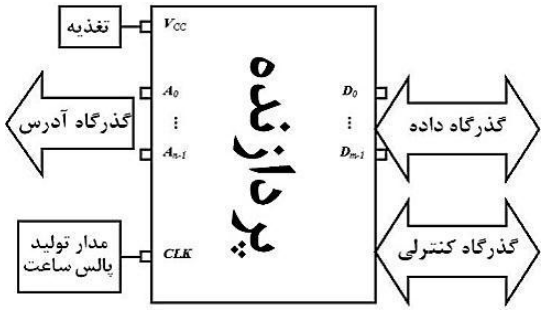
متفاوت می باشد؛ به طوری‌که هر کیلوبایت شامل ۱۰۲۴ بایت است؛ به طور مشابه هر مگابایت برابر ۱۰۲۴ کیلو بایت، هرگیگابایت برابر ۱۰۲۴ مگابایت و هر ترابایت برابر ۱۰۲۴ گیگابایت است.

پردازنده 8086/88 دارای ۲۰ خط آدرس و فضای آدرس‌دهی یک مگابایت است؛ بنابراین آدرسهای حافظه در آن ۲۰ بیتی هستند. در آینده خواهیم دید چگونه با ثباتهای آدرس‌دهی ۱۶ بیتی پردازنده و سیستم سگمنت/آفست، این آدرس ۲۰ بیتی تولید می‌شود.

چون گذرگاه آدرس فقط برای آدرس‌دهی و توسط پردازنده به کار می‌رود، بنابراین یک گذرگاه یک‌جهته[1] و سمت آن به سوی خارج پردازنده است.

گذرگاه کنترل: این گذرگاه شامل پینهایی از پردازنده است که برای کنترل ارتباطات داخل سیستم مورد استفاده قرار می‌گیرد و در پردازنده‌های مختلف، نوع و تعداد آنها متفاوت است. مثلاً تعدادی از این سیگنالها مشخص می‌کنند آیا پردازنده مشغول دریافت اطلاعات است یا ارسال اطلاعات ؟ دسته دیگر مشخص می‌کنند وسیله‌ای که آدرس آن توسط گذرگاه آدرس مشخص گردیده، از نوع حافظه است یا ورودی/خروجی ؟ و ....

شکل عمومی یک پردازنده با n خط آدرس و m خط داده را در شکل ۱-۵ مشاهده می‌کنید.



شکل ۱-۵-شکل عمومی یک پردازنده

اکنون که با اجزاء یک سیستم پردازنده‌ای آشنا شده‌اید، وقت آن رسیده که با چند مثال ساده، چگونگی عملکرد پردازنده در اجرای برنامه‌ها را مورد بررسی قرار دهیم.

[1] Uni-directional

فصل دوم

طراحی سیستم‌های مبتنی بر پردازنده

در این بخش می‌خواهیم چگونگی اجرای برنامه‌های مختلف در یک پردازنده را به صورت گام به گام بررسی کنیم. مثالهای این بخش عملکرد عمومی پردازنده‌ها را تشریح خواهند کرد.

برای طراحی یک سیستم مبتنی بر پردازنده دو کار اساسی باید انجام شود:

شناخت سخت‌افزار که شامل شناخت عملکرد پینهای پردازنده مورد نظر، نحوه عملکرد حافظه‌ها و اتصال آنها به پردازنده و سخت‌افزارهای ورودی/خروجی و ... می‌شود.

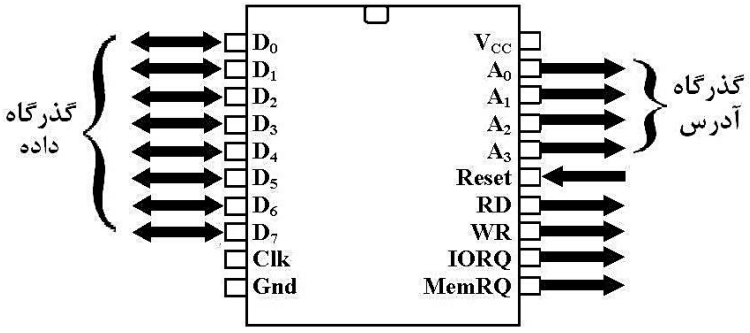
شناخت نرم‌افزار که شامل شناخت ساختار داخلی پردازنده به ویژه ثباتهای آن و نیز آشنایی با قواعد زبان اسمبلی پردازنده مورد نظر می باشد.

معرفی یک پردازنده فرضی

برای طراحی مثالهای این بخش از یک پردازنده فرضی مطابق شکل ۲-۱ استفاده خواهیم کرد که دارای ۴ خط آدرس و ۸ خط داده (شکل زیر)، ثبات انباره A و ثباتهای همه‌منظوره B و C و D است. واضح است که این ثباتها ۸ بیتی هستند (چرا؟).

در برنامه نویسی به زبان اسمبلی پردازنده فرضی ما چند قاعده باید رعایت شود (این قواعد در زبان اسمبلی اکثر پردازنده ها وجود دارد):

- ✓ یکی از عملوند های عمل جمع، حتماً باید در ثبات A باشد.
- ✓ حاصل عمل جمع در ثبات A قرارمی گیرد.
- ✓ نمی توان در دستور جمع مستقیماً به حافظه مراجعه کرد.
- ✓ تنها رابط پردازنده با خارج آن، ثبات A است.



شکل ۲-۱-پایه‌های یک پردازنده نوعی

در ادامه با این نکات برخورد خواهیم کرد.

به طور کلی برای طراحی سیستمها روال زیر را در پیش خواهیم گرفت :

الف) نوشتن برنامه مورد نظر.

ب) بررسی آنچه در پردازنده برای اجرای برنامه انجام می شود.

ج) طراحی سخت افزار سیستم بر اساس گام (ب).

اتصال حافظه به پردازنده

حافظه محل ذخیره برنامه است؛ بنابراین هر سیستمی حتماً دارای یک حافظه است و ضروریست قبل از وارد شدن به جزییات طراحی سیستم، با نحوه ذخیره و بازیابی اطلاعات در حافظه‌ها آشنا شویم.

عملکرد حافظه

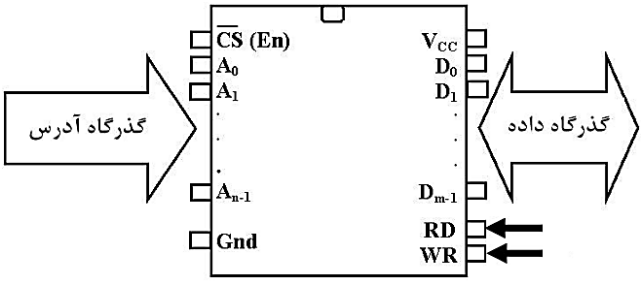
همانطور که قبلاً ذکر شد، حافظه محل ذخیره دستورات و نیز بعضی داده‌های ورودی و خروجی است.

مشخصات مهم یک حافظه عبارتند از :

الف) ظرفیت حافظه: بیانگر حجم اطلاعاتی است که می‌تواند در حافظه ذخیره شود و معمولاً بر حسب بیت یا بایت بیان می‌شود.

ب) ساختار حافظه: حافظه از تعدادی ثبات یکسان (خانه) تشکیل می‌شود که در هر خانه یک قلم داده ذخیره می‌شود. منظور از ساختار حافظه، حجم هر یک از خانه‌ها و تعداد خانه‌هاست.

حجم هر خانه حافظه از یک تا ۸ و گاهی ۱۶ بیت معمول است. تعداد خانه‌های یک حافظه نیز معمولاً توانی از ۲ است. نمای کلی یک حافظه را در شکل ۲-۲ مشاهده می‌کنید.



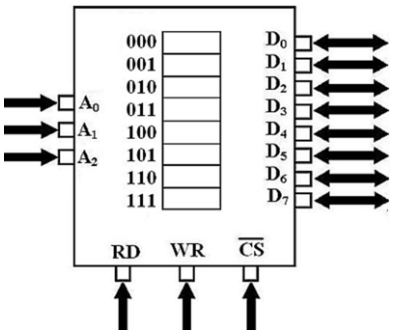
شکل ۲-۲- شکل کلی یک حافظه

خطوط A_0 تا A_{n-1} که گذرگاه آدرس حافظه نامیده می‌شود، آدرس خانه‌ای از حافظه که باید اطلاعات آن خوانده یا در آن اطلاعاتی نوشته شود را مشخص می‌کند. طول هر قلم اطلاعاتی (حجم هر خانه) که اطلاعات آن از طریق خطوط داده D0 تا D_{m-1} رد و بدل می‌شود، m بیت است.

پینهای RD و WR برای تعیین نحوه تبادل اطلاعات با حافظه (خواندن یا نوشتن) به‌کارمی‌روند. اگر سیگنال RD را فعال کنیم یعنی می‌خواهیم از حافظه اطلاعاتی بخوانیم. با فعال کردن سیگنال WR به حافظه اعلام می‌کنیم که اطلاعاتی در یکی از خانه‌هایش نوشته می‌شود. در هر دو عمل خواندن و نوشتن، تبادل اطلاعات با خانه‌ای از حافظه که شماره (آدرس) آن روی گذرگاه آدرس حافظه گذاشته می‌شود انجام می‌گیرد.

به عنوان مثال حافظه نوعی شکل ۲-۳ را در نظر بگیرید. این حافظه ۸ خط داده (D_0 تا D_7) و ۳ خط آدرس (A_0 تا A_2) دارد؛ چون در این حافظه می‌توان 2^3 خانه حافظه را آدرس‌دهی کرد (با ۳ بیت می‌توان 2^3 عدد دودویی مختلف را مشخص کرد) و ظرفیت هر خانه نیز ۸ بیت است (چون ۸ پین برای تبادل اطلاعات با خارج دارد)، پس این حافظه دارای $8(2^3)$ خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۸ بایت یا ۶۴ بیت است.

به عنوان یک مثال عملی، تراشه 2732 ، ۸ خط داده (D_0 تا D_7) و ۱۲ خط آدرس (A_0 تا A_{11}) دارد؛ چون در این حافظه می‌توان 2^{12} خانه حافظه را آدرس‌دهی کرد (با ۱۲ بیت می‌توان 2^{12} عدد دودویی مختلف را مشخص کرد)، پس این حافظه دارای $۴ \times ۱۰۲۴ = ۲^{۱۰} \times ۲^{۲}$ (یا ۴ کیلو) خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۴ کیلوبایت یا ۳۲ کیلوبیت ($۴ \times ۱۰۲۴ \times ۸$) است که دو رقم آخر شماره آن نیز این مطلب را نشان می‌دهد.



شکل ۲-۳- یک حافظه نوعی

پرسش : ظرفیت حافظه شکل ۲-۳ بر حسب بیت چقدر است؟

ذخیره و بازیابی اطلاعات در حافظه

اکنون فرض کنید می‌خواهیم در خانه شماره ۷ حافظه 2732، عدد دودویی ۸ بیتی 10110001 را ذخیره کنیم. کافی است شماره خانه فوق یعنی معادل دودویی ۱۲ بیتی عدد ۷ (000000000111) را روی پینهای A_0 تا A_{11} و عدد مورد نظر (10110001) را روی پینهای D_0 تا D_7 تراشه قرار داده و پین WR (Write) را فعال کنیم تا عدد فوق در خانه شماره ۷ ذخیره شود. اگر این حافظه از نوع RAM باشد تا زمان قطع برق و اگر از نوع ROM باشد به طور دائم این داده در این خانه باقی خواهد ماند.

برای خواندن محتویات یک خانه مثلاً همان خانه شماره ۷ ، کافی است عدد ۱۲ بیتی معادل ۷ یعنی 000000000111 را روی پینهای آدرس حافظه قرار داده و پین RD (Read) را فعال کنیم. عدد ذخیره شده در خانه ۷ که در اینجا 10110001 است روی پینهای D_0 تا D_7 (گذرگاه داده حافظه) قرار می‌گیرد.

توجه کنید که چه برای نوشتن در حافظه و چه برای خواندن از آن، پین فعال ساز تراشه (CS یا En) باید فعال باشد.

مراحل خواندن و نوشتن در حافظه را می‌توان به صورت زیر خلاصه کرد؛

برای نوشتن عدد a در آدرس b حافظه:

- ✓ آدرس b را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- ✓ عدد a را روی پینهای داده حافظه (گذرگاه داده حافظه) قرار می‌دهیم.
- ✓ پین WR حافظه را فعال می‌کنیم تا عدد a در آدرس b حافظه ثبت شود.

برای خواندن محتویات خانه حافظه به آدرس x:

- ✓ آدرس x را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- ✓ پین RD حافظه را فعال می‌کنیم تا محتویات خانه حافظه به آدرس x روی پینهای داده حافظه (گذرگاه داده حافظه) قرار گیرد.
- ✓ پینهای داده حافظه (گذرگاه داده حافظه) را می‌خوانیم.

پرسش) راجع به دو مفهوم گسترش طول کلمه و گسترش طول آدرس حافظه تحقیق کنید.

ج) زمان دسترسی: مشخصه سرعت حافظه و عبارتست از زمانی که بعد از گذاشتن آدرس خانه مورد نظر روی خطوط آدرس و فعال کردن پین RD به طول می‌انجامد تا داده خانه فوق روی خطوط داده تراشه حافظه به صورت پایدار ظاهر شود.

اکنون که با نحوه خواندن و نوشتن در حافظه آشنا شدیم، به مثالهایی از طراحی سیستم می‌پردازیم. اولین مثال راجع به یک سیستم بسیار ساده است.

مثال ۱) اولین کاری که می‌خواهیم پردازنده فرضی ما انجام دهد این است که مقدار ثباتهای B و C خودش را با هم جمع کند و در ثبات D قرار دهد.

یادآوری می کنیم برای طراحی سیستمها به ترتیب زیر عمل می کنیم:

الف) نوشتن برنامه مورد نظر.

ب) بررسی آنچه در پردازنده برای اجرای برنامه انجام می شود.

ج) طراحی سخت افزار سیستم بر اساس گام (ب).

به عنوان گام اول طراحی این سیستم، برنامه آن را می نویسیم.

این برنامه می تواند به سادگی یک خط برنامه باشد :

D ⟵ B + C

اما طبق قواعد (الف) و (ب) که مطرح شد، نمی‌توان جمع دو ثبات را به‌طور مستقیم انجام داد؛ یعنی دستور فوق نامعتبر است و با توجه به اینکه ثبات A ،انباره یعنی برگه موقت کار است، مراحل زیر باید انجام شود :

A ⟵ B
A ⟵ A + C
D ⟵ A

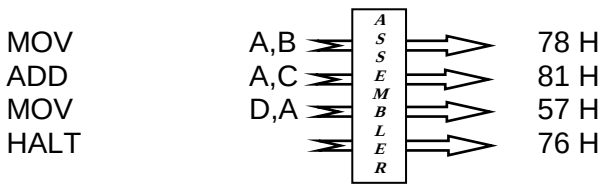
مراحل ذکر شده باید به زبانی که پردازنده متوجه شود، یعنی زبان صفر و یک یا همان زبان ماشین به پردازنده دیکته شود. در عمل برنامه هیچگاه مگر در موارد خاص به زبان ماشین نوشته نمی‌شود و این کار در پایین‌ترین سطح با زبان اسمبلی انجام می‌شود. معادل اسمبلی فرضی دستورات بالا به صورت زیر است (مشابه این دستورات در زبان اسمبلی پردازنده‌های مختلف وجود دارد) :

```
MOV     A,B
ADD     A,C
MOV     D,A
HALT
```

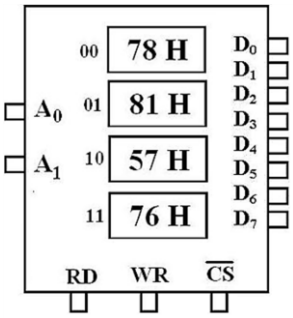
قبلاً گفته شد که زبان اسمبلی شباهتهایی به زبان انگلیسی دارد؛ مثلاً دستور MOV A,B یعنی MOVe B to A .

همچنین دستور HALT به معنای **قطع** اجرای برنامه است.

این دستورات با کمک اسمبلر پردازنده ما به زبان ماشین تبدیل می‌شوند. کدهای زبان ماشین معمولاً در قالب هگزادسیمال نشان داده می‌شوند.



پس از تبدیل دستورات برنامه به زبان ماشین، آنها را در یک تراشه حافظه ذخیره می‌کنیم (شکل ۲-۴ را ببینید) تا پردازنده یکی یکی آنها را واکشی و اجرا کند؛ مثلاً وقتی پردازنده کد یک بایتی 78 H (یا 01111000 B) را از حافظه دریافت می‌کند، با رمزگشایی آن متوجه می‌شود که باید محتویات ثبات B را در ثبات A قرار دهد و به همین ترتیب بقیه دستورات را اجرا می‌کند تا به کد 76 H (کد HALT) برسد. این کد به پردازنده می‌گوید که اجرای برنامه به پایان رسیده است. به کدهای فوق که پردازنده را به اجرای عملیات‌های گوناگون وامی‌دارند، کد عمل[1] گفته می‌شود. چون برنامه فوق چهار بایتی است (تعداد کدهای عمل آن ۴ بایت است)، از یک حافظه ۴ بایتی که دو خط آدرس دارد استفاده می‌کنیم.



شکل ۲-۴-ذخیره برنامه اول در حافظه

طراحی سخت افزار

اکنون گام (الف) به پایان رسیده است؛ برنامه (نرم‌افزار) سیستم آماده است و نوبت به طراحی سخت‌افزار می‌رسد. در این گام باید عملکرد پردازنده برای اجرای این برنامه را بررسی کنیم تا در گام بعد بتوانیم بر اساس این دانسته‌ها، سخت افزار نهایی را طراحی کنیم.

سخت‌افزار سیستم فوق بسیار ساده است و فقط باید مکانیسمی در آن پیش‌بینی شود که برنامه زبان ماشین را که در حافظه ذخیره شده، به موقع به پردازنده برساند. برای طراحی این مکانیسم باید با بررسی نحوه اجرای برنامه (گام **ب**) ببینیم پردازنده با چه روشی بایتهای محتوی برنامه را از حافظه درخواست می‌کند؟

پردازنده چگونه کدهای عمل را از حافظه می‌خواند؟

همانطور که قبلاً گفته شد، در پردازنده‌ها ثباتی به نام شمارنده برنامه یا PC وجود دارد که وظیفه آن، نگهداری شماره دستوری است که قرار است اجرا شود. این شماره دستور در بسیاری از پردازنده‌ها ازجمله پردازنده فرضی ما، شماره خانه‌ای از حافظه است که دستور بعدی را در خود نگه می‌دارد[2]. چون در پردازنده مورد بررسی ما گذرگاه آدرس ۴ بیتی است، بنابراین تمام شماره‌ها (از جمله شماره دستور بعدی که در ثبات PC ذخیره می‌شود) نیز ۴ بیتی است؛ بنابراین ثبات PC در این پردازنده یک ثبات چهار بیتی است.

اولین اصل طراحی سیستم در مورد نحوه درخواست کدهای عمل از حافظه توسط پردازنده است:

اصل اول طراحی سیستم: پردازنده برای درخواست یک کد عمل از حافظه، مقدار ثبات PC را روی خطوط آدرسش گذاشته و سیگنال RD خود را فعال می کند؛ در این لحظه پردازنده انتظار دارد که کد عمل مورد نظر که همان دادهٔ ذخیره شده در PCامین خانه حافظه است، روی گذرگاه داده اش بیاید تا بتواند آن را بخواند و اجرا کند؛ وظیفه طراح سیستم برآوردن این انتظار پردازنده است.

[1] Operational Code : OP-Code

[2] بعدها خواهید دید که در پردازنده‌های سری 80xx اینتل دقیقاً این‌گونه نیست.

فرض می‌کنیم در پردازنده فرضی ما، بعد از روشن شدن سیستم، PC برابر صفر می‌شود[۱]. بنابراین پردازنده به خانه صفر حافظه مراجعه و اولین دستور را از آنجا درخواست می‌کند. این درخواست توسط پردازنده با قرار دادن آدرس 0000 (یعنی مقدار PC) روی گذرگاه آدرس و فعال کردن سیگنال خواندن (RD) و سپس خواندن گذرگاه داده به هدف خواندن اولین کد عمل برنامه انجام می‌شود.

پس از خواندن اولین دستور برنامه (78 H) از حافظه و اجرای آن، PC به طور خودکار یک واحد افزایش می‌یابد و برای خواندن خانه بعدی از حافظه به کار می‌رود؛ به این صورت که پردازنده محتوای PC یعنی 0001 را روی گذرگاه آدرس قرار داده و پس از فعال کردن سیگنال RD ، گذرگاه داده را که انتظار دارد محتوی دستور بعدی (81 H) باشد می‌خواند.

کد HALT (76 H) به پردازنده می‌گوید که باید افزایش PC و درخواست دستور بعدی از حافظه را متوقف کند.

در زیر مراحل اجرای این برنامه را می بینید :

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000	78 H(کد عمل صفر)
	---- **اجرای کد** 78 h ----	
0001	0001	81 H(کد عمل یک)
	---- **اجرای کد** 81 h ----	
0010	0010	57 H(کد عمل دو)
	---- **اجرای کد** 57 h ----	
0011	0011	76 H(کد عمل سه)
	---- **اجرای کد** 76 h ----	

با اجرای کد 76h (دستور HALT) پردازنده دیگر به مقدار PC اضافه نمی‌کند تا دستور بعدی را بخواند؛ بنابراین اجرای برنامه قطع می‌شود.

بنا بر آنچه گفته شد، مکانیسم پردازنده برای دریافت دستورات از حافظه، قرار دادن PC روی گذرگاه آدرس و فعال کردن سیگنال RD است؛ پس باید سیستمی طراحی شود که وقتی پردازنده اعمال فوق را انجام داد، در پاسخ آن بایت ذخیره شده در PCامین خانه حافظه که در واقع کد عمل دستوری است که باید اجرا شود را روی گذرگاه داده پردازنده قرار دهد. به بیان دیگر، باید اتصالات بین پردازنده و حافظه را به نحوی برقرار کنیم که وقتی پردازنده یک آدرس را روی گذرگاه آدرس گذاشته و سیگنال RD را فعال می کند، بایت مورد نظر از حافظه به گذرگاه داده پردازنده متقل شود.

با گذرگاههای داده چه باید کرد؟ کدهای عمل باید از طریق گذرگاه داده پردازنده به آن وارد شوند. از طرف دیگر حافظه کدهای عمل را از طریق گذرگاه داده‌اش به خارج ارسال می‌کند. بنابراین طبیعی است که:

گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.

[۱] مقدار هر ثبات بعد از روشن شدن سیستم، یکی از مشخصات خاص پردازنده‌هاست که Register Wake up Value نام دارد.

این موضوع با گفته قبلی ما که در سیستم یک گذرگاه داده بیشتر وجود ندارد مطابقت دارد. توجه کنید اگر هر دو جزء سیستم برای ارتباط با هم بخواهند گذرگاههای داده مجزا داشته باشند، سیستم ما مملو از سیمهای ارتباطی می‌شود که هزینه و خطاپذیری سیستم را بسیار افزایش می‌دهد.

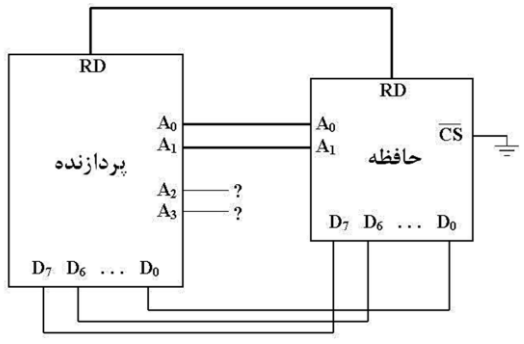
برای دانستن چگونگی اتصال گذرگاههای آدرس پردازنده و حافظه به هم، به مراحل اجرای برنامه به ویژه آنچه روی گذرگاه آدرس پردازنده می‌آید، توجه کنید. اولین نکته‌ای که جلب توجه می‌کند این است که خطوط آدرس A_2 و A_3 در طول اجرای برنامه صفر هستند و تغییر نمی‌کنند. بنابراین دقت خود را به بیتهای A_0 و A_1 معطوف می‌کنیم.

اکنون روند واکشی دستور اول را ببینید؛ هنگامی که پردازنده روی خطوط A_0 و A_1 عدد 00 را گذاشته و سیگنال RD خود را فعال می‌کند، انتظار دارد کد 78h روی گذرگاه داده‌اش بیاید. از طرف دیگر وقتی روی خطوط آدرس A_0 و A_1 حافظه عدد 00 گذاشته شده و سیگنال RD آن فعال شود، حافظه محتویات خانه صفر خود (78h) را روی گذرگاه داده‌اش می‌گذارد.

واکشی دستور دوم نیز همین روند را دنبال می‌کند؛ هنگامی که پردازنده روی خطوط A_0 و A_1 عدد 01 را گذاشته و سیگنال RD خود را فعال می‌کند، انتظار دارد کد 81h روی گذرگاه داده‌اش بیاید. از طرف دیگر وقتی روی خطوط آدرس A_0 و A_1 حافظه عدد 01 گذاشته شده و سیگنال RD آن فعال شود، حافظه محتویات خانه صفر خود (81h) را روی گذرگاه داده‌اش می‌گذارد.

بررسی آنچه گفته شد نشان می‌دهد اتصالات لازم برای این سیستم به شرح زیر است:

- ✓ اتصال بیت به بیت گذرگاه داده پردازنده به گذرگاه داده حافظه
- ✓ اتصال خطوط A_0 و A_1 پردازنده به خطوط A_0 و A_1 حافظه
- ✓ اتصال سیگنال RD حافظه به سیگنال RD پردازنده



شکل ۲-۵-سخت‌افزار سیستم اول

به شکل ۲-۵ و نحوه عملکرد سیستم توجه کنید. در ابتدا پردازنده مقدار PC یعنی 0000 را روی خطوط آدرس خود یعنی A_0 تا A_3 می‌گذارد؛ اما تنها خطوط A_0 و A_1 به حافظه متصل هستند که مقدار آنها 00 است. سپس پردازنده برای خواندن کد عمل، پین RD خود که به پین RD حافظه متصل است را فعال می‌کند. با فعال شدن پین RD حافظه توسط پردازنده، محتویات خانه 00 حافظه (که قبلاً مقدار H 78 در آن ذخیره شده) روی گذرگاه داده حافظه که به گذرگاه داده پردازنده متصل است، قرار می‌گیرد و پردازنده آن را می‌خواند. چون پردازنده می‌داند که اولین بایتی که از حافظه می‌خواند، یک کد عمل است، این بایت را مستقیماً به ثبات دستورالعمل ارسال می‌کند. با افزایش PC ، بایتهای بعدی برنامه که در حافظه ذخیره شده‌اند، خوانده و سپس اجرا می‌شوند.

چون خطوط A_2 و A_3 پردازنده تا پایان اجرای برنامه صفر هستند، تنها اتصال خطوط پایینی آدرس پردازنده یعنی A_0 و A_1 به حافظه کافی است. با خطوط A_2 و A_3 چه باید کرد ؟ فعلاً نیازی به اتصال آنها نیست؛ اما بعداً برای تکمیل کار به آنها نیاز خواهیم داشت. **پرسش)** حافظه مورد استفاده در این مثال ۴ بایت ظرفیت دارد؛ اما حتی اگر ظرفیت آن بیشتر هم بود، باز برنامه را با شروع از خانه صفر حافظه ذخیره می کردیم. چرا؟

پرسش) اگر کد HALT در انتهای برنامه استفاده نشود، چه می شود؟

زمانبندی سیستم

ذکر یک نکته ضروری به نظر می رسد؛ وقتی پردازنده می خواهد یک دستور را از حافظه بخواند، شماره آن دستور را روی گذرگاه آدرس خود گذاشته و سیگنال RD را فعال می کند و سپس گذرگاه داده خود را می خواند. پردازنده در این لحظه **انتظار دارد** که محتوی PC–امین خانه حافظه روی گذرگاه داده آن آمده باشد؛ در حالی که اگر حافظه سیستم کندتر از پردازنده باشد (زمان دسترسی حافظه طولانی باشد که معمولاً چنین است)، عمل خواندن PC–امین خانه حافظه به طول می انجامد و ممکن است باعث شود اطلاعات خوانده شده از گذرگاه داده توسط پردازنده معتبر نباشد. بعدها راجع به چگونگی لحاظ کردن مسایل زمانبندی در طراحی یک سیستم بحث خواهیم کرد.

مثال ۲) هر بار که برنامه مثال ۱ اجرا شود، یک عدد ثابت در ثبات D ذخیره می‌شود. چون در ابتدای کار B و C همیشه یک مقدار دارند (Wake up Value).

پس برنامه را به نحوی اصلاح می کنیم که B و C ابتدا با مقادیر دلخواه (مثلاً ۴۲ و ۶۸) مقداردهی و سپس با هم جمع شوند. برنامه مورد نظر به همراه کدهای عمل آن در زیر آمده است.

MOV	B,42	ASSEMBLER	06 h	2A h
MOV	C,68		0E h	44 h
MOV	A,B		78 h	
ADD	A,C		81 h	
MOV	D,A		57 h	
HALT			76 h	

توجه کنید که در زبان ماشین معمولاً کدها در قالب هگزادسیمال نشان داده می‌شوند؛ به‌همین دلیل عدد ۴۲ به صورت 2A H نشان داده شده است.

می‌بینید که کد عمل بار کردن یک مقدار ثابت در ثبات B ، دو بایتی است که بایت اول (06 H) که کد عمل اصلی نام دارد، نشان می‌دهد که بایت بعدی (2A H) باید در ثبات B ذخیره شود. همچنین توجه کنید که کد بارگذاری ثبات B با مقدار ثابت (06 H) با کد بارگذاری ثبات C با مقدار ثابت (0E H) متفاوت است.

این برنامه ۸ بایت کد عمل دارد که باید در حافظه ذخیره شود؛ بنابراین از یک حافظه ۸ بایتی استفاده می کنیم که ۳ خط آدرس دارد.

روند اجرای این برنامه با برنامه قبل تفاوت مهمی دارد؛ در برنامه قبلی هنگامی که کد 78h از حافظه خوانده می‌شود، پردازنده آن را اجرا می کند (محتویات ثبات B را به ثبات A منتقل می کند). اما در این برنامه وقتی پردازنده کد 06 را از حافظه می‌خواند، متوجه می‌شود باید عدد ثابتی را در ثبات B کپی کند؛ این عدد چند است؟! چون پردازنده

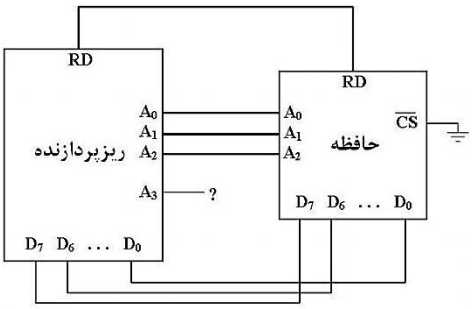
هنوز نمی‌داند این عدد ثابت چقدر است، باید کد عمل بعدی که این عدد را نشان می‌دهد، از حافظه بخواند. بنابراین اجرای این دستور تا خواندن کد عمل بعدی به تأخیر می‌افتد.

پرسش) کد عمل دستور MOV B, 43 و MOV C, 29 را بنویسید.

قسمتی از روند اجرای این برنامه در ذیل دیده می‌شود.

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000	06 H(کد عمل صفر)
0001	0001	2A H(کد عمل یک)
	---- **اجرای کد** 2A h 06 h ----	
0010	0010	0E H(کد عمل دو)
0011	0011	44 H(کد عمل سه)
	---- **اجرای کد** 44 h 0E h ----	
....	...	...

پرسش) روند اجرای برنامه را تکمیل کنید و به کمک آن نشان دهید سخت‌افزار سیستم مانند شکل ۲-۶ است:



شکل ۲-۶-سخت‌افزار سیستم دوم

در سخت‌افزار مثال اول که حافظه دو خط آدرس A_0 و A_1 دارد، این دو خط به خطوط پایین آدرس پردازنده (A_0 و A_1) متصل شده‌اند. در سخت‌افزار مثال دوم که حافظه سه خط آدرس A_0 و A_1 و A_2 دارد، باز این خطوط به خطهای پایین آدرس پردازنده (A_0 و A_1 و A_2) متصل شده‌اند. این یک قانون کلی است:

خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.

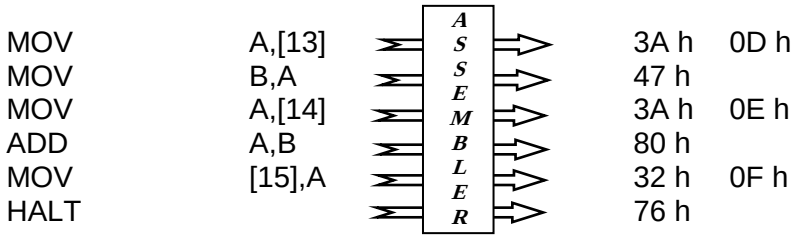
دلیل این موضوع ساده است؛ روند تغییرات خطوط پایین آدرس پردازنده (A_1A_0 در مثال ۱ و $A_2A_1A_0$ در مثال ۲) با روند تغییرات خطوط آدرس حافظه یکسان است. مثلاً در مثال ۲ پردازنده برای خواندن ۸ کد عمل ابتدای برنامه از آدرس 000 تا آدرس 111 را روی خطوط آدرس $A_2A_1A_0$ قرار می‌دهد. از سوی دیگر، برای خواندن ۸ کد عملی که در حافظه ذخیره شده است، باید از آدرس 000 تا آدرس 111 روی خطوط آدرس حافظه قرار گیرد. بنابراین برای هماهنگ شدن عملکرد حافظه با پردازنده، باید خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل شوند.

با خطوط بالای آدرس پردازنده چه باید کرد؟ در مثالهای بعدی این موضوع را بررسی خواهیم کرد.

بنابراین تاکنون از مثالهای اول و دوم، قوانینی برای طراحی سیستم یافته‌ایم که در مثالهای بعدی بدون بحث از آنها استفاده می کنیم:

- ✓ گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.
- ✓ خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
- ✓ سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.

مثال ۳) این مثال ما را با نحوه خواندن و نوشتن در حافظه آشنا می‌کند. برنامه‌های مثال ۱ و ۲ به درستی اجرا می‌شوند، اما نمی‌توانیم نتیجه اجرا را مشاهده کنیم؛ چون نتیجه اجرا در ثبات D که یکی از ثباتهای داخلی پردازنده است ذخیره می‌شود. به علاوه اعداد ورودی را نمی‌توان جز با تغییر برنامه، تغییر داد. در حالی که سیستمی مطلوب است که بتوان با تغییر ورودیها، خروجیهای مختلفی از آن گرفت. برنامه مثال ۲ را به نحوی اصلاح می‌کنیم که محتویات خانه‌های شماره ۱۳ و ۱۴ حافظه را جمع و حاصل را در خانه شماره ۱۵ حافظه ذخیره کند:

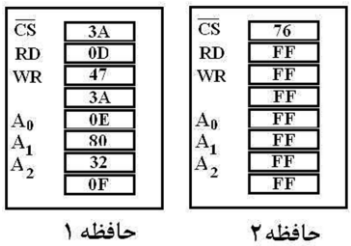


توجه کنید که دستور MOV A, 13 عدد ۱۳ را در ثبات A ذخیره می‌کند؛ در حالی که دستور MOV A, [13]، عددی که در خانه شماره ۱۳ حافظه ذخیره شده را در ثبات A قرار می‌دهد. با مقایسه این برنامه با برنامه قبلی واضح است کد عمل این دو دستور نیز با هم متفاوت است. این موضوع یکی از قوانین معمول زبانهای اسمبلی است[۱].

کد عمل دستور MOV A,[13] دو بایتی است که بایت اول آن (کد عمل اصلی که 3A h است) نشان می‌دهد عددی که در خانه حافظه‌ای به شماره کد عمل دوم (0D H یا ۱۳دهدهی) قرار دارد باید در ثبات A ذخیره شود. همچنین توجه کنید که مطابق آنچه در قواعد زبان اسمبلی پردازنده فرضی ما گفته شد، تنها یک ثبات (معمولاً ثبات انباره) می‌تواند برای تبادل اطلاعات با حافظه مورد استفاده قرار گیرد که در پردازنده فرضی ما ثبات A است؛ بنابراین دستور MOV B,[13] نامعتبراست و به جای آن باید از دو دستور اول برنامه استفاده کرد.

پرسش) چرا کد عمل اصلی دستورات اول و پنجم با هم متفاوت است؟ چرا کد عمل اصلی دستورات اول و سوم مشابه است؟

تعداد کدهای عمل، ۹ بایت است که در یک حافظه ۸ بایتی جا نمی‌گیرد. بنابراین باید از یک حافظه ۱۶ بایتی با چهار خط آدرس استفاده کنیم. فرض کنید فقط حافظه‌های ۸ بایتی در اختیار داریم؛ بنابراین باید از دو تراشه حافظه ۸ بایتی استفاده کنیم که هر کدام ۳ خط آدرس دارند (در انتهای این مثال خواهیم گفت چرا چنین فرضی را در نظر گرفته‌ایم). نحوه ذخیره سازی اطلاعات را در شکل ۲-۷ می‌بینید.



شکل ۲-۷-ذخیره برنامه مثال ۳ در دو حافظه

[۱] در بعضی زبانهای اسمبلی (مانند اسمبلی پردازنده Z80)، به جای علامت [ ] از علامت ( ) استفاده می‌شود.

معمولاً خانه‌های خالی حافظه‌ها حاوی 11111111 (FF h) هستند.

چگونه این دو حافظه باید به پردازنده متصل شوند؟

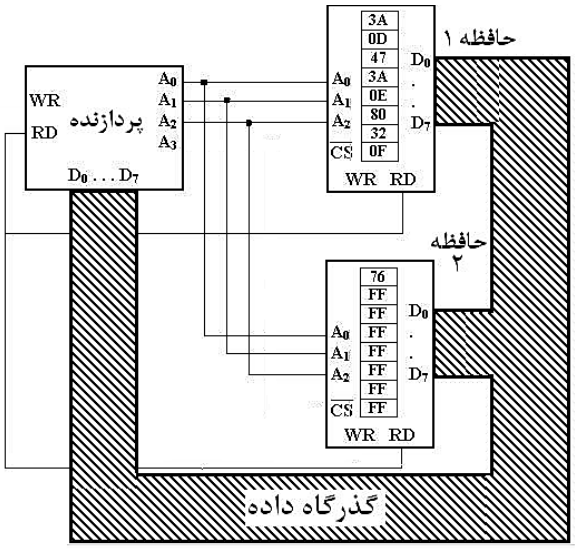
طبق قرار قبلی، گذرگاههای داده این دو حافظه باید به هم متصل و مشترک شده و به گذرگاه داده پردازنده متصل شوند.

آنچه باقی می‌ماند نحوه اتصال خطوط آدرس پردازنده برای آدرس‌دهی مناسب به این حافظه هاست. پردازنده ۴ خط آدرس و حافظه‌ها هرکدام ۳ خط آدرس دارند. طبق قرار قبلی، خطوط آدرس حافظه‌ها باید به خطوط پایین آدرس پردازنده و سیگنالهای RD نیز به هم متصل می‌شوند.

شکل ۲-۸ اتصال ابتدایی بین این دو حافظه و پردازنده را نشان می‌دهد.

برای فهم چگونگی تکمیل اتصالات، باید عملکرد پردازنده برای اجرای این برنامه را بررسی کنیم. قبلاً گفته شد که پردازنده برای دریافت کد عمل از حافظه مقدار PC را روی گذرگاه آدرس خود گذاشته و سیگنال RD را فعال می‌کند.

دستورات خواندن اطلاعات از حافظه (مانند دستور MOV A, [13]) چگونه اجرا می‌شوند؟



شکل ۲-۸-سخت‌افزار ابتدایی مثال ۳

اصل دوم طراحی سیستم: پردازنده برای خواندن محتویات یکی از خانه‌های حافظه، شماره (آدرس) آن خانه را روی گذرگاه آدرس خود گذاشته و سیگنال RD خود را فعال می‌کند. در این لحظه پردازنده انتظار دارد محتویات خانه حافظه ذکرشده روی گذرگاه داده‌اش ظاهر شود.

دستورات نوشتن اطلاعات در حافظه (مانند دستور MOV [15], A) چگونه اجرا می‌شوند؟

اصل سوم طراحی سیستم: پردازنده برای نوشتن مقداری در یکی از خانه‌های حافظه، شماره (آدرس) آن خانه را روی گذرگاه آدرس خود و آن مقدار را روی گذرگاه داده خود گذاشته و سیگنال WR خود را فعال می‌کند.

مراحل اجرای این برنامه به صورت زیر است:

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	* 0000	3A(دریافت کد عمل)
0001	* 0001	0D(دریافت کد عمل)
	---- **اجرای کد** 3A 0D ----	
0001	1101	**محتویات خانه ۱۳ حافظه**
0010	* 0010	47(دریافت کد عمل)
	---- **اجرای کد** 47 h ----	
0011	* 0011	3A(دریافت کد عمل)
0100	* 0100	0E(دریافت کد عمل)
	---- **اجرای کد** 3A 0E ----	
0100	1110	**محتویات خانه ١٤ حافظه**
0101	* 0101	80(دریافت کد عمل)
	---- **اجرای کد** 80 h ----	
0110	* 0110	32(دریافت کد عمل)
0111	* 0111	0F(دریافت کد عمل)
	---- **اجرای کد** 32 0F ----	
0111	1111	**محتویات ثبات** A
	(که پردازنده روی گذرگاه داده قرار می دهد.)	
1000	1000	76(دریافت کد عمل)
	---- **اجرای کد** 76 h ----	

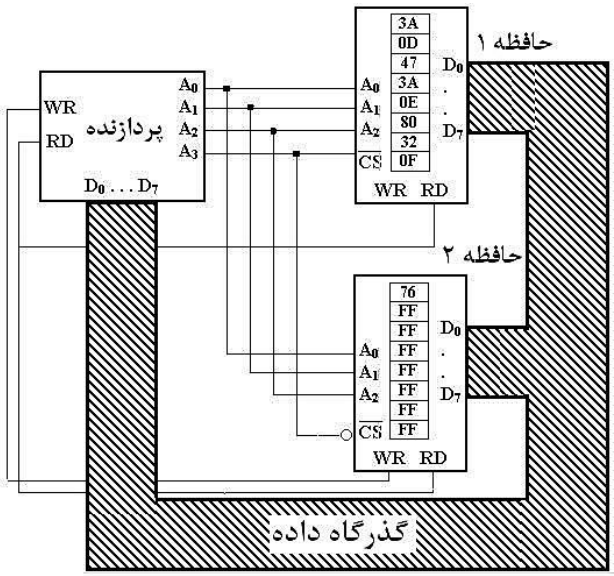
پرسش) با توجه به اصول دوم و سوم طراحی سیستم، نحوه اجرای دستور 3A 0E و دستور 32 0F را توضیح دهید.

با دقت در روند اجرای برنامه، مشکل مهمی در مدار رسم شده به چشم می‌آید. وقتی پردازنده می‌خواهد دستور اول و دستور آخر را از حافظه واکشی کند، در هر دو حالت روی خطوط پایین آدرس خود ($A_2A_1A_0$) آدرس 000 قرار می‌دهد. چون این خطوط به خطوط آدرس هر دو حافظه متصل است، هر دو حافظه می‌خواهند محتویات اولین خانه خود را روی گذرگاه داده قرار دهند که طبیعتاً مشکل ایجاد می‌کند که نشان می‌دهد هنوز اتصالات مربوط به حافظه تکمیل نشده است.

برای حل این مشکل به آنچه برای اجرای این برنامه روی گذرگاه آدرس می‌آید دقت کنید. در کنار تعدادی از این آدرسها علامت * گذاشته شده است. با دقت در این آدرسها می‌توان دریافت که اولاً در تمام این آدرسها بیت پرارزش آدرس (A_3) صفر است؛ ثانیاً تمام دستوراتی که با این آدرسها مورد مراجعه قرار گرفته‌اند در حافظه اول قرار دارند.

اکنون به آدرسهایی توجه کنید که علامت * ندارند. بیت A_3 در تمام این آدرسها برابر 1 بوده و ضمناً خانه‌هایی که به کمک این آدرس مورد مراجعه قرار می‌گیرند، همگی در حافظه دوم قرار دارند.

نتیجه مهمی به دست می‌آید: «هنگامی که پین A_3 صفر است پردازنده با خانه‌های حافظه اول کار می‌کند. وقتی این پین یک می‌شود، پردازنده با خانه‌های حافظه دوم تبادل اطلاعات می‌کند». به عبارت دیگر، هنگامی که پردازنده پین A_3 خود را صفر می‌کند باید حافظه اول فعال و حافظه دوم غیرفعال شود. وقتی پردازنده پین A_3 خود را یک می‌کند، باید حافظه اول غیرفعال و حافظه دوم فعال شود. فعالسازی حافظه‌ها از طریق پین $\overline{CS}$ انجام می‌شود. شکل ۲-۹ اتصال مناسب برای استفاده از این ویژگی را نشان می دهد.



شکل ۲-۹-سخت‌افزار نهایی مثال ۳

در این سیستم از گسترش طول آدرس استفاده شده است.

شاید این سؤال برای شما پیش آمده باشد که وقتی در اینجا فقط با حافظه‌های ۸ خانه ای سروکار داریم، معنای خانه شماره ۱۳ حافظه چیست ؟

معمولاً تمام بایتهای فضای آدرس‌دهی به دنبال هم شماره‌گذاری و خوانده می شوند؛ مثلاً در این پردازنده که چهار خط آدرس (۱۶ مکان آدرس‌دهی شده) داریم، فرض می‌کنیم فضای آدرس‌دهی از صفر تا ۱۵ پشت سر هم قرار دارد که حافظه ۸ خانه‌ای اول، ۸ آدرس ابتدایی و حافظه ۸ خانه دوم، ۸ آدرس دوم را پوشش می‌دهد. مثلاً منظور از خانه شماره ۱۳ حافظه، **خانه شماره ۵ حافظه دوم** است. پردازنده برای درخواست ارتباط با این خانه عدد 1101 (۱۳) را روی گذرگاه آدرسش قرار می‌دهد.

پرسش) اگر به جای دو حافظه هشت خانه‌ای، از چهار حافظه چهارخانه‌ای استفاده کنیم، مکان خانه شماره ۱۳ حافظه در این پیکربندی را تعیین کنید.

چگونه این سیستم با وجود اتصال گذرگاههای داده حافظه‌ها به هم درست کار می‌کند؟ از بیت A_3 برای انتخاب حافظه استفاده شده است؛ وقتی پردازنده ۸ بایت اول برنامه که در حافظه ۱ ذخیره شده را درخواست می‌کند، PC که روی گذرگاه آدرس قرار می‌گیرد از 0000 تا 0111 تغییر می‌کند. می‌بینید که برای خواندن ۸ بایت اول برنامه، بیت A_3 پردازنده صفر است؛ بنابراین حافظه ۱ فعال و حافظه ۲ غیرفعال است (به $\overline{CS}$ها دقت کنید) و محتویات حافظه ۱ روی گذرگاه داده مشترک قرار می‌گیرد؛ اما وقتی پردازنده بایت شماره ۸ (1000) را درخواست می‌کند (یعنی این آدرس را روی گذرگاه آدرس خود می‌گذارد) یا با خانه‌های ۱۳ (1101) یا ۱۴ (1110) یا ۱۵ (1111) فضای آدرس‌دهی کار می‌کند، بیت A_3 «یک» بوده و حافظه اول فعال و حافظه دوم غیرفعال می‌شود.

پرسش) چون می‌خواهیم در خانه شماره ۱۵ بنویسیم، حافظه استفاده شده در این آدرس باید از نوع RAM باشد. مکان خانه شماره ۱۵ را در حافظه‌های متصل شده مشخص کنید.

پرسش) با توجه به اینکه حافظه دوم از نوع RAM است، اگر برق سیستم قطع شود، کد 76 H نیز پاک می‌شود و با وصل مجدد برق باید راهی برای اضافه کردن این کد به انتهای برنامه اندیشید. یک راه نرم‌افزاری این است که در ابتدای برنامه دستوراتی بنویسیم که کد 76h را در انتهای برنامه (در خانه مناسب حافظه RAM) بنویسد. روش سخت‌افزاری، اضافه کردن یک تراشه ثبات با محتویات 76h به سیستم است. در این صورت باید اتصالات سیستم به نحوی انجام شود که با وصل شدن برق به سیستم، محتویات ثبات فوق در خانه مناسب حافظه RAM کپی شود. این دو راه را پیاده‌سازی و با هم مقایسه کنید.

پرسش) چرا سیگنال WR پردازنده فقط به حافظه ۲ متصل شده است ؟

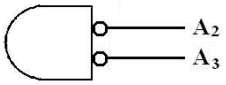
پرسش) اگر آخرین کد برنامه این سیستم، کد HALT نباشد چه می شود؟

پرسش) اگر به جای دو حافظه ۸ خانه‌ای، چهار حافظه ۴ خانه‌ای داشته باشیم، به کمک مراحل اجرای برنامه شرح دهید چگونه این چهار حافظه باید به پردازنده متصل شوند؟

راهنمایی: دو خط آدرس حافظه‌ها را به خطوط پایین آدرس پردازنده (A_1A_0) متصل و از خطوط بالای آدرس پردازنده (A_3A_2) برای انتخاب یک حافظه در هر لحظه استفاده کنید. برای این کار، به نمایش دودویی آدرسهای 0000 تا 1111 در کادر مقابل توجه کنید.

A3	A2	A1	A0	
0	**0**	0	0	= 0
0	**0**	0	1	= 1
0	**0**	1	0	= 2
0	**0**	1	1	= 3
0	1	0	0	= 4
0	1	0	1	= 5
0	1	1	0	= 6
0	1	1	1	= 7
1	**0**	0	0	= 8
1	**0**	0	1	= 9
1	**0**	1	0	= 10
1	**0**	1	1	= 11
1	1	0	0	= 12
1	1	0	1	= 13
1	1	1	0	= 14
1	1	1	1	= 15

می‌بینید که وقتی پردازنده با چهار خانه اول حافظه تبادل داده می‌کند، A_3A_2 برابر 00 است؛ بنابراین هنگامی که A_3A_2 برابر 00 است باید حافظه اول فعال و بقیه حافظه‌ها غیرفعال شوند. با اتصال خروجی گیت AND مقابل به ورودی $\overline{CS}$ حافظه اول این موضوع تضمین خواهد شد. این طراحی را برای سه حافظه بعدی تکمیل کنید.



شاید این پرسش برای شما پیش آمده باشد که وقتی با انتخاب یک حافظه ۱۶ بایتی می‌توان به راحتی خطوط آدرس حافظه را به خطوط آدرس پردازنده متصل کرد، چرا دو حافظه ۸ بایتی انتخاب کرده‌ایم تا به مشکلات انتخاب حافظه به کمک CS و خطوط بالای آدرس پردازنده برخورد کنیم؟

این موضوع در سیستمهای مبتنی بر پردازنده بسیار پیش می‌آید. مثلاً به جای یک حافظه RAM یک گیگابایتی در کامپیوتر می‌توان از دو حافظه ۵۱۲ مگابایتی یا ۴ حافظه ۲۵۶ مگابایتی (که ممکن است بیشتر در دسترس باشند) استفاده نمود. بنابراین باید با تکنیک استفاده از حافظه‌های کم‌حجم‌تر به منظور پوشش فضای یک آدرس‌دهی بزرگ آشنا باشیم.

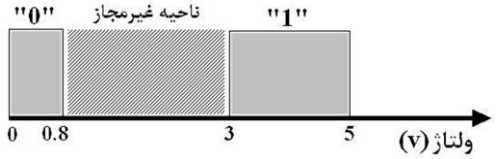
تاکنون مجموعه قوانین زیر را برای طراحی سیستم دیده‌ایم:

- ✓ گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.
- ✓ خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
- ✓ از خطوط بالای آدرس پردازنده برای انتخاب حافظه مناسب استفاده می‌شود.
- ✓ سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.
- ✓ سیگنال WR پردازنده باید به سیگنال WR حافظه متصل شود.

اتصال وسایل ورودی/خروجی به پردازنده

همانطور که قبلاً ذکر شد، مهمترین وظیفه پردازنده، پردازش اطلاعات ورودی به آن و ارسال نتایج به خارج است. در مثالهای قبلی داده‌های ورودی و خروجی در حافظه ذخیره می‌شدند. در مثالهای بعدی در پی این هستیم که ورودی و خروجی سیستم را نه از طریق حافظه که از راههای قابل‌لمس‌تر پیاده کنیم.

اطلاعاتی که در سیستمهای مبتنی بر پردازنده و مدارهای منطقی وجود دارد، همه در قالب «صفر و یک منطقی» است. بنابراین اگر بتوانیم یک بیت (یک رقم دودویی) را تولید کنیم، می‌توانیم اطلاعات چند بیتی را نیز به عنوان ورودی به سیستم بدهیم. به همین ترتیب اگر بتوان یک بیت را نمایش داد، می‌توان اطلاعات چندبیتی را نیز به راحتی به نمایش درآورد.

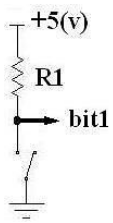


شکل ۲-۱۰-محدوده ولتاژهای TTL

در مدارهای منطقی و سیستمهای مبتنی بر پردازنده، از «صفر و یک منطقی» تعبیر ولتاژی می‌شود. شکل ۲-۱۰ نشان می‌دهد، اگر ولتاژی بین صفر تا ۰/۸ ولت باشد از آن به «صفر منطقی» و اگر بین ۳ تا ۵ ولت باشد از آن به «یک منطقی» تعبیر می‌شود[1]. مثلاً اگر به پین ورودی یک گیت NOT (پین ۱ تراشه 7404) ولتاژی حدود صفر (مثلاً ۰/۲ ولت) اعمال کنیم، روی پین خروجی این گیت (پین ۲ تراشه 7404) ولتاژی حدود ۵ ولت (مثلاً ۴/۸ ولت) به کمک ولت‌متر قابل مشاهده است. ولتاژ بین ۰/۸ تا ۳ ولت در منطق ولتاژی فوق غیرمجاز است؛ یعنی اگر خروجی یک مدار منطقی مثلاً ۲ ولت باشد، مدار خراب است و به درستی کار نمی‌کند. اگر به ورودی یک مدار منطقی ولتاژ ۲ ولت داده شود، مدار ممکن است از آن تعبیر «صفر» یا «یک» داشته باشد.

پس برای ایجاد یک بیت، باید مداری دوحالته داشته باشیم که دو ولتاژ صفر و ۵ ولت را بتواند تولید کند. برای نمایش یک بیت نیز باید مداری ببینیم که در مقابل دو ولتاژ صفر و ۵ ولت، دو واکنش متفاوت ایجاد کند.

چگونه یک بیت را تولید کنیم ؟



ساختار مقابل با استفاده از یک کلید دوحالته این کار را انجام می دهد.

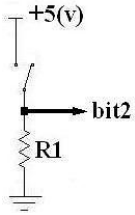
در این ساختار اگر کلید باز باشد، سیگنال bit1 توسط مقاومت R1 به ۵ ولت متصل است؛ بنابراین «یک منطقی» ایجاد می شود.

اگر کلید بسته باشد، سیگنال bit1 به زمین متصل شده و «صفر منطقی» ایجاد می شود.

[1] این منطق ولتاژ موسوم به TTL است. در منطق‌های ولتاژی دیگر مانند CMOS، ولتاژهای دیگری به «صفر» و «یک» منطقی نسبت داده می‌شوند.

به طور خلاصه :

bit1	کلید
"1"	باز
"0"	بسته



یک بیت با استفاده از ساختار روبرو نیز قابل ایجاد است. در این ساختار اگر کلید باز باشد، سیگنال bit2 توسط مقاومت R1 به زمین متصل است؛ بنابراین «صفر منطقی» ایجاد می شود.

اگر کلید بسته شود، سیگنال bit2 به ۵ ولت متصل شده و «یک منطقی» ایجاد می‌شود.

به طور خلاصه :

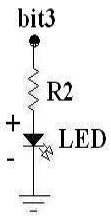
bit2	کلید
"0"	باز
"1"	بسته

مقاومت R1 در هردو ساختار برای جلوگیری از اتصال کوتاه شدن منبع تغذیه هنگام بسته شدن کلید به کار رفته و مقدار نوعی آن 10 KΩ است.

همانطور که مشاهده می کنید bit1 در هنگام **باز بودن کلید** برابر «یک منطقی» است و bit2 در هنگام **بسته بودن کلید**.

دو ساختار ذکر شده از نظر فنی متفاوت هستند که بررسی این تفاوت به عنوان تحقیق به شما واگذار شده است.

چگونه یک بیت را نمایش دهیم ؟



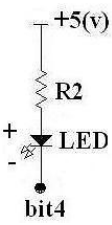
ساده ترین راه برای نمایش یک بیت، استفاده از دیود نوری (LED[1]) است. دیود نوری یک عنصر الکترونیکی شبیه یک لامپ کوچک با دوپایه مثبت (آند) و منفی (کاتد) است. اگر ولتاژ پایه مثبت دیود نوری از ولتاژ پایه منفی آن از حدی بیشتر شود، دیود نوری روشن و در غیر این صورت خاموش می‌شود. ساختار روبرو برای نمایش یک بیت با دیود نوری به کار می رود.

اگر bit3، «صفر منطقی» باشد، ولتاژی حدود صفر دارد و با توجه به صفر بودن ولتاژ سر منفی، دیود نوری روشن نخواهد شد. اگر bit3، «یک منطقی» باشد دارای ولتاژی بالاتر از ۳ ولت است و باعث روشن شدن دیود نوری می شود.

به طور خلاصه :

bit3	وضعیت دیود نوری
"0"	خاموش
"1"	روشن

ساختار دیگر نمایش یک بیت را در شکل روبرو می بینید.



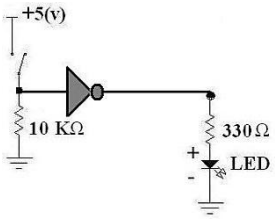
پرسش) با توجه به ساختار مقابل، جدول را کامل کنید.

bit4	وضعیت دیود نوری
"0"	
"1"	

[1] Light Emitting Diode

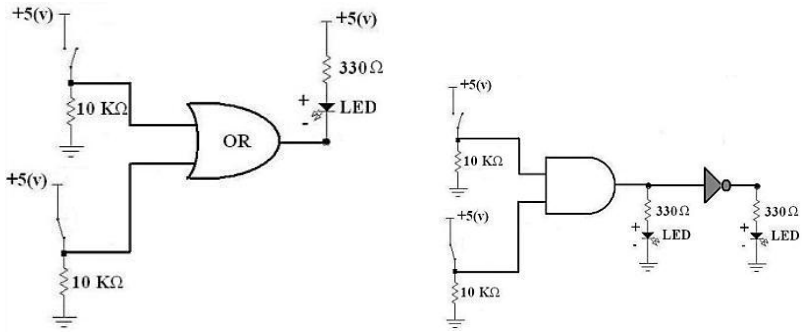
مقاومت R2 در هر دو ساختار برای جلوگیری از سوختن دیود نوری در اثر عبور جریان زیاد به کار رفته و مقدار نوعی آن 330 Ω است.

مثال) در ساختار روبرو مشخص کنید دیود نوری خاموش است یا روشن ؟

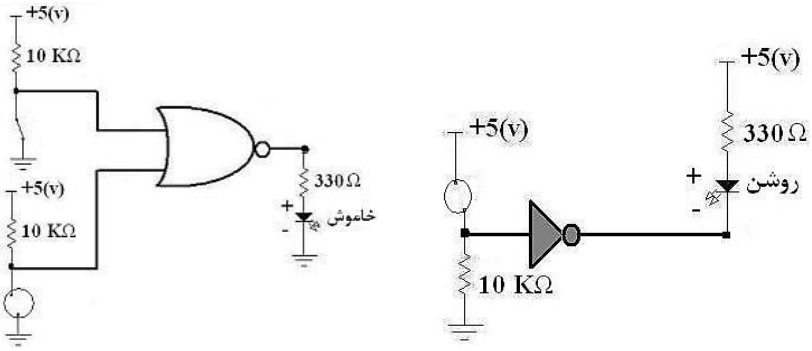


چون کلید باز است، ورودی گیت NOT با مقاومت 10KΩ به زمین متصل شده است و «صفر منطقی» است. اگر ورودی گیت NOT «صفر» باشد، خروجی آن «یک» (یعنی حدود ۵ ولت) است؛ چون ولتاژ سر مثبت دیود نوری از سر منفی آن بیشتر است، دیود نوری روشن می شود.

پرسش) در هریک از ساختارهای زیر مشخص کنید هر یک از دیودهای نوری روشن است یا خاموش؟



پرسش) در هریک از شکلهای زیر معلوم کنید کلید مشخص شده باز است یا خیر؟



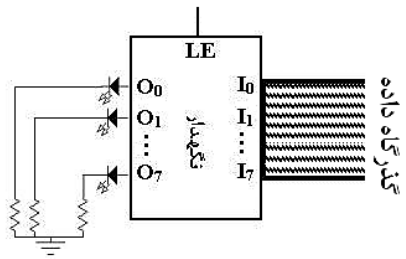
مثال ٤) این مثال ما را با نحوه اتصال خروجی به سیستم آشنا می‌کند. می‌خواهیم سیستم مثال ۳ را به نحوی تغییر دهیم که محتوای خانه‌های ۱۳ و ۱۴ حافظه را با هم جمع کند و نتیجه که یک عدد ۸ بیتی است را روی ۸ عدد دیود نوری نشان دهد؛ به نحوی که روشن بودن هر دیود نوری نشان‌دهنده "یک منطقی" و خاموش بودن آن نشانه "صفر منطقی" باشد.

این ۸ عدد دیود نوری باید به کجا متصل شوند؟ اولین پاسخی که به ذهن می‌رسد این است که چون گذرگاه داده محل دسترسی به اطلاعات است، این ۸ دیود نوری را به سیمهای گذرگاه داده متصل کنیم. اما این عقیده خوبی نیست؛ چون می‌خواهیم از روشن و خاموش بودن دیودهای نوری نتیجه جمع را بدانیم؛ در حالی که اگر دیودهای نوری به

گذرگاه داده متصل شوند، در هر لحظه محتویات گذرگاه داده را نشان خواهند داد که ممکن است کد عمل، آدرس مکانی از حافظه و ... باشد!

مشکل دیگری که وجود دارد این است که نتیجه عمل جمع تنها یک لحظه کوتاه روی گذرگاه داده ظاهر می‌شود و سپس گذرگاه داده به اطلاعات دیگر اختصاص می‌یابد؛ اگر دیودهای نوری مستقیماً به گذرگاه داده متصل باشند، نمی‌توانند این لحظه کوتاه را ثبت کنند.

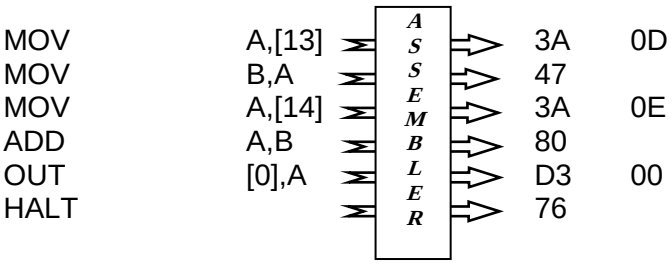
بنابراین باید سیستم به گونه‌ای طراحی شود که اولاً تنها لحظه‌ای که گذرگاه داده محتوی حاصل جمع است، دیودهای نوری را به گذرگاه داده متصل کند، ثانیاً وقتی نتیجه جمع از روی گذرگاه داده برداشته شد، اتصال بین دیودهای نوری و گذرگاه داده را قطع کند و نتیجه جمع را روی دیودهای نوری حفظ کند. بخشی از سیستم که این کار را انجام می‌دهد، تراشه‌ای به نام نگهدار[1] است.



تراشه نگهدار (مثلاً تراشه 74LS273) دارای ۸ پین ورودی، ۸ پین خروجی و یک پین فعال‌ساز به نام LE[2] است. هرگاه پین LE فعال شود، خروجیهای نگهدار به ورودیهای آن متصل خواهند شد و پس از غیرفعال شدن LE خروجیهای نگهدار در حالت قبلی خود باقی می‌مانند (قفل می‌شوند). بنابراین برای طراحی سیستم، گذرگاه داده را به ورودی نگهدار و ۸ دیود نوری را به خروجی آن متصل می‌کنیم و ترتیبی اتخاذ می‌کنیم که در لحظه‌ای که گذرگاه داده محتوی حاصل جمع مورد نظر است، پین LE نگهدار فعال و پس از قفل کردن حاصل جمع در خروجی (دیودهای نوری) غیرفعال شود. شکل تراشه نگهدار و نحوه اتصال آن به سیستم را در شکل بالا می‌بینید. به عنوان اولین گام برای طراحی این سیستم، برنامه را می‌نویسیم. پیش از این آموختیم که هرکدام از ورودیها و خروجی‌ها در سیستم مبتنی بر پردازنده دارای یک آدرس هستند. به همان شیوه که خانه‌های حافظه را آدرس‌دهی می‌کردیم، به خروجی (در اینجا تراشه نگهدار) هم یک آدرس (مثلاً آدرس صفر) نسبت می‌دهیم.

طراح سیستم باید به هر خروجی یک آدرس نسبت دهد.

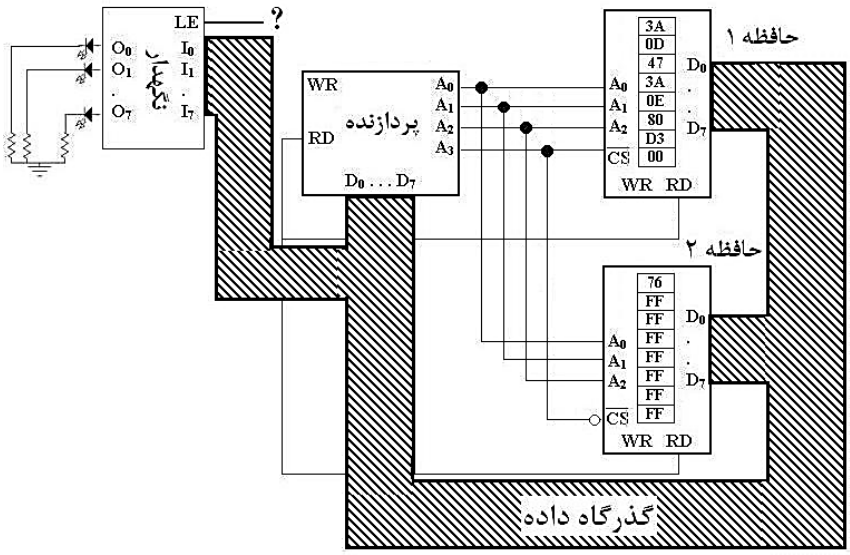
برنامه زیر را ببینید:



این برنامه را در دو حافظه ذخیره و به انضمام تراشه نگهدار به پردازنده متصل می‌کنیم. یک اتصال ابتدایی برای این سیستم را در شکل ۲-۱۱ می‌بینید.

[1] Latch

[2] Latch Enable : LE



شکل ۲-۱۱-سخت‌افزار ابتدایی مثال ۴

تنها کاری که باقی‌مانده این است که پین LE را در لحظه مناسب فعال کنیم. این لحظه مناسب، وقتی است که پردازنده نتیجه عمل جمع را روی گذرگاه داده قرار می‌دهد. دقت در برنامه بالا نشان می‌دهد که این اتفاق در هنگام اجرای دستور Out [0], A رخ می‌دهد. این دستور مقدار ثبات A که حاصل جمع مورد نظر ماست را به **خروجی** با آدرس صفر ارسال می‌کند؛

بنابراین پین LE باید در لحظه اجرای این دستور فعال شود. نشانه‌های اجرای این دستور چیست؟ به بیان دیگر، این دستور چگونه توسط پردازنده اجرا می‌شود؟

اصل چهارم طراحی سیستم: پردازنده برای ارسال داده‌ای به یک خروجی، آدرس آن خروجی را روی گذرگاه آدرس و داده مورد نظرش را روی گذرگاه داده قرار داده و پین WR را فعال می‌کند.

این اصل شباهت عجیبی به اصل سوم (مربوط به نوشتن در حافظه) دارد! در واقع تفاوتی اساسی بین نوشتن در حافظه و ارسال داده به خروجی وجود دارد که چون نمی‌خواهیم در حال حاضر بحث را بیهوده پیچیده کنیم، در اصل چهارم ذکر نشده است. بعداً این اصل را تکمیل می‌کنیم.

مراحل اجرای این برنامه در زیر نشان داده شده است :

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000 (آدرس حافظه)	(دریافت کد عمل)3A
0001	0001	(دریافت کد عمل)0D
	---- **اجرای کد** 0D 3A ----	
0001	1101	**محتویات خانه ۱۳ حافظه**
0010	0010	(دریافت کد عمل)47
	---- **اجرای کد** 47 ----	
0011	0011	(دریافت کد عمل)3A
0100	0100	(دریافت کد عمل)0E
	---- **اجرای کد** 0E 3A ----	
0100	1110	**محتویات خانه ١٤ حافظه**
0101	0101	(دریافت کد عمل)80
	---- **اجرای کد** 80----	

0110	0110	(دریافت کد عمل)D3
0111	0111	(دریافت کد عمل)00

---- **اجرای کد** 00 D3 ----

0111	0000 (آدرس خروجی)	**محتویات ثبات**A

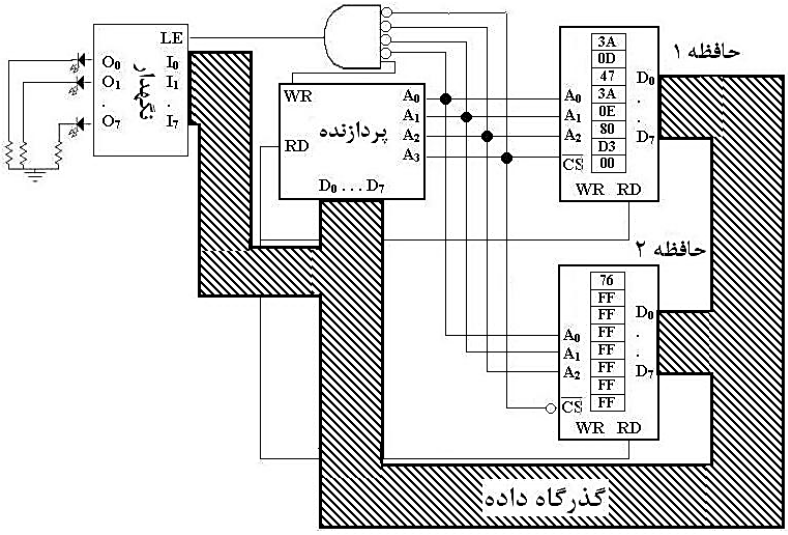
(که پردازنده روی گذرگاه داده قرار می دهد)

پین LE در این لحظه باید فعال شود.

1000	1000	(دریافت کد عمل) 76

---- **اجرای کد** 76 ----

به لحظه‌ای که پین LE باید فعال شود توجه کنید. به نظر می‌آید، در لحظه‌ای که آدرس 0000 روی گذرگاه آدرس می‌آید باید پین LE فعال شود. این ایده کاملاً درست نیست؛ چون همانطور که در روند اجرای برنامه مشاهده می‌کنید وقتی پردازنده می‌خواهد اولین کد عمل برنامه را بخواند نیز آدرس 0000 را روی گذرگاه آدرسش قرار می‌دهد. تفاوت این دو حالت این است پردازنده هنگام خواندن اولین کد عمل برنامه سیگنال RD و هنگام ارسال داده به خروجی سیگنال WR را فعال می‌کند.



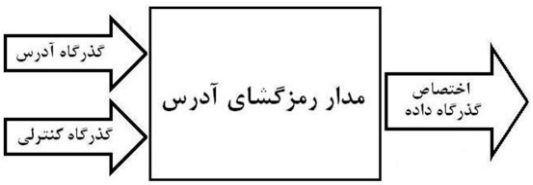
شکل ۲-۱۲-سخت‌افزار ابتدایی مثال ۴

بنابراین در لحظه‌ای که:

- ✓ پردازنده آدرس 0000 را روی گذرگاه آدرس خود می‌گذارد،
- ✓ سیگنال WR را فعال می‌کند،

پین LE باید فعال شود. مدار مورد نظر را در شکل ۲-۱۲ می‌بینید.

همانطور که گفته شد، پردازنده برای اجرای دستور OUT [0],A ، محتویات A (حاصل جمع مورد نظر) را روی گذرگاه داده و آدرس 0000 را روی گذرگاه آدرس خود گذاشته و سیگنال WR را فعال می‌کند. در این لحظه خروجی گیت AND در مدار بالا «یک» شده و نگهدار فعال می‌شود و مقدار گذرگاه داده را در خروجی خود (روی دیودهای نوری) نشان می‌دهد. وقتی پردازنده می‌خواهد دستور بعد را اجرا کند، PC دستور بعدی (آدرس ۹ یعنی 1001) را روی گذرگاه آدرس می‌گذارد که طبعاً نگهدار را غیرفعال خواهد کرد (چون خروجی گیت AND صفر می شود)؛ اما جای نگرانی نیست، چون عدد مورد نظر ما روی خروجی نگهدار (دیودهای نوری) قفل شده است.

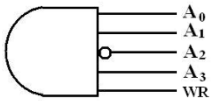


شکل ۲-۱۳-مدار رمزگشای آدرس

گیت AND استفاده شده در این مثال، شکل ساده‌ای از مدار رمز گشای آدرس[1] (شکل ۲-۱۳) است. وظیفه این مدار در سیستمهای مبتنی بر پردازنده، اختصاص گذرگاه داده به متقاضی مجاز با استفاده از اطلاعات گذرگاه آدرس و کنترل است.

در واقع گیت AND در این مثال، آدرس 0000 را رمزگشایی می‌کند.

پرسش) اگر بخواهیم نتیجه عمل جمع علاوه بر نمایش روی دیودهای نوری در خانه شماره ۱۵ حافظه هم ذخیره شود، چه تغییراتی در سخت افزار و نرم افزار سیستم باید انجام شود؟



پرسش) گفتیم طراح سیستم باید به هر خروجی یک آدرس نسبت دهد. اگر آدرس خروجی را به جای صفر، ۱۱ در نظر بگیریم (از دستور OUT [11], A استفاده کنیم) نشان دهید که ورودیهای مدار رمزگشای آدرس (گیت AND) به شکل مقابل خواهند بود. توجه کنید که در اینجا ورودیهای گیت AND باید به نحوی به گذرگاه آدرس متصل شوند که تنها وقتی عدد ۱۱ روی گذرگاه آدرس می آید، خروجی گیت AND برابر ۱ شود. در اینجا چون غیر از خروجی فوق آدرس ۱۱ دیگری در سیستم وجود ندارد، استفاده از سیگنال WR لزومی ندارد؛ اما بهتر است عادت کنیم در ورودی مدار رمزگشای آدرس خروجی از سیگنال WR استفاده نماییم.

به کمک مثال چهار، قوانین طراحی سیستم خود را کاملتر می‌کنیم:

- ✓ گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه و ورودی تراشه نگهدار متصل می‌شود.
- ✓ خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
- ✓ از خطوط بالای آدرس پردازنده برای انتخاب حافظه مناسب استفاده می‌شود.
- ✓ سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.
- ✓ سیگنال WR پردازنده باید به سیگنال WR حافظه متصل شود.
- ✓ برای اتصال خروجی به سیستم، باید از تراشه نگهدار استفاده کنیم که گذرگاه داده به ورودی این تراشه متصل می‌شود. تراشه نگهدار وقتی باید فعال شود که پردازنده آدرس آن خروجی (که توسط طراح سیستم به آن نسبت داده می‌شود) را روی گذرگاه آدرس خود قرار داده و سیگنال WR خود را فعال کند.
- ✓ برای اتصال خروجی به سیستم، باید به هر خروجی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه نگهدار) مورد توجه قرار دهیم.

پرسش) دستور NOT A محتویات ثبات A را معکوس می‌کند و کد عمل آن 73h است. نرم‌افزار و سخت‌افزار سیستم بالا را به نحوی تغییر دهید که مجموع خانه‌های ۱۳ و ۱۴ حافظه را به خروجی با آدرس صفر و معکوس آن را به خروجی با آدرس ۱۱ ارسال کند.

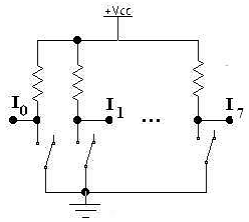
[1] Address Decoder Circuit

برخلاف وعده‌ای که کرده بودیم، در مدار رسم‌شده سیگنال WR پردازنده را به پین WR حافظه‌ها متصل نکرده‌ایم. دلیل این موضوع این است که وقتی پردازنده آدرس 0000 را روی گذرگاه آدرسش قرار داده و پین WR را فعال می‌کند، داده باید به خروجی شماره صفر ارسال شود؛ در حالی که اگر سیگنال WR پردازنده به پین WR حافظه اول متصل باشد، در این لحظه تلاشی برای نوشتن در خانه صفر حافظه اول هم انجام می‌شود که صحیح نیست.

دلیل این خلف وعده این است که هنوز تفاوت دقیق اصل سوم (نوشتن در حافظه) و اصل چهارم (ارسال داده به خروجی) که به دو سیگنال MemRQ و IORQ برمی‌گردد را بیان نکرده‌ایم. خوشبختانه چون در این سیستم نوشتن در حافظه نداریم، این موضوع مشکلی پیش نمی‌آورد. اما در مثال بعد با تکمیل اصول طراحی پردازنده‌ها، این خلف وعده را جبران خواهیم کرد.

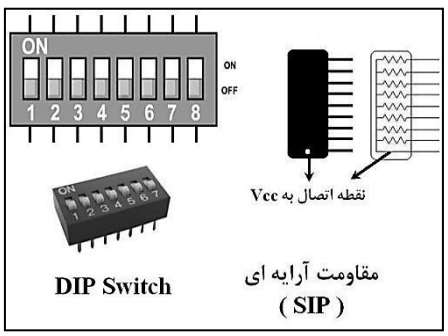
در اینجا لازم است باز به مسأله زمانبندی سیستم توجه کنیم. هنگامی که پردازنده آدرس خروجی (0000) را روی گذرگاه آدرس خود قرار داده و سیگنال WR را فعال می‌کند، گیت AND رمزگشای آدرس، تراشه نگهدار را فعال کرده و گذرگاه داده سیستم را به LEDها متصل می‌کند. توجه داشته باشید که زمان اجرای دستور OUT [0],A (که طی آن آدرس 0000 روی گذرگاه آدرس قرار دارد و سیگنال WR فعال است) به دلیل سرعت بالای پردازنده، بسیار کوتاه است. معمولاً سرعت تراشه نگهدار از سرعت پردازنده بسیار پایین‌تر است و به همین لحاظ در طی زمان کوتاه اجرای دستور، تراشه نگهدار موفق به قفل کردن داده‌ها نمی‌شود. بنابراین باید مکانیسمی اندیشیده شود تا اجرای این دستور چند برابر حالت عادی به طول بیانجامد تا تراشه نگهدار فرصت کافی برای قفل کردن داده‌ها داشته باشد.

مثال ۵) در مثال ۴ متوجه شدید که چگونه می‌توان حاصل پردازش داده‌های ورودی را روی تعدادی دیود نوری نشان داد. اکنون می‌خواهیم پا را فراتر نهاده و داده‌های ورودی را نیز به جای حافظه، از خارج پردازنده بخوانیم؛ به بیان دیگر می‌خواهیم سیستمی طراحی کنیم که دو عدد را از کاربر سیستم دریافت کرده و آنها را با هم جمع کند و نتیجه را روی تعدادی دیود نوری نشان دهد.



سؤال اول این است که چگونه می‌توان اعداد را به سیستم وارد کرد ؟ ساده‌ترین راه، ورود اعداد به صورت دودویی با استفاده از ۸ عدد کلید به صورت شکل مقابل است.

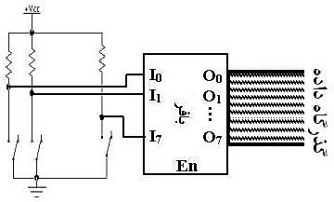
همانطور که می‌بینید، به‌عنوان مثال اگر کلید اول بسته باشد، ولتاژ I_0 برابر صفر ولت («صفر منطقی») و اگر کلید باز باشد، این ولتاژ حدود ۵ ولت («یک منطقی») است. بنابراین با باز و بسته کردن این کلیدها می‌توان یک عدد ۸ بیتی (بین صفر تا ۲۵۵) تولید کرد.



معمولاً برای اتصال ساختار فوق (۸ کلید دوحالته)، از کلیدهای کوچک کنار هم که در تعداد ۴ تایی یا ۸ تایی ساخته می شوند و به نام میکروسوییچ یا DIP Switch معروفند، استفاده می‌شود. همچنین برای خودداری از به کار بردن ۸ مقاومت مجزا که باعث شلوغی مدار می شود، از مقاومتهای آرایه‌ای که به نام SIP معروفند، استفاده می‌شود که هم جای کمتری اشغال می کنند و هم اتصال آنها ساده‌تر است.

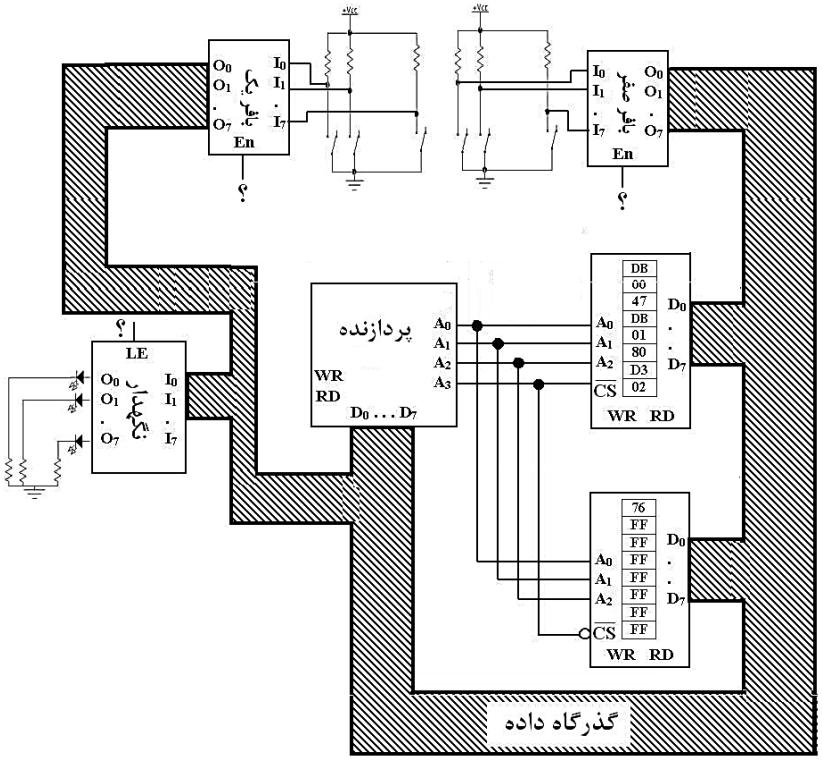
سؤال دوم این است که این کلیدها باید به کجا متصل شوند؟ به گذرگاه داده؟ واضح است که این راه عملی نیست؛ چون اگر کلیدها را مستقیماً به گذرگاه داده متصل کنیم، عدد روی گذرگاه داده همیشه همان عددی است که کلیدها نشان می‌دهند و گذرگاه داده از انجام وظایف دیگر خود مانند انتقال کدهای عمل و آدرسهای حافظه و آدرسهای ورودی/خروجی باز می‌ماند.

در واقع تنها در لحظه‌ای که پردازنده می‌خواهد داده‌های روی گذرگاه داده را بخواند، باید این گذرگاه به کلیدها متصل شود. قسمتی از سیستم که این کار را انجام می‌دهد، تراشه‌ای به نام بافر سه‌حالته[1] است.



این تراشه دارای تعدادی ورودی و خروجی و یک پین فعالساز En است. خروجی این تراشه در صورت غیرفعال بودن پین En ، امپدانس بالا[2] (مثل حالت قطع) و در صورت فعال بودن این پین برابر مقدار ورودی است. بنابراین ۸ عدد کلید را به ۸ پین ورودی بافر سه‌حالته و گذرگاه داده را به خروجی آن متصل می‌کنیم و ترتیبی می‌دهیم که این تراشه فقط هنگامی که پردازنده می‌خواهد عددی که کلیدها روی آن تنظیم شده‌اند را بخواند فعال شود و کلیدها را به گذرگاه داده متصل کند. نحوه اتصال بافر سه‌حالته به پردازنده را در شکل مقابل می‌بینید. چون می‌خواهیم دو عدد را بخوانیم، به دو مجموعه کلید و دو تراشه بافر سه‌حالته ۸ ورودی نیاز داریم.

اتصال اولیه این سیستم را در شکل ۲-۱۴ می‌بینید:



شکل ۲-۱۴-سخت‌افزار ابتدایی مثال ۵

[1] Tri-State Buffer

[2] High Impedance

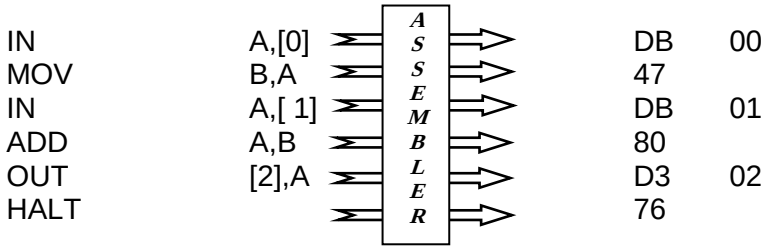
سه علامت سؤال در شکل وجود دارد که باید تکمیل شود؛ به عبارت دیگر با توجه به روند اجرای برنامه، باید لحظه‌ای که پینهای En بافرها و پین LE نگهدار باید فعال شوند را پیدا کرده و به کمک مدار مناسب بازسازی کنیم.

مطابق قرار قبلی، برای تکمیل مدار فوق برنامه مورد نظر را نوشته و طبق روندی که پردازنده برای اجرای برنامه در پیش می‌گیرد، اتصالات مدار را کامل می‌کنیم. همانطور که در مورد خروجی و تراشه نگهدار عمل کردیم، به ورودیها نیز آدرس می‌دهیم؛ مثلاً به مجموعه کلید اول که قرار است عدد اول را تولید کند، آدرس «صفر» و به مجموعه کلید دوم (تراشه بافر سه‌حالته دوم) که عدد دوم را از آن می‌خوانیم، آدرس «یک» اختصاص می‌دهیم.

طراح سیستم باید به هر ورودی یک آدرس نسبت دهد.

بدیهی است که خروجی سیستم یعنی تراشه نگهدار هم باید دارای آدرس باشد که آدرس آن را در این سیستم «دو» درنظر می‌گیریم. توجه کنید که آدرس ورودی/خروجی‌ها دلخواه است و توسط طراح سیستم انتخاب می‌شود.

به برنامه زیر دقت کنید:



دستور IN A, [0] محتویات ورودی شماره صفر را خوانده و به ثبات A منتقل می‌کند. در لحظه اجرای این دستور، باید گذرگاه داده سیستم به مجموعه کلید اول (ورودی شماره صفر) متصل شود. نشانه‌های اجرای این دستور چیست؟ به بیان دیگر، این دستور چگونه توسط پردازنده اجرا می‌شود؟

اصل پنجم طراحی سیستم: پردازنده برای دریافت داده‌ای از یک ورودی، آدرس آن ورودی را روی گذرگاه آدرس خود قرار داده و پین RD را فعال می‌کند و محتویات گذرگاه داده را (که باید در این لحظه مقدار ورودی را روی خود داشته باشد) می‌خواند.

این اصل نیز شبیه اصل سوم (خواندن یکی از خانه‌های حافظه) شده است! به زودی تفاوت را بیان خواهیم کرد.

مراحل اجرای این برنامه را در زیر مشاهده می کنید :

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000 (آدرس حافظه)	DB(دریافت کد عمل)
0001	0001	00(دریافت کد عمل)
	---- **اجرای کد** 00 DB ----	
0001	0000 (آدرس ورودی)	**محتویات بافر شماره صفر**
	پین EN تراشه بافر صفر در این لحظه باید فعال شود.	
0010	0010	47(دریافت کد عمل)
	---- **اجرای کد** 47 ----	
0011	0011	DB(دریافت کد عمل)
0100	0100	01(دریافت کد عمل)
	---- **اجرای کد** 01 DB ----	

0100	0001	**محتویات بافر شماره ۱**
	پین EN تراشه بافر یک در این لحظه باید فعال شود.	
0101	0101	(دریافت کد عمل)80
	---- **اجرای کد** 80 ----	
0110	0110	(دریافت کد عمل)D3
0111	0111	(دریافت کد عمل)02
	---- **اجرای کد** 02 D3 ----	
0111	0010	**محتویات ثبات**A
		(که پردازنده روی گذرگاه داده قرار می‌دهد)
	پین LE تراشه نگهدار در این لحظه باید فعال شود.	
1000	1000	(دریافت کد عمل) 76
	---- **اجرای کد** 76 ----	

به لحظه‌ای که پین En بافر صفر باید فعال شود توجه کنید. به نظر می‌آید، در لحظه‌ای که آدرس 0000 روی گذرگاه آدرس می‌آید باید پین En فعال شود. این ایده کاملاً درست نیست؛ چون همانطور که در روند اجرای برنامه مشاهده می‌کنید وقتی پردازنده می‌خواهد اولین کد عمل برنامه را بخواند نیز آدرس 0000 را روی گذرگاه آدرسش قرار می‌دهد. متأسفانه چون در هر دو حالت پردازنده مشغول عملیات خواندن گذرگاه داده است، دیگر برای تمایز دو حالت فوق نمی‌توانیم مانند مثال قبل از سیگنالهای RD و WR استفاده کنیم. تنها تفاوت این دو حالت این است که پردازنده در هنگام دریافت کد عمل با **حافظه** و هنگام خواندن بافر با **ورودی/خروجی** در ارتباط است.

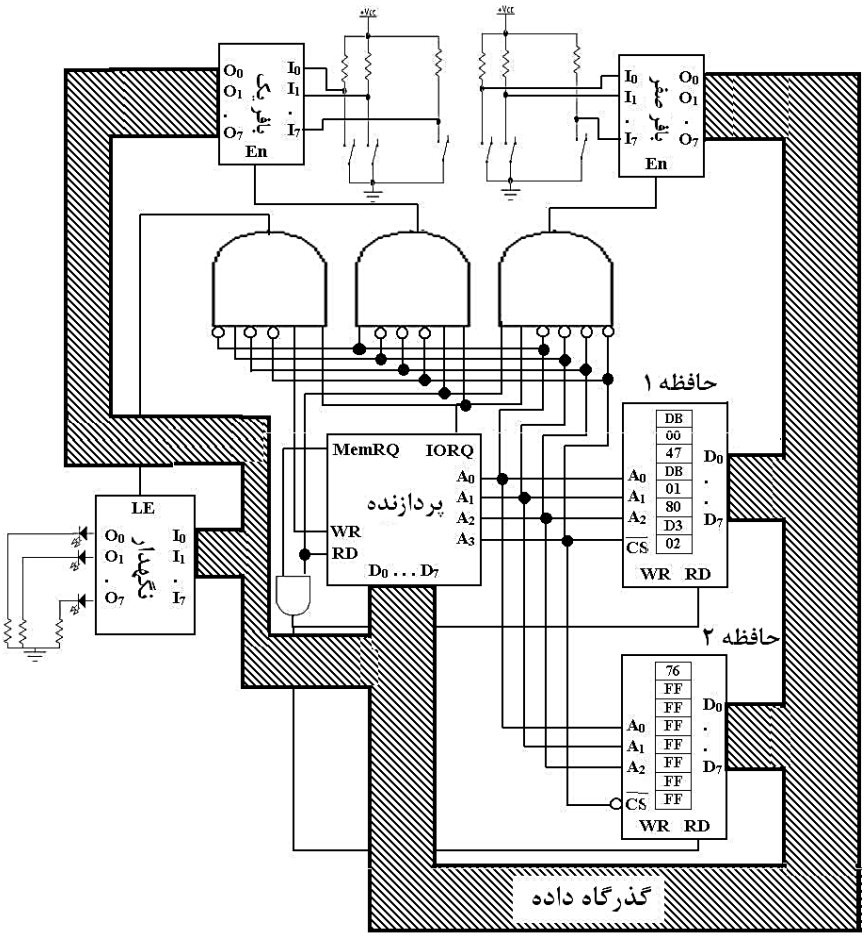
دو سیگنال مهم در پردازنده‌ها این مشکل را حل می‌کنند. هنگامی که پردازنده در حال تبادل اطلاعات با حافظه است **سیگنال** MemRQ و در هنگام تبادل اطلاعات با ورودی/خروجی **سیگنال** IORQ را فعال می‌کند.

بنابراین پردازنده در هنگام واکشی کد عمل از آدرس صفر به خانه صفر حافظه اول (MemRQ فعال و IORQ غیرفعال) و هنگام خواندن عدد اول به بافر سه‌حالته اول (MemRQ غیرفعال و IORQ فعال) مراجعه می‌کند. به بیان دیگر، سیستم باید به نحوی طراحی شود که تنها در هنگام اجرای دستور IN کلیدها به گذرگاه داده پردازنده متصل شوند؛ این لحظه دو نشانه دارد که به کمک آن نشانه‌ها سیستم را طراحی می‌کنیم:

- ✓ فعال شدن سیگنالهای RD و IORQ
- ✓ قرار گرفتن آدرس ورودی روی گذرگاه آدرس پردازنده

جزییات مدار را در شکل ۲-۱۵ زیر می‌بینید.

نکته دیگری نیز در این شکل وجود دارد. سیگنال RD حافظه به سیگنال RD پردازنده متصل نشده است؛ بلکه از خروجی یک گیت AND گرفته شده که ورودیهای آن سیگنالهای RD و MemRQ هستند. یعنی حافظه باید هنگامی برای خواندن فعال شود که سیگنالهای RD و MemRQ فعال باشند. اگر سیگنال RD حافظه به سیگنال RD پردازنده متصل شود، هنگامی که پردازنده می‌خواهد از یک ورودی با آدرسی مشترک با حافظه (مثلاً از ورودی شماره صفر) نیز اطلاعات بخواند، حافظه برای خواندن فعال خواهد شد که درست نیست.



شکل ۲-۱۵-سخت‌افزار کامل مثال ۵

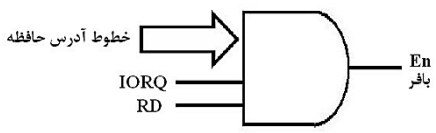
طراحی یک مدار رمزگشای آدرس کامل

پرسشی که ممکن است پیش آید این است که چرا در ورودی گیتهای AND فعالساز بافرها از سیگنال RD استفاده کرده‌ایم؟ در این سیستم دو آدرس صفر داریم که یکی برای خواندن از حافظه و یکی برای خواندن از ورودی استفاده می‌شود که به کمک سیگنالهای MemRQ و IORQ از هم جدا شده‌اند. در این سیستم از آدرس صفر برای هیچگونه خروجی (حافظه یا نگهدار) استفاده نشده است که برای تفکیک آن از آدرسهای صفر ورودی ناچار به استفاده از سیگنالهای RD و WR باشیم. بنابراین به نظر می‌آید استفاده از سیگنال RD در ورودی گیتهای AND فعالساز بافرها لزومی ندارد.

اصل زیر را همیشه (چه لازم باشد و چه لازم نباشد) رعایت می‌کنیم:

> در ورودی گیت AND فعالساز بافرهای ورودی، همیشه از سیگنالهای IORQ و RD استفاده کنید.

شکل کلی گیت AND فعالساز بافرهای ورودی به صورت مقابل است:

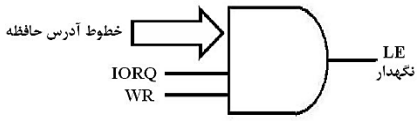


پرسش دیگر این است که با وجود اینکه ما در این برنامه فقط در خروجی (نگهدار) می‌نویسیم و نوشتن در حافظه نداریم، چرا در ورودی گیت AND فعالساز نگهدار هم از سیگنال IORQ استفاده کرده‌ایم، در حالیکه به نظر می‌آید جایی که یک مکان حافظه

و یک ورودی یا خروجی، آدرس مشابه داشته باشند و باید هر دو خوانده یا هردو نوشته شوند، از سیگنالهای MemRQ و IORQ برای تمایز آنها استفاده می‌کنیم؟

در ورودی گیت AND فعالساز تراشه‌های نگهدار خروجی، همیشه از سیگنالهای IORQ و WR استفاده کنید.

شکل کلی گیت AND فعالساز تراشه‌های نگهدار به صورت مقابل است:



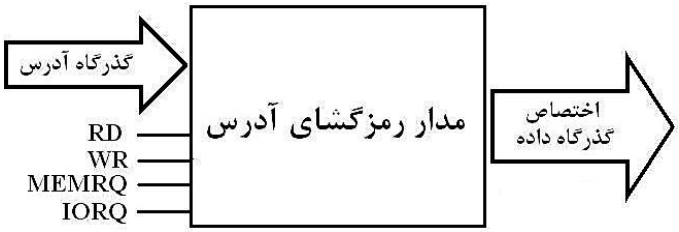
برای اینکه خود را درگیر جزییات مدار (آیا آدرسی مشترک داریم؟ کدام مخصوص خواندن و کدام ویژه نوشتن است؟ کدام مربوط به حافظه و کدام مربوط به ورودی/خروجی است؟ و...) سیگنالهای زیر را می‌سازیم و همیشه (چه لازم باشد و چه نباشد) از آنها استفاده می‌کنیم:

شکل ۲-۱۶-سیگنال‌های مدار رمزگشای آدرس

بنابراین از این پس:

- ✓ در ورودی گیت AND فعالساز بافرهای ورودی، همیشه از سیگنال IORD استفاده می‌کنیم.
- ✓ در ورودی گیت AND فعالساز تراشه‌های نگهدار خروجی، همیشه از سیگنال IOWR استفاده می‌کنیم.
- ✓ سیگنال RD حافظه‌هایی که محتویات آنها باید خوانده شود را به سیگنال MemRD متصل می‌کنیم.
- ✓ سیگنال WR حافظه‌هایی که در آنها باید اطلاعاتی نوشته شود را به سیگنال MemWR متصل می‌کنیم.

شکل ۲-۱۷ مدار کلی رمزگشای آدرس را نشان می‌دهد.



شکل ۲-۱۷-طرح کلی مدار رمزگشای آدرس

اصول کامل طراحی سیستم

اصل اول: پردازنده در هنگام واکشی کد عمل یک دستور، محتویات ثبات PC را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و MemRQ خود را فعال می‌کند و انتظار دارد در پاسخ این کار، کد عمل مناسب روی گذرگاه داده‌اش ظاهر شود تا بتواند آن را بخواند و اجرا کند.

اصل دوم: پردازنده هنگام خواندن محتویات یک خانه حافظه، شماره آن خانه را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و MemRQ خود را فعال می‌کند و انتظار دارد در پاسخ این کار، محتویات آن خانه حافظه روی گذرگاه داده‌اش ظاهر شود تا بتواند آن را بخواند.

اصل سوم: پردازنده هنگام نوشتن عددی در یک خانه حافظه، شماره آن خانه را روی گذرگاه آدرس خود و عدد مورد نظر را روی گذرگاه داده‌اش قرار می‌دهد و سیگنالهای WR و MemRQ خود را فعال می‌کند.

اصل چهارم: پردازنده هنگام ارسال یک داده به خروجی، آن داده را روی گذرگاه داده و آدرس خروجی مزبور را روی گذرگاه آدرسش قرار می‌دهد و سیگنالهای WR و IORQ خود را فعال می‌کند.

اصل پنجم: پردازنده هنگام خواندن اطلاعات یک دستگاه ورودی، شماره (آدرس) آن ورودی را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و IORQ خود را فعال می‌کند و انتظار دارد در پاسخ این عمل، دستگاه ورودی مزبور اطلاعاتش را روی گذرگاه داده پردازنده قرار دهد.

اصل اول مربوط به زمان واکشی دستورات و چهار اصل دیگر به زمان اجرای دستورات مربوط می‌شوند.

از این پنج اصل، چند قانون کلی برای طراحی و سیم‌بندی سیستمها نتیجه می‌شود که باید آنها را همیشه به خاطر داشته باشیم:

- ✓ تنها یک گذرگاه داده در سیستم وجود دارد. بنابراین گذرگاه داده پردازنده و گذرگاه داده حافظه و ورودی تراشه نگهدار و خروجی تراشه بافر سه‌حالته همگی باید به هم متصل شوند.
- ✓ برای اتصال حافظه به پردازنده، خطوط آدرس آن حافظه را به **خطوط پایین آدرس پردازنده** متصل کرده و از خطوط بالای آدرس پردازنده برای انتخاب حافظه استفاده می‌کنیم. خطوط RD و WR حافظه(ها) باید به سیگنالهای MemRD و MemWR متصل شوند.
- ✓ برای اتصال خروجی به گذرگاه داده باید از تراشه نگهدار استفاده کنیم. نگهدار باید تنها هنگام اجرای دستور OUT مربوط به خودش یعنی وقتی آدرس آن خروجی روی گذرگاه آدرس پردازنده می‌آید و سیگنالهای WR و IORQ فعال می‌شوند، فعال شود و خروجی را به گذرگاه داده پردازنده متصل کند.
- ✓ برای اتصال خروجی به سیستم، باید به هر خروجی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه نگهدار) مورد توجه قرار دهیم.
- ✓ برای اتصال ورودی به گذرگاه داده باید از بافر سه‌حالته استفاده کنیم. بافر سه‌حالته باید تنها هنگام اجرای دستور IN مربوط به خودش یعنی وقتی آدرس آن ورودی روی گذرگاه آدرس پردازنده می‌آید و سیگنالهای RD و IORQ فعال می‌شوند، فعال شود و ورودی را به گذرگاه داده پردازنده متصل کند.
- ✓ برای اتصال ورودی به سیستم، باید به هر ورودی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه بافر سه‌حالته) مورد توجه قرار دهیم.

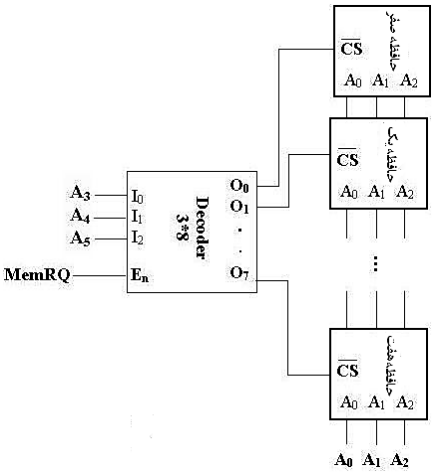
پرسش) اگر آدرس ورودیها ۹ و ۱۰ و آدرس خروجی ۱۱ باشد و بخواهیم نتیجه جمع را در خانه ۱۳ حافظه هم بنویسیم، مدار رمزگشای آدرس به چه شکل در می آید؟ (از سیگنالهای IORD و IOWR و MemRD و MemWR استفاده کنید).

پرسش) در مورد مسأله زمانبندی در طراحی سیستم مثال ۵ بحث کنید.

پرسش) خروجی سیستم ما یک عدد ۸ بیتی است که روی ۸ عدد دیود نوری نشان داده می‌شود و حداکثر آن ۱۱۱۱۱۱۱۱ یا ۲۵۵ است. حال اگر مجموعه کلید اول عدد ۱۵۰ و مجموعه کلید دوم عدد ۲۰۰ را نشان دهند، چه اتفاقی می‌افتد[1] ؟ با نوشتن معادل دودویی اعداد فوق به پرسش پاسخ دهید.

پرسش) در مثالهای ذکر شده، حداکثر تعداد کدهای عمل برنامه‌ها، ۹ بایت بود. پردازنده فرضی ما چون دارای ۴ خط آدرس است حداکثر می‌تواند تا ۱۶ بایت (0000 تا 1111) را آدرس‌دهی کند. بنابراین اگر تعداد کدهای عمل برنامه‌ای بیش از ۱۶ بایت باشد، نمی‌توان از این پردازنده استفاده کرد.

فرض کنید پردازنده‌ای با ۶ خط آدرس (A_0 تا A_5) داریم. این پردازنده می‌تواند تا 2^6 یعنی ۶۴ بایت را آدرس‌دهی کند. حال فرض کنید برنامه‌ای به حجم ۶۰ بایت داریم و تنها حافظه‌های موجود، حافظه‌های ۸ بایتی است. بنابراین باید ۸ عدد از این حافظه‌ها را به پردازنده متصل کنیم.



الف) توضیح دهید مدار رمز گشای آدرس شکل مقابل چگونه آدرس‌دهی مناسب ۶۴ بایت را فراهم می‌کند ؟ (خروجی دکودر، فعال پایین است).

ب) در صورت استفاده از مدار بالا، می‌توان سیگنال RD حافظه را مستقیماً به سیگنال RD پردازنده متصل کرد و نیازی به ساختن سیگنال MemRD نیست. چرا؟

روشهای اتصال ورودی/خروجی به سیستم

شاید تاکنون به این اندیشیده باشید که چرا اصلاً باید آدرسهای ورودی/خروجی با آدرسهای حافظه مشترک باشند تا برای متمایز کردن آنها به زحمت اسفاده از سیگنالهای IORD و IOWR و MemRD و MemWR بیافتیم؟ آیا بهتر نیست برای ورودی/خروجی‌ها از آدرسهایی استفاده کنیم که در آن آدرسها حافظه‌ای نصب نشده باشد تا دیگر نیازی به استفاده از سیگنالهای کنترلی نباشد؟

روشی که در مثال ۵ برای اتصال ورودی/خروجی به پردازنده آموختیم، به روش **اتصال ورودی/خروجی با نگاشت ورودی/خروجی** (I/O Mapped I/O) موسوم است. در این روش، آدرسهایی که به ورودی/خروجی‌ها نسبت داده می‌شود ممکن است با آدرسهای حافظه مشترک باشد و از سیگنالهای کنترلی برای متمایز کردن آنها استفاده می‌کنیم. دستورات IN و OUT در برنامه‌نویسی به این روش به کار می‌روند.

روش دیگری برای اتصال ورودی/خروجی به پردازنده وجود دارد که **اتصال ورودی/خروجی با نگاشت حافظه** (Memory Mapped I/O) نام دارد. در این روش، آدرسهای ورودی/خروجی‌ها را به گونه‌ای انتخاب می‌کنند که با آدرس خانه‌های حافظه یکسان نباشد و دیگر از سیگنالهای کنترلی برای رمزگشایی آدرس استفاده نمی کنند؛ به

[1] این مسأله به مشکل سرریز (Overflow) معروف است.

بیان دیگر آدرس ورودی/خروجی‌ها را در ادامه آدرسهای کدهای عمل برنامه در نظر گرفته و با ورودی/خروجی‌ها همانند خانه‌های حافظه رفتار می‌کنند تا سیستم ساده‌تر باشد.

در این حالت آدرس مشترکی در سیستم نخواهیم داشت و صِرف بیان آدرس یک موجودیت برای مشخص کردن آن کافی است و دیگر به سیگنالهای کنترلی برای رمزگشایی آدرس نیازی نخواهد بود (البته طبعاً سیگنالهای RD و WR پردازنده باید به حافظه متصل شوند؛ اما هدف دیگر رمزگشایی آدرس نیست).

به عنوان مثال اگر بخواهیم مثال ۵ را با این روش طراحی کنیم، آدرس ورودیها (بافرها) و خروجی (نگهدار) نمی‌تواند با آدرسهای حافظه (که در این سیستم از آدرس صفر تا ۸ به کدهای عمل برنامه اختصاص یافته است) یکسان باشد؛ مثلاً باید به بافرهای صفر و ۱ به ترتیب آدرسهای ۹ و ۱۰ و به تراشه نگهدار آدرس ۱۱ را اختصاص دهیم.یعنی دستور MOV A,[9] برای خواندن بافر شماره «صفر»، دستور MOV A,[10] برای خواندن بافر شماره «یک» و دستور MOV [11],A برای نوشتن در نگهدار مورد استفاده قرار می‌گیرد.

برنامه سیستم در این روش به شکل زیر است:

```
MOV    A, [9] ;input 1
MOV    B, A
MOV    A, [10] ;input 2
ADD    A, B
MOV    [11], A ;output
HALT
```

روش ورودی/خروجی با نگاشت حافظه از نظر طراحی ساده‌تر است؛ چون دیگر نیازی به استفاده از سیگنالهای کنترلی نیست. اما چون در این روش برای ورودی/خروجی هم از دستور MOV باید استفاده شود و تمایزی بین خواندن و نوشتن در حافظه با خواندن و نوشتن در ورودی/خروجی نیست، سیستمهای طراحی شده با این روش از نظر فهم پیچیده‌تر هستند. در این حالت کسی که برنامه این سیستم را می خواند بدون مراجعه به مدار نمی تواند بفهمد که آیا منظور از مثلاً آدرس ۱۲، آدرس یک خانه حافظه است یا یک واحد ورودی/خروجی؟ این موضوع فهم کلی سیستم را پیچیده می کند.

روش اتصال ورودی/خروجی با نگاشت ورودی/خروجی از نظر طراحی سخت افزاری (به خاطر استفاده از سیگنالهای کنترلی) مشکل‌تر و از نظر نرم افزاری ساده‌تر است. مزیت مهم این روش آن است که بر خلاف روش قبلی، در این روش یک ورودی یا خروجی می‌تواند دارای آدرس مشترک با یک خانه حافظه باشد که تفاوت آن دو از طریق سیگنالهای MemRQ و IOREQ معلوم می‌شود. به همین دلیل یک آدرس در یک سیستم می‌تواند در آنِ واحد آدرس یک ورودی و آدرس یک خروجی و آدرس یک خانه حافظه برای خواندن و آدرس یک خانه حافظه برای نوشتن باشد.

توجه کنید که در مدار مثال ۵ در هر ۱۶ بایت فضای آدرس‌دهی پردازنده حافظه نصب شده است. به همین جهت در این سیستم استفاده از روش اتصال ورودی/خروجی با نگاشت حافظه چندان معنایی ندارد (مگر اینکه سیستم را طوری طراحی کنیم که تنها در ۹ بایت اول فضای آدرس‌دهی، حافظه نصب شده باشد تا بتوان از بقیه آدرسها برای ورودی/خروجی استفاده کرد).

پرسش) علیرغم اینکه پردازنده فرضی ما دارای فضای آدرس دهی ۱۶ بایتی است، اما با استفاده از روش ورودی/خروجی با نگاشت ورودی/خروجی می‌توان به این پردازنده ۱۶ خانه حافظه بعنوان ورودی (دستور یا اطلاعات)، ۱۶ خانه حافظه به عنوان خروجی، ۱۶ ورودی و ۱۶ خروجی متصل شود. با استفاده از عملکرد سیگنالهای RD و WR و MemRQ و IORQ این موضوع را نشان دهید.

پرسش) سیستم مثال ۵ را از نظر سخت افزاری و نرم افزاری به گونه‌ای تغییر دهید که خروجی علاوه بر نمایش روی دیودهای نوری، در خانه شماره ۱۳ حافظه نیز نوشته شود. طراحی خود را با دو روش گفته شده انجام دهید.

پرسش) سیستمی طراحی کنید که محتویات یک ورودی به آدرس ۷ و یک ورودی به آدرس ۸ را بخواند و محتویات آنها را با محتویات خانه شماره ۱۵ حافظه جمع کند و نتیجه را در خانه شماره ۱۴ حافظه و خروجی شماره ۱۰ بنویسد.

با مطالبی که تاکنون گفته شد، عملکرد کلی پردازنده‌ها و چگونگی اتصال حافظه و ورودی/خروجی به آنها تا حدودی روشن شد. در فصول بعد جزییات بیشتری راجع به پردازنده های 8088/8086 را بیان خواهیم کرد.

زنگ تفریح!

نگاهی به صفحه اول کاتالوگ پردازنده ۸۰۸۸ بیاندازید و ببینید چه نکاتی از آن برداشت می‌کنید؟ در فصل بعد به این پردازنده می‌پردازیم!

intel

8088
8-BIT HMOS MICROPROCESSOR
8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with 8086 CPU
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Two Clock Rates:
  — 5 MHz for 8088
  — 8 MHz for 8088-2
- Available in EXPRESS
  — Standard Temperature Range
  — Extended Temperature Range

The Intel 8088 is a high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS-II), and packaged in a 40-pin CERDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with 8086 software and 8080/8085 hardware and peripherals.

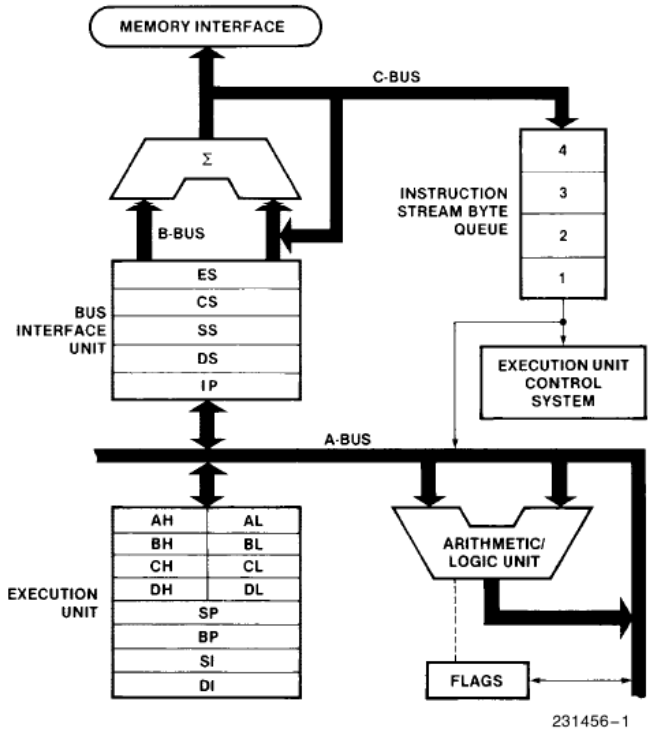


Figure 1. 8088 CPU Functional Block Diagram

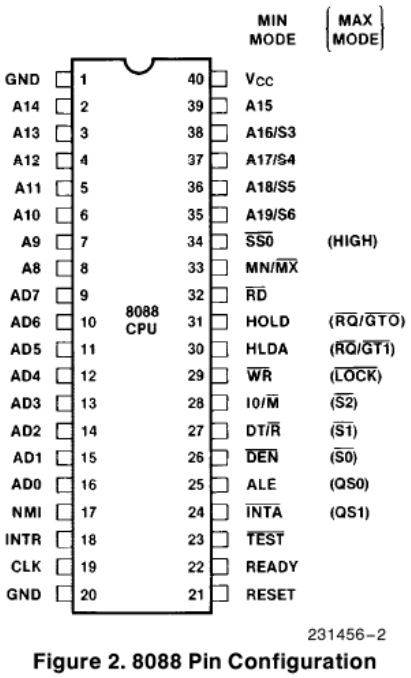


Figure 2. 8088 Pin Configuration

فصل سوم

پردازنده های ۸۰۸۶/۸۸

در فصل قبل با مباحث اساسی طراحی سیستم های مبتنی بر پردازنده ها همچون اتصال حافظه، اتصال وسایل ورودی/خروجی و رمز گشایی آدرس آشنا شدید. در این فصل می‌خواهیم پردازنده های ۸۰۸۶ و ۸۰۸۸ اینتل را به عنوان پردازنده های نمونه بررسی کنیم.

پردازنده های ۸۰۸۶/۸۸ در اواخر دهه ۱۹۷۰ به بازار عرضه شدند. ویژگی های پیشرفته این دو پردازنده از جمله معماری ۱۶ بیتی، وجود دستور العملهای ۸ بیتی و ۱۶ بیتی، فضای آدرس دهی یک مگابایتی، فضای آدرس دهی ورودی/خروجی ۶۴ کیلو بایتی، فرکانس کاری مناسب در آن زمان (حداکثر 8 MHz) و ... باعث شد این پردازنده ها محبوبیت زیادی کسب کنند و سرانجام در اوایل دهه‌ی ۱۹۸۰، اولین کامپیوترهای شخصی بر پایه پردازنده ۸۰۸۸ به بازار آمدند.

این موضوع تحول عظیمی در صنعت پردازنده ها ایجاد کرد که مهمترین آن سوق دادن راه تکامل پردازنده ها به سمت استفاده بهتر از آنها در کامپوترهای شخصی بود. امکانات حافظه مجازی، چند کاربری بودن، مدیریت حافظه و ... که در نسلهای بعدی این پردازنده یعنی ۸۰۲۸۶ گنجانده شده بود، در حقیقت پاسخی به نیازهای کاربران کامپیوترهای شخصی بود. در کنار تحولات سخت افزاری پردازنده ها، سیستم های عامل نیز به سرعت رشد کردند و اینکه امروز شاهد وجود چند صد میلیون کامپیوتر شخصی در دنیا هستیم، جز با پیشرفت موازی پردازنده ها و نرم افزارهای مناسب برای استفاده از آنها نمی توانست محقق شود.

علیرغم اینکه پردازنده‌های پنتیوم که در کامپیوترهای امروزی به صورت گسترده مورد استفاده قرار می‌گیرند، دارای امکانات فوق العاده‌ای هستند که به بخشی از آنها در فصل قبل اشاره شد، اما پایه‌ی اساسی آنها همان پردازنده های ۸۰۸۶/۸۸ است؛ بنابراین برای آشنایی با پردازنده های پیشرفته و امکانات آنها آشنایی با ۸۰۸۶/۸۸ ضروری است. به علاوه معماری کامپیوترها که در ابتدا بر اساس ۸۰۸۸ بنا نهاده شد، تاکنون به صورت اساسی تغییر نکرده و آشنایی با پردازنده ۸۰۸۸ و کامپوتر PC-XT که اولین نسل کامپوترهای شخصی مبتنی بر ۸۰۸۸ است، برای شناسایی ساختار داخلی کامپیوتر و نحوه استفاده از امکانات آن مفید است.

هدف این فصل معرفی سخت‌افزار و نرم افزار ۸۰۸۶/۸۸ است؛ البته به جای توضیح مفصل در رابطه با کلیه امکانات این پردازنده ها که موضوعی پیچیده، وقتگیر و تاحدی غیرضروری است، کوشیده ایم با پررنگ کردن ویژگیهایی از این پردازنده ها که در نسلهای بعدی پردازنده های اینتل گسترش یافته و به تغییرات مهمی در معماری پردازنده ها منجر شد، به تشریح ساختار کاری ۸۰۸۶/۸۸ بپردازیم؛ به صورتی که پس از مطالعه این فصل نکات لازم جهت طراحی یک سیستم کنترلی مبتنی بر ۸۰۸۸ را آموخته باشید.

ساختار سخت افزاری پردازنده های ۸۰۸۶/۸۸

همانطورکه گفته شد، پردازنده ۸۰۸۵، پردازنده ای با معماری ۸ بیتی بود. اینتل طی یک تحول مهم در سال ۱۹۷۸ پردازنده ۸۰۸۶ را معرفی کرد که چه از نظر ساختار داخلی (گذرگاه داده داخلی) و چه از نظر تعداد پینهای داده

(گذرگاه داده خارجی)، یک پردازنده ۱۶ بیتی بود. به رغم انتظار اینتل، از پردازنده ۸۰۸۶ استقبال چندانی نشد؛ دلیل اصلی این امر، وجود هزاران دستگاه کنترلی مبتنی بر گذرگاههای داده ۸ بیتی منطبق بر۸۰۸۵ بود که تغییر ناگهانی آنها به سیستم های ۱۶ بیتی هیچ توجیه اقتصادی نداشت.

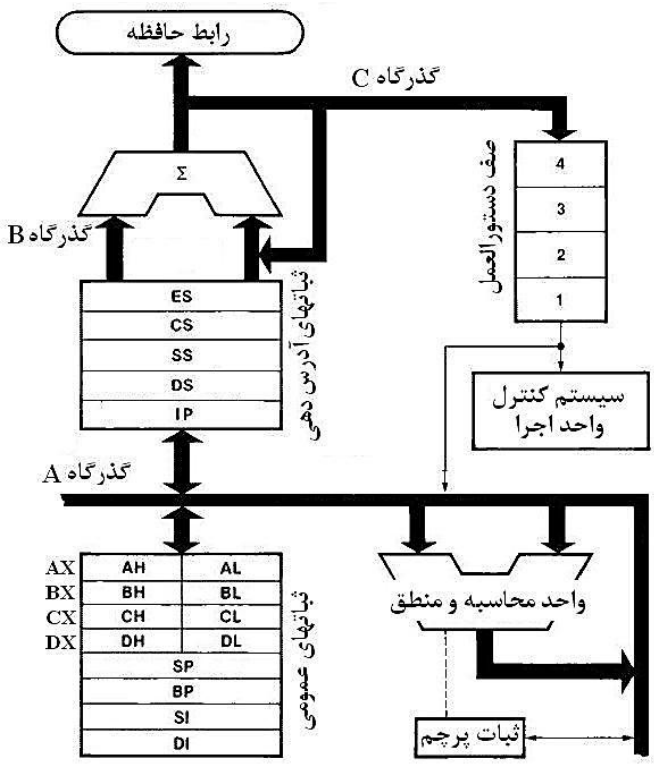
فروش ۸۰۸۶ در سال ۱۹۷۸ یک شکست مطلق بود؛ اما اینتل به جای عقب نشینی در مقابل این مشکل، دست به کاری جالب زد که در نوع خود در آن زمان بی نظیر بود. اینتل در سال ۱۹۷۹ پردازنده ۸۰۸۸ را معرفی کرد که از نظر معماری داخلی تطابق کامل با ۸۰۸۶ داشت؛ بنابراین می‌توانست از ویژگیهای مفید معماری آن مثل صف دستورالعمل، ثباتها و دستورات ۱۶ بیتی و ... استفاده نماید. اما ۸۰۸۸ بر خلاف ۸۰۸۶ دارای گذرگاه داده خارجی ۸ بیتی (۸ پین برای گذرگاه داده) بود تا بتواند به راحتی جایگزین پردازنده سیستم های ۸ بیتی آن زمان شود. استقبال فوق‌العاده از ۸۰۸۸ و استفاده از آن به عنوان پردازنده اولین نسل کامپیوترهای شخصی، پاسخی به ایده جالب اینتل در مورد تغییر ساختار ۸۰۸۶ بود.

به مرور با ایجاد سیستم های ۱۶ بیتی، استفاده از پردازنده ۸۰۸۶ و نسلهای بعدی آن متداول شد و جایگاه ۸۰۸۸ به عنوان پلی بین پردازنده های ۱۶ بیتی و ۸ بیتی در عرصه پردازنده‌ها محفوظ ماند.

بنابر آنچه گفته شد، پایه اساسی بحث ما پردازنده ی ۸۰۸۸ خواهد بود و در قسمت آخر این فصل گذری مختصر بر ۸۰۸۶ نیز خواهیم داشت. کاتالوگ پردازنده ۸۰۸۸ نیز در پیوست کتاب آورده شده است.

معماری پردازنده ۸۰۸۸

معماری داخلی پردازنده ۸۰۸۸ را در شکل ۳-۱ می‌بینید.



شکل ۳-۱- معماری داخلی پردازنده ۸۰۸۸

قسمتهای مهم این ساختار عبارتند از :

- واحد محاسبه آدرس

در۸۰۸۸ یک واحد محاسبه آدرس در نظر گرفته شده که با استفاده از ثباتهای آدرس دهی IP، DS، SS، CS و ES عمل استخراج آدرس و تبدیل آدرس منطقی به آدرس واقعی (فیزیکی) را انجام دهد. در بخش آینده به معنای این عمل پی خواهیم برد.

- رابط حافظه

آدرس محاسبه شده در واحد محاسبه آدرس به رابط حافظه داده می شود تا با استفاده از آن تبادل داده مناسب با خارج پردازنده را انجام دهد.

- صف دستورالعمل

یکی از مهمترین قسمتهای ساختار داخلی ۸۰۸۸، بخش صف دستورالعمل است. این بخش مبنای مباحث مهمی در علم پردازنده ها مانند Cache، چند وظیفگی، موازی‌سازی و ... است.

در فصل قبل خواندید که پردازنده برای اجرای یک دستورالعمل باید سه مرحله واکشی یا دریافت دستور از حافظه (Fetch) ، رمزگشایی دستور (Decode) و اجرای دستور (Execution) را انجام دهد.

در پردازنده نسل قبل از ۸۰۸۸ ، یعنی ۸۰۸۵، این عمل به صورت پشت سرهم انجام می شد؛ یعنی ۸۰۸۵ برای اجرای یک دستورالعمل، آن را از حافظه دریافت، رمزگشایی و سپس اجرا می کرد. سپس این سه مرحله برای دستور بعدی دنبال می شد. این روش ساده است، اما با استفاده از امکانات ۸۰۸۸ می توان با نوعی موازی سازی این روش را بهبود بخشید.

همانطور که در شکل معماری داخلی ۸۰۸۸ می بینید، واحد محاسبه آدرس از واحد اجرای دستور مجزاست. این موضوع به ۸۰۸۸ این امکان را می دهد که در حین اجرای یک دستور (**موازی با اجرای یک دستور**) تا ۴ دستور بعد را از حافظه دریافت کرده و در یک صف ذخیره کند تا به موقع رمزگشایی و اجرا شوند. این کار باعث کاهش زمان کلی اجرای برنامه می شود.

طول صف دستورالعمل در ۸۰۸۶/۸۸ ، ۴ دستور است که در پردازنده های بعدی افزایش یافت و در نهایت به ایجاد Cache که در پردازنده های فعلی یکی از معیارهای قدرت آنها به شمار می رود، منجر گردید؛ بعلاوه ایده صف دستورالعمل آغاز بحث موازی‌سازی در پردازنده ها نیز به شمار می‌رود.

ایده جدا کردن مرحله واکشی از مراحل رمزگشایی و اجرا، در پردازنده های بعدی اینتل منجر به ایده خط لوله[1] شد که یکی از موضوعات بسیار مهم معماری پردازنده های فعلی به شمار می رود.

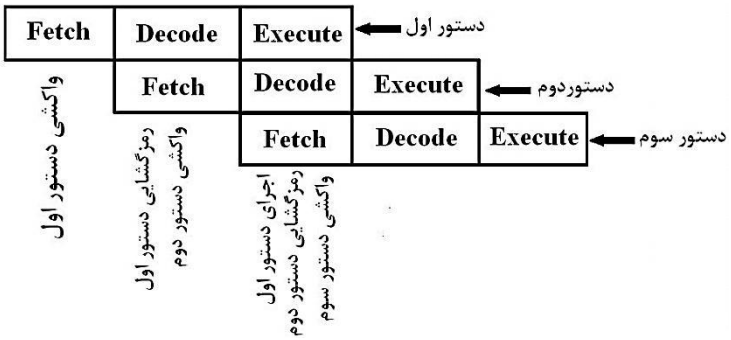
خط لوله

همانطور که گفته شد پردازنده ۸۰۸۵ سه مرحله واکشی، رمزگشایی و اجرا را برای هر دستور به طور مجزا و **مستقل از دستورات قبلی و بعدی** انجام می دهد. این موضوع باعث می شود که نتوانیم از امکانات پردازنده به درستی استفاده کنیم. *چون وقتی ۸۰۸۵ دستور اول را واکشی می کند و سپس آن را به واحد رمزگشایی می سپارد، در حین*

[1] Pipelining

رمزگشایی دستور اول واحد واکشی بیکار است، در حالی که واکشی دستور دوم به رمزگشایی دستور اول هیچ ارتباطی ندارد و این دو عمل وابسته به هم نیستند و می توانند موازی هم اجرا شوند. به نحو مشابه وقتی ۸۰۸۵ دستور اول را پس از رمزگشایی به واحد اجرا می فرستد، واحدهای واکشی و رمزگشایی بیکارند؛ در حالی که در همین زمان می‌توانند به ترتیب به واکشی دستور سوم و رمزگشایی دستور دوم بپردازند.

اینتل در پردازنده های بعدی خود، با بهره گیری از ایده صف دستورالعمل ۸۰۸۸ به موضوع استفاده از واحدهای واکشی، رمزگشایی و اجرای دستور در زمانهای بیکاری جامه عمل پوشاند. این تکنیک **به نام خط لوله** معروف است که نمای آن را در شکل ۳-۲ مشاهده می کنید. این نام از تفاوت روشهای آوردن آب از جایی به جای دیگر گرفته شده است؛ به جای آوردن آب به صورت سطل سطل از چاه (مانند آنچه ۸۰۸۵ برای اجرای دستورات انجام می‌دهد) می توان یک خط لوله از چاه به منزل کشید تا آب به صورت جاری و پشت سرهم منتقل شود (مانند آنچه درپردازنده های دارای خط لوله می گذرد).



شکل ۳-۲- خط لوله

- سیستم کنترل واحد اجرا

این واحد پس از دریافت دستورالعمل بعدی، دستورات کنترلی لازم برای اجرای آن را به قسمتهای مختلف پردازنده ارسال می کند.

- واحد محاسبه و منطق (ALU)

این واحد علاوه بر انجام اعمال اساسی محاسباتی و منطقی، در ۸۰۸۸ قادر به انجام اعمال ضرب و تقسیم نیز می باشد.

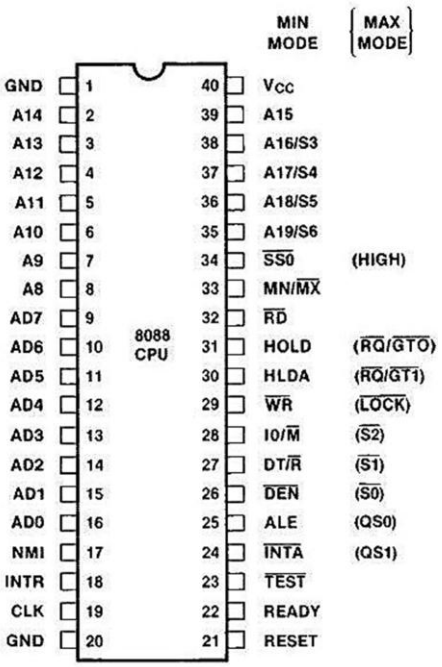
- ثبات های داخلی

۸۰۸۸ به جز ثباتهای آدرس دهی، دارای ۸ ثبات عمومی ۱۶ بیتی به نام های AX،BX، CX،DX (۴ ثبات فوق، به ثباتهای همه منظوره معروفند)، SP،BP، SI و PI است که با کاربردهای آنها در بخشهای بعد آشنا خواهید شد.

نکته قابل ذکر این است که چون ۸۰۸۸ قابلیت اجرای دستورات ۸ بیتی و ۱۶ بیتی را دارد، ثباتهای همه منظوره آن به صورت دو ثبات ۸ بیتی مجزا نیز قابل دسترسی اند؛ مثلاً ثبات ۱۶ بیتی AX خود شامل ۲ ثبات ۸ بیتی AH (۸ بیت بالای AX) و AL (۸ بیت پایین AX) است.

-گذرگاههای A و B و C

این گذرگاهها، مسیرهای داخلی پردازنده برای جابجایی اطلاعات مختلف مانند کدهای عملیاتی یا داده هاست.



شکل ۳-۳- پایه‌های پردازنده ۸۰۸۸

پینهای پردازنده ۸۰۸۸: پس از بررسی معماری داخلی ۸۰۸۸، به بررسی پینهای این تراشه۴۰ که در شکل ۳-۳ نشان داده شده است، می‌پردازیم.

پین ورودی $MN/\overline{MX}$ (33): چون تعداد پینهای ۸۰۸۸ محدود است، برای گنجاندن کاربردهای زیادتر در همین تعداد پین محدود، دو مدکاری برای ۸۰۸۸ به نام مدهای مینیمم (Min) و ماکزیمم (Max) تعریف شده است. اگر این پین به ولتاژ ۵+ ولت («یک» منطقی) متصل باشد، ۸۰۸۸ در مد مینیمم کار می کند و پینهای آن طبق نامهایی که در جلو آنها ذکر شده است، نامیده شده و عمل می کنند؛ اما اگر این پین به زمین (صفر منطقی) متصل باشد، معانی تعدادی از پینها عوض شده و به جای معانی اصلی، طبق نامهایی که در پرانتز جلوی آنها ذکر شده است، نامیده شده و عمل می کنند. در این فصل به شرح پینها در مد مینیمم می‌پردازیم.

برای شناخت پینهای مد مینیمم، در ابتدا به جستجوی پینهایی می‌پردازیم که در پردازنده فرضی فصل قبل به آنها برخورده ایم.

گذرگاه داده: پینهای AD0 تا AD7 (پینهای ۱۶ تا ۹) ، پینهای مربوط به گذرگاه داده خارجی ۸۰۸۸ هستند (به یاد می آورید که گذرگاه داده **داخلی** ۸۰۸۸ مانند ۸۰۸۶، ۱۶ بیتی است و گذرگاه **خارجی** آن برای تطابق با دستگاههایی که گذرگاه داده ۸ بیتی دارند، به صورت ۸ بیتی در نظر گرفته شده است). همانطور که در فصل قبل دیدید، چون پردازنده برای ارسال و دریافت اطلاعات از گذرگاه داده خود استفاده می کند، این ۸ پین به صورت دوطرفه[1] طراحی شده اند.

گذرگاه آدرس: پینهای A15 تا A19 (۳۹ تا ۳۵) و A8 تا A14 (۸ تا ۲) ، پینهای مربوط به گذرگاه آدرس ۸۰۸۸ هستند. پس A0 تا A7 کجاست؟! باز هم پینهای ۱۶ تا ۹ را ببینید. نام AD (Address/Data) بیانگر این نکته است که این پینها هم به عنوان گذرگاه داده و هم به عنوان ۸ بیت ابتدایی گذرگاه آدرس به کار رفته اند. در واقع به دلیل صرفه جویی در تعداد پینهای تراشه، در پردازنده های ۸۰۸۶/۸۸ پینهای مربوط به گذرگاه داده با تعدادی از پینهای گذرگاه آدرس مشترکند[2]؛ در واقع تعدادی از پینها هم نقش گذرگاه داده و هم نقش قسمتی از گذرگاه آدرس را بازی می کنند.

چگونه چنین چیزی ممکن است؟!

طبیعی است که در یک لحظه یک پین باید یا محتوی داده باشد یا محتوی آدرس؛ بنابراین در لحظه T، ۸ پین AD0 تا AD7 محتوی آدرس (A0 تا A7) هستند که به همراه پینهای A8 تا A19 ، گذرگاه آدرس ۸۰۸۸ را تشکیل می‌دهند و در لحظه 'T این ۸ پین محتوی داده (D0 تا D7) می‌باشند.

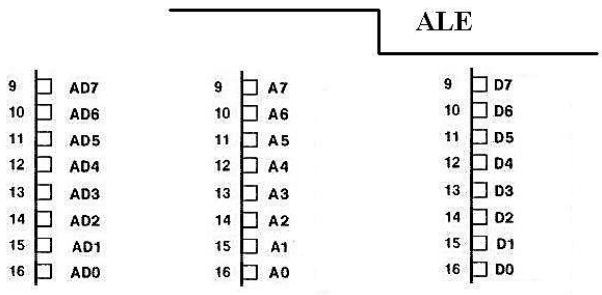
[1] Bidirectional
[2] Multiplexed

اولین سؤالی که به ذهن می رسد این است که آیا ابتدا لحظه T فرا می رسد یا 'T ؟ به بیان دیگر برای اجرای یک دستور آیا لازم است که پینهای AD0 تا AD7 ابتدا برای آدرس دهی به کار روند یا برای ارسال و دریافت داده ها؟

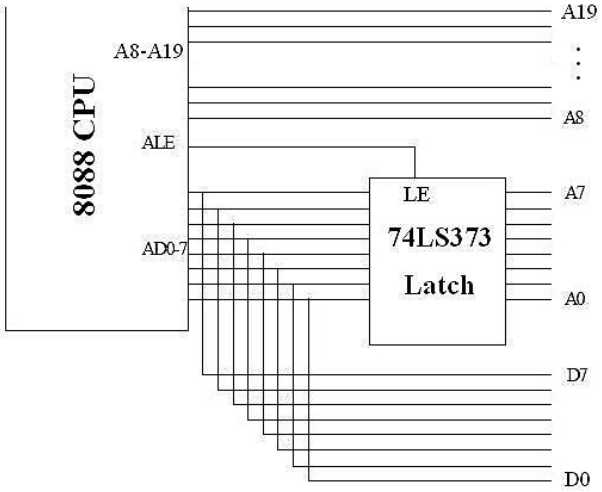
اگر مطالب فصل قبل را به دقت خوانده باشید، پاسخ شما «**آدرس**» خواهد بود. همانگونه که گفته شد، شماره ای (آدرسی) که پردازنده روی گذرگاه آدرس خود می گذارد، بیانگر هویت واحد جانبی است که مجاز به استفاده از گذرگاه داده می باشد. *بنابراین طبیعی است که ابتدا باید آدرس به صورت کامل مشخص شود تا هویت واحدی که مجاز به استفاده از گذرگاه داده است تعیین گردد و سپس گذرگاه داده در اختیار آن قرار گیرد.* بنابراین پینهای AD0 تا AD7 در ابتدا حاوی ۸ بیت ابتدایی آدرس (A0 تا A7) هستند تا بعلاوه پینهای دیگر آدرس (A8 تا A19) شماره واحدی که می‌تواند از گذرگاه داده استفاده کند را معلوم کنند و سپس این ۸ بیت به صورت گذرگاه داده (D0 تا D7) عمل می‌کنند تا برای ارسال و دریافت داده ها مورد استفاده قرار گیرند.

پرسش دوم این است که سیستم باید به چه نحوی طراحی شود تا در زمان مناسب با این پینها به صورت پین آدرس و در زمان مناسب به صورت پین داده رفتار کند؟

پین خروجی ALE (Address Latch Enable) (۲۵): این پین توسط پردازنده برای اعلان وجود آدرس روی پینهای AD0 تا AD7 به کار می‌رود. بنا بر آنچه گفته شد، در ابتدای اجرای دستور این پین به نشانه وجود آدرس، «یک» می‌شود و پس از مدت زمانی، به نشانه اختصاص پینهای AD0 تا AD7 به گذرگاه داده صفر می‌شود. توجه کنید که سیستم کنترلی برای رمزگشایی آدرس به طور دائم به آدرس نیاز دارد. پینهای A8 تا A19 که در طول اجرای دستور حاوی بیتهای آدرس هستند؛ بنابراین باید به نحوی ۸ بیت پایینی آدرس در مدت زمان کوتاهی که در ابتدای اجرای دستور روی پینهای AD0 تا AD7 قرار می گیرند، جایی قفل یا نگهداشته شوند تا در ادامه کار که این پینها به گذرگاه داده اختصاص می‌یابند، بتوانند در کنار A8 تا A19 برای رمزگشایی آدرس به کار روند.



برای این کار از سیستم مقابل استفاده می کنیم. وقتی پردازنده A0 تا A7 را روی پینهای AD0 تا AD7 قرار می دهد، پین ALE را «یک» می‌کند. بنابراین نگهدار فعال شده و A0 تا A7 را در خروجی خود می‌گذارد. پس از مدت زمانی که پردازنده می‌خواهد AD0 تا AD7 را به D0 تا D7 اختصاص دهد، ALE را صفر می‌کند. پس نگهدار غیرفعال می شود و A0 تا A7 را در خروجی خود نگه می دارد که در کنار A8 تا A19 آدرس ۲۰ بیتی مورد نظر ۸۰۸۸ را تشکیل دهد. از این پس پینهای AD0 تا AD7 به D0 تا D7 اختصاص خواهد داشت.



پرسش) با توجه به اینکه گذرگاه آدرس یکطرفه و گذرگاه داده دو طرفه است، در چه زمانی AD0 تا AD7 یکطرفه و در چه زمانی دو طرفه است؟

پرسش) باتوجه به ۲۰ بیتی بودن گذرگاه آدرس ۸۰۸۸، فضای آدرس دهی این پردازنده چقدر است؟

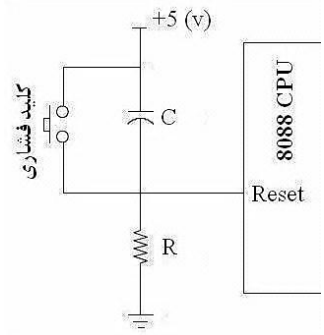
پرسش) پردازنده ۸۰۸۸ برای مراجعه به ورودی/خروجی تنها از ۱۶ بیت گذرگاه آدرس ۲۰ بیتی خود استفاده می کند. این پردازنده مجموعاً می تواند چند وسیله ورودی/خروجی را بشناسد؟ (این فضای آدرس دهی ورودی/خروجی در پردازنده های بعدی اینتل نیز ثابت ماند).

پین ورودی CLK (Clock) (۱۹): پالس هماهنگ کننده فعالیتهای داخلی ۸۰۸۸ (پالس ساعت) از طریق این پین به پردازنده داده می‌شود. در فصل بعد با نحوه تولید این پالس ساعت توسط تراشه 8284 آشنا خواهید شد. فرکانس این پالس ساعت در ۸۰۸۸ پایه برابر 5MHZ و در ۲-۸۰۸۸ که نسخه پیشرفته تر ۸۰۸۸ است، برابر 8 MHZ می باشد.

پینهای خروجی $\overline{RD}$ و $\overline{WR}$ (۳۲ و ۲۹): پردازنده از این دو پین برای اعلام اینکه در حال خواندن یا نوشتن داده است استفاده می کند.

پین خروجی IO/$\overline{M}$ (۲۸): پردازنده با قرار دادن «یک» روی این پین اعلام می کند که مشغول تبادل داده با وسایل ورودی و خروجی است و وجود «صفر» روی این پین بیانگر تبادل داده پردازنده با حافظه است. معمولاً از ترکیب ۳ سیگنال اخیر، ۴ سیگنال مهم در پردازنده ها می‌سازیم که رده تبادل داده پردازنده با خارج را نشان می دهد.

پرسش : با توجه به ۶۴ کیلو بایتی بودن فضای آدرس دهی ورودی/خروجی در ۸۰۸۸، هنگامی که IO/M برابر یک است، مقادیر موجود روی کدامیک از پینهای گذرگاه آدرس معتبر هستند؟



پین ورودی Reset (۲۱): با فعال شدن این پین، ۸۰۸۸ اجرای برنامه فعلی را قطع کرده و به وضعیت اولیه پردازنده هنگام روشن شدن (که بعداً بررسی خواهیم کرد) باز می‌گردد و اجرای برنامه را از سر می‌گیرد. این پین حداقل برای ۴ سیکل ساعت باید (یک) باقی بماند تا عمل Reset به درستی انجام شود.

معمولاً برای اینکه علاوه بر هنگام نیاز، هنگام اتصال برق به پردازنده نیز عمل Reset به طور اتوماتیک انجام شود، از مداری به نام Auto-Power on-Reset استفاده می شود که شکل مقابل این مدار را نشان می دهد.

هنگام روشن شدن سیستم (متصل شدن ولتاژ ۵+ ولت)، خازن اتصال کوتاه است و پین Reset به ۵+ ولت متصل و فعال می‌شود. خازن شروع به شارژ شدن می کند و پس از مدتی که به مقادیر C و R بستگی دارد، شارژ شده و اتصال بین پین Reset و ۵+ ولت را قطع می کند.

هرگاه نیاز به Reset مدار بود، با فشردن کلید فشاری پین Reset به ۵+ ولت متصل شده و خازن تخلیه می‌شود. با رها کردن کلید، خازن مجدداً شارژ شده و ارتباط پین Reset با ۵+ ولت را قطع می کند.

پرسش: مقادیر R و C چگونه انتخاب می شوند؟

معمولاً سیگنال Reset پردازنده ۸۰۸۸ توسط تراشه ۸۲۲۴ تأمین می شود که در فصل بعد مورد بررسی قرار خواهد گرفت.

پینهای HOLD و HLDA (۳۱ و ۳۰): این دو پین برای در اختیار گرفتن گذرگاههای سیستم توسط وسایلی غیر از پردازنده (مثلاً در DMA) به کار می‌روند

پینهای خروجی $\overline{DT/R}$ و $\overline{DEN}$ (۲۷ و ۲٦): این دو پین از سیگنالهای لازم در طراحی سیستم هستند. به طور خلاصه بدانید که پین DEN برای فعال کردن بافرهای متصل به گذرگاه سیستم و پین DT/R برای مشخص کردن جهت جریان داده در این بافرها به کار می رود.

پین ورودی Ready (۲۲): اگر حافظه یا وسیله ورودی/خروجی قادر نباشد با سرعت پردازنده کار کند باید به نحوی پردازنده را معطل کند تا بتواند قبل از آنکه پردازنده به سراغ دستور بعدی برود، تبادل داده مورد نظر دستور فعلی با پردازنده را به پایان رساند. پین Ready برای وادار کردن پردازنده به طولانی تر کردن جریان کار عادی به کار می‌رود.

پینهای INTR و NMI و $\overline{INTA}$ (۱۸ و ۱۷ و ۲٤): این پینها مخصوص سرویس دادن به وقفه هستند. اجمالاً بدانید که واحدهای جانبی سیستم می‌توانند با ارسال سیگنالهایی به پینهای INTR و NMI پردازنده، باعث شوند پردازنده در کاری که مشغول انجام آن است وقفه ایجاد کرده و به واحد وقفه دهنده سرویس ارائه کند و سپس به ادامه کار قبلی خود بپردازد. پردازنده از طریق پین INTA ، رسیدن سیگنال وقفه را تصدیق می کند.

پین ورودی $\overline{Test}$ (۲۳): این پین برای هماهنگی سخت افزار و نرم افزار به کار می‌رود.

پینهای خروجی آدرس/وضعیت (A19/S6–A18/S5 –A17/S4 –A16/S3) (۳۵-۳۸): این پینها در ابتدای اجرای دستور حاوی ۴ بیت بالای آدرس و در ادامه نشان دهنده وضعیت جاری سیستم از جمله سگمنت در حال استفاده و وضعیت پرچم وقفه می باشند. به کمک این پینها ارتباط مناسبی با تراشه های کمکی برقرار می شود. جزئیات را بعدها خواهید دید.

پین خروجی $\overline{SSO}$ (۳٤): این پین به همراه پینهای IO/M و DT/R ، برای نشان دادن حالت فعلی گذرگاه سیستم و تشخیص سیکلها (مانند سیکل خواندن حافظه، سیکل نوشتن در ورودی/خروجی و ...) به کار می‌رود.

الگوی برنامه نویسی ۸۰۸۶/۸۸

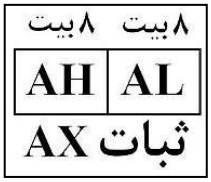
الگوی برنامه نویسی یک پردازنده شامل دو بخش است:

۱- آشنایی با معماری داخلی پردازنده: که شامل آشنایی با ثباتها، عملکرد و اندازه آنها و شیوه دسترسی پردازنده به حافظه و ورودی/خروجی هاست.

۲- آشنایی با قواعد زبان اسمبلی پردازنده

برای فراگیری مدل برنامه نویسی پردازنده ۸۰۸۶/۸۸ ابتدا به معماری داخلی آن می‌پردازیم.

ثباتهای ۸۰۸۶/۸۸



الف) ثباتهای همه منظوره: ثباتهای همه منظوره AX و BX، CX و DX ثباتهای همه منظوره هستند که برای مقاصد مختلف قابل استفاده هستند. ویژگی منحصر به فرد این ثباتها قابلیت استفاده از آنها به صورت ۸ بیتی و ۱۶ بیتی است. مثلاً ثبات ۱۶ بیتی AX شامل یک بخش AH (۸ بیت بالایی) و AL (۸ بیت پایینی) است که می توان به هر بخش با نام آن مراجعه کرد.

در زیر با این ثباتها به صورت مختصر آشنا می‌شویم.

ثبات AX (Accumulator): ثبات انباره اصلی است و در دستورالعملهای ورودی/خروجی و محاسباتی مورد استفاده قرار می گیرد.

ثبات BX (Base): این ثبات به عنوان یک اندیس برای توسعه آدرس دهی و مراجعات جدولی به حافظه به کار می رود. کاربرد دیگر آن در انجام محاسبات است.

ثبات CX (Counter): از این ثبات معمولاً برای شمارش دفعات تکرار یک حلقه و نیز در محاسبات استفاده می شود.

ثبات DX (Data): از این ثبات در عملیات ورودی/خروجی به عنوان آدرس پورت و نیز در عملیاتهای ضرب و تقسیم با اعداد بزرگ استفاده می شود.

ب) ثبات پرچم: در هر پردازنده یک ثبات به نام پرچم وجود دارد که بیتهای آن، وضعیت پردازنده بعد از انجام یک عمل را نشان می‌دهند. بیتهای ثبات پرچم ۱۶ بیتی ۸۰۸۶/۸۸ را در شکل ۳-۴ می بینید.

--	--	--	--	O	D	I	T	S	Z	--	A	--	P	--	C

شکل ۳-۴- ثبات پرچم پردازنده ۸۰۸۸

بیتهای خالی در ۸۰۸۶/۸۸ تعریف نشده اند. در زیر شرح مختصر بیتهای تعریف شده را می‌بینید.

بیت C (Carry): بیت رقم نقلی است. چنانچه یک عمل محاسباتی، تولید رقم نقلی کند (مثلاً جمع دو عدد بدون علامت که تولید سرریز کند) این بیت یک می شود. به‌علاوه این بیت حاوی رقم نقلی در عملیات منطقی شیفت نیز می باشد.

بیت D (Direction): بیت جهت است و توسط برنامه نویس برای کنترل جهت اعمال رشته ای مانند مقایسه یا انتقال رشته مقداردهی می شود.

بیت P (Parity): بیت توازن است و برای کنترل صحت اطلاعات ۸ بیت پایینی نتیجه یک عمل از طریق توازن فرد به کار می رود.

بیت A (Auxilary Carry): بیت رقم نقلی کمکی است و چنانچه در محاسبات ۸ بیتی رقم نقلی در بیت سوم ایجاد شود، برابر یک می شود. از این بیت در اعمال ریاضی BCD استفاده می شود.

بیت Z (Zero): بیت نشانگر نتیجه صفر است. چنانچه نتیجه یک عمل محاسباتی یا منطقی صفر باشد، این بیت برابر یک می شود.

بیت S (Sign): بیت علامت و منعکس کننده بیت علامت نتیجه آخرین عمل پردازنده است و از آن معمولاً برای تعیین علامت نتیجه استفاده می‌شود.

بیت T (Trap): بیت اجرای قدم به قدم است. چنانچه این بیت توسط برنامه نویس یک شده باشد، پردازنده بعد از اجرای هر دستور دچار وقفه خاصی می‌شود که نتیجه آن اجرای قطعه برنامه خاصی است که توسط برنامه نویس در مکان خاصی از حافظه نوشته شده و شامل دستوراتی برای نمایش محتویات ثباتهای داخلی و حافظه است.پس از استفاده کاربر از نتیجه‌این کار، با صدور فرمانی اجرای برنامه ادامه می‌یابد. از این بیت برای اشکال‌زدایی برنامه‌ها به کار می رود.

بیت I (Interrupt): بیت وقفه است و چنانچه توسط برنامه نویس صفر شده باشد، پردازنده به وقفه‌ها پاسخ نمی دهد.

بیت O (Overflow): بیت سرریز است و در صورت ایجاد سرریز در اعمال محاسباتی علامتدار، یک می شود.

ج) ثباتهای آدرس دهی: مجموع ثباتهای سگمنت (CS، DS، SS و ES) و ثباتهای اشاره‌گر (BP، SP، IP) و ثباتهای اندیس (SI و DI) که همگی ۱۶ بیتی هستند، برای آدرس دهی حافظه به کار می‌روند.
پیش از مرور عملکرد آنها، نحوه مراجعه به حافظه در پردازنده ۸۰۸۶/۸۸ را بررسی می کنیم.

سگمنت[1]های حافظه: برای مراجعه به حافظه در ۸۰۸۶/۸۸ (و پردازنده‌های بعدی اینتل)، حافظه به صورت قطعه‌هایی در نظر گرفته می‌شود که سگمنت نامیده می‌شوند. اندازه یک سگمنت حافظه حداکثر ۶۴ کیلوبایت است.

چهار نوع سگمنت حافظه وجود دارد که آنها را در ذیل بررسی می کنیم:

سگمنت کد (Code Segment): دستورالعملهای زبان ماشین که باید توسط پردازنده اجرا شوند در ناحیه ای از حافظه به نام سگمنت کد ذخیره می شوند؛ یعنی اولین دستور اجرایی در ابتدای این سگمنت قرار می گیرد. اگر کد برنامه بیش از ۶۴ کیلوبایت باشد، باید چند سگمنت کد در حافظه تعریف شود.

[1] Segment

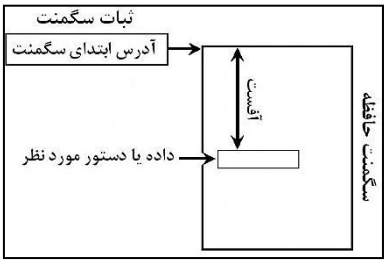
سگمنت داده (Data Segment): داده‌های مورد نیاز برنامه در این سگمنت قرار می گیرند. یک برنامه می تواند چند سگمنت داده داشته باشد.

سگمنت پشته (Stack Segment): اطلاعاتی که باید پیش از فراخوانی زیربرنامه‌ها در حافظه ذخیره شود تا بعد از بازگشت به برنامه اصلی بازیابی شود، در این سگمنت ذخیره می شود.

سگمنت اضافی (Extra Segment): این سگمنت در اعمال رشته ای مورد استفاده قرار می‌گیرد.

ثباتهای سگمنت: برای مراجعه به محتویات یک سگمنت باید آدرس آغاز آن سگمنت را بدانیم. ثباتهای ۱۶ بیتی سگمنت برای همین منظور به کار می روند: ثبات CS آدرس ابتدای سگمنت کد، ثبات DS آدرس ابتدای سگمنت داده، ثبات SS آدرس ابتدای سگمنت پشته و ثبات ES آدرس ابتدای سگمنت اضافی را در خود نگه می‌دارند.

توجه به یک نکته ضروری است. چون گذرگاه آدرس ۸۰۸۶/۸۸، ۲۰ بیتی است، قاعدتاً باید آدرس آغاز سگمنت‌ها که مکانهایی در حافظه هستند نیز ۲۰ بیتی باشد؛ پس چگونه آدرس آغاز یک سگمنت که ۲۰ بیتی است در ثبات ۱۶ بیتی سگمنت ذخیره می شود ؟!



آدرس آغاز یک سگمنت در حافظه که آدرسی ۲۰ بیتی است، باید همیشه آدرسی بخشپذیر بر ۱۶ باشد؛ مثل 00000 H یا F23E0 H یا 2A320 H یا .... به همین دلیل همیشه ۴ بیت پایینی آدرس آغاز سگمنت صفر است و نیازی به ذخیره ندارد. مثلاً وقتی گفته می شود آدرس آغاز سگمنت کد که در CS ذخیره می شود، 2B3F H است، در واقع منظور آدرس ۲۰ بیتی 2B3F0 H است. بنابراین برای دستیابی به آدرس واقعی سگمنت، باید آن را ۴ بیت به سمت چپ شیفت دهیم.

ثباتهای اشاره گر و اندیس: ثباتهای ۱۶ بیتی اشاره‌گر و اندیس حاوی فاصله دستورات و داده‌ها از مبدأ سگمنت هستند که اصطلاحاً آفست[1] نامیده می شود. منظور از آفست، فاصله آدرس داده یا دستورالعمل از ابتدای سگمنت آن است.

پرسش) با توجه به ۱۶ بیتی بودن ثباتهای آفست، نشان دهید که حداکثر اندازه یک سگمنت ۶۴ کیلوبایت است.

ثباتهای اشاره گر

ثبات اشاره گر دستور (IP[2]): این ثبات حاوی فاصله دستور اجرایی بعدی از آغاز سگمنت کد است. آدرس دستورالعملها در برنامه‌نویسی به صورت CS:IP مشخص می شود؛ مثلاً وقتی به دستور واقع در مکان 2E35:0100 رجوع می شود، در واقع به دستوری که از مکان آغاز سگمنت کد (2E350) به اندازه 0100 واحد (۲۵۶ بایت) فاصله دارد، مراجعه می شود. آدرس این دستور در حافظه برابر 2E450 = 0100 + 2E350 است. به این آدرس، آدرس حقیقی یا فیزیکی و به آدرس IP آدرس منطقی گفته می شود.

---
[1] Offset
[2] Instruction Pointer

ثبات اشاره گر مبنا (BP[1]): این ثبات حاوی فاصله‌ای در سگمنت پشته است و قابلیت دسترسی به محتویات پشته را برای برنامه نویس فراهم می آورد.

ثبات اشاره گر پشته (SP[2]): این ثبات حاوی فاصله ذخیره سازی داده‌ها در سگمنت پشته است و توسط پردازنده برای ذخیره و بازیابی اطلاعات در پشته مورد استفاده قرار می گیرد.

ثباتهای اندیس

ثبات اندیس مبدأ (SI[3]): این ثبات در عملیات رشته ای برای نگهداری اندیس رشته منبع به کار می رود.

ثبات اندیس مقصد (DI[4]): این ثبات برای نگهداری اندیس رشته مقصد در عملیات رشته ای به کار می‌رود.

آدرسهای منطقی و فیزیکی: آدرس داخل یک سگمنت می تواند از صفر تا FFFF (۶۵۵۳۵) تغییر کند. به آدرس داخل یک سگمنت، آدرس منطقی گفته می شود. مثلاً آدرس منطقی IP = 2F0A در سگمنت کد که CS = 204E معادل آدرس حقیقی (فیزیکی) ۲۰ بیتی 204E0 + 2F0A = 233EA در حافظه است. برای به دست آوردن آدرس حقیقی ۲۰ بیتی، آدرس شروع سگمنت را چهار بیت به سمت چپ شیفت داده و با آدرس آفست (آدرس منطقی) جمع می کنیم.

همانطور که می بینید آدرس حقیقی یک آدرس ۲۰ بیتی است که توسط پردازنده برای مراجعه به یک خانه حافظه روی گذرگاه آدرس گذاشته می شود؛ اما آدرس منطقی یک آدرس ۱۶ بیتی است که تنها درون محدوده ۶۴ کیلوبایتی یک سگمنت معنا دارد.

مهمترین مزیت استفاده از حافظه سگمنت‌بندی شده این است که برنامه‌هایی که تنها به آدرس منطقی ارجاع دهند را می توان در هر جای حافظه ذخیره و اجرا کرد؛ چون تغییر مکان ذخیره برنامه فقط آدرسهای شروع سگمنت را تغییر می‌دهد که مستقل از آدرسهای منطقی است. بعنوان یک برنامه نویس به ندرت نیاز به دانستن آدرس فیزیکی یک مکان پیدا می کنید و معمولاً دانستن آدرس منطقی کافی است.

به این ترتیب مانند آنچه در سیستمهای عامل (مانند DOS و ویندوز) انجام می شود، می توان برای بهره برداری درست از فضای حافظه، انتخاب مکان بار کردن برنامه (مکان آغاز سگمنتها) را بدون نگرانی از درهم ریختن آدرسها به سیستم عامل سپرد. تنها شرط لازم، عدم مراجعه به آدرسهای فیزیکی و نیز عدم تغییر ثباتهای سگمنت در برنامه است.

ذخیره داده‌ها در حافظه: پردازنده ۸۰۸۸ از نظر معماری داخلی مانند ۸۰۸۶ یک پردازنده ۱۶ بیتی است؛ یعنی گذرگاه داده داخلی آن ۱۶ بیتی است. اما گذرگاه داده خارجی آن (پینهای تراشه ۸۰۸۸ که به تبادل داده اختصاص یافته اند) ۸ بیتی است. سؤالی که به ذهن می آید این است که دستورات تبادل اطلاعات ۱۶ بیتی با خارج، چگونه در ۸۰۸۸ اجرا می شوند ؟

---

[1] Base Pointer
[2] Stack Pointer
[3] Source Index
[4] Destination Index

چون گذرگاه داده خارجی ۸۰۸۸ ۸ بیتی است، برای تبادل داده‌های ۱۶ بیتی چاره ای نیست جز اینکه داده‌های فوق را دو بار (در هر بار ۸ بیت) مبادله کنیم.

در پردازنده ۸۰۸۶ به دلیل ۱۶ بیتی بودن گذرگاه داده خارجی، این مشکل پیش نمی آید و ۸۰۸۶ می تواند هربار ۱۶ بیت اطلاعات را با خارج مبادله کند. به همین دلیل اجرای یک عملیات مشابه در ۸۰۸۸ کندتر از ۸۰۸۶ صورت می گیرد.

مشکلی که گذرگاه داده خارجی ۱۶ بیتی ۸۰۸۶ به وجود می آورد این است که تراشه‌های حافظه اکثراً دارای خانه‌های ۸ بیتی هستند، نه ۱۶ بیتی! شاید تحمیل حافظه‌های ۸ بیتی به یک پردازنده ۱۶ بیتی یک ضعف محسوب شود، اما حسن آن این است که کد ماشین بعضی دستورات، دارای تعداد فرد بایت (مثلاً یک یا ۳ یا ۵ بایت) است و پردازنده می‌تواند با آنها نیز به نحو مناسب رفتار کند.

در ۸۰۸۶ برای حل این مشکل، $۲^{۲۰}$ بایت فضای حافظه در دو بانک زوج و فرد در نظر گرفته می‌شود که ظرفیت هرکدام $۲^{۱۰}$ بایت است.

به یک نکته دیگر نیز دقت کنید. هنگامی که اسمبلر، برنامه‌ای که حاوی عدد ۱۶ بیتی 2E34 است را به کد ماشین ترجمه می کند، در ترتیب کدهای ماشین ابتدا 34 و سپس 2E می آید و به‌همین ترتیب در حافظه ذخیره می‌شوند. پردازنده نیز هنگام واکشی یک عدد ۱۶ بیتی از حافظه به ترتیب عکس معمول عمل می‌کند؛ یعنی ابتدا ۸ بیت کم ارزش و سپس ۸ بیت پرارزش را دریافت می‌کند. چون پردازنده اعداد ۱۶ بیتی را به صورت معکوس واکشی می کند، در نهایت عدد 2E34 به صورت واقعی در می‌آید.

واحدهای حافظه: کوچکترین واحد اطلاعاتی در پردازنده‌ها، واحد بیت یعنی یک رقم دودویی (صفر یا یک) است. واحدهای دیگر Nibble (یک رشته ۴ بیتی) و بایت (یک رشته ۸ بیتی یا مجموع دو Nibble) هستند.

واحدهای اطلاعاتی دیگری نیز به صورت زیر تعریف می شوند:

کلمه (Word): یک قلم داده ۲ بایتی (۱۶ بیتی)

کلمه مضاعف (Double Word): یک قلم داده ۴ بایتی (۳۲ بیتی)

کلمه چهارگانه (Quad Word): یک قلم داده ۸ بایتی (۶۴ بیتی)

پاراگراف (Paragraph): یک قلم داده ۱۶ بایتی (۱۲۸ بیتی)

هر ۱۰۲۴ بایت یک کیلوبایت، هر ۱۰۲۴ کیلوبایت یک مگابایت و هر ۱۰۲۴ مگابایت یک گیگابایت نامیده می‌شود.

مثالهایی از برنامه‌نویسی به زبان اسمبلی ۸۰۸۶/۸۸

مثال) قطعه برنامه ای بنویسید که مرتباً مقدار پورت ورودی شماره 27 H را خوانده و منفی آن را در پورت شماره 28 H نشان دهد.

```
AGAIN:
    IN      AL,27H
    NEG     AL
    OUT     AL,28H
JMP   AGAIN
```

چون عمل خواسته شده باید مرتباً انجام شود، آخرین دستور برنامه، دستور بازگشت به ابتدای برنامه باید باشد.

برای این کار به نقطه ابتدایی برنامه نام AGAIN را نسبت می دهیم. نام انتخاب شده دلخواه است و می تواند شامل حروف و ارقام باشد (اولین کاراکتر حتماً باید حرف باشد). بعد از نام انتخاب شده علامت ":" آورده می شود. این نام را برچسب[1] می‌نامند.

دستور JMP AGAIN، سبب انجام پرش به نقطه ای از برنامه به نام AGAIN که نقطه ابتدایی برنامه است، می‌شود. بنابراین اجرای قطعه برنامه فوق بی‌نهایت بار (تا زمانی که پردازنده خاموش یا بازنشانده نشده یا وقفه‌ای برای آن اتفاق نیفتاده است) ادامه می‌یابد.

پرسش) الف) اگر در پایان برنامه ای دستورHERE: JMP HERE نوشته شود، با اجرای دستور فوق چه می شود؟

ب) آیا می توان از این دستور به جای HALT استفاده کرد؟

پرشهای شرطی و ساختارهای تصمیم

دستور پرش غیر شرطی JMP در هر نقطه از برنامه که نوشته شود، سبب انجام پرش به نقطه ای دیگر از برنامه می‌شود.

برای تصمیم گیری در برنامه های اسمبلی از دستورات پرش شرطی استفاده می شود. این دستورات که تنوع زیادی نیز دارند، پس از یک دستور مقایسه مورد استفاده قرار می گیرند و بر اساس نتیجه مقایسه، تصمیم می گیرند. یک دستور پرش شرطی همیشه باعث پرش نمی شود، بلکه ابتدا یک شرط را بررسی کرده و در صورت صحیح بودن شرط، پرش انجام می شود؛ در غیر این صورت دستور بعد از دستور پرش شرطی اجرا خواهد شد.

قالب کلی استفاده از دستورات پرش شرطی به صورت زیر است:

```
CMP          عملوند ۱, عملوند ۲
دستور پرش شرطی     مقصد پرش
                    (نام نقطه‌ای از برنامه)
```

```
مجموعه دستورات ۱

JMP          Next
مقصد پرش:

مجموعه دستورات ۲

Next: ----
```

دستور CMP[2] دستور مقایسه دو عملوند است و بر اساس این مقایسه در دستور پرش شرطی یک شرط بررسی می شود. در صورت صحت شرط پرش به "مقصد پرش" انجام شده و مجموعه دستورات ۲ اجرا می شود. در صورت صحیح نبودن شرط پرش انجام نشده و مجموعه دستورات ۱ اجرا شده و سپس با پرش به نقطه Next برنامه ادامه می یابد.

---

[1] Label
[2] Compare

توجه کنید که دستور JMP Next ضروری است. اگر این دستور نباشد، بعد از مجموعه دستورات ۱، مجموعه دستورات ۲ نیز اجرا می شود! در حالی که می خواهیم در صورت صحیح نبودن شرط، فقط مجموعه دستورات ۱ اجرا شود. بنابراین بعد از اجرای این دستورات به نقطه Next که ادامه برنامه است پرش انجام شده است. البته نقطه Next می تواند جایی قبل از دستور CMP نیز باشد.

پرسش) نشان دهید چنانچه بخواهیم در صورت صحیح نبودن شرط فقط مجموعه دستورات ۲ و در صورت صحیح بودن شرط مجموعه دستورات ۱ و ۲ اجرا شوند، نیازی به دستور JMP Next نیست.

همانطور که گفته شد، دستورات پرش شرطی تنوع زیادی دارند. برای آشنایی با کاربرد آنها چند مثال می آوریم.

مثال) قطعه برنامه ای بنویسید که مرتباً مقدار پورت 32H را بخواند و تا زمانی که مقدار آن برابر صفر نشده، عدد 20 H را در پورت 35H بنویسد و با صفر شدن آن، عدد FFH را در پورت نوشته و به اجرای برنامه پایان دهد.

```
AGAIN:
    IN      AL,32H
    CMP     AL,0
    JZ    EXIT_PRGM
    MOV   AL,20H
    OUT   AL,35H
JMP    AGAIN

EXIT_PRGM:
MOV   AL,0FFH
OUT   AL,35H
HALT
```

دستور CMP AL,0، محتویات ثبات AL (مقدار پورت ورودی 32H) را با صفر مقایسه می کند. این دستور تعدادی از بیتهای ثبات پرچم را تغییر می دهد که دستورات پرش شرطی براساس این بیتها تصمیم می گیرند.

دستور JZ EXIT_PRGM[1] در صورت صفر بودن نتیجه مقایسه، به جایی از برنامه به نام EXIT_PRGM پرش انجام می دهد. به جای دستور JZ، می توان از دستور JE[2] استفاده کرد. هردو دستور بر اساس بیت Z ثبات پرچم تصمیم می گیرند و در صورت یک بودن این بیت، پرش را انجام می دهند.

اگر AL صفر باشد، دستور JZ اجرا شده و پرش به EXIT_PRGM انجام می‌شود. اگر AL صفر نباشد، دستور بعد از JZ اجرا می شود که شامل ارسال 20H به پورت 35H و بازگشت به ابتدای برنامه برای خواندن مجدد پورت 32H است.

پرسش) این برنامه را با استفاده از دستور JNZ (یا JNE) بنویسید.

مثال) قطعه برنامه‌ای بنویسید که مرتباً پورتهای ورودی 40H و 41H را بخواند و اگر مقدار پورت 40H از پورت 41H بیشتر یا با آن مساوی بود، عدد 55H و در غیر این صورت عدد AA H را به پورت خروجی 30H ارسال کند.

```
AGAIN:
    IN      AL,40H
    MOV   BL,AL
```

[1] Jump if Zero
[2] Jump if Equal

```
        IN      A,41H
        CMP     AL,BL
        JAE     SEND_AAH
        MOV     AL,55H
        OUT     AL,30H
JMP             AGAIN
SEND_AAH:
        MOV     AL,0AAH
        OUT     AL,30H
JMP             AGAIN
```

دستور CMP AL,BL محتویات ثباتهای AL و BL را با هم مقایسه می کند.

دستور JAE SEND_AAH[1] در صورتی که عملوند اول دستور CMP (AL) بزرگتر یا مساوی عملوند دوم (BL) باشد، به مکانی به نام SEND_AAH پرش انجام می دهد که دستورات آن نقطه شامل ارسال AAH به پورت 30H و بازگشت به ابتدای برنامه است.

در صورتی که AL < BL، دستور JAE باعث پرش نمی شود و دستورات بعد از آن که شامل ارسال 55H به پورت 30H و بازگشت به ابتدای برنامه است، انجام می شود.

به جای دستور JAE، می توان از دستور JNB[2] نیز استفاده کرد.

ساختارهای تصمیم گیری: در این بخش به طور مختصر ساختارهای تصمیم گیری مورد استفاده در زبانهای برنامه سازی سطح بالا را در زبان اسمبلی شبیه سازی می کنیم.

ساختار if-then-else: در این ساختار ابتدا شرطی بررسی می شود و در صورت درست بودن شرط، مجموعه دستورات زیر if و در غیر این صورت مجموعه دستورات زیر else اجرا می شود:

```
if    شرط     then
```

مجموعه دستورات ۱

```
else
```

مجموعه دستورات ۲

در صورت صحیح بودن شرط، مجموعه دستورات ۱ و در غیر این صورت مجموعه دستورات ۲ اجرا می شود.

مثلاً دستورات زیر را در نظر بگیرید:

```
if    sum > 100      then
```

مجموعه دستورات ۱

```
else
```

مجموعه دستورات ۲

معادل این دستورات در زبان اسمبلی به شرح زیر است:

[1] Jump if Above or Equal
[2] Jump if Not Below

```
CMP             sum,100
JA        P1
```

مجموعه دستورات ۲

```
JMP             end_if

P1:
```

مجموعه دستورات ۱

```
end_if: ------
```

پیاده‌سازی دیگر به صورت زیر است:

```
CMP             sum,100
JBE             P1
JMP             end_if

P1:
```

مجموعه دستورات ۲

```
end_if: -------
```

ساختار switch (case): در این ساختار مقدار یک متغیر با مقادیر ثابت مختلف مقایسه شده و در صورت تساوی با هرکدام عمل خاصی انجام می شود.

مثالی از ساختار switch در زبان C را در زیر می بینید:

```
switch (average) {
case  10:
```

مجموعه دستورات ۱

```
case  15:
```

مجموعه دستورات ۲

```
default:
```

مجموعه دستورات ۳

```
}
```

معادل اسمبلی ساختار بالا را در زیر می بینید:

```
CMP             average,10
JE        L1
CMP             average,15
JE        L2
JMP             L3
L1:
```

مجموعه دستورات ۱

```
JMP             Continue
L2:
```

مجموعه دستورات ۲

JMP Continue
L3:

مجموعه دستورات ۳

Continue: ----

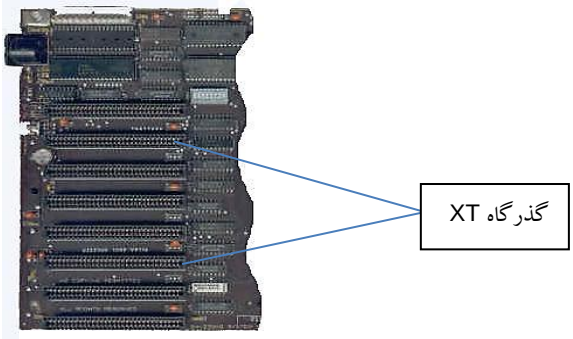
فصل چهارم
گذرگاههای کامپیوتری

هنگام خرید کامپیوتر باید راجع به امکانات آن تصمیم بگیرید. بسیاری از امکانات مهم کامپیوترها، در قالب کارتهایی عرضه می‌شود که داخل شکافهایی[۱] که روی بورد اصلی تعبیه شده‌اند قرار می‌گیرد. کارت گرافیکی، کارت صوتی، کارت شبکه، کارت مودم و ... رابطهای کامپیوتر با دنیای خارج هستند. بعضی از این کارتها ورودی (مانند کارت TV[۲])، بعضی خروجی (مانند کارت گرافیکی) و بعضی ورودی/خروجی (مانند کارت صوتی[۳]) هستند.

کارتهایی که در اولین کامپیوتر های IBM مورد استفاده قرار گرفتند، از خانواده کارتهای XT[۴] بودند. نسلهای بعدی کارتهای کامپیوتری عبارتند از ISA[۵]، EISA[۶]، MCA[۷] و PCI[۸]. کارتهای گرافیکی به دلیل نیاز به پهنای باند و سرعت بالا راه خود را در پیش گرفتند؛ کارتهای AGP[۹] و PCI-EXPRESS در زمره کارتهای گرافیکی پرسرعت هستند. لپ‌تاپها نیز دارای کارتهایی ویژه (مانند کارتهای خانواده PCMCIA[۱۰]) هستند. چون هر کارتی داخل یک شکاف روی بورد اصلی قرار می‌گیرد، هرکدام از این نامها هم به کارت و هم به شکاف متناظر با آن اطلاق می‌شود. مثلاً کارت مودم PCI در شکاف PCI بورد اصلی کامپیوتر قرار می‌گیرد. در این فصل ما از نام کلی گذرگاه برای اشاره به هر دو استفاده می‌کنیم. بنابراین در یک دید کلی می‌توان گفت:

> گذرگاه، واسطه ارتباط کامپیوتر با دنیای خارج است.

در این بخش اصول طراحی کارتهای کامپیوتری XT و ISA را مطالعه خواهیم کرد که بسیار شبیه به آن چیزی است که در مورد اتصال ورودی/خروجی به پردازنده‌ها آموختیم. هرچند این دو گذرگاه در کامپیوترهای شخصی امروزی کمیابند؛ اما آموختن معماری پایه کارتهای کامپیوتری به کمک آنها بسیار ساده است. در انتهای این بخش، مروری بر گذرگاههای دیگر و مقایسه‌ای بین ویژگیهای آنها خواهیم داشت.



شکل ۴-۱- گذرگاه **XT** روی مادربورد

گذرگاه **XT:** در سال ۱۹۸۱ در اولین کامپیوترهای شخصی IBM برای اتصال دستگاههای ورودی/خروجی به کامپیوتر ایجاد شد. شکل ۴-۱ گذرگاه XT روی بورد اصلی کامپیوتر را نشان می‌دهد.

تمام کارتهای کامپیوترهای اولیه از نوع XT بودند و در

---

[1] Slot

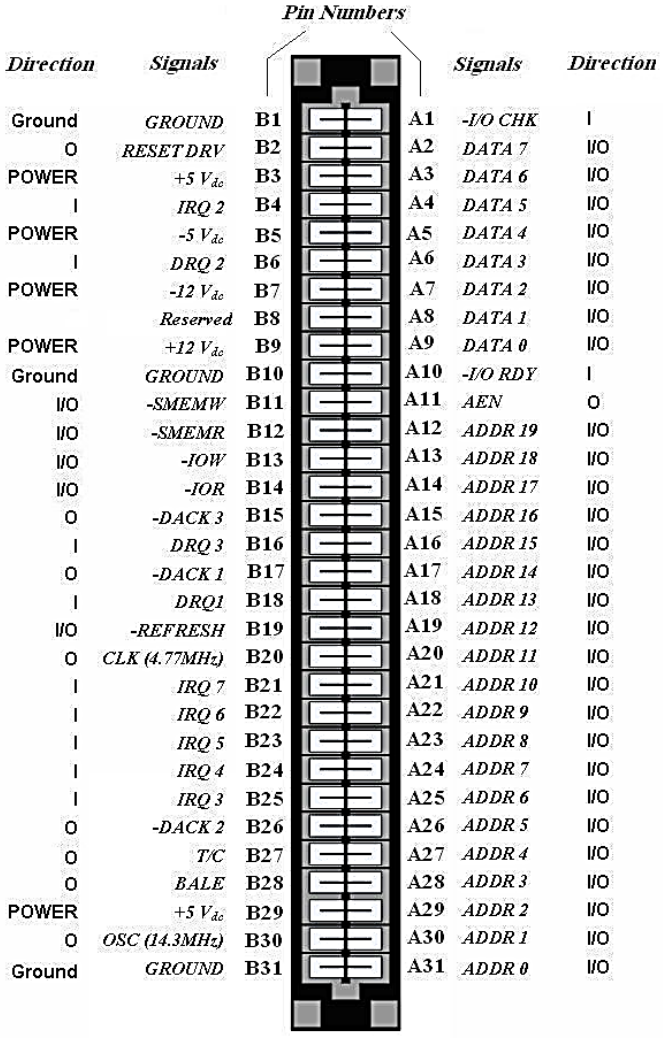
[۲] کارت TV، سیگنال تلویزیون را از آنتن دریافت کرده و روی کامپیوتر نمایش می‌دهد.

[۳] درگاه بلندگو (Speaker) خروجی کارت صوتی و درگاه Microphone ورودی کارت صوتی است.

[4] Extended Slot
[5] Industrial Standard Architecture
[6] Extended ISA
[7] Micro-Channel Architecture
[8] Prepheral Component Interconnect
[9] Accelerating Graphic Port
[10] Personal Computer Memory Card International Association

این شکافها قرار می‌گرفتند. هم‌اکنون نیز نمونه‌های زیادی از این کارتها را در ماشینهای صنعتی می‌توانید پیدا کنید. از کارت XT عموماً برای اتصال ورودی/خروجی به کامپیوتر استفاده می‌شود (که البته می‌تواند حاوی حافظه هم باشد).

به شکل ۳-۳ که پینهای پردازنده ۸۰۸۸ را نشان می‌دهد دقت کنید. انتظار داریم گذرگاه XT، کدام پایه‌های این پردازنده را در اختیار ما قرار داده باشد؟ حضور پایه‌های داده، پایه‌های آدرس، سیگنالهای $IO/\overline{M}$ و $\overline{RD}$ و $\overline{WR}$ الزامی است. اکنون به پینهای گذرگاه XT در شکل ۲-۴ دقت کنید.



شکل ۴-۲- پایه‌های گذرگاه XT

گذرگاه XT دارای ۶۲ پین است که با نامهای A_1~A_{31} و B_1~B_{31} مشخص می‌شوند. در شکل بالا روبروی هر کدام از پینها، جهت سیگنال نسبت به شکاف کامپیوتر نوشته شده است (I برای ورودی به شکاف، O برای خروجی از شکاف و I/O برای ورودی/خروجی). علامت منفی قبل از نام سیگنال، نشانه فعال پایین بودن آن است.

گذرگاههای آدرس و داده: پینهای A_2 تا A_9، هشت پایه داده پردازنده (DATA 0 ~ DATA 7) و پینهای A_{12} تا A_{31}، بیست پایه آدرس پردازنده (ADDR 0 ~ ADDR 19) را در اختیار ما قرار می‌دهند. دقت کنید برخلاف

پایه‌های اصلی پردازنده ۸۰۸۸ که پایه‌های داده و آدرس بر هم نهاده[1] شده بودند، در گذرگاه XT این پایه‌ها از هم جدا شده‌اند و زحمت نگهداری موقت اطلاعات آدرس (Latch) از دوش طراح کارت برداشته شده است.

پایه‌های داده که طبیعتاً باید دوطرفه باشند؛ اما پایه‌های آدرس که اصولاً باید خروجی از پردازنده باشند، چرا در گذرگاه XT دوطرفه هستند؟ اگر کارت XT بتواند گذرگاه کامپیوتر را در اختیار بگیرد (به اصطلاح حاکم گذرگاه[2] شود)، آدرس‌دهی نیز در اختیار اوست و باید به این منظور از همین پینهای آدرس به صورت ورودی استفاده کند.

سیگنالهای کنترل: پایه‌های B_{11} تا B_{14} هم برایمان آشنا هستند. $\overline{SMEMW}$[3] هنگام نوشتن در حافظه توسط پردازنده فعال می‌شود. $\overline{SMEMR}$[4] نیز ویژه اعلام خواندن از حافظه است. پایه‌های $\overline{IOW}$ و $\overline{IOR}$ نیز به ترتیب برای اعلام نوشتن در خروجی و خواندن از ورودی توسط پردازنده مورد استفاده قرار می‌گیرند. هر چند انتظار داریم این سیگنالها خروجی باشند، اما به لحاظ اینکه ممکن است گاهی کارت، حاکم گذرگاه شود و از این سیگنالها استفاده کند، دوطرفه طراحی شده‌اند.

با این مقدمه، طراحی کارت ورودی و خروجی XT ساده به نظر می‌رسد؛ اما پیش از بررسی مثالهای عملی، بهتر است با کارکرد پایه‌های دیگر گذرگاه XT نیز آشنا شویم.

پینهای تغذیه: پایه‌های B_3 و B_{29}، ولتاژ ۵+ ولت با حداکثر جریان ۱۱ آمپر برای تغذیه تراشه‌های TTL روی کارت را در اختیار طراح می‌گذارند. به علاوه ولتاژهای ۵- ولت (پایه B_5 با حداقل جریان ۰/۳ آمپر)، ۱۲+ ولت (پایه B_9 با حداکثر جریان ۴/۵ آمپر) و ۱۲- ولت (پایه B_7 با حداقل جریان ۰/۲۵ آمپر) و نیز سه پایه زمین (B_1 و B_{10} و B_{31}) در گذرگاه XT وجود دارند. خواهیم دید این تعداد کم پایه زمین یکی از موانع افزایش سرعت گذرگاه است.

پینهای درخواست وقفه: پایه‌های ورودی B_4 و B_{21}~B_{25} برای درخواست وقفه از کامپیوتر مورد استفاده قرار می‌گیرند و به پایه‌های IRQ 2~IRQ 7 تراشه کنترل‌کننده وقفه کامپیوتر متصل می‌شوند. اگر طراح کارت بخواهد از وقفه‌های سخت‌افزاری (خارجی) کامپیوتر سود ببرد، باید از این پینها استفاده کند.

پینهای DMA: گاهی طراح کارت تمایل دارد برای تبادل اطلاعات کارت خود با حافظه از روش DMA استفاده کند. سه پایه ورودی گذرگاه XT با نامهای DRQ 1 و DRQ 2 و DRQ 3 برای درخواست DMA[5] توسط کاربر و سه پایه خروجی با نامهای DACK 1 و DACK 2 و DACK 3 برای پاسخ به DMA[6] توسط پردازنده مورد استفاده قرار می‌گیرند.

پایه خروجی RESET DRV: این پایه از سیگنال Reset تراشه کنترل‌کننده گذرگاه پردازنده می‌آید و برای Reset همزمان تراشه‌های کارت با پردازنده به کار می‌رود.

---

[1] Multiplexed
[2] Bus Master
[3] System MEMory Write
[4] System MEMory Read
[5] Dma ReQuest
[6] Dma ACKnowledge

پایه $\overline{\text{REFRESH}}$: اگر این پایه در نقش خروجی ظاهر شود، فعال بودن آن نشان‌دهنده پیشروی سیکل تازه‌سازی[1] حافظه‌های RAM پویا[2] در سیستم است. این سیگنال می‌تواند ورودی نیز باشد و توسط کارت برای تعیین سیکل تازه‌سازی مورد استفاده قرار گیرد.

پایه خروجی CLK (4.77MHz): یک پالس مربعی با فرکانسی همزمان با پالس ساعت داخلی کامپیوتر را برای استفاده تراشه‌های کارت در اختیار طراح می‌گذارد. این فرکانس می‌تواند نوعاً از ۴/۷۷ مگاهرتز تا ۸ مگاهرتز باشد[3]. در گذرگاههای بعدی این فرکانس بالاتر رفته و به عنوان مثال در اولین نسخه گذرگاه PCI به ۳۳ مگاهرتز رسیده است.

پایه خروجی OSC (14.3MHz): یک پالس مربعی با فرکانس ۱۴/۳ مگاهرتز غیرهمزمان با پالس ساعت کامپیوتر در اختیار طراح می‌گذارد که سه برابر فرکانس کاری سیستم (۴/۷۷ مگاهرتز) است. در کارتهای گرافیکی اولیه از این فرکانس برای تازه‌سازی نقاط[4] تصویر استفاده می‌شد.

پایه خروجی T/C (Terminal Count): برای اعلام پایان شمارش تعداد بایتهای انتقالی در عملیات DMA توسط کامپیوتر مورد استفاده قرار می‌گیرد.

پایه خروجی BALE (Buffered ALE): سیگنال بافر شده ALE پردازنده است که فعال بودن آن نشانه معتبر بودن آدرس روی پایه‌های A_{12}~A_{31} گذرگاه XT است. این سیگنال در طول عملیات DMA برابر «یک» خواهد بود.

پایه ورودی $\overline{\text{I/OCHK}}$ (I/O CHecK): طراح کارت می‌تواند از این پین برای اعلام خطای غیرقابل جبران به صورت وقفه غیرقابل چشم‌پوشی[5] به کامپیوتر استفاده کند. اگر کارت حافظه به گذرگاه XT متصل شود از این پایه برای اعلام خطای توازن استفاده می‌شود.

پایه ورودی $\overline{\text{I/ORDY}}$ (I/O ReaDY): سرعت عملکرد کارت XT معمولاً کمتر از پردازنده است. طراح کارت می‌تواند از این سیگنال برای ایجاد حالت انتظار[6] در پردازنده و ایجاد فرصت برای انجام عملیات کارت استفاده کند. برای ایجاد صحیح حالت انتظار، پس از اینکه کارت از طریق مدار رمزگشای آدرسش متوجه شد مخاطب پردازنده قرار گرفته است، باید بلافاصله این سیگنال را «صفر» (Not Ready) کند. اگر کارت این سیگنال را به مدت طولانی (معمولاً بیش از ۱۵ میکروثانیه) در سطح «صفر» نگه دارد، ممکن است باعث جلوگیری از تازه‌سازی RAM پویا و به هم ریختن محتویات آن شود.

پایه خروجی AEN (Address Enable): این سیگنال برای مشخص کردن اینکه کنترل گذرگاه در اختیار پردازنده است یا حاکم گذرگاه دیگر (مثلاً کنترل‌کننده DMA) مورد استفاده قرار می‌گیرد. اگر مقدار این سیگنال «صفر» باشد یعنی کنترل گذرگاه در اختیار پردازنده است. وجود «یک» روی این پایه نشان‌دهنده آن است که حاکم

---

[1] Refresh
[2] Dynamic

[3] در بعضی کامپیوترها، این فرکانس در بایوس سیستم قابل انتخاب است.

[4] Pixels
[5] Non Maskable Interrupt (NMI)
[6] Wait State

گذرگاه دیگری وارد عمل شده و گذرگاههای داده و آدرس و سیگنالهای $\overline{IOW}$ و $\overline{IOR}$ و $\overline{SMEM۱}$ و $\overline{SMEMR}$ در اختیار آن است. به بیان دیگر اگر این سیگنال «یک» باشد، نشان‌دهنده این است که آدرس روی گذرگاه آدرس سیستم متعلق به عملیات DMA است و از نظر عملیات ورودی/خروجی بی‌اعتبار می‌باشد و کارت نباید به این آدرس واکنش نشان دهد. از این سیگنال در طراحی مدار رمزگشای آدرس کارتهای ورودی/خروجی استفاده می‌شود. چون زمانی می‌توان از سیگنالهای گذرگاه در رمزگشایی آدرس کارت استفاده کرد که گذرگاه در اختیار پردازنده باشد.

از بین پایه‌های تشریح شده، تعدادی برای طراحی کارتهای ورودی/خروجی ضروری هستند و تصمیم به استفاده از بقیه با طراح کارت است.

سیگنالهای ضروری عبارتند از: پینهای داده، پینهای آدرس، پینهای ۵+ ولت و زمین و AEN و $\overline{IOW}$ و $\overline{IOR}$.

سیگنالهای دیگری مانند $\overline{I/ORDY}$ و CLK نیز در بسیاری از کارتها مورد استفاده قرار می‌گیرند.

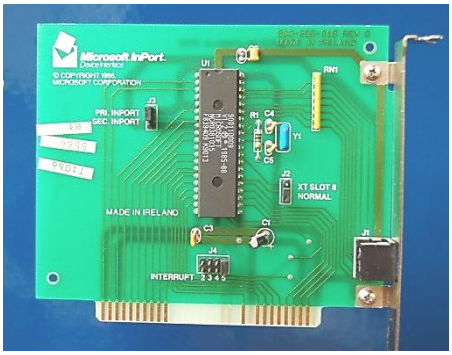
به یک نکته توجه کنید. همانطور که قبلاً گفته شد، با وجود زیاد شدن تعداد خطوط آدرس پردازنده‌ها در نسلهای متوالی پردازنده‌های اینتل، تعداد خطوط آدرس برای آدرس‌دهی ورودی/خروجی ثابت مانده است؛ یعنی در تمامی نسلهای پردازنده اینتل از ۸۰۸۵ به بعد، تعداد خطوط آدرس ورودی/خروجی ثابت و برابر ۱۶ خط هست و به همین لحاظ آدرسهای ورودی/خروجی (حتی در کامپیوترهای امروزی) به صورت ۱۶ بیتی بیان می‌شوند. شکل ۴-۳ آدرس‌های ورودی/خروجی کامپیوترهای IBM اولیه از نسخه AT (که از پردازنده ۸۰۲۸۶ استفاده می‌کرد) به بعد را نشان می‌دهد. به جهت رعایت تطابق با قبل، این آدرسها همچنان در کامپیوترهای امروزی معتبرند.

Port (hex)	Port Assignments
000-00F	DMA Controller
010-01F	DMA Controller (PS/2)
020-02F	Master Programmable Interrupt Controller (PIC)
030-03F	Slave PIC
040-05F	Programmable Interval Timer (PIT)
060-06F	Keyboard Controller
070-071	Real Time Clock
080-083	DMA Page Register
090-097	Programmable Option Select (PS/2)
0A0-0AF	PIC #2
0C0-0CF	DMAC #2
0E0-0EF	reserved
0F0-0FF	Math coprocessor, PCJr Disk Controller
100-10F	Programmable Option Select (PS/2)
110-16F	AVAILABLE
170-17F	Hard Drive 1 (AT)
180-1EF	AVAILABLE
1F0-1FF	Hard Drive 0 (AT)
200-20F	Game Adapter
210-217	Expansion Card Ports
220-22F	Soundblaster cards
278-27F	Parallel Port 3
280-2A1	AVAILABLE

Port (hex)	Port Assignments
2A2-2A3	clock
2B0-2DF	EGA/Video
2E2-2E3	Data Acquisition Adapter (AT)
2E8-2EF	Serial Port COM4
2F0-2F7	Reserved
2F8-2FF	Serial Port COM2
300-31F	Prototype Adapter, Periscope Hardware Debugger،Data acquisition cards
320-32F	AVAILABLE
330-33F	Reserved for XT/370
340-35F	AVAILABLE
360-36F	Network
370-377	Floppy Disk Controller
378-37F	Parallel Port 2
380-38F	SDLC Adapter
390-39F	Cluster Adapter
3A0-3AF	reserved
3B0-3BF	Monochrome Adapter
3BC-3BF	Parallel Port 1
3C0-3CF	EGA/VGA
3D0-3DF	Color Graphics Adapter
3E0-3EF	Serial Port COM3
3F0-3F7	Floppy Disk Controller
3F8-3FF	Serial Port COM1

شکل ۴-۳- آدرس‌های ورودی/خروجی کامپیوترهای **IBM-PC-AT**

آدرس کارتهای XT (و ISA) که توسط کاربر طراحی می‌شوند باید در بازه 0300 H تا 031F H باشند.

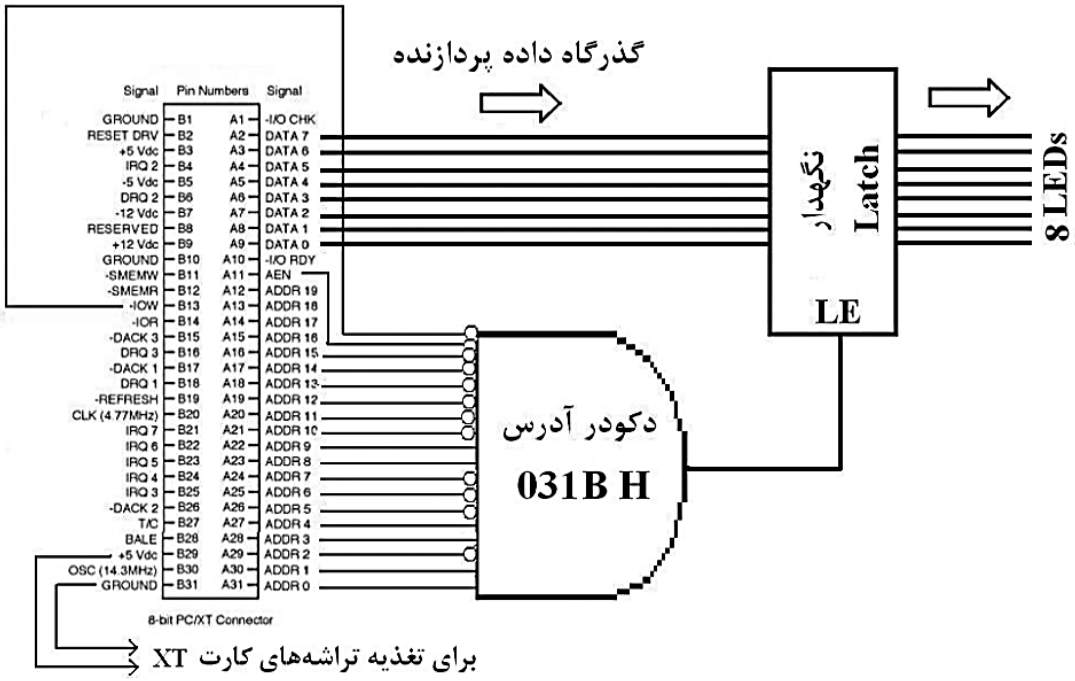


دو مثال ساده طراحی کارت ورودی و کارت خروجی را در ادامه بررسی می‌کنیم. برای نرم‌افزار کامپیوتر از زبان C استفاده کرده‌ایم که کاربر می‌تواند به دلخواه خود آن را تغییر دهد. شکل مقابل نمونه‌ای از کارتهای XT (کارت ماوس که در کامپیوترهای اولیه که فاقد پورت ویژه ماوس بودند استفاده می‌شد) را نشان می‌دهد.

طراحی کارت XT خروجی

می‌خواهیم سیستمی طراحی کنیم که یک رقص نور روی هشت LED به صورت زوج و فرد ایجاد کند؛ یعنی ابتدا LED های زوج روشن و LED های فرد خاموش و مدتی بعد عکس این عمل انجام شود و این کار مرتباً ادامه یابد.

برای این کار یک کارت XT طراحی می‌کنیم که روی آن هشت LED باشد و با اجرای یک برنامه در کامپیوتر، این رقص نور اجرا شود. بنا بر آنچه در فصل قبل گفته شد، نمی‌توان LED ها را مستقیماً به گذرگاه داده شکاف XT متصل کرد و باید از تراشه نگهدار استفاده کنیم که در موقع مقتضی، LED ها را به گذرگاه داده کامپیوتر متصل کند. شکل زیر نحوه طراحی این کارت را نشان می‌دهد. آدرس 031B H برای این کارت انتخاب شده است.



شکل ۴-۴- نمونه‌ای از کارت XT خروجی

به نکات زیر توجه کنید:

- ✓ چون برای تبادل اطلاعات با این کارت، پردازنده (و نه واحد دیگری) آدرس را روی گذرگاه آدرس قرار می‌دهد، سیگنال AEN به نشانه این موضوع صفر می‌شود. می‌بینید که یکی از شرطهای فعال شدن تراشه نگهدار، صفر بودن AEN است.
- ✓ چون آدرسهای ورودی/خروجی در کامپیوتر ۱۶ بیتی هستند، از خطوط آدرس بالاتر از A_{15} در رمزگشایی استفاده نشده است.

- ✓ چون این کارت برای اتصال خروجی به پردازنده طراحی شده و در این حال پردازنده سیگنال $\overline{IOW}$ را فعال می‌کند، این سیگنال با واسطه یک گیت NOT به مدار رمزگشای آدرس متصل شده است.
- ✓ به جای گیت AND با تعداد ورودی زیاد (مانند تراشه 7430 که AND با ۸ ورودی است) می‌توان از تراشه دکودر (مثلاً 74LS138) برای طراحی مدار رمزگشای آدرس استفاده کرد.

بنابراین هنگامی LED ها به گذرگاه داده متصل می‌شوند که پردازنده آدرس 031B H را روی گذرگاه آدرس خود قرار داده و سیگنالهای AEN و $\overline{IOW}$ را نیز فعال کرده باشد.

```
// This program writes the 8-bit data to  the
// address 031BH every 500 milli-second. Press
// any key to terminate.
#include <STDIO.H>
#include <CONIO.H>
#include <DOS.H>
void main(void){
        unsigned char data = 0x55;
        while ( !kbhit() ){
                asm {
                        push ax
                        push dx

                        mov  dx,31Bh
                        mov  al,data
                        out  dx,al

                        pop  dx
                        pop  ax
                }

                data = ~data;
                delay(500);
        }
}
```

outportb (0x031B , data) ;

در برنامه کامپیوتری، به ترتیب اعداد 55 H (01010101) و AA H (10101010) هر ۵۰۰ میلی‌ثانیه یک بار به آدرس 031B h ارسال می‌شوند تا رقص نور مورد نظر روی LED ها انجام شود. برنامه صفحه بعد این کار را انجام می‌دهد.

در این برنامه از متغیر یک بایتی data برای ارسال اطلاعات به خروجی استفاده شده است. مقدار اولیه این متغیر برابر 55 H است و در هر دور اجرای برنامه به کمک عملگر "~" معکوس می‌شود تا رقص نور مورد نظر را روی LED ها ایجاد کند.

در قسمتی از برنامه C که به زبان اسمبلی نوشته شده، عمل ارسال متغیر data به خروجی انجام می‌شود. دستور *out dx,al* مقدار یک‌بایتی AL را به پورت خروجی که شماره آن در DX قرار داده شده ارسال می‌کند. چون می‌خواهیم مقدار متغیر data را به پورت شماره 031B H ارسال کنیم، data را در AL و شماره پورت را در DX کپی و دستور out را اجرا می‌کنیم. استفاده از push و pop قبل و بعد از دستور out برای جلوگیری از تغییر محتویات ثباتهای AL و DX در اثر انجام این عملیات است.

به جای قطعه برنامه اسمبلی گفته شده، می‌توان از دستور outportb یا outp در زبان C استفاده کرد. دستور outportb(PortID,value); یا outp(PortID,value); عدد یک بایتی value را به پورت شماره PortID ارسال می‌کند[1].

دستور delay(500)، پانصد میلی‌ثانیه تأخیر ایجاد می‌کند. برنامه با فشار هر کلیدی پایان می‌یابد.

نکات زیر در مورد این برنامه قابل توجه هستند:

---

[1] دستورات outport و outpw در زبان C برای ارسال اعداد دوبایتی به پورتها استفاده می‌شوند.

- ✓ در سیستمهایی با سرعت بالا، استفاده از دو دستور ورودی/خروجی به صورت متوالی به درستی کار نمی‌کند و باید حتماً بین آنها از دستورات ایجاد تأخیر استفاده کرد. دلیل این موضوع بالا بودن سرعت پردازنده در اجرای دستورات نسبت به ارتباط با درگاه‌های ورودی/خروجی است.
- ✓ به جای استفاده از دستورات ورودی/خروجی که ویژه روش «ورودی/خروجی با نگاشت ورودی/خروجی» است، می‌توان از روش نگاشت حافظه و دستورات MOV استفاده کرد. عیب این روش در کامپیوترهای IBM این است که برای دسترسی به حافظه در این کامپیوترها از روش سگمنت/آفست استفاده می‌شود و طراح کارت در نوشتن برنامه باید به این نکته دقت کند. به علاوه در سیم‌بندی باید از تمام بیتهای گذرگاه آدرس پردازنده استفاده نمود.

چگونه باید این کارت را سیم‌بندی کرد؟ معمولاً در بازار بوردهای آماده‌ای وجود دارد که در شکاف XT قرار می‌گیرد و سیگنالهای این گذرگاه را در اختیار طراح قرار می‌دهد. به علاوه می‌توانید به کمک ابزارهای طراحی PCB خودتان این کار را انجام دهید. برای جلوگیری از سوختن بورد اصلی کامپیوتر، توصیه می‌شود تمام خطوط داده و آدرس و کنترل گذرگاه XT با استفاده از بافر به خارج متصل شوند.

برنامه این کارت به کمک C تحت DOS نوشته شده که اگر آن را در سیستم‌عامل DOS یا ویندوز ۹۵ یا ۹۸ یا Millenium (Me) اجرا کنید، به درستی با کارت ارتباط برقرار می‌کند. اما قصه ارتباط با پورتهای ورودی/خروجی در سیستم‌عامل‌های ویندوز NT به بعد (2000، XP، Vista و ...) متفاوت است. در این سیستم‌عامل‌ها، برنامه‌های کاربر در مد کاربردی[1] و توابع سیستمی (از جمله توابع دسترسی به پورتهای ورودی/خروجی) در مد هسته سیستم‌عامل[2] اجرا می‌شوند. این کار برای بالا بردن امنیت سیستم‌عامل و جلوگیری از حملات مخرب انجام می‌شود و آزادی عملی که کاربران DOS و ویندوز ۹۸ برای اجرای توابع سیستمی داشتند را از آنان می‌گیرد. بنابراین اگر برنامه گفته شده را در یکی از سیستم‌عاملهای ویندوز NT به بعد (حتی در محیط مجازی DOS آنها) اجرا کنید، برنامه به پورتها دسترسی نداشته و پاسخ آن صحیح نخواهد بود. حتی اگر برنامه خود را به زبانهای تحت ویندوز (مانند Delphi و VC++) بنویسید و بخواهید مستقیماً به پورتهای ورودی/خروجی دسترسی داشته باشید، در ویندوزهای NT به بعد ناکام خواهید ماند. برای تصحیح این مشکل، باید از برنامه‌های کتابخانه‌ای استفاده کنیم که به صورت واسط بین مد کاربردی و مد هسته عمل می‌کنند؛ به بیان دیگر درخواستهای کاربر برای دسترسی به پورتها را دریافت و به هسته سیستم‌عامل ارائه کرده و پاسخ را از هسته گرفته و به کاربر می‌دهند. کتابخانه پویای inpout32.dll یا برای همین هدف نوشته شده است. با الحاق آن به پروژه‌های خود در زبانهای تحت ویندوز (مانند Delphi و C#) می‌توانید به پورتهای کامپیوتر دسترسی داشته باشید. روش دیگر استفاده از برنامه کاربردی PortTalk است که می‌تواند پورتهای مورد نظر شما را در محیط DOS ویندوز باز کند. نحوه طراحی مدارهای واسط کاربردی به کمک برنامه‌های تحت ویندوز در پیوست کتاب تشریح شده است.

به جای نوشتن برنامه، می‌توان دستورات ارتباط با ورودی/خروجی را در محیط Debug نوشت و با زحمت کمتری اجرا کرد. البته به تذکر قبلی در مورد دسترسی به پورتها در محیط ویندوز توجه کنید.

---

[1] Application Mode
[2] Kernel Mode

ملاحظات زمانبندی سیستم

آیا این سیستم از نظر زمانبندی درست کار می‌کند؟

پیشتر دیدیم که در طراحی سیستم‌های مبتنی بر پردازنده باید مسایل زمانبندی را در نظر گرفت؛ به این معنا که طراح این سیستم باید مطمئن شود مدت زمانی که پردازنده به اجرای دستور خروجی اختصاص می‌دهد، برای انتقال داده‌ها از گذرگاه داده پردازنده به خروجی تراشه نگهدار کافی است. به بیان دیگر، هنگامی که پردازنده دستور خروجی را اجرا می‌کند، تا پایان زمان اجرای دستور باید عمل رمزگشایی آدرس، فعال شدن نگهدار و ظاهر شدن داده‌ها روی خروجی تراشه نگهدار انجام شود. با این توضیحات، آیا کارت بالا به درستی کار می‌کند؟

پاسخ مثبت است. دستورات در کامپیوتر PC/XT در چهار سیکل ساعت انجام می‌شوند که معمولاً عملیات انتقال داده در **سیکل سوم** صورت می‌گیرد. در کامپیوتر PC/XT، فرکانس کاری گذرگاه 5 MHz بود که در گذرگاه ISA (نسخه گسترش‌یافته گذرگاه XT که در ادامه آن را خواهیم دید) به 8 MHz رسید. با فرض اینکه فرکانس کاری گذرگاه در سریعترین مقدار ممکن (8 MHz) باشد، زمان یک سیکل $\frac{1}{8\mathbf{MHz}} = 125\mathbf{ns}$ است. تأخیر مدار رمزگشا و تراشه نگهدار معمولاً از ۷۰ نانوثانیه کمتر است. بنابراین انتقال داده‌ها در زمانی کمتر از یک سیکل ساعت انجام می‌شود و کارت فوق به درستی کار می‌کند.

توجه کنید این موضوع ارتباطی به پردازنده کامپیوتر ندارد؛ چون فرکانس کاری گذرگاه ISA به 8 MHz محدود است و هرچه پردازنده دارای فرکانس کاری بالایی نیز باشد، باز ناچار است با فرکانس 8 MHz با گذرگاه ISA به تبادل داده بپردازد.

سؤالی که ممکن است پیش آید این است که چگونه پردازنده‌ای با سرعت بالا (مثلاً پنتیومی که با فرکانسی بالاتر از 2 GHz کار می‌کند) با فرکانس 8 MHz با گذرگاه ISA تبادل داده می‌کند؟

در پاسخ باید به چند نکته توجه داشت:

- ✓ فرکانس کاری بورد اصلی کامپیوتر حدود 400 MHz است که در بوردهای جدید به حدود 800 MHz افزایش یافته است. در واقع فرکانس پردازنده می‌تواند برای اجرای دستورات داخلی آن بسیار بالا باشد، اما برای کار با تجهیزات بورد اصلی باید به مقادیر گفته شده کاهش یابد.
- ✓ هرچه مدل پردازنده بالاتر می‌رود، مدت زمان اجرای دستورات ورودی/خروجی در آن افزایش می‌یابد؛ مثلاً در پردازنده ۸۰۴۸۶ زمان اجرای این دستورات در مد حقیقی ۱۲ سیکل ساعت است[1]؛ حال آنکه اجرای دستورات دیگر تنها ۲ سیکل ساعت به طول می‌انجامد. همین موضوع به کاهش سرعت پردازنده در ارتباط با گذرگاههای ورودی/خروجی کمک می‌کند.
- ✓ در بوردهای اصلی، یک سیکل انتظار (Wait State) به صورت خودکار به دستورات ورودی/خروجی تزریق می‌شود که زمان اجرای آنها را افزایش می‌دهد[2].

---

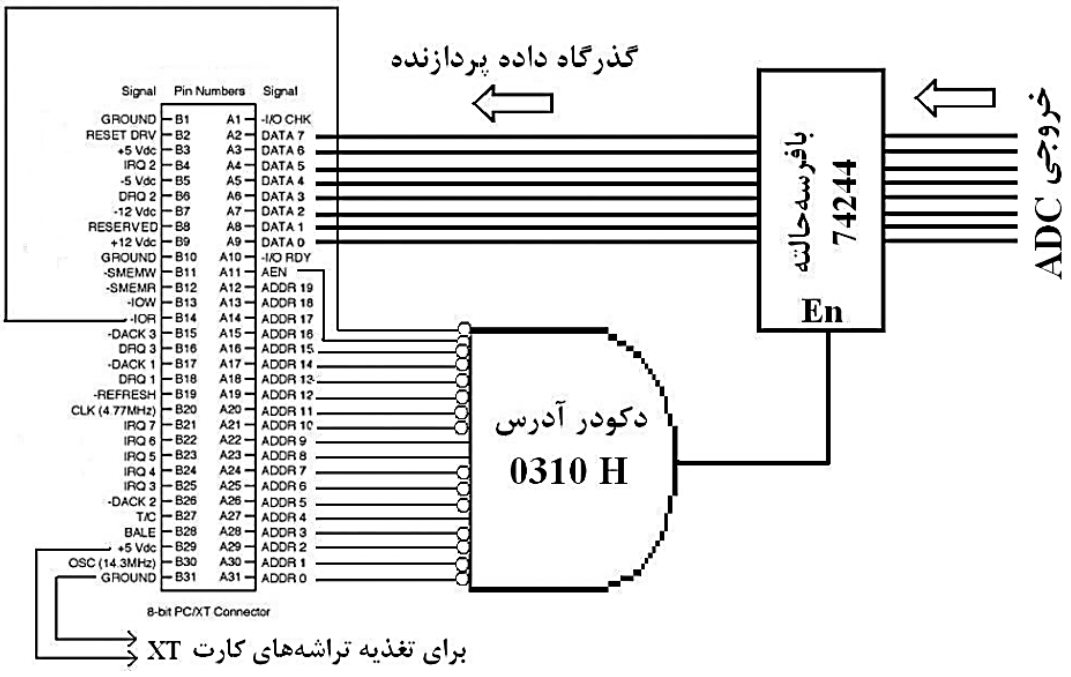
[1] این زمان در مد محافظت شده ۱۷ سیکل ساعت است.

[2] چنانچه مدار روی کارت سریع عمل کند، می‌توانید به کمک سیگنال NOWS که بعدها خواهیم دید، این سیکل انتظار اضافه را حذف کنید.

- ✓ با فرض تمام این موارد، چنانچه باز هم سرعت پردازنده زیاد بود، مدار کنترل‌کننده گذرگاه ISA در بورد اصلی به کمک سیگنال IO RDY زمان اجرای دستورات ورودی/خروجی مربوط به این گذرگاه را طولانی می‌کند.

طراحی کارت ورودی XT

فرض کنید می‌خواهیم یک کارت XT طراحی کنیم که به کمک آن هر یک ثانیه یک‌بار دمای اتاق را خوانده و روی نمایشگر کامپیوتر نشان دهیم. بدیهی است چون دمای اتاق (که با حسگر[1] دما اندازه گرفته می‌شود) یک کمیت آنالوگ است، برای وارد کردن آن به کامپیوتر (یا هر سیستم دیجیتال) باید از تراشه مولد آنالوگ به دیجیتال[2] (مانند مبدل ۸ بیتی ADC804) استفاده کنیم. این تراشه، یک سیگنال آنالوگ را به عنوان وردی دریافت کرده و مقدار آن را به صورت یک عدد ۸ بیتی نشان می‌دهد. کارت ما باید هر یک ثانیه یک بار مقدار این عدد ۸ بیتی را به عنوان ورودی خوانده و روی نمایشگر کامپیوتر نمایش دهد. بنا برآنچه در فصل قبل آموختیم، نمی‌توان ورودی را مستقیماً به گذرگاه داده کامپیوتر متصل کرد و باید از بافر سه‌حالته برای این اتصال استفاده کنیم که در موقع مقتضی،کلیدها را به گذرگاه داده کامپیوتر متصل کند. شکل صفحه بعد نحوه طراحی این کارت را نشان می‌دهد. آدرس 0310 H برای این کارت انتخاب شده است. تفاوت اساسی سخت افزار این کارت با نمونه قبلی، استفاده از سیگنال $\overline{IOR}$ به جای $\overline{IOW}$ است؛ چون این کارت برخلاف قبلی یک کارت ورودی است. به علاوه مدار رمزگشای آدرس طوری تغییر یافته که با قرار گرفتن آدرس 0310 H (و «صفر» شدن سیگنالهای AEN و $\overline{IOR}$) بافر را فعال کند. دیگر نکات طراحی این کارت مانند نمونه قبلی خواهد بود.



شکل ۴-۵- نمونه‌ای از کارت XT ورودی

برنامه این سیستم نیز بسیار شبیه به سیستم قبلی است؛ فقط به جای دستور out که برای ارسال داده‌ها به خروجی است، باید از دستور in استفاده کنیم. دستور *in al,dx* یک عدد یک بایتی از پورت ورودی که شماره آن در DX قرار

[1] Sensor
[2] Analog to Digital Converter: ADC

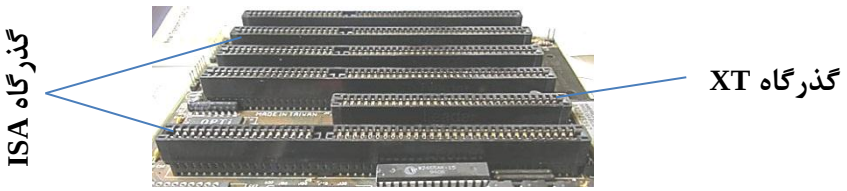
گرفته می‌خواند و در AL ذخیره می‌کند. مانند قبل به جای دستورات اسمبلی خواندن ورودی، می‌توان از دستور inportb یا inp در زبان C استفاده کرد. دستور data = inportb (port); یا data = inp(port); یک عدد یک بایتی از پورت ورودی به شماره port را خوانده و در متغیر یک بایتی data (که باید در برنامه تعریف شود) ذخیره می‌کند[1]. با استفاده از دستور printf و فرمت %X می‌توان یک عدد را در دستگاه شانزده‌تایی نمایش داد.

```
// This program reads the 8-bit data at
// address 310H every 1 second and
// displays it on the screen. Press any
// key to terminate.

#include <STDIO.H>
#include <CONIO.H>
#include <DOS.H>

void main(void){

        clrscr();
         unsigned char data;
         while ( !kbhit() ){

                asm {
                        push ax
                        push dx

                        mov  dx,0310h
                        in      al,dx
                        mov  data,al

                        pop  dx
                        pop  ax
                }
                 printf("%X\n",data);
                delay(1000);
        }
}
```

data = inportb (0x0310) ;

گذرگاه ISA یا گذرگاه AT

گذرگاه ISA در سال ۱۹۸۴ برای جای دادن قابلیتهای پردازنده ۸۰۲۸۶ در گذرگاه سیستم ارائه شد. گذرگاه ISA دارای دو قسمت ۶۲ پایه‌ای و ۳۶ پایه‌ای است که بخش ۶۲ پایه‌ای آن عیناً همان گذرگاه XT است و بخش ۳۶ پایه‌ای گذرگاه داده ۱۶ بیتی و گذرگاه آدرس ۲۴ بیتی پردازنده ۸۰۲۸۶ به اضافه چند سیگنال کنترلی جدید را ارائه می‌کند. شکل ۴-۶ یک بورد اصلی کامپیوتر را نشان می‌دهد که هم گذرگاه XT و هم گذرگاه ISA دارد.

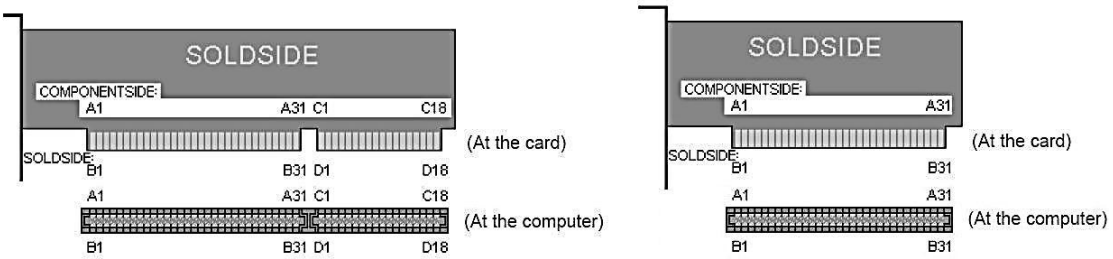


شکل ۴-۶- یک مادربورد دارای گذرگاه‌های XT و ISA

همانطور که در شکل بالا دیده می‌شود، اگر بورد اصلی کامپیوتر فاقد شکاف XT باشد، می‌توان کارت XT را در قسمت بزرگتر شکاف ISA قرار داد. شکل ۴-۷ مقایسه‌ای بین کارت ISA و کارت XT (که گاهی کارت ۸ بیتی ISA نامیده می‌شود) را نشان می‌دهد.

شکل ۴-۸ پایه‌های گذرگاه ISA را نشان می‌دهد. پایه B8 گذرگاه XT رزرو شده است؛ اما در گذرگاه ISA این پایه عملیات خاصی انجام می‌دهد.

[1] دستورات inport و inpw در زبان C برای خواندن اعداد دوبایتی از پورتها استفاده می‌شوند.



کارت **ISA** و شکاف آن — کارت **XT** (یا کارت ۸ بیتی **ISA**) و شکاف آن

شکل ۴-۷- مقایسه شکاف و کارت گذرگاه‌های **XT** و **ISA**

پایه ورودی $\overline{\text{NOWS}}$[1]: کارت ورودی/خروجی با فعال کردن این سیگنال به سیستم اطلاع می‌دهد نیازی به تزریق حالت انتظار در سیکل خواندن و نوشتن ندارد و می‌تواند انتقال داده را در دو سیکل (که حالت معمول در ۸۰۴۸۶ و پنتیوم است) به پایان برساند.

این کار باعث تسریع عملیات ورودی/خروجی می‌شود. اگر کارت XT ساختید که قرار باشد از شکاف ISA استفاده کند و پردازنده کامپیوتر از ۸۰۴۸۶ به بعد بود، این سیگنال را به زمین متصل کنید.

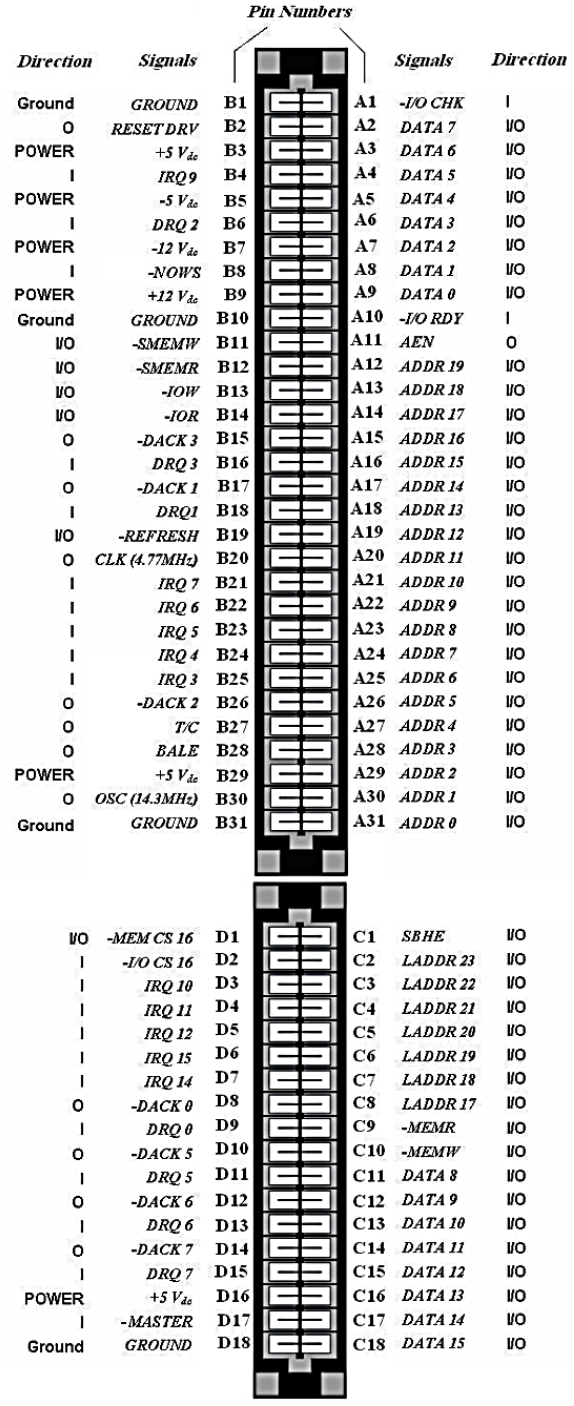
از آنجا که به جز تفاوت بالا، قسمت ۶۲ پایه‌ای گذرگاه ISA با گذرگاه XT که قبلاً معرفی شد یکسان است، به مرور بخش کوچکتر ۳۶ پایه‌ای این گذرگاه که برای استفاده از قابلیتهای پردازنده ۸۰۲۸۶ ایجاد شده می‌پردازیم.

پایه SBHE: این سیگنال، انتقال داده در ۸ بیت بالای گذرگاه داده ۱۶ بیتی را مشخص می‌کند.

پایه‌های آدرس LADDR17~LADDR23: پردازنده ۸۰۲۸۶ دارای ۲۴ خط آدرس است که ۲۰ خط اول آن به صورت نگهداری (Latch) شده در بخش XT گذرگاه ISA و بقیه پایه‌های آدرس بدون قفل شدن در بخش کوچکتر این گذرگاه آمده‌اند که باید به کمک تراشه نگهدار و سیگنال BALE نگهداری شوند. انتظار داریم تنها پایه‌های A_{20} به بعد به گذرگاه اضافه شده باشند، اما می‌بینید که پایه‌های A_{17} و A_{18} و A_{19} نیز اضافه شده‌اند. سیکل حافظه پردازنده‌های ۸۰۲۸۶ به بعد از سیکل حافظه ۸۰۸۸ کوتاهتر است و به همین لحاظ، پایه‌های بالای آدرس پردازنده بدون قفل شدن در پایه‌های LADDR آمده‌اند تا با استفاده از تراشه‌های نگهدار سریع بتوان زمان تأخیر قفل (Latch) شدن آدرس را کاهش داد. بدیهی است از این خطوط آدرس تنها برای کارتهای حافظه استفاده می‌شود و آدرسهای ورودی/خروجی همچنان ۱۶ بیتی هستند.

پایه‌های خروجی $\overline{\text{MEMR}}$ **و** $\overline{\text{MEMW}}$**:** پردازنده از این سیگنالها برای اعلام تبادل اطلاعات با حافظه در کل فضای آدرس‌دهی ۱۶ مگابایتی پردازنده ۸۰۲۸۶ استفاده می‌کند. در بخش XT گذرگاه ISA نیز سیگنالهای $\overline{\text{SMEMR}}$ و $\overline{\text{SMEMW}}$ وجود دارند که تنها برای تبادل اطلاعات در یک مگابایت اول حافظه (که با ۲۰ خط آدرس ذکر شده در گذرگاه XT قابل آدرس‌دهی است) معتبر هستند. پایه‌های $\overline{\text{MEMR}}$ و $\overline{\text{MEMW}}$ مانند $\overline{\text{SMEMR}}$ و $\overline{\text{SMEMW}}$ دوطرفه هستند و به DMA حاکم نیز اجازه دسترسی به فضای حافظه را می‌دهند.

[1] No Wait State یا 0WS



شکل ۴-۸- پایه‌های گذرگاه **ISA**

خطوط داده DATA 8~DATA 15: هشت خط بالای گذرگاه داده پردازنده ۱۶ بیتی ۸۰۲۸۶ در این پایه‌ها آمده است.

پایه‌های ورودی $\overline{\text{MEMCS16}}$ و $\overline{\text{I/O CS16}}$: اگر کارت حافظه بخواهد داده‌های ۱۶ بیتی انتقال دهد، باید سیگنال $\overline{\text{MEMCS16}}$ را فعال نماید. همین وضعیت در مورد کارتهای ورودی/خروجی ۱۶ بیتی وجود دارد که باید سیگنال $\overline{\text{I/O CS16}}$ را فعال نمایند. برای حفظ سازگاری با گذرگاه XT، گذرگاه ISA فرض می‌کند انتقال داده‌ها ۸ بیتی است مگر اینکه خلاف آن از طریق فعال کردن این دو سیگنال توسط طراح کارت به سیستم اطلاع داده شود. اگر در هنگام انتقال داده‌های ۱۶ بیتی از این دو سیگنال استفاده‌ای نشود، یک انتقال ۱۶ بیتی در دو سیکل انتقال ۸ بیتی انجام خواهد شد. این پایه‌ها کلکتور باز هستند.

پایه ورودی $\overline{\text{MASTER}}$: کارت ISA می‌تواند با فعال کردن این سیگنال، حاکم گذرگاه شده و کنترل آن را به دست گیرد و وسایل دیگری در سیستم به صورت تابع[1] آن عمل کنند. در این حالت کارت باید به گونه‌ای طراحی شده باشد که کنترل گذرگاه را جهت تازه‌سازی RAM دینامیک آزاد کند، وگرنه خرابی اطلاعات را به دنبال خواهد داشت. به یاد دارید که در گذرگاه XT راهی برای حاکم شدن کارت XT وجود نداشت. مشکل این شیوه در گذرگاه ISA این است که فعال کردن سیگنال فوق باید به دنبال دریافت یک سیگنال DACK (تصدیق DMA) انجام شود. یعنی کارت نمی‌تواند مستقل از DMA حاکم گذرگاه شود. این موضوع در سیستمهای چند پردازنده‌ای که هر پردازنده می‌خواهد مستقل از DMAکنترل گذرگاه را به دست گیرد مشکل‌ساز می‌شود. این محدودیت در گذرگاههای بعدی برداشته شده است.

پایه‌های دیگر

پایه‌های درخواست وقفه (IRQ)، درخواست DMA ۱۶ بیتی (DRQ)، پذیرش DMA (DACK) و تغذیه اضافی در بخش ۳۶ پایه‌ای گذرگاه ISA آمده‌اند. جریان پایه‌های تغذیه تقریباً دو برابر پایه‌های مشابه در گذرگاه XT است.

[1] Slave

شکل ۴-۸ نشان می‌دهد کارتهای مختلف ISA چگونه از سیگنالهای این گذرگاه استفاده می‌کنند. دقت کنید که جهت سیگنالها در این جدول نسبت به کارت بیان شده است. علامت منفی قبل از نام سیگنال نشانه فعال پایین بودن آن است.

Signal Name	ISA Bus Master	ISA 16-bit Mem Slave	ISA 16-bit I/O Slave	ISA 8-bit Mem Slave	ISA 8-bit I/O Slave	ISA DMA Device
AEN	-	-	I	-	I	-
BALE	-	I	-	(I)	-	-
CLK	(I)	(I)	(I)	(I)	(I)	(I)
-DACK	I	-	-	-	-	I
DRQ	O	-	-	-	-	O
-IO CS16	I	-	O	-	-	-
-I/O CH CK	(O)	(O)	(O)	(O)	(O)	(O)
I/O CH RDY	I	(O)	(O)	(O)	(O)	-
-IOR	O	-	I	-	I	I
-IOW	O	-	I	-	I	I
IRQ	(O)	(O)	(O)	(O)	(O)	(O)
LA(23:17)	O	I	-	(I)	-	-
-MASTER	O	-	-	-	-	-
-MEMCS16	I	0	-	-	-	-
-MEMR	O	I	-	(I)	-	-
-MEMW	O	I	-	(I)	-	-
OSC	(I)	(I)	(I)	(I)	(I)	(I)
-REFRESH	(O)	I	-	I	-	-
RESETDRV	I	I	I	I	I	I
SA(16:0)	O	I	I	I	I	-
SA(19:17)	-	(I)	-	(I)	-	-
SD(7:0)	I/O	I/O	I/O	I/O	I/O	I/O
SD(15:8)	I/O	I/O	I/O	-	-	(I/O)
-SBHE	O	I	I	-	-	-
-SMEMR	-	-	-	I	-	-
-SMEMW	-	-	-	I	-	-
TC	-	-	-	-	-	(I)
-0WS	-	(O)	-	(O)	(O)	-

شکل ۴-۹- نحوه استفاده از پایه‌های گذرگاه ISA در کارت‌های مختلف

وقفه‌ها

برای اطلاع از روی دادن یک رخداد دو روش وجود دارد:

روش سرکشی (سرشماری – Polling) که در آن، دستگاه سرویس‌دهنده (مثلاً کامپیوتر) مرتباً به دستگاه سرویس‌دهنده (مثلاً کنترل‌کننده ارتفاع ماده مذاب در دیگ) سرکشی می‌کند تا ببیند نیاز به سرویس دارد یا خیر؟ این روش ساده است؛ اما زمان زیادی از دستگاه سرویس‌دهنده را تلف می‌کند.

در روش وقفه (Interrupt) ، دستگاه سرویس‌دهنده به کارهای معمول خود می‌پردازد و هرگاه دستگاه سرویس‌گیرنده نیاز به ارائه خدماتی داشت، این نیاز خود را به دستگاه سرویس‌دهنده اعلام می‌کند. سرویس‌دهنده در عملیات معمول خود وقفه آی ایجاد کرده و خدمت مورد نیاز را در قالب اجرای برنامه‌ای به نام ISR[1] ارائه می‌کند و سپس به ادامه عملیات عادی خود می‌پردازد.

به عنوان مثال، فرض کنید یک روبات مین‌یاب طراحی کرده‌ایم و آن را به کامپیوتر متصل کرده‌ایم. کامپیوتر از یک سو باید حرکت روبات را کنترل کند و از سوی دیگر هرگاه روبات به یک مین برخورد کرد، مختصات محل مین را در نقشه میدان مین ثبت کند. اگر از شیوه سرکشی برای کنترل پیدا شدن مین استفاده کنیم، کامپیوتر ناچار است علاوه بر

[1] Interrupt Service Routin

کنترل حرکت روبات، مرتباً حالت سنسور فلزیاب روبات را چک کند و در صورت پیدا شدن مین، مختصات آن را ثبت کند.

روش دیگر این است که کامپیوتر فقط به کنترل روبات بپردازد و روبات هرگاه مین یافت، خودش به کامپیوتر اطلاع دهد؛ در این حال، کامپیوتر کار عادی خودش را قطع می‌کند و قطعه‌برنامه‌ای (ISR) را اجرا می‌کند که مختصات مین را ثبت کند و مجدداً به کار عادی خود برمی‌گردد.

چون در پاسخ به هر وقفه‌ای باید ISR خاصی اجرا شود، کنترلر باید جدولی داشته باشد که به ازای هر شماره وقفه، آدرس آغاز ISR ویژه آن وقفه را مشخص کند. به این جدول بردار وقفه (Interrupt Vector) گفته می‌شود.

وقفه‌های کامپیوتر PC/XT: پردازنده ۸۰۸۸ (و اعضای بعدی خانواده اینتل) دارای ۲۵۶ منبع وقفه است که بعضی از آنها توسط سخت‌افزار جانبی (وقفه‌های IRQ)، بعضی توسط سیستم (مانند وقفه تقسیم بر صفر) و بعضی توسط نرم‌افزار (مانند دستور int 21h در زبان اسمبلی) مورد استفاده قرار می‌گیرند.

در بردار وقفه کامپیوتر، به ازای هر شماره وقفه باید آدرس آغاز ISR آن وقفه نگهداری شود که یک آدرس ۴ بایتی (CS:IP) است. بنابراین طول بردار وقفه کامپیوتر ۱۰۲۴ = ۴ × ۲۵۶ بایت است که در ابتدای حافظه قرار می‌گیرد؛ یعنی ۴ بایت اول بردار وقفه (بایت شماره صفر تا ۳) آدرس آغاز ISR وقفه شماره صفر، ۴ بایت دوم (بایت شماره ۴ تا ۷) آدرس آغاز ISR وقفه شماره یک، ۴ بایت سوم (بایت شماره ۸ تا ۱۱) آدرس آغاز ISR وقفه شماره دو و ... را نگهداری می‌کند. پس برای دسترسی به آدرس آغاز ISR یک وقفه در بردار وقفه کامپیوتر کافی است شماره آن وقفه را در ۴ ضرب کنیم و به بایتی با آن شماره در بردار وقفه مراجعه کنیم.

در کل می‌توان مراحل کاری کامپیوتر پس از دریافت وقفه را به صورت زیر خلاصه کرد:

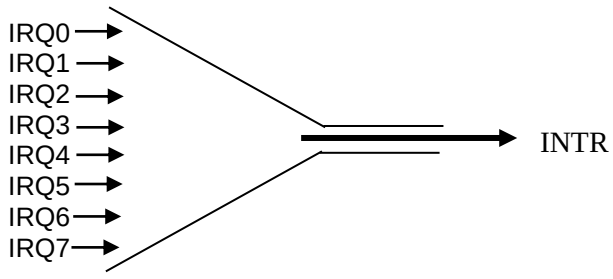
- ✓ کامپیوتر اجرای دستور فعلی را به پایان می‌رساند.
- ✓ آدرس دستور بعدی (نقطه بازگشت) را در پشته ذخیره می‌کند.
- ✓ از کنترلر وقفه شماره وقفه رخ داده را می‌پرسد.
- ✓ شماره وقفه را در ۴ ضرب می‌کند.
- ✓ به بایتی با آن شماره در بردار وقفه مراجعه می‌کند و آدرس ۴ بایتی آغاز ISR را از آنجا می‌خواند.
- ✓ به آدرس مذکور مراجعه کرده و ISR را اجرا می‌کند.
- ✓ به کمک آدرس ذخیره شده در پشته، به نقطه قطع اجرای برنامه بازمی‌گردد و به روال کاری عادی خود ادامه می‌دهد.

لیست وقفه‌های کامپیوتر اولیه IBM PC/XT را در جدول زیر می‌بینید. بسیاری از این آدرسها در کامپیوترهای امروزی نیز معتبرند.

با کاربرد بعضی از این وقفه‌ها در زبان اسمبلی آشنا هستیم (وقفه 10h برای تصویر، وقفه 16h برای کار با صفحه کلید، وقفه 21h برای استفاده از توابع DOS و ...).

شماره وقفه	هدف	شماره وقفه	هدف
0	خطای تقسیم (Divide Error)	1B	کنترل ctrl+Break
1	اجرای تک‌گام (Single Step)	1C	کنترل تایمر
2	وقفه NMI	1D	جدول پارامترهای تصویر
3	نقطه توقف (Break point)	1E	جدول پارامترهای فلاپی دیسک
4	سرریز محاسبات علامت‌دار	1F	جدول کاراکترهای گرافیکی
5	چاپ صفحه نمایش (Print Screen)	20	خاتمه برنامه DOS
6,7	رزرو شده	21	فراخوانی توابع DOS
8	**IRQ0 از ۸۲۵۹ (وقفه تایمر)**	22	آدرس پایان DOS
9	**IRQ1 از ۸۲۵۹ (وقفه صفحه کلید)**	23	آدرس خروج ctrl+Break
A	**IRQ2 از ۸۲۵۹ (رزرو شده- اتصال به ۸۲۵۹ دوم در سیستم‌های AT)**	24	بردار مدیریت خطای بحرانی DOS
B	**IRQ3 از ۸۲۵۹ (پورت COM2)**	25	خواندن مطلق دیسک در DOS
C	**IRQ4 از ۸۲۵۹ (پورت COM1)**	26	نوشتن مطلق دیسک در DOS
D	**IRQ5 از ۸۲۵۹ (دیسک سخت یا درگاه LPT2)**	27	پایان برنامه DOS و مقیم‌سازی در حافظه
E	**IRQ6 از ۸۲۵۹ (فلاپی دیسک)**	28,2E	رزرو برای DOS
F	**IRQ7 از ۸۲۵۹ (درگاه LPT1)**	2F	وقفه Multiplex
10	Video I/O	30-3F	رزرو برای DOS
11	بررسی پیکربندی تجهیزات	40	ورودی/خروجی دیسکت (XT)
12	بررسی اندازه حافظه	41	پارامترهای دیسک سخت (XT)
13	ورودی/خروجی دیسک	42-5F	رزرو برای DOS
14	ورودی/خروجی پورت سریال	60-66	تعریف شده توسط کاربر
15	ورودی/خروجی نوار	67	مدیریت حافظه توسعه یافته (Expanded)
16	ورودی/خروجی صفحه کلید	68-7F	به کار نرفته است
17	ورودی/خروجی چاپگر	80-85	رزرو برای BASIC
18	Load ROM BASIC	86-F0	پارامترهای BASIC
19	Load Boot Strap	F1-FF	به کار نرفته است
1A	تاریخ		

به وقفه‌های 08 تا 0F دقت کنید. این وقفه‌ها که به IRQ[1] معروفند، در واقع وقفه‌های سخت‌افزاری کامپیوتر هستند. پیشتر دیدیم که پردازنده کامپیوتر تنها یک پایه وقفه سخت‌افزاری (INTR) دارد. بنابراین برای اتصال وقفه‌های سخت‌افزاری مختلف به پردازنده، باید از ساختاری



مانند قیف استفاده کنیم که سیگنالهای وقفه سخت‌افزاری را دریافت و آنها را مدیریت کند؛ به بیان دیگر با رعایت اولویت، درخواست وقفه را به کامپیوتر انتقال داده و شماره وقفه را نیز به پردازنده اطلاع دهد. این ساختار قیف‌مانند،

[1] Interrupt Request

PIC[1] نام دارد که در کامپیوتر تراشه ۸۲۵۹ این وظیفه را بر عهده دارد. اولویت وقفه‌ها از IRQ0 تا IRQ7 کاهش می‌یابد.

تعداد وقفه‌های سخت‌افزاری در کامپیوتر PC/XT (مبتنی بر ۸۰۸۸) هفت (یا هشت با احتساب وقفه NMI) و در کامپیوتر PC/AT (مبتنی بر ۸۰۲۸۶ به بعد) پانزده (یا شانزده با احتساب وقفه NMI) می‌باشد.

در کامپیوتر PC/XT وقفه IRQ0 (INT 08h) توسط تایمر کامپیوتر برای تازه‌سازی حافظه RAM پویا به کار رفته است. از آنجا که این وقفه هر ثانیه ۱۸/۲ مرتبه رخ می‌دهد (به اصطلاح تیک می‌زند)، از آن در نوشتن برنامه‌های مقیم در حافظه‌ای که باید در زمانهای مشخصی مرتباً اجرا شوند استفاده می‌شود. برای این کار، باید قطعه برنامه مورد نظر خودتان را بنویسید و آدرس آن را در جدول وقفه‌ها جایگزین آدرس قبلی ISR وقفه تایمر کنید. توجه کنید چون در کامپیوتر اعمال اساسی بر اساس زمانبندی تایمر در ISR این وقفه انجام می‌شود، حتماً باید در انتهای ISR خودتان پرشی به ابتدای ISR اصلی کامپیوتر انجام دهید.

وقفه IRQ1 نیز در بوردهای اصلی کامپیوتر PC/XT برای اتصال صفحه‌کلید به کامپیوتر استفاده شده است. صفحه‌کلید به کمک این سیگنال به کامپیوتر اطلاع می‌دهد که کلیدی توسط کاربر فشرده شده است. دقت کنید که این وقفه سخت‌افزاری است و با وقفه نرم‌افزاری 16h که برای کار با صفحه‌کلید (مثلاً خواندن کاراکتر) به کار می‌رود متفاوت است. اگر می‌خواهید کامپیوتر در هنگام فشرده شدن کلیدی، واکنش خاصی نشان دهد، باید برنامه‌ای مقیم در حافظه بنویسید که طی آن ISR وقفه 09h بازنویسی شود. پرش به ابتدای ISR اصلی کامپیوتر در انتهای ISR خودتان را فراموش نکنید. در مراجع زبان اسمبلی و برنامه‌نویسی سیستم طریقه نوشتن این برنامه‌ها مشخص شده است.

پرسش) ویروسی بنویسید که جای کلیدهای A و S را با هم عوض کند و کلیدهای D و F را نیز از کار بیندازد.

پرسش) ویروسی بنویسید که صفحه کلید کامپیوتر را از کار بیندازد.

وقفه‌های IRQ2 تا IRQ7 برای منابع دیگر وقفه سخت‌افزاری استفاده شده‌اند. همانطور که قبلاً مشاهده کردید، این وقفه‌ها در پایه‌های گذرگاه XT ظاهر شده‌اند. برای استفاده از این وقفه‌ها، کافی است کاربر ISR وقفه مربوطه را بازنویسی کند (در واقع ISR جدیدی بنویسد و آدرس آن را در بردار وقفه جایگزین آدرس ISR اصلی بنماید). مثلاً اگر طراح کارت XT می‌خواهد از منبع وقفه سخت‌افزاری IRQ7 استفاده کند، باید ISR مربوط به وقفه INT 0Fh را بازنویسی کند.

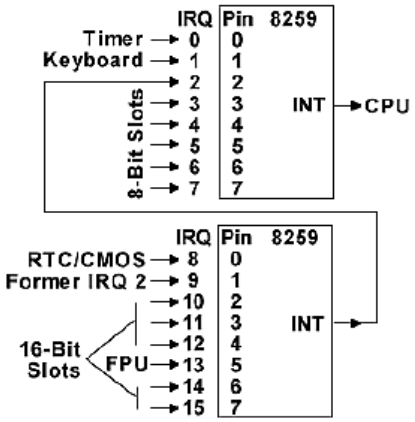
پرسش) وقفه IRQ4 برای پورت سریال COM1 استفاده می‌شود. کاربر برای استفاده از این وقفه چه باید بکند؟ اگر ISR اصلی مربوط به پورت سریال هم مهم باشد، آیا باز هم می‌توان از این IRQ در کارتی که خود طراحی کرده‌ایم، استفاده کنیم؟

پرسش) فرض کنید می‌خواهیم از روبات مین‌یاب (که پیشتر در مورد آن سخن گفتیم) به شیوه وقفه استفاده کنیم. سیگنالی در روبات تعبیه می‌کنیم که هرگاه به یک مین برخورد کرد، این سیگنال را فعال کند. در کارت ورودی XT

[1] Peripheral Interrupt Controller

که برای نظارت بر مین‌یابی طراحی شده، این سیگنال را به IRQ6 متصل می‌کنیم. توضیح دهید نحوه برنامه‌ریزی این کارت باید به چه صورتی باشد؟

پرسش) با توجه به اولویت وقفه‌های IRQ، چرا وقفه‌های IRQ0 و IRQ1 به تایمر و صفحه‌کلید اختصاص یافته‌اند؟



وقفه‌های کامپیوتر PC/AT: در کامپیوترهای PC/AT (مبتنی بر پردازنده‌های ۸۰۲۸۶ به بالا) یک تراشه ۸۲۵۹ در کنار تراشه ۸۲۵۹ قبلی اضافه شد که به صورت مقابل به هم متصل شده‌اند. اولین ۸۲۵۹ در مد حاکم[1] و دیگری در مد تابع[2] مورد استفاده قرار گرفته است.

همانطور که در شکل می‌بینید، خروجی وقفه تراشه ۸۲۵۹ دوم به پایه IRQ2 متصل شده است؛ بنابراین رخ دادن هرکدام از وقفه‌های IRQ8 تا IRQ15 از طریق IRQ2 مربوط به تراشه ۸۲۵۹ اول به پردازنده اطلاع داده می‌شود.

پرسش) نشان دهید اولویت وقفه‌های IRQ در PC/AT به شکل زیر کاهش می‌یابد (از چپ به راست).

IRQ0- IRQ1- IRQ8- IRQ9- IRQ10- IRQ11- IRQ12- IRQ13- IRQ14- IRQ15- IRQ3- IRQ4- IRQ5- IRQ6- IRQ7

وقفه‌های IRQ جدید به صورت معمول طبق جدول زیر مورد استفاده قرار گرفته‌اند:

وقفه سخت‌افزاری	شماره وقفه	هدف معمول
IRQ8	70H	ساعت بلادرنگ تراشه CMOS RAM
IRQ9	71H	تغییر جهت نرم‌افزاری به IRQ2 (INT 0Ah)
IRQ10	72H	-
IRQ11	73H	-
IRQ12	74H	ماوس PS/2
IRQ13	75H	کمک‌پردازنده (FPU)
IRQ14	76H	دیسک سخت
IRQ15	77H	-

اگر به قسمت ۳۶ پایه‌ای گذرگاه ISA دقت کنید، می‌بینید که IRQ10، IRQ11، IRQ12، IRQ14 و IRQ15 در این قسمت در اختیار طراح قرار دارند. پایه‌های IRQ8 و IRQ13 برای مقاصد سیستمی (جدول بالا را ببینید) استفاده شده‌اند.

IRQ9 کجاست؟

در PC/XT، وقفه IRQ2 از طریق گذرگاه XT در اختیار طراح کارت بود. اما چون در PC/AT IRQ2 به عنوان ورودی سیگنال وقفه تراشه ۸۲۵۹ دوم به کار رفت، IBM آن را با سیگنال IRQ9 (IRQ دوم تراشه ۸۲۵۹ دوم) جایگزین کرد. اگر به قسمت ۶۲ پایه‌ای گذرگاه ISA دقت کنید، می‌بینید که سیگنال IRQ9 دقیقاً سر جای سیگنال IRQ2 قرار

[1] Master
[2] Slave

گرفته است. طراح کارت باید در نوشتن برنامه و بازنویسی ISR مناسب، مراقب این نکته باشد. دلیل این جایگزینی به شیوه فوق این است که بالاترین اولویت وقفه در PC-XT مربوط به IRQ2 و در PC-AT مربوط به IRQ9 است. چون در PC-AT سیگنال IRQ2 دیگر وجود ندارد، بنابراین IRQ9 دقیقاً در جای IRQ2 قرار گرفته تا چنانچه یک کارت XT در شکاف ISA جا زده شود، اولویت وقفه آن تغییری نکند.

پرسش) اگر برای کارت ورودی ISA روبات مین‌یاب بخواهیم از سیگنال IRQ11 استفاده کنیم، کارت باید به چه صورتی برنامه‌ریزی شود؟

وقفه‌های گذرگاه ISA حساس به لبه[1] هستند. این موضوع سبب می‌شود که سیگنالهای مزاحم نیز بتوانند ایجاد وقفه کنند. به همین لحاظ در گذرگاههای بعدی وقفه‌ها حساس به سطح[2] شده‌اند. با این کار به اشتراک گذاشتن یک وقفه IRQ بین چند منبع وقفه نیز به سادگی انجام می‌شود. چون در حالت حساس به لبه، اعلام وقفه تنها در یک لحظه انجام می‌شود و تشخیص اینکه کدامیک از منابع مشترک وقفه اعلام وقفه کرده‌اند پیچیده خواهد بود. این موضوع به ویژه در سالیان اخیر که برای تجهیزات جدید با اتمام پذیرنده‌های وقفه خارجی (IRQ) مواجهیم حائز اهمیت است.

پرسش) با الهام از چگونگی اضافه کردن ۸ وقفه سخت‌افزاری جدید به PC/AT، توضیح دهید چگونه می‌توانید یک کارت XT طراحی کنید که ۱۰ منبع ایجاد وقفه بتواند داشته باشد.

استثناءها[3] در کامپیوترهای PC/AT

شرکت IBM برای کامپیوتر PC/XT از پردازنده ۸۰۸۸ بهره برد. در آن زمان تنها پنج منبع وقفه INT 00h تا INT 04h پیش‌بینی شده بود که IBM نیز از آنها استفاده کرد. تنها وضعیت خطای غیرعادی که باعث بروز استثناء شود، وضعیت تقسیم بر صـــفر بود که با INT 0h پوشش داده می‌شد.

اینتل ۳۲ وقفه اول (INT 00h تا INT 1Fh) را رزرو کرده بود تا برای پوشش استثناءها در پردازنده‌های آینده از آنها استفاده کند؛ اما متأسفانه IBM به این موضوع توجهی نشان نداد و از بعضی وقفه‌های این فضای رزرو شده نیز استفاده کرد. به همین لحاظ در کامپیوترهای AT که از تعدادی از وقفه‌های رزرو شده برای پوشش استثناهای پردازنده در مد حفاظت شده استفاده شده است، طراحان کامپیوتر ناچار شده‌اند همه وقفه‌های جدید را برای جلوگیری از تداخل وقفه با IRQهای کامپیوتر به کار گیرند.

جدول صفحه بعد، وقفه‌های پردازنده‌های ۸۰۲۸۶ به بعد را نشان می‌دهد.

پیش از آن که به بحث راجع به گذرگاههای دیگر بپردازیم ذکر نکته‌ای لازم است. اکنون شما با اصول کلی طراحی کارتهای کامپیوتری آشنا هستید. اما این آشنایی به این معنا نیست که برای حل هر مسأله و انجام هر پروژه‌ای باید یک کارت «طراحی» کرد. هنر یک مهندس تنها طراحی دقیق و ماهرانه یک سیستم نیست؛ بلکه باید بتواند به تناسب موضوع پروژه، از سیستمهای از پیش طراحی شده نیز استفاده کند. به عنوان مثال شرکت Advantek طیف وسیعی از کارتهای کامپیوتری با ویژگیها و عملکردهای بسیار متنوع را در اختیار کاربران قرار می‌دهد. شما باید علاوه بر توانایی طراحی

[1] Edge Trigerred
[2] Level Trigerred
[3] Exceptions

کارت، در مواقع ضروری بتوانید با مراجعه به کاتالوگهای شرکتهای تولیدکننده سخت‌افزارهای آماده، محصولات مناسب برای کار خود را نیز انتخاب کنید. در دنیای امروزی، یک مهندس موفق، لزوماً یک «طراح[1]» موفق نیست؛ بلکه یک «جمع‌کننده سیستم[2]» موفق است.

شماره وقفه	پردازنده	عملکرد
0	همه	خطای تقسیم
1	همه	تک‌گام (Single Step)
2	همه	NMI
3	همه	نقطه توقف (Break Point)
4	همه	وقفه سرریز (INTO)
5	۸۰۲۸۶ تا پنتیوم IV	نادیده گرفتن مرز (Bound)
6	۸۰۲۸۶ تا پنتیوم IV	کد عملیاتی نامعتبر
7	۸۰۲۸۶ تا پنتیوم IV	پیدا نشدن کمک‌پردازنده
8	۸۰۲۸۶ تا پنتیوم IV	خطای دوگانه (Double Fault)
9	۸۰۳۸۶	Co-processor Segment Overrun
A	۸۰۲۸۶ تا پنتیوم IV	TSS نامعتبر
B	۸۰۲۸۶ تا پنتیوم IV	سگمنت پیدا نشد
C	۸۰۲۸۶ تا پنتیوم IV	خطای پشته
D	۸۰۲۸۶ تا پنتیوم IV	خطای حفاظت (General Protection Fault)
E	۸۰۳۸۶ تا پنتیوم IV	خطای صفحه (Page Fault)
F	–	رزرو شده
10	۸۰۲۸۶ تا پنتیوم IV	خطای پردازنده کمکی (Floating Point erro
11	۸۰۴۸۶ مدل SX	بررسی انطباق (Allignment Check)
12	پنتیوم به بعد	بررسی ماشین (Machine Check)
13-1F	–	رزرو شده
20-FF	–	وقفه‌های کاربر

محدودیت‌های گذرگاه ISA

گذرگاه ISA سالها در کامپیوترها مورد استفاده قرار می‌گرفت و حتی در بوردهای کامپیوتر با پردازنده پنتیوم هم یک یا دو شکاف ISA به چشم می‌خورد. اما محدودیتهای این گذرگاه باعث شد کم‌کم استفاده از آن محدود به دستگاههای صنعتی شود. این محدودیتها عبارتند از:

- ✓ گذرگاه داده آن محدود به ۱۶ بیت است و نمی‌تواند از گذرگاههای داده ۳۲ بیتی پردازنده‌های بعدی حمایت کند.

[1] Designer
[2] System Integrator

- ✓ گذرگاه آدرس آن محدود به ۲۴ بیت و تنها قادر به آدرس‌دهی ۱۶ مگابایت است. در حالی که گذرگاه آدرس پردازنده‌های بعدی ۳۲ بیتی و قادر به آدرس‌دهی ۴ گیگابایت هستند.
- ✓ به دلیل سطح بزرگ شکاف، به هر سیگنال بار خازنی و القایی بزرگی وارد می‌شود که باعث محدودیت فرکانس در ۸ مگاهرتز می‌شود[۱]. یعنی پردازنده می‌تواند در فرکانسهای بسیار بالاتر (مثلاً ۲۳۳ مگاهرتز) کار کند، اما هنگام تبادل داده با کارت ISA این سرعت به ۸ مگاهرتز کاهش می‌یابد. نداشتن پایه‌های زمین اضافی برای کاهش اثرات تداخل و تشعشع‌های رادیویی به این موضوع دامن می‌زند[۲].
- ✓ سطح بزرگی را روی بورد اصلی اشغال می‌کند.

مشکل Plug & Play: شاید بتوان مهمترین مشکل گذرگاههای ISA و XT را بعد از محدودیت سرعت، عدم پیکربندی خودکار آنها دانست. هنگامی که مدار رمزگشای آدرس یک کارت XT یا ISA یا پایه‌های DRQ و DACK و IRQ را روی کارت سیم‌بندی می‌کنید، این آدرس و منابع وقفه و DMA سیم‌بندی شده به این کارت نسبت داده می‌شود و قابل تغییر نیست. البته روی بعضی کارتهای XT یا ISA به کمک DIP Switch می‌توان کمی انعطاف‌پذیری ایجاد کرد. در این روش مثلاً ۴ بیت پایین آدرس کارت را به کمک یک سوییچ چهارتایی ایجاد می‌کنند. به این صورت کارت می‌تواند دارای ۱۶ آدرس مختلف باشد.

حالت عملیاتی مناسب‌تر این است که سیستم انعطاف‌پذیر باشد و بتواند نسبت به نصب کارتهایی با منابع درخواستی مختلف واکنش مناسب نشان داده و پیکربندی آنها را خود بر عهده گیرد. گذرگاههای جدید (مانند PCI) دارای خاصیت پیکربندی خودکار هستند که به اصطلاح به آن Plug & Play (یا به اختصار PnP) گفته می‌شود. آدرس کارتهای PnP و نیز مشخصات دقیق منابعی که به آن اختصاص داده می‌شود، از پیش تعیین شده نیست و هر بار با روشن شدن سیستم و طی یک همکاری بین BIOS و سیستم‌عامل مشخص می‌شود. عبارت Plug & Play به معنای «نصب کن و استفاده کن» است؛ یعنی کاربر تنها باید کارت را داخل شکاف وارد کند و دیگر نگرانی از جانب تداخل آدرسها و تخصیص منابع ندارد.

این در حالی است که در کارتهای ISA و XT آدرسی که به کارت نسبت داده شده قابل تغییر نیست و خود کاربر باید مراقب باشد تداخل[۳] بین آدرس و منابع کارتهای نصب شده در سیستم به وجود نیاید.

وقتی در سیستمی هم کارتهای PnP و هم کارتهای ISA یا XT وجود داشته باشد، این نگرانی افزایش می‌یابد. چون کاربر از آدرسها و منابعی که سیستم به کارتهای PnP اختصاص می‌دهد بی‌اطلاع است و نمی‌داند مجاز به استفاده از چه آدرسها و منابعی است که مشکل تداخل ایجاد نشود. به این خاصیت کارتهای فوق به اصطلاح Plug & Pray گفته می‌شود؛ یعنی کاربر باید کارت را در شکاف بگذارد و دعا کند که مشکل تداخل به وجود نیاید! البته IBM تلاشهایی

---

[۱] در واقع امپدانس معادل خازن برابر $\frac{1}{jc\omega}$ است که با بالاتر رفتن C باید ω کاهش یابد تا امپدانس معادل خط ثابت بماند.

[۲] حالت ایده‌آل از نظر الکترونیکی این است که هر سیگنال دارای مسیر برگشت (زمین) جداگانه‌ای باشد. اگر مسیرهای برگشت یکسان باشند، باعث جمع شدن آثار برگشتی سیگنالهای مختلف و ایجاد نویز روی زمین مدار می‌شوند. بنابراین هرچه تعداد خطوط تغذیه و زمین گذرگاه بیشتر باشد، آثار نویز در آن کمتر است. به علاوه وجود سیگنالهای زمین متعدد طبق قانون قفس فارادی باعث کاهش آثار تداخل‌های الکترومغناطیسی می‌شود.

[3] conflict

برای ارائه ISA Plug & Play انجام داد که به موفقیت چندانی نیانجامید. در حال حاضر ویندوز Vista تنها از این نوع گذرگاه ISA حمایت محدودی می‌کند.

در بعضی سیستمهای صنعتی گذرگاهی به نام PC/104 وجود دارد که سیگنالهای آن مانند ISA با اتصالهای متفاوت است. گسترش گذرگاه ISA بیشتر برای اتصال دیسکهای سخت به کامپیوتر دنبال شد که گذرگاههای ATA[1] (که به گذرگاه IDE[2] نیز معروف است) و نوع فعلی آن SATA[3] حاصل این گسترش هستند. گذرگاه PCMCIA نسخه‌ای مشتق‌شده از گذرگاه ATA است.

امروزه طراحی کارتهای ISA کاربرد محدودی در سیستمهای صنعتی و قدیمی دارد. اما آشنایی با اصول طراحی این کارت، دید مهندسی نسبت به موضوع را تقویت می‌کند. در حال حاضر برای طراحی مدارهای واسطه با کامپیوتر معمولاً از پورتهای کامپیوتر (موازی، سریال، USB، Joystick و ...) استفاده می‌شود. واسطهای پیشرفته‌تر در قالب کارتهای PCI هستند که در ادامه به آنها خواهیم پرداخت.

گذرگاه MCA: این گذرگاه توسط IBM در سال ۱۹۸۷ برای استفاده از ویژگیهای پردازنده ۸۰۳۸۶ در کامپیوترهای IBM-PS/2 ارائه شد. گذرگاه MCA قربانی اشتباهات IBM شد که می‌تواند برای طراحان بعدی عبرت‌آموز باشد. طرح پایه و مشخصات سیگنالهای MCA کلاً با گذرگاه ISA متفاوت بود و همین موضوع باعث ناسازگاری تعداد زیادی از کارتهای موجود در بازار با کامپیوترهای جدید IBM می‌شد. اما قصه به همین جا ختم نشد. IBM گذرگاه MCA را یک گذرگاه انحصاری اعلام و از سازندگان دیگر کامپیوتر که تمایل به استفاده از آن داشتند تقاضای حق مجوز کرد و حتی کار را به جایی رساند که برای هر کامپیوتر فروخته شده تا آن زمان درخواست حق امتیاز ثابت نمود! با این تفاصیل، تحریم MCA توسط سازندگان دیگر امری دور از انتظار نبود. ۹ شرکت بزرگ کامپیوتری آن زمان[4]، با یکدیگر متحد شده و ضمن مستندسازی ویژگیهای گذرگاه ISA[5]، نسخه توسعه‌یافته‌ای از آن به نام EISA را معرفی نمودند که کاربرد وسیعی یافت.

بد نیست نگاهی به ویژگیهای گذرگاه ناکام MCA بیندازیم:

- ✓ گذرگاه آدرس آن ۳۲ بیتی است.
- ✓ می‌تواند با گذرگاههای داده ۸ و ۱۶ و ۳۲ بیتی کار کند.
- ✓ تطبیق با ویژگیهای پردازنده‌های ۸۰۳۸۶/۴۸۶
- ✓ ساختار چندحاکمی[6]
- ✓ قابلیت آرایش نیمه‌خودکار (در ویژگیهای گذرگاه EISA توضیح بیشتری داده خواهد شد)

---

[1] Advanced Technology Attachment
[2] Integrated Drive Electronics
[3] Serial ATA
[4] AST Research, Compaq Computer, Epson, Hewlett-Packard, NEC, Olivetti, Tandy, WYSE, Zenith Data Systems – این ۹ شرکت به گروه Gang of Nine معروف شدند.

[5] در واقع نام رسمی این گذرگاه AT بود که بعد از مستندسازی ویژگیهایش به ISA تغییر نام یافت. مستند IEEE در مورد گذرگاه ISA بـه P996 معـروف است.

[6] Multi-Matser

- ✓ فرکانس کاری آن ۱۰ مگاهرتز است که از نظر تئوری می‌تواند تا ۳۰ مگاهرتز نیز افزایش یابد. این افزایش سرعت به کمک اضافه شدن خطوط تغذیه و زمین بین هر سه خط دیگر که منجر به کاهش نویز می‌شود، میسر شده است. به علاوه ابعاد شکاف MCA و سطح اتصال آن نسبت به ISA کوچکتر است و با کاهش خاصیت خازنی، اجازه کار در فرکانسهای بالا را می‌دهد.
- ✓ شکاف MCA دارای پایه‌های استفاده نشده برای گسترش آینده است.

گذرگاه EISA: برای رفع محدودیتهای گذرگاه ISA و برای مقابله با انحصارطلبی IBM در مورد گذرگاه MCA، نسخه گسترش‌یافته این گذرگاه به نام EISA توسط سازندگان دیگر در سال ۱۹۸۸ ارائه شد. ایده اصلی این گذرگاه، جای دادن به پایه‌های جدید پردازنده ۸۰۳۸۶ بود. اما چون تعداد زیادی از کامپیوترها تنها گذرگاه ISA داشتند و نیز تعداد زیادی کارت ISA در بازار موجود بود، گذرگاه EISA باید به نحوی طراحی می‌شد که در صورت لزوم کارت EISA بتواند با شکاف ISA و بالعکس کار کند. به همین دلیل، پایه‌های جدید بین پایه‌های قبلی شکاف ISA اضافه شدند. شکل ۴-۱۰ یک کارت EISA را نشان می‌دهد. اگر به یک شکاف EISA به دقت نگاه کنید، خواهید دید که گیرنده‌های پایه‌های ویژه EISA که بین گیرنده‌های پایه‌های ISA تعبیه شده‌اند، در عمق بیشتری قرار دارند؛ بنابراین اگر یک کارت ISA در یک شکاف EISA قرار گیرد، فقط با گیرنده‌های پایه‌های ویژه ISA در تماس است و به درستی کار می‌کند. اگر یک کارت EISA نیز در شکاف ISA قرار بگیرد، فقط پایه‌های ویژه ISA از کارت با شکاف در تماس است و می‌تواند از قابلیتهای گذرگاه ISA استفاده کند.

پایه‌هایی که در گذرگاه ISA قرار می‌گیرند. بقیه پایه‌ها که بین آنها هستند مربوط به گذرگاه EISA هستند.

شکل ۴-۱۰- گذرگاه EISA

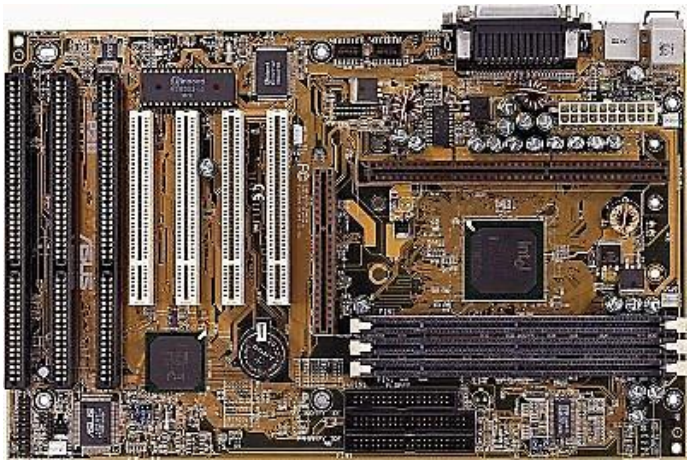
ویژگیهای گذرگاه EISA عبارتند از:

- ✓ اضافه شدن خطوط آدرس جدید (تا A_{31}~A_{23}) و خطوط داده جدید (تا D_{31}~D_{16}) برای جای دادن به پهنای بیشتر گذرگاه داده و آدرس پردازنده ۸۰۳۸۶
- ✓ چون پایه‌های جدید بین پایه‌های قبلی اضافه شده‌اند، سطح گذرگاه تغییر نکرده و به همین لحاظ سرعت گذرگاه ISA همچنان در ۸ مگاهرتز محدود است.

- ✓ کارتهای EISA آرایش نیمه خودکار دارند. به همراه کارت EISA یک فایل پیکربندی نیز دریافت می‌کنید که با اجرای آن، کامپیوتر اطلاعات مربوط به آن کارت از قبیل آدرس پورتهای ورودی/خروجی و نیز حافظه‌هایی که به کارت اختصاص داده شده و کانالهای IRQ و DMA را در حافظه‌ای که روی کارت قرار دارد، ذخیره کرده و در دفعات بعدی از آن استفاده می‌کند.

گذرگاه محلی[1]**:** همانطور که بروز توانایی یک خودروی سریع‌السیر بستگی به وجود یک جاده مناسب دارد، پردازنده‌های پیشرفته نیز به گذرگاه مناسبی نیاز دارند تا بتوانند از تمام توانایی خود استفاده کنند. در حالی که سرعت کاری پردازنده‌ها رو به افزایش بود، محدودیت سرعت گذرگاههای ISA و EISA در ۸ مگاهرتز ایجاد مشکل می‌نمود. سرعت دستیابی پردازنده به حافظه از دیگر مشکلات مهمتر بود. به همین لحاظ شرکت Compaq شکافهای مخصوص حافظه و مجزا از ISA طراحی کرد که هم‌اکنون نیز روی بوردهای اصلی استفاده می‌شود. این گذرگاهها در سرعتی نزدیک به سرعت پردازنده کار می‌کنند و به همین خاطر از سرعت بالایی برخوردارند. این سیستمها به دوگذرگاهی[2] معروف شدند.

برای افزایش سرعت ارتباط کارتهای جانبی (کارت گرافیکی، دیسک سخت یا شبکه) با پردازنده، بعضی سازندگان برای کامپیوترهای خود، گذرگاههای اختصاصی ارائه نمودند که به دلیل انحصاری بودن موجب هزینه بالایی در تولید می‌شد. به همین جهت طراحی گذرگاهی که بتواند با سرعتی نزدیک به پردازنده کار کند، در دستور کار سازندگان کامپیوتر قرار گرفت. این گذرگاه به نام گذرگاه محلی معروف شد. از آنجا که سرعت بالا برای نمایش تصاویر و گرافیک بیش از بقیه کاربردها اهمیت دارد، سازمان استانداردهای تصویر الکترونیک[3] در طراحی این گذرگاه پیش‌قدم شد و گذرگاهی به نام VL[4] ارائه کرد که با تغییراتی تبدیل به گذرگاه PCI شد.



گذرگاه PCI: شرکت اینتل بعد از معرفی پردازنده ۸۰۴۸۶، کار بر روی گذرگاه محلی PCI را برای استفاده مناسب از تواناییهای پردازنده‌هایش در سال ۱۹۹۰ آغاز نمود و پس از تکمیل، آن را به رایگان در اختیار تمام سازندگان قرار داد تا تجهیزات خود را تحت استاندارد آن ارائه کنند. کارتهای PCI سالهاست به صورت گسترده مورد استفاده قرار می‌گیرند. کارتهای صوتی، مودم، شبکه و حتی USB از نمونه‌های کارتهای PCI هستند. اگر درب جعبه کامپیوتر خود را باز کنید، تعدادی شکاف سفیدرنگ می‌بینید (شکافهای XT و ISA و EISA سیاه‌رنگ بودند) که همان شکافهای PCI هستند و کارتهای PCI در آنها نصب می‌شوند.

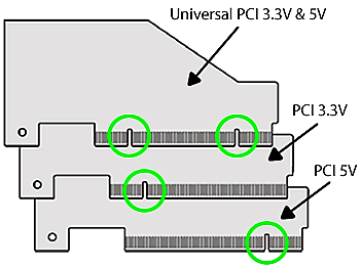
---

[1] Local Bus
[2] Dual Bus
[3] Video Electronics Standard Association: VESA
[4] VESA Local Bus

پرسش) در شکل صفحه قبل، کدام گذرگاهها را می‌توانید تشخیص دهید؟

ویژگیهای گذرگاه PCI عبارتند از:

- ✓ با قرار دادن پایه‌های زمین و تغذیه بین سیگنالهای دیگر و نیز کاهش سطح اتصال گذرگاه به بود اصلی، سرعت گذرگاه به ۳۳ مگاهرتز رسیده است.
- ✓ توانایی کار با گذرگاه داده ۳۲ بیتی و ۶۴ بیتی
- ✓ ساختار چندحاکمی
- ✓ ساختار مستقل از پردازنده (با پردازنده‌های غیر اینتل نیز می‌تواند کار کند)
- ✓ به کمک قطعه‌ای به نام پل[1]، می‌تواند با کارتهای دیگر (ISA، EISA و MCA) نیز کار کند.
- ✓ در دو نوع ۵ ولتی و ۳/۳ ولتی ارائه شده است. ساختار قسمت پایین این دو نوع کارت با هم متفاوت است تا به اشتباه کارت ۵ ولتی در شکاف ۳/۳ ولتی قرار داده نشود. شکل روبرو این دو نوع کارت به اضافه کارتی که در هر دو شکاف قرار می‌گیرد را نشان می‌دهد. اگر تعداد شکافها به بیش از پنج عدد برسد سرعت گذرگاه از ۳۳ مگاهرتز کمتر می‌شود.



- ✓ خاصیت Plug & Play دارد و تراشه کنترل‌کننده PCI وظیفه تخصیص منابع را بدون نیاز به دخالت کاربر بر عهده می‌گیرد. در کامپیوترهای فعلی، گذرگاههای معمول برای اتصال وسایل جانبی (PCI، PCI EXPRESS، AGP، USB، IEEE1394 (Fire Wire) و ...) همگی دارای این خاصیت هستند.

طراحی کارتهای PCI: همانطور که گفته شد، امروزه طراحی کارتهای ISA کاربرد محدودی در سیستمهای صنعتی دارد. در حال حاضر برای اتصال یک دستگاه ورودی/خروجی به کامپیوتر معمولاً از گذرگاه PCI استفاده می‌شود. اصول طراحی کارت PCI با کارت ISA که قبلاً آموختیم شباهت زیادی دارد. اما شناخت سیگنالهای گذرگاه PCI و نیز مفاهیم جدید آن مانند ساختار چندحاکمی و Plug & Play ضروری است که در مراجع [۱] و [۵] به آنها پرداخته شده است.

برای طراحی کارتهای PCI معمولاً از دو روش استفاده می‌شود. در روش اول، از تراشه‌های کنترل‌کننده گذرگاه PCI که به صورت آماده توسط شرکتهای سخت‌افزاری (مانند PLX) ارائه می‌شود، استفاده می‌کنند. این تراشه‌ها که به رابط PCI[2] معروفند، از یک طرف به گذرگاه PCI متصل شده و پاسخگوی عملیات و سیگنالهای گذرگاه هستند و از سوی دیگر یک گذرگاه ساده برای اتصال به وسایل ورودی/خروجی فراهم می‌آورند.

در روش دیگر، می‌توان از تراشه‌های قابل برنامه‌ریزی FPGA استفاده کرد. در این روش، طراح کارت باید کد توصیف سخت‌افزار[3] گذرگاه PCI را بنویسد (یا خریداری کند) و به همراه کد عملیاتی که باید توسط کارت انجام شود، داخل یک FPGA قرار دهد. این روش از روش قبلی دشوارتر است؛ اما این مزیت را دارد که طرح سیستم در

---

[1] Bridge یا Daughter-Board

[2] PCI Interface

[3] Hardware Description Language: HDL

اختیار طراح است و می‌تواند به سادگی آن را تغییر دهد. به علاوه اگر کد PCI نیز توسط طراح نوشته شود، هزینه کمتری خواهد داشت.

پهنای باند گذرگاه: منظور از سرعت یک گذرگاه چیست؟ چرا گفته می‌شود سرعت گذرگاههای ISA و حتی PCI برای عملیات گرافیکی مناسب نیست؟ آیا فرکانس کاری یک گذرگاه (۸ مگاهرتز برای گذرگاه ISA و ۳۳ مگاهرتز برای گذرگاه PCI) همان سرعت گذرگاه است؟

این جمله کاملاً صحیح نیست؛ منظور از سرعت گذرگاه، توانایی آن در انتقال داده‌ها در واحد زمان است که بر حسب بایت در ثانیه بیان می‌شود و به فرکانس گذرگاه، تعداد خطوط داده گذرگاه و مد انتقال داده بستگی دارد.

هر عمل انتقال داده شامل دو سیکل کاری (یک سیکل برای تعیین آدرس و یک سیکل برای خواندن یا نوشتن داده‌ها) است. وقتی گفته می‌شود فرکانس کاری گذرگاه ISA برابر ۸ مگاهرتز است، در واقع می‌توان به کمک این گذرگاه ۴ میلیون انتقال داده در ثانیه انجام داد. چون پهنای گذرگاه داده در ISA برابر ۲ بایت (۱۶ بیت) است، پهنای باند گذرگاه ISA برابر ۸ میلیون بایت در ثانیه یا 8 MegaByte/Sec خواهد بود.

جدول زیر، پهنای باند گذرگاههای مختلف کامپیوتری را با هم مقایسه می‌کند.

نوع گذرگاه	پهنای گذرگاه داده (بیت)	حداکثر فرکانس گذرگاه (MHz)	حداکثر پهنای باند گذرگاه در حالت انتقال معمولی (MByte/Sec)	حداکثر پهنای باند گذرگاه در حالت انتقال پیاپی (MByte/Sec)
ISA	۱۶	۸	۸	–
EISA	۳۲	۸/۳	۱۶	۳۲
MCA	۳۲	۱۰	۲۰	۴۰
PCI	۳۲	۳۳/۳۳	۶۶/۶۶	۱۳۳/۳۳
PCI-2X	۳۲	۶۶/۶۶	۱۳۳/۳۳	۲۶۶/۶۶
PCI-64bit	۶۴	۳۳/۳۳	۱۳۳/۳۳	۲۶۶/۶۶
PCI2X-64bit	۶۴	۶۶/۶۶	۲۶۶/۶۶	۵۳۳/۳۳

ذکر نکته‌ای در اینجا لازم است. از گذرگاه EISA به بعد، قابلیتی به نام انتقال در حالت پیاپی[1] وجود دارد که برای انتقال بلوکی داده‌ها به کار می‌رود. در این حالت، حاکم ابتدا آدرس ابتدایی مقصد را برای تابع ارسال کرده و اطلاعات مربوط به آن آدرس را منتقل می‌کند. برای انتقالهای بعدی، دیگر نیازی به تولید آدرس نیست و تابع به صورت خودکار یک واحد به آدرس اضافه کرده و داده بعدی را انتقال می‌دهد. در این حالت هر انتقال داده فقط شامل یک سیکل است و پهنای باند ذکر شده در جدول بالا به دوبرابر افزایش می‌یابد.

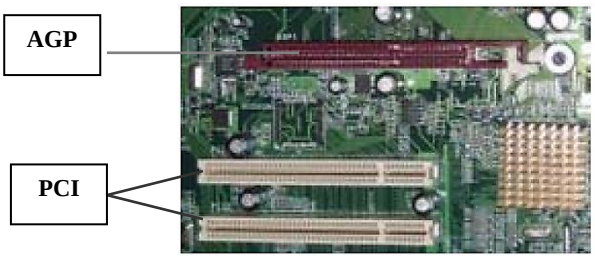
برای نمایش گرافیکی با دقت[2] بسیار بالا و نیز اجرای نرم‌افزارهایی که نیاز به کیفیت گرافیکی بالایی دارند (مانند بعضی بازیها) به پهنای باند بالا نیاز است.

مثلاً اگر دقت تصویر نمایشگر کامپیوتر روی ۱۲۰۰×۱۶۰۰ پیکسل و تفکیک رنگ آن روی ۴ میلیارد رنگ (سیستم رنگ ۳۲ بیتی که به سیستم نمایش با کیفیت بالا[3] معروف است) تنظیم شده باشد و در هر ثانیه ۸۰ بار تصویر تازه‌سازی

[1] Burst Mode Transfer
[2] Resolution
[3] High Quality

شود، در هر ثانیه باید حدود ۶۱۴ مگابایت (۱۲۰۰×۱۶۰۰×۳۲×۸۰ بیت) اطلاعات برای نمایشگر کامپیوتر ارسال شود[۱]. به همین لحاظ هرچند کارتهای گرافیکی ISA و PCI نیز وجود داشتند (و دارند)، اما در حال حاضر اکثر کارتهای گرافیکی (و کارتهایی که در رابطه با نمایش گرافیکی کار می‌کنند مانند کارت Capture و کارت TV) از نوع AGP یا PCI-EXPRESS هستند که برای رهایی از کندی ارتباطات با پردازنده، خود دارای پردازنده مجزا روی کارت هستند.

در انتها بد نیست مروری بر گذرگاههای معمول دیگر داشته باشیم.



گذرگاه AGP: این گذرگاه در سال ۱۹۹۶ توسط اینتل برای کارتهای گرافیکی پیشنهاد شد. اگر بورد اصلی کامپیوتر خود را نگاه کنید، یک شکاف به رنگ قهوه‌ای یا بنفش می‌بینید که گذرگاه AGP است.

گذرگاه AGP با فرکانس پایه ۶۶ مگاهرتز کار می‌کند و تاکنون سرعتهای 1X و 2X و 4X و 8X آن به بازار عرضه شده است. جدول صفحه بعد، ویژگیهای این نسخه‌های AGP را با هم مقایسه می‌کند. توجه کنید با وجودی که فرکانس گذرگاه AGP ثابت است، پهنای باند انتقال داده گذرگاه به کمک تکنیک‌هایی (مثلاً ارسال داده‌ها در هر دو لبه پایین‌رونده و بالارونده) افزایش یافته است.

نوع گذرگاه	AGP-1X	AGP-2X	AGP-4X	AGP-8X
پهنای گذرگاه داده (بیت)	۳۲	۳۲	۳۲	۳۲
حداکثر فرکانس گذرگاه (MHz)	۶۶/۶۶	۶۶/۶۶	۶۶/۶۶	۶۶/۶۶
پهنای باند انتقال داده (MB/Sec)	۲۶۶/۶۶	۵۳۳/۳۳	۱۰۶۶/۶۶	۲۱۳۳/۳۲

گذرگاه PCI-EXPRESS (PCI-E یا PCIe): این گذرگاه در سال ۲۰۰۴ توسط اینتل به عنوان جایگزینی برای گذرگاههای PCI و AGP معرفی شد. برخلاف گذرگاههای قبلی، PCIe یک گذرگاه یک‌بیتی سریال یک‌طرفه و نقطه‌به‌نقطه است. در هر گذرگاه از یک تا ۳۲ گذرگاه تک‌بیتی وجود دارد که در نسخه فعلی PCIe (۱/۱) سرعت انتقال داده در هر خط بالغ بر ۲۵۰ مگابایت در ثانیه در هر جهت است که این سرعت در نسخه ۲ (سال ۲۰۰۷) دو برابر شده است و در نسخه ۳ که تا سال ۲۰۱۰ عرضه خواهد شد افزایش خواهد یافت. بنابراین سرعت گذرگاه PCIe-X32 برابر ۸ گیگابایت در ثانیه (۲۵۰ مگابایت در ثانیه × ۳۲ خط) است. سرعت گذرگاه PCIe-X8 از سریعترین نسخه گذرگاه AGP بیشتر است.

شکل زیر، دو گذرگاه PCIe-X1 و PCIe-X16 را نشان می‌دهد.

PCI Express x16 slot
PCI Express x1 slot

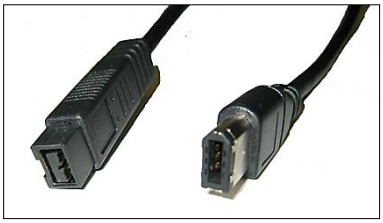
[۱] البته این پنهای باند در عمل به خاطر «سکوت» سیگنال تصویر در مواقع برگشت افقی و عمودی، کمتر است.

گذرگاه PCMCIA یا PC-CARD: این گذرگاه ویژه اتصال تجهیزات جانبی مانند حافظه، کارت شبکه، کارت مودم، کارت صوتی، کارت TV و ... به کامپیوترهای همراه[1] است. ویژگیهای آن عبارتند از:

- ✓ پهنای گذرگاه ۳۲ بیتی با فرکانس ۳۳ مگاهرتز
- ✓ ولتاژ کاری و توان مصرفی پایین
- ✓ پشتیبانی از تکنولوژیهای DMA، APM[2]، PnP، ATA
- ✓ واسط ZV[3] که به کارت اجازه می‌دهد اطلاعات ویدیو را مستقیماً به کنترل‌کننده VGA ارسال کند و معطل بافر شدن انتقال از طریق گذرگاه سیستم نشود.

گذرگاه USB: معروفترین گذرگاه کامپیوتری که حتماً با آن آشنایی دارید، گذرگاه USB است. این گذرگاه خارجی بستری مناسب برای اتصال طیف وسیعی از دستگاههای خارجی (چاپگر، اسکنر، حافظه FLASH، MP3 Player و ...) را فراهم می‌آورد. ویژگی جالب گذرگاه USB این است که با اتصال دستگاه به این پورت، نیازی به تنظیم آدرس یا ایجاد تغییراتی در سیستم یا حتی روشن و خاموش کردن سیستم نیست و تمام عملیات پیکربندی بلافاصله توسط سیستم انجام می‌شود (این ویژگی به اصطلاح Hot PnP نامیده می‌شود). ویژگی دیگر پورت USB این است که مبدلهای آن برای تبدیل به گذرگاههای دیگر در بازار موجود هستند. مبدلهای USB به موازی، USB به سریال، USB به SCSI، USB به Ethernet، USB به ماوس و صفحه‌کلید، USB به ویدیو و تلویزیون و ... به شما این امکان را می‌دهند از پورت USB برای طراحی هر واسطی استفاده کنید.

نسخه ۱/۱ گذرگاه USB که در سال ۱۹۹۸ عرضه شد دارای سرعتهای ۲/۵ و ۱۲ مگابایت است. کابلهای با کیفیت بالاتر (شیلددار) سرعت بالاتر را حمایت می‌کنند و معمولاً برای اتصال چاپگر و اسکنر و ... به کار می‌روند. کابلهای ارزانتر که کیفیت پایینتری دارند برای اتصال صفحه‌کلید و ماوس استفاده می‌شوند که نیاز به سرعت بالایی ندارد. نسخه ۲ USBکه در سال ۲۰۰۲ به بازار آمد، با سرعتی بالاتر یعنی ۶۰ مگابایت در ثانیه کار می‌کند. بستر گذرگاه USB، گذرگاه PCI است؛ یعنی در هرم تجهیزات سیستمی، گذرگاه USB زیرمجموعه گذرگاه PCI است.



گذرگاه IEEE1394 (Firewire یا I.Link): این گذرگاه معمولاً برای انتقال صوت و تصویر و نیز اتصال دیسکهای سخت خارجی به کار می‌رود و اکثر دوربینهای دیجیتال فعلی از آن حمایت می‌کنند. نسخه‌های مختلف آن در سرعتهای ۱۰۰ و ۲۰۰ و ۴۰۰ و ۸۰۰ مگابیت در ثانیه ارائه شده‌اند و در نیمه اول سال ۲۰۰۸، نسخه 1394c با سرعت ۱۶۰۰ مگابیت در ثانیه عرضه شده است.

[1] Laptop
[2] Advanced Power Management
[3] Zoomed Video

منابع این فصل

[1] "The 80x86 IBM PC & Compatible Computers", Muhammed Ali Mazidi and et al, Prentice Hall,2000.

[2] "PC Interfaces under Windows", Burkhard Kaink and Hans-Joachim Berndt, Elektor Electronics Publications, 2002.

(این کتاب با ترجمه کیهان حدادشرق توسط انتشارات نشر علوم به چاپ رسیده است)

[۳] "مرجع علمی-کاربردی سخت‌افزار"، شیرزاد شهریاری، انتشارات جهاد دانشگاهی مشهد، ۱۳۸۴.

[۴] "اسلاتهای توسعه و طراحی کارتها"، شیرزاد شهریاری، انتشارات جهاد دانشگاهی مشهد، ۱۳۷۸.

[۵] "گذرگاهها و درگاههای کامپیوترهای شخصی"، عبدالمجید منصوریان‌فر و اصغر کریمی، انتشارات دانش‌پژوهان برین - ارکان، ۱۳۸۳

[6] ISA & EISA Theory and Operation, by Edward Solari. (Annabooks) (ISBN 0-929392-15-9)

[7] ISA System Architecture, by Don Anderson and Tom Shanley (MindShare) (ISBN 0-201-40996-8)

فصل پنجم

پورت‌های ارتباطی کامپیوتر

در بخش قبل با نحوه طراحی کارتهای کامپیوتری آشنا شدیم. امروزه بسیاری از وسایل ورودی/خروجی مانند کارت صوتی و گرافیکی (حتی مودم و کارت شبکه) روی بورد اصلی کامپیوتر جاسازی شده‌اند[۱] و شکافهای بورد نیز برای اضافه کردن کارتهای دیگر تعبیه شده‌اند. اما رفته‌رفته پورتهای خارجی کامپیوتر، جای کارتها را برای اتصال ورودی/خروجی به کامپیوتر می‌گیرند. در بازار امروز تجهیزات بسیاری از جمله کارت حافظه، چاپگر، CD-ROM خارجی[۲]، اسکنر، کارت TV، دیسک سخت و ... از طریق پورت USB به کامپیوتر متصل می‌شوند. مزیت مهم این اتصال این است که نیازی به باز کردن درب جعبه کامپیوتر نیست و به راحتی می‌توان تجهیزات ورودی/خروجی را به کامپیوتر متصل و نیز قطع نمود. به علاوه اگر در طراحی کارت برای انجام یک عملیات ورودی/خروجی اشتباهی رخ دهد، با قرار دادن کارت در شکاف بورد اصلی کمترین حاصل روشن نشدن کامپیوتر و بدترین نتیجه سوختن بورد اصلی کامپیوتر خواهد بود. در عوض سرعت تعامل با کامپیوتر از طریق شکاف‌ها بیشتر است.

در این نوشتار دو پورت موازی[۳] و سریال[۴] کامپیوتر را معرفی و نحوه اتصال تجهیزات به آنها را بررسی خواهیم کرد. چاپگرهای قدیمی‌تر (و حتی بعضی انواع جدید) به پورت موازی کامپیوتر متصل می‌شوند. به همین لحاظ پورت موازی به پورت چاپگر نیز معروف است. پورت سریال نیز قبلاً برای اتصال اسکنر و مودمهای خارجی[۵] مورد استفاده قرار می‌گرفت. با مطالبی که در این بخش می‌آموزید خواهید توانست به راحتی دستگاههایی که خودتان می‌سازید (مانند دستگاههای جمع‌آوری اطلاعات یا روباتها) را با این دو پورت متداول کامپیوتر کنترل کنید. به علاوه از این دو پورت برای ارتباط اطلاعاتی بین دو کامپیوتر نیز استفاده می‌شود. البته پورت USB رفته‌رفته جای دیگر پورت‌ها را در کامپیوترهای شخصی می‌گیرد. اما گذشته از آنکه اصول ارتباط USB مبتنی بر ارتباط سریال است، پورت‌های موازی و سریال در بسیاری از دستگاهها و مادربوردهای صنعتی مورد استفاده قرار می‌گیرند. مطالبی که در ادامه می‌آموزید به شما کمک می‌کند حتی وقتی کامپیوتری در بین نباشد بتوانید ارتباط بین سیستم‌های مبتنی بر پردازنده را برقرار کنید.

پیش از بررسی جزیی ویژگیهای این دو پورت، بیان مطالبی راجع به ارتباطات کامپیوتری و طبقه‌بندی آنها ضروری به نظر می‌رسد. این مطالب در مورد ارتباط بین هر دو سیستم مبتنی بر پردازنده نیز صدق می‌کند.

دسته‌بندی ارتباطات کامپیوتری

برای بررسی و مقایسه ویژگیهای انواع ارتباطات کامپیوتری، به دسته‌بندی آنها از دیدگاههای مختلف می‌پردازیم:

- ✓ از دید حجم سیمهای رابط
- ✓ از دید شیوه کنترل جریان داده
- ✓ از دید جهت انتقال اطلاعات

---

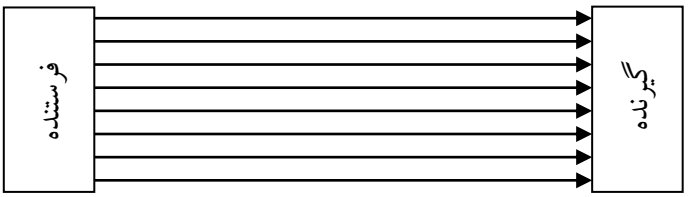
[۱] به اصطلاح on-board هستند.

[2] External
[3] Parallel
[4] Serial
[5] External

ارتباطات کامپیوتری از دید حجم سیمهای رابط

از این دیدگاه، ارتباطات کامپیوتری به دو دسته ارتباط موازی و ارتباط سریال تقسیم می‌شوند.

فرض کنید می‌خواهیم یک داده ۸ بیتی را از یک سیستم به سیستم دیگری انتقال دهیم. هر بیت، یک ولتاژ دیجیتال است که باید روی سیم انتقال داده شود. یک راه برای انتقال این داده ۸ بیتی این است که ۸ سیم از سمت فرستنده به گیرنده برقرار شود و هر بیت داده روی یک سیم منتقل شود[1]. شکل ۵-۱ این نوع ارتباط را نشان می‌دهد:

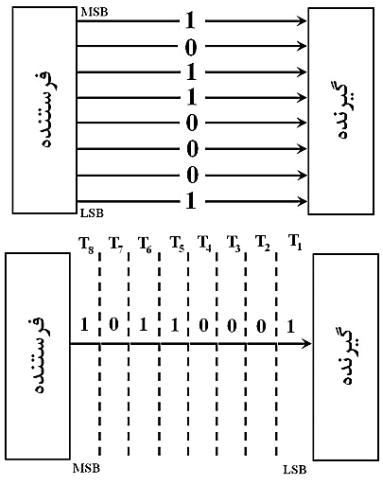


شکل ۵-۱-ارتباط موازی

چون هر ۸ بیت موازی با هم از فرستنده ارسال و در گیرنده دریافت می‌شوند، این نوع ارتباط به **ارتباط موازی** معروف است. هنگامی که از یکی از پورتهای ورودی/خروجی یک میکروکنترلر اطلاعاتی را ارسال یا دریافت می‌کنید و یا هنگام استفاده از پورت چاپگر کامپیوتر، از ارتباط موازی استفاده می‌کنید.

راه دیگر برای برقراری ارتباط بین فرستنده و گیرنده این است که به جای استفاده از ۸ سیم و ارسال هر بیت داده روی یک سیم، تنها از یک سیم بین فرستنده و گیرنده برای انتقال داده‌ها استفاده کنیم. واضح است یک سیم در هر لحظه تنها می‌تواند یک ولتاژ (یک بیت داده) را روی خود نگه دارد؛ بنابراین فرستنده باید ۸ بیت داده را یکی یکی و به صورت متوالی (سریال) روی سیم انتقال داده‌ها قرار دهد و گیرنده نیز بیتها را یکی یکی از رسانه انتقال داده‌ها بخواند و بایت انتقالی را بسازد. این ارتباط به **سریال** معروف است. پورت COM و پورت USB کامپیوتر و نیز شبکه‌های کامپیوتری از این نوع ارتباط استفاده می‌کنند. اکثر میکروکنترلرها امکانات ارتباط سریال دارند.

برای مثال فرض کنید می‌خواهیم بایت 10110001 را از فرستنده به گیرنده انتقال دهیم. شکل ۵-۲ این ارسال را در دو حالت موازی و سریال نشان می‌دهد.



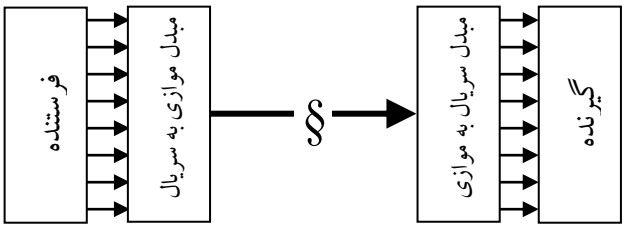
شکل ۵-۲-مقایسه ارتباط موازی با سریال

[1] البته واضح است که دست‌کم یک سیم زمین به عنوان ولتاژ مرجع مورد نیاز است. به طوری که خواهیم دید، در پورت موازی کامپیوتر به ازای هر سیم داده یک سیم زمین وجود دارد.

همانطور که می‌بینید در ارتباط سریال ابتدا کم‌ارزش‌ترین بیت ارسال می‌شود. هر بیت مدت زمانی روی خط می‌ماند تا گیرنده فرصت خواندن آن را داشته باشد. معمولاً زمانهای T_1 تا T_8 با هم مساوی هستند.

با مشخصاتی که گفته شد، مقایسه‌ای بین ارتباط موازی و سریال مفید است:

- ✓ حجم سیمهای ارتباطی در ارتباط موازی بیشتر از ارتباط سریال است. این موضوع از طرفی باعث بالا رفتن هزینه سیم‌کشی می‌شود و از طرف دیگر باعث می‌شود سیمهای ارتباطی روی یکدیگر ایجاد نویز کنند. به همین لحاظ، فاصله دو سیستم با ارتباط موازی حداکثر می‌تواند تا ۵ متر باشد؛ اما طول ارتباط سریال می‌تواند در محیط‌های کم‌نویز تا ۱۵ متر باشد.
- ✓ چون در ارتباط موازی هر ۸ بیت داده به صورت موازی ارسال می‌شوند، سرعت آن بیشتر از ارتباط سریال است که در آن، بیتها یکی یکی فرستاده می‌شوند. سرعت ارتباط موازی می‌تواند تا ۱۰۰ کیلوبایت در ثانیه نیز باشد، در حالی که سرعت ارتباط سریال حداکثر ۲۵۶۰۰۰ بیت در ثانیه (۳۲ کیلوبایت در ثانیه) است.
- ✓ وقتی در ارتباط سریال سیم رابط قطع شود، کل ارتباط قطع می‌شود؛ اما در ارتباط موازی با قطع یک سیم، فقط یکی از بیتها مخدوش می‌شود.
- ✓ اگر در طول ارتباط، نویز شدید و کوتاه‌مدتی رخ دهد، بیتهای ارسالی سریال را خراب می‌کند. اما در ارتباط موازی چون هر ۸ بیت اطلاعات مدت زمانی روی خطوط ارتباطی می‌مانند، امکان خراب نشدن اطلاعات بیشتر است.
- ✓ ارتباط موازی ساده است؛ چون به ازای هر بیت داده یک سیم وجود دارد. اما در ارتباط سریال چون تنها یک سیم برای انتقال داده موجود است، در فرستنده باید یک «مبدل موازی به سریال» و در گیرنده یک «مبدل سریال به موازی» باید وجود داشته باشد تا هماهنگی ارسال و دریافت اطلاعات را انجام دهند:



شکل ۵-۳-ارتباط سریال

وظیفه مبدل موازی به سریال، دریافت اطلاعات به صورت موازی از فرستنده و گذاشتن (به اصطلاح «سوار کردن») اطلاعات روی خط انتقال است. مبدل سریال به موازی، اطلاعات را به اصطلاح از روی خط «پیاده کرده» و به صورت موازی در اختیار گیرنده قرار می‌دهد. سیستمهای آماده‌ای وجود دارند که عمل سوار کردن (مدولاسیون-MOdulation) و پیاده کردن (دمدولاسیون-DEModulation) اطلاعات را انجام می‌دهند که به اصطلاح مودم (MODEM) نامیده می‌شوند[1]. حتماً از مودم خط تلفن[1] یا مودم ADSL برای اتصال به اینترنت استفاده کرده‌اید. این

[1] این موضوع یک دیدگاه کلی است. از دید مخابراتی، اگر بیتها به صورت دو ولتاژ مختلف روی رسانه انتقال قرار داده شوند، به مدولاسیون نیازی نیست. بلکه در حالتی که سیگنالهای دیجیتال باید باید به شکل دیگری از سیگنال (معمولاً آنالوگ) تبدیل شوند تا قابل حمل روی رسانه باشند (مثلاً وقتی

کارتها، رده خاصی ار مودمها هستند که از خط تلفن برای انتقال اطلاعات به صورت سریال بین کامپیوتر شما و شرکت سرویس‌دهنده اینترنت[۲] استفاده می‌کنند. در این مثال فاصله بین فرستنده و گیرنده دو سر ارتباط می‌تواند میلیونها کیلومتر باشد.

بنا به مقایسه‌ای که انجام شد، برای برقراری ارتباط بین دو سیستم می‌توان از هرکدام از ارتباطات موازی یا سریال استفاده کرد.

دسته‌بندی ارتباطات کامپیوتری از دید شیوه کنترل جریان داده[۳]

فرض کنید سیستم فرستنده می‌خواهد ۱۰۰۰ قلم اطلاعات را برای سیستم گیرنده ارسال کند. باید مکانیسمی بین فرستنده و گیرنده پیش‌بینی شود تا جریان داده بین آنها کنترل‌شده باشد. به بیان دیگر، فرستنده باید پس از ارسال هر قلم اطلاعات، مطمئن شود که گیرنده آن را دریافت و ذخیره (و در بعضی مواقع پردازش) کرده و بعد قلم بعدی اطلاعات را ارسال کند. به اصطلاح فرستنده نباید گیرنده را در اطلاعات غرق[۴] کند.

برای کنترل جریان داده دو روش **تأخیر**[۵] و **دست‌دهی**[۶] وجود دارد. برای بررسی این دو روش بیان یک مثال مفید است.

فرض کنید شما می‌خواهید برای دوستتان اسم ده نفر را بخوانید تا او بنویسد. طبیعی است که نمی‌توان اسم ده نفر را پشت سرهم بخوانید؛ وگرنه دوستتان نمی‌تواند آنها را بنویسد. یک راه ساده این است که بعد از خواندن هر اسم، مدت زمان ثابتی (مثلاً ۳۰ ثانیه) صبر کنید و سپس اسم بعدی را بخوانید. این روش ساده است، اما ممکن است منجر به ایجاد مشکل شود؛ مثلاً سرعت نوشتن دوستتان زیاد باشد و زمانی که بین خواندن دو اسم صرف می‌کنید، به هدر رود. از طرف دیگر، ممکن است در حین نوشتن مشکلی برای دوستتان پیش بیاید (مثلاً نوک مدادش بشکند!) و نتواند در مدت زمان ثابتی که بین خواندن دو اسم صبر می‌کنید، اسم قبلی را بنویسد. به صورت خلاصه اشکال روش ایجاد تأخیر ثابت بین ارسال دو قلم اطلاعات این است که ممکن است این زمان زیاد باشد که باعث اتلاف وقت شود یا کم باشد که گیرنده نتواند اطلاعات را به درستی ثبت کند.

روش دیگر این است که یک اسم را بخوانید، صبر کنید تا دوستتان به شما بگوید که اسم قبلی را نوشته و سپس اسم بعدی را بخوانید. زحمت این روش کمی بیشتر از روش قبل است؛ اما در عوض زمان به هدر نمی‌رود و در صورت بروز مشکل در ثبت اطلاعات، فرستنده قبل از تصدیق دریافت اطلاعات قبلی توسط گیرنده اطلاعات بعدی را ارسال نخواهد کرد. به این روش به اصطلاح دست‌دهی[۷] گفته می‌شود.

به صورت خلاصه برای کنترل جریان داده دو روش وجود دارد:

مودم کامپیوتر «صفر» و «یک»ها را به سیگنالهای قابل حمل به کمک حامل‌های روی خط تلفن تبدیل می‌کند)، به مدولاسیون و در مقصد به دمدولاسیون نیاز است.

[1] Dial-up Modem
[2] Internet Service Proider: ISP
[3] Flow Control
[4] Overwhelm
[5] Delay
[6] Hand-Shaking

[۷] اصطلاح دست‌دهی معادل فارسی Hand Shaking است که در زبان انگلیسی به معنای دست دادن (و به نوعی توافق و تفاهم کردن) می‌باشد.

روش تأخیر: در این روش، فرستنده پس از ارسال هر قلم اطلاعات، مدت زمان ثابتی (مثلاً T) تأخیر ایجاد کرده و سپس داده بعدی را ارسال می‌کند. گیرنده نیز در فواصل زمانی T خطوط ارتباطی را خوانده و اطلاعات را ثبت می‌کند. این روش ساده است؛ اما ثابت بودن زمان تأخیر ایجاد مشکل می‌کند. ممکن است این زمان زیاد باشد که باعث اتلاف وقت شود یا کم باشد که گیرنده نتواند اطلاعات را به درستی ثبت کند.

روش دست‌دهی: در این روش، فرستنده پس از ارسال هر قلم اطلاعات، صبر می‌کند تا گیرنده دریافت آن را تصدیق کند و سپس داده بعدی را می‌فرستد. در این روش علاوه بر داده‌ها، سیگنالهای کنترلی هم باید رد و بدل شوند و به همین خاطر ماهیتی پیچیده‌تر از روش قبل دارد؛ اما در عوض زمان به هدر نمی‌رود و در صورت بروز مشکل در ثبت اطلاعات، فرستنده قبل از تصدیق دریافت اطلاعات قبلی توسط گیرنده اطلاعات بعدی را ارسال نخواهد کرد.

مراحل ارسال هر قلم اطلاعات به این روش توسط فرستنده عبارتند از:

۱) ارسال داده

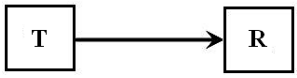
۲) آگاه کردن گیرنده از ارسال داده

۳) انتظار برای تصدیق دریافت داده توسط گیرنده

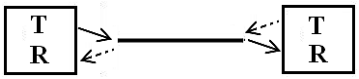
توجه کنید دسته‌بندی ارتباطات از دید شیوه کنترل جریان داده، ارتباطی با دسته‌بندی از دید حجم سیمهای ارتباطی ندارد؛ یک ارتباط موازی (یا سریال) می‌تواند از هرکدام از دو شیوه تأخیر یا دست‌دهی برای کنترل جریان داده استفاده کند.

دسته‌بندی ارتباطات کامپیوتری از دید جهت انتقال داده‌ها

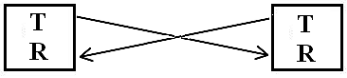
از این دیدگاه، سه نوع ارتباط وجود دارد:



ارتباط یک‌طرفه[1]**:** در این نوع ارتباط یک فرستنده[2] و یک گیرنده[3] داریم و جهت انتقال داده‌ها همواره از فرستنده به گیرنده است.



ارتباط نیمه دوطرفه[4]**:** در این نوع ارتباط هرکدام از طرفین ارتباط می‌توانند فرستنده یا گیرنده باشند؛ اما چون تنها یک کانال داده بین آنها وجود دارد، در هر لحظه یکی از دو طرف فرستنده و دیگری گیرنده است. به بیان دیگر در هر لحظه یکی از طرفین فرستنده و دیگری گیرنده است.



ارتباط کاملاً دوطرفه[5]**:** در این نوع ارتباط نیز هرکدام از طرفین ارتباط می‌توانند فرستنده یا گیرنده باشند. اما دو کانال مجزا بین فرستنده و گیرنده وجود دارد و هر دو طرف می‌توانند همزمان هم فرستنده و هم گیرنده باشند.

---

[1] Simplex
[2] Transmitter
[3] Receiver
[4] Half-Duplex
[5] Full-Duplex

مانند قبل، دسته‌بندی ارتباطات از این دیدگاه، ارتباطی با دسته‌بندی‌های قبلی ندارد. به عنوان مثال یک ارتباط یک‌طرفه می‌تواند برای کنترل جریان داده از روش تأخیری یا دست‌دهی استفاده کند. توجه کنید تصدیق دریافت داده قبلی توسط گیرنده که در روش دست‌دهی استفاده می‌شود، سیگنالی از سمت گیرنده به فرستنده است؛ اما منظور از ارتباط یکطرفه، انتقال یک‌طرفه داده‌هاست و سیگنالهای کنترلی می‌تواند از سمت گیرنده به سمت فرستنده باشد.

تشخیص خطا[1] و تصحیح خطا[2]

در حین یک ارتباط ممکن است به علت وجود نویز، قطع و وصل رسانه انتقال داده و ... خطایی در ارتباط رخ دهد و اطلاعاتی که فرستنده ارسال می‌کند به درستی در گیرنده دریافت نشود. به همین لحاظ معمولاً در ارتباطات کامپیوتری مکانیسمی برای تشخیص و در صورت امکان تصحیح خطا در نظر گرفته می‌شود.

روش اساسی برای تشخیص خطا این است که در فرستنده، محاسباتی روی داده‌هایی که قرار است ارسال شود انجام شده و نتیجه این محاسبات به عنوان اطلاعات کشف خطا به همراه داده‌ها ارسال می‌شود. در گیرنده همین محاسبات روی داده‌های دریافتی انجام شده و نتیجه آن با نتیجه‌ای که از فرستنده دریافت شده (اطلاعات کشف خطا) مقایسه می‌شود. اگر دو نتیجه یکسان نباشند، نشان‌دهنده این است که در طول ارتباط خطایی رخ داده است. اما نکته مهم این است که اگر نتیجه‌ها یکسان باشند نمی‌توان با قاطعیت گفت خطایی رخ نداده است!

به عنوان مثال فرض کنید قرار است ۵ عدد از فرستنده به گیرنده ارسال شود. فرستنده برای کشف خطا، اعداد فوق را با هم جمع کرده و رقم یکان مجموع را به عنوان اطلاعات کشف خطا ارسال می‌کند. در گیرنده نیز اعداد دریافتی با هم جمع می‌شوند و رقم یکان مجموع با اطلاعات کشف خطا مقایسه می‌شود تا وقوع خطا بررسی شود. بسته اطلاعات زیر را در نظر بگیرید:

داده‌های ارسالی					اطلاعات کشف خطا
۱۰	۲۴	۳۳	۱۸	۶	۱

فرستنده داده‌های ارسالی را با هم جمع می‌کند (۹۱ = ۱۰ + ۲۴ + ۳۳ + ۱۸ + ۶) و رقم یکان آن را به عنوان اطلاعات کشف خطا به همراه داده‌ها ارسال می‌کند.

در گیرنده داده‌های دریافتی با هم جمع می‌شوند و اگر رقم یکان مجموع «۱» شود، گیرنده به این نتیجه می‌رسد که داده‌ها به درستی دریافت شده‌اند.

حال فرض کنید خطایی رخ دهد و به جای عدد ۳۳ عدد ۳۵ دریافت شود. گیرنده اعداد فوق را با هم جمع می‌کند (۹۳ = ۱۰ + ۲۴ + ۳۵ + ۱۸ + ۶) و با مقایسه رقم یکان مجموع «۳» با اطلاعات کشف خطا (۱) متوجه وقوع خطا می‌شود.

اما آیا ممکن است خطایی رخ دهد و گیرنده متوجه وقوع خطا نشود؟! فرض کنید خطایی رخ دهد و به جای عدد ۳۳ عدد ۳۵ و به جای عدد ۶ عدد ۴ دریافت شود. گیرنده اعداد فوق را با هم جمع می‌کند (۹۱ = ۱۰ + ۲۴ + ۳۵ + ۱۸ + ۴) و با مقایسه رقم یکان مجموع «۱» با اطلاعات کشف خطا (۱) به این نتیجه نادرست می‌رسد که خطایی رخ نداده است! در حالی که نه تنها خطا رخ داده، بلکه دو قلم اطلاعات را نیز خراب کرده است.

---

[1] Error Detection
[2] Error Correction

روشهای معمول کشف خطا در ارتباطات کامپیوتری از همین شیوه پیروی می‌کنند. در روش ارسال بیت توازن[1]، بر مبنای تعداد «یک»ها در یک بسته داده عمل می‌شود. روش بیت توازن که از یک بیت به عنوان اطلاعات کشف خطا استفاده می‌کند، به دو صورت توازن زوج[2] و توازن فرد[3] عمل می‌کند. در روش توازن زوج، فرستنده تعداد «یک»های بسته داده را می‌شمارد و در صورتی که تعداد آنها زوج باشد بیت توازن را «صفر» و اگر فرد باشد بیت توازن را «یک» می‌کند. به بیان دیگر مجموع تعداد «یک»های بسته داده و بیت توازن باید زوج باشد. در روش توازن فرد مجموع تعداد «یک»های بسته داده و بیت توازن باید فرد باشد. مثالهای زیر بسته‌های ۸ بیتی به همراه بیت توازن زوج را نشان می‌دهند.

بسته داده								بیت توازن زوج
۰	۱	۱	۰	۰	۰	۱	۱	۰

بسته داده								بیت توازن زوج
۱	۱	۰	۱	۱	۰	۰	۱	۱

اگر در جریان انتقال داده‌ها خطایی رخ دهد و یک بیت «یک» به «صفر» یا یک بیت «صفر» به «یک» تبدیل شود، گیرنده با ارزیابی بیت توازن متوجه وقوع این خطا می‌شود. مثلاً فرض کنید بسته داده دومی به صورت زیر دریافت شود:

بسته داده								بیت توازن زوج
۱	۱	۱	۰	①	۱	۱	۰	۱

گیرنده با شمارش تعداد «یک»های بسته داده و بیت توازن به عدد ۷ می‌رسد و چون از قبل مکانیسم توازن زوج بین فرستنده و گیرنده توافق شده و ۷ یک عدد فرد است، گیرنده متوجه وقوع خطا می‌شود.

واضح است که اگر تعداد بیتهای خطا زوج باشد (دو بیت یا چهار بیت یا شش بیت داده دچار خطا شود)، با مکانیسم توازن (چه توازن زوج و چه توازن فرد) نمی‌توان به آن پی برد. چون تعداد «یک»ها تغییری نمی‌کند.

در شبکه‌های TCP/IP از مکانیسمی به نام CRC[4] برای کشف خطا استفاده می‌شود. در این روش، فرستنده اطلاعات بسته داده که شامل «صفر» و «یک» است را بر یک چندجمله‌ای ویژه تقسیم می‌کند و باقیمانده آن (موسوم به CRC) را به همراه بسته داده می‌فرستد. گیرنده بسته دریافتی را بر همان چندجمله‌ای تقسیم می‌کند و باقیمانده را با CRC دریافتی مقایسه می‌کند. تحقیقات فراوانی برای انتخاب چندجمله‌ای مزبور انجام شده تا امکان وقوع خطا و کشف نشدن آن توسط مکانیسم CRC (مانند آنچه در مورد بیت توازن دیدیم) را به حداقل برساند.

حال فرض کنید وقوع خطا کشف شده است. اکنون چه باید کرد؟!

می‌توان با تغییراتی در روش کشف خطا، آن را به روش تصحیح خطا تبدیل کرد؛ به گونه‌ای که گیرنده بتواند به کمک این اطلاعات محل وقوع بیت خطا را تشخیص دهد. تعداد بیتهایی که فرستنده برای تصحیح خطا باید به همراه بسته داده ارسال کند، از تعداد بیتهای کشف خطا بیشتر است و قسمت زیادی از پهنای باند را هدر می‌دهد؛ به همین لحاظ

---

[1] Parity
[2] Even Parity
[3] Odd Parity
[4] Cyclic Redundancy Check

معمولاً از روش تصحیح خطا استفاده نمی‌شود. شیوه معمول این است که گیرنده با کشف خطا، درخواست ارسال مجدد بسته[1] را می‌نماید که از دید آماری نسبت به روش تصحیح خطا، پهنای باند کمتری صرف می‌کند.

پروتکل[2]

به مجموعه قراردادهای مابین فرستنده و گیرنده برای یک ارتباط صحیح، پروتکل گفته می‌شود. در پروتکل باید موارد زیر مشخص شود:

- ✓ نحوه آغاز و پایان ارتباط
- ✓ سرعت ارتباط
- ✓ شیوه کنترل جریان داده
- ✓ نوع ارتباط از دید حجم سیمهای ارتباطی (موازی یا سریال)
- ✓ جهت انتقال داده‌ها
- ✓ نحوه تشخیص و تصحیح خطا
- ✓ ...

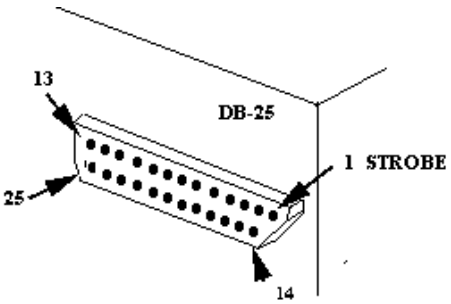
[1] Retransmission
[2] Protocol

فصل ششم
پروتکل ارتباط موازی کامپیوتر

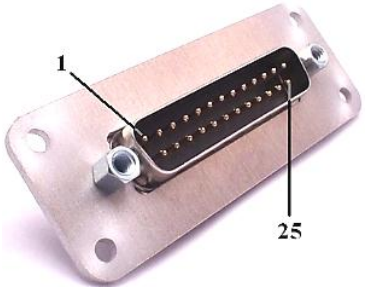
اگر به پشت بورد اصلی کامپیوتر خود نگاه کنید، یک پورت با ۲۵ پایه مانند شکل مقابل می‌بینید که به پورت موازی، پورت چاپگر، پورت LPT[1] یا در اصطلاح فنی الکترونیک به پورت DB25 معروف است.

در این بخش به بررسی سیگنالهای پورت موازی کامپیوتر می‌پردازیم. با شناخت این سیگنالها دو مطلب را خواهیم آموخت؛ اول چگونگی کنترل یک چاپگر بدون نیاز به کامپیوتر و دوم نحوه کنترل یک دستگاه جانبی به کمک پورت موازی کامپیوتر.

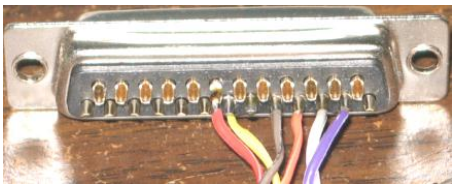
پیش از توضیح عملکرد پایه‌ها، توضیحی راجع به نحوه شماره‌گذاری پایه‌ها لازم است. پورتی که پشت کامپیوتر مشاهده می‌کنید، دارای ۲۵ **حفره** است که سیگنالها را در اختیار کاربر قرار می‌دهد[2]. شماره‌گذاری این حفره‌ها به صورت مقابل است:



معمولاً برای استفاده از این پورت، قطعه‌ای به کار می‌رود که به جای حفره دارای **میله** است و میله‌ها درون حفره‌ها قرار می‌گیرند[3]. این قطعه نیز به نام DB25 معروف است. شکل بعدی را ببینید. سر کابل چاپگر که درون پورت موازی قرار می‌گیرد نیز چنین شکلی دارد.



دقت کنید که در پورت اصلی کامپیوتر، پایه بالا سمت راست و در قطعه روبرو که درون پورت قرار می‌گیرد، پایه بالا سمت چپ شماره یک است. چون وقتی قطعه بالا درون پورت قرار می‌گیرد، سمت چپ آن با سمت راست پورت مقابل هم قرار می‌گیرد. پشت این قطعه در محل هر میله، جایی برای لحیم کردن سیم تعبیه شده تا به راحتی بتوانید از سیگنالهای پورت موازی استفاده کنید. شکل زیر را ببینید:



جدول صفحه بعد سیگنالهای پورت موازی را نشان می‌دهد:

[1] Line Printer

[2] این نوع پورت معمولاً به Socket یا Female معروف است.

[3] این نوع پورت معمولاً به Plug یا Male معروف است.

شماره پایه	عملکرد	جهت سیگنال (از دید چاپگر)	نوع سیگنال	شماره پایه	عملکرد	جهت سیگنال (از دید چاپگر)	نوع سیگنال
۱	-STROBE	ورودی	کنترل	۱۰	-ACK	خروجی	وضعیت
۲	+DATA BIT 0	ورودی/ خروجی	داده	۱۱	+BUSY	خروجی	وضعیت
۳	+DATA BIT 1	ورودی/ خروجی	داده	۱۲	+PE	خروجی	وضعیت
۴	+DATA BIT 2	ورودی/ خروجی	داده	۱۳	+SLCT OUT	خروجی	وضعیت
۵	+DATA BIT 3	ورودی/ خروجی	داده	۱۴	-AUTO FEED	ورودی	کنترل
۶	+DATA BIT 4	ورودی/ خروجی	داده	۱۵	-ERROR	خروجی	وضعیت
۷	+DATA BIT 5	ورودی/ خروجی	داده	۱۶	-INIT	ورودی	کنترل
۸	+DATA BIT 6	ورودی/ خروجی	داده	۱۷	-SLCT IN	ورودی	کنترل
۹	+DATA BIT 7	ورودی/ خروجی	داده	۱۸-۲۵	GND	-	داده

چون پورت موازی از ابتدا برای اتصال چاپگر به کامپیوتر استاندارد شد، سیگنالهای این پورت نیز با نامهایی که در ارتباط کامپیوتر و چاپگر به کار می‌روند نامگذاری شده‌اند.

سه نوع سیگنال در پورت موازی کامپیوتر وجود دارد:

سیگنالهای داده[1] که برای تبادل اطلاعات استفاده می‌شوند و در پورتهای موازی جدید[2] می‌توانند هم ورودی و هم خروجی باشند.

سیگنالهای وضعیت[3] که برای اطلاع دادن وضعیت چاپگر به کامپیوتر به کار می‌رود. همانطور که در جدول می‌بینید، جهت تمام سیگنالهای وضعیت، خروجی از چاپگر و ورودی به کامپیوتر است.

سیگنالهای کنترل[4] که برای کنترل چاپگر توسط کامپیوتر به کار می‌رود. همانطور که در جدول می‌بینید، جهت تمام سیگنالهای کنترل، ورودی به چاپگر و خروجی از کامپیوتر است.

هنگامی که کامپیوتر به کمک یک چاپگر می‌خواهد یک جمله را چاپ کند، باید کاراکترهای آن جمله را یکی‌یکی و با کنترل جریان داده برای چاپگر ارسال کند تا چاپگر فرصت چاپ هر کاراکتر را داشته باشد. در ارتباط کامپیوتر باچاپگر، از روش دست‌دهی برای کنترل جریان داده استفاده می‌کند. برای ارسال داده‌ها به روش دست‌دهی طبق روندنمای صفحه بعد باید عمل شود.

برای شرح سیگنالهای پورت موازی، ابتدا به سیگنالهایی می‌پردازیم که در این رابطه دست‌دهی شرکت دارند.

سیگنالهای داده: هشت پایه (از شماره ۲ تا شماره ۹) برای ارسال کد اسکی کاراکترها به چاپگر به کار می‌روند. پایه‌های ۱۸ تا ۲۵ نیز هشت پایه زمین هستند. هرچند استفاده از یکی از این پایه‌های زمین نیز به عنوان ولتاژ مرجع کافی است؛ اما اگر از سیمهای داده و زمین در کابل انتقال داده به صورت یکی در میان استفاده شود، در کاهش نویز تأثیر به سزایی دارد و به این صورت می‌توان فاصله کامپیوتر و چاپگر را افزایش داد. به علاوه استفاده از یک سیم زمین به ازای هر سیم داده، نویز ناشی از تداخل جریانهای برگشتی را کاهش می‌دهد.

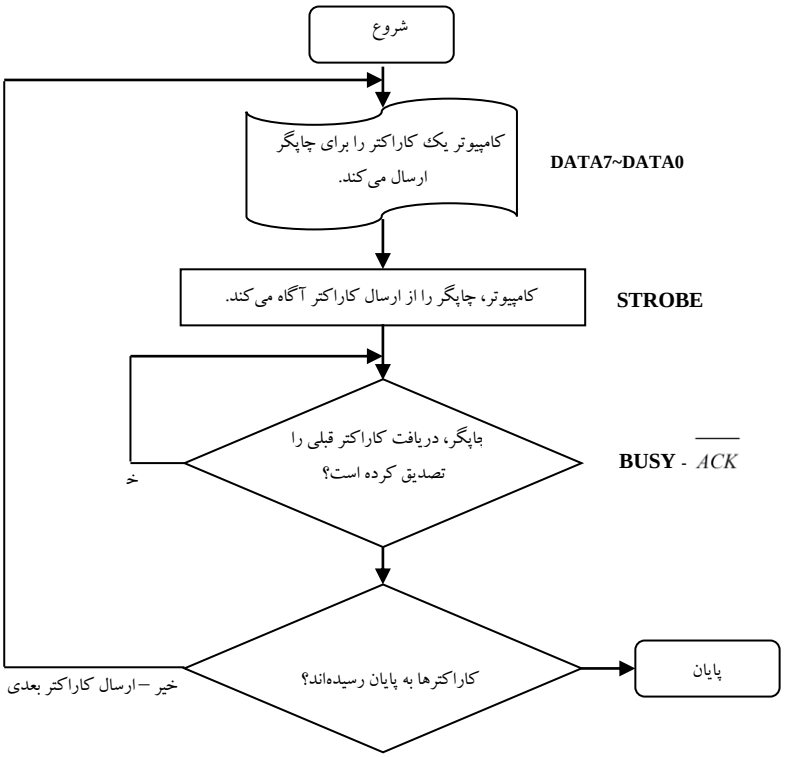
---

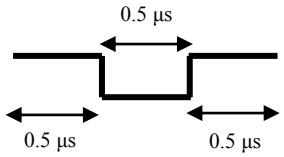
[1] Data

[2] با استانداردهای EPP و ECP که بعداً خواهیم دید.

[3] Status

[4] Control





سیگنال $\overline{\text{STROBE}}$: یک سیگنال کنترلی فعال پایین و جهت آن از کامپیوتر به چاپگر می‌باشد. هنگامی که کامپیوتر کد اسکی کاراکتری که باید چاپ شود را روی خطوط داده قرار می‌دهد، یک پالس H-L-H[1] روی این پایه ایجاد می‌کند تا چاپگر را از ارسال کاراکتر آگاه کند. کد اسکی کاراکتر باید دست‌کم ۰/۵ میکروثانیه قبل و ۰/۵ میکروثانیه بعد از فعال شدن STROBE، روی پایه‌های داده پورت موازی قرار داشته باشد. شکل بالا را ببینید.

سیگنال BUSY: یک سیگنال وضعیت فعال بالا و جهت آن از چاپگر به سمت کامپیوتر می‌باشد. هنگامی که چاپگر از طریق سیگنال STROBE متوجه می‌شود کامپیوتر برایش کاراکتری ارسال کرده، سیگنال BUSY را فعال می‌کند تا به کامپیوتر نشان دهد که مشغول چاپ کاراکتر است. کامپیوتر مرتباً این سیگنال را کنترل می‌کند و مادامی که فعال است، کاراکتر بعدی را ارسال نمی‌کند.

سیگنال $\overline{\text{ACK}}$[2]: یک سیگنال وضعیت فعال پایین و جهت آن از چاپگر به سمت کامپیوتر می‌باشد. هنگامی که چاپگر به چاپ یک کاراکتر پایان می‌دهد، یک پالس H-L-H (که دست‌کم باید ۰/۵ میکروثانیه در حالت فعال باقی بماند) روی این پایه برای کامپیوتر ارسال می‌کند. کامپیوتر با دریافت این پالس متوجه می‌شود چاپ کاراکتر قبلی به پایان رسیده و می‌تواند کاراکتر بعدی را برای چاپگر ارسال کند. معمولاً از این سیگنال در ارتباط با پایه وقفه دستگاه درخواست‌کننده چاپ (مثلاً کامپیوتر) استفاده می‌شود.

[1] High to Low to High

[2] مخفف Acknowledgment و به معنای تصدیق است.

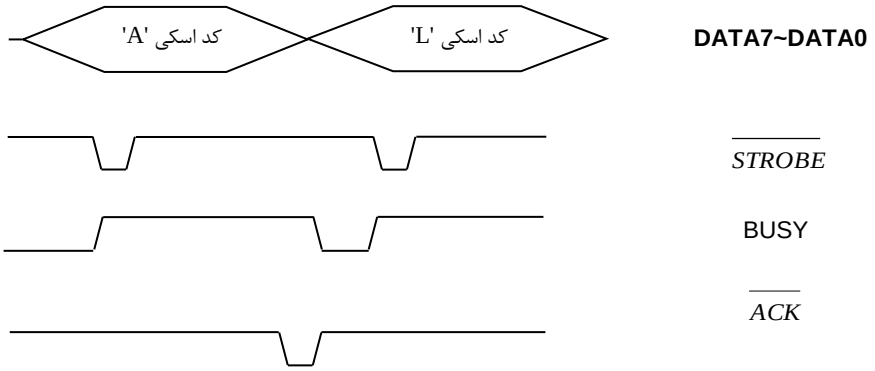
به عنوان مثال روند چاپ رشته ALI در کامپیوتر را بررسی می‌کنیم. برای این کار کامپیوتر:

- ✓ کد اسکی کاراکتر A (01000001) را روی خطوط داده پورت موازی قرار می‌دهد.
- ✓ یک پالس روی پایه STROBE ایجاد می‌کند.
- ✓ منتظر می‌ماند تا چاپگر سیگنال BUSY را غیرفعال کند و یک پالس روی سیگنال ACK برایش ارسال کند.
- ✓ کد اسکی کاراکتر L را روی خطوط داده پورت موازی قرار می‌دهد.
- ✓ یک پالس روی پایه STROBE ایجاد می‌کند.
- ✓ منتظر می‌ماند تا چاپگر سیگنال BUSY را غیرفعال کند و یک پالس روی سیگنال ACK برایش ارسال کند.
- ✓ کد اسکی کاراکتر I را روی خطوط داده پورت موازی قرار می‌دهد.
- ✓ یک پالس روی پایه STROBE ایجاد می‌کند.
- ✓ منتظر می‌ماند تا چاپگر سیگنال BUSY را غیرفعال کند و یک پالس روی سیگنال ACK برایش ارسال کند.

در سمت چاپگر چه روی می‌دهد؟ چاپگر:

- ✓ منتظر می‌ماند تا پالس STROBE را از کامپیوتر دریافت کند.
- ✓ با دریافت سیگنال STROBE، سیگنال BUSY را فعال می‌کند.
- ✓ اطلاعات مربوط به کد اسکی ارسالی توسط کامپیوتر را از روی خطوط داده می‌خواند و مشغول به چاپ آن می‌شود.
- ✓ پس از پایان چاپ، سیگنال BUSY را غیرفعال کرده و یک پالس روی پایه ACK برای کامپیوتر می‌فرستد.
- ✓ منتظر می‌ماند تا پالس STROBE را از کامپیوتر دریافت کند.
- ✓ ...

شکل زیر بخشی از نمودار زمانبندی چاپ رشته ALI را نشان می‌دهد:



وظیفه عملیاتی سیگنالهای BUSY و ACK مشابه یکدیگر است و در بعضی سیستم‌ها از اولی و در بعضی از دومی استفاده می‌شود. در زیر به معرفی سیگنالهای دیگر پورت موازی کامپیوتر می‌پردازیم:

سیگنال PE[1]: یک سیگنال وضعیت فعال بالا و جهت آن از چاپگر به کامپیوتر است. هنگامی که چاپگر با اتمام کاغذ مواجه شود این سیگنال را فعال می‌کند تا مراتب را به کامپیوتر اطلاع دهد.

[1] Paper End

سیگنال $\overline{\text{SLCT IN}}$: یک سیگنال کنترلی فعال پایین و جهت آن از کامپیوتر به چاپگر می‌باشد. در رابطه کامپیوتر با چاپگر این سیگنال به نشانه انتخاب شدن چاپگر باید فعال باشد.

سیگنال SLCT OUT: یک سیگنال وضعیت فعال بالا و جهت آن از چاپگر به کامپیوتر است. هنگامی که کامپیوتر برای انتخاب چاپگر سیگنال SLCT IN را فعال می‌کند، چنانچه چاپگر مشکلی برای این ارتباط نداشته باشد، در پاسخ سیگنال SLCT OUT را فعال می‌نماید.

سیگنال $\overline{\text{AUTOFEED}}$: یک سیگنال کنترلی فعال پایین و جهت آن از کامپیوتر به چاپگر می‌باشد. اگر کامپیوتر این سیگنال را فعال کرده باشد، به چاپگر فرمان می‌دهد پس از پایان چاپ یک خط (دریافت کاراکتر ۱۳ یا CR[1]) به صورت خودکار به خط بعد برود.

سیگنال $\overline{\text{ERROR}}$: یک سیگنال وضعیت فعال پایین و جهت آن از چاپگر به کامپیوتر است. اگر در حین چاپ خطایی (غیر از اتمام کاغذ) رخ دهد، چاپگر به کمک این سیگنال کامپیوتر را از وقوع این خطا مطلع می‌کند.

سیگنال $\overline{\text{INIT}}$: یک سیگنال کنترلی فعال پایین و جهت آن از کامپیوتر به چاپگر می‌باشد. در ابتدای ارتباط، کامپیوتر با ایجاد یک پالس روی این پایه (که دست‌کم باید ۵۰ میلی‌ثانیه فعال بماند)، چاپگر را Reset می‌کند. با این کار، چاپگر مجدداً مقداردهی درونی اولیه‌اش را انجام می‌دهد (مثلاً حافظه بافر داخلی‌اش را پاک می‌کند).

حال که نحوه تعامل بین کامپیوتر و چاپگر را شناختیم، به دو موضوع مهم میپردازیم. اول اینکه چگونه می‌توان بدون کمک کامپیوتر (مثلاً با یک میکروکنترلر) از چاپگر استفاده کرد و دوم اینکه چگونه به کمک پورت موازی کامپیوتر و با الهام از نحوه ارتباط بین کامپیوتر و چاپگر، می‌توان یک دستگاه جانبی (مثلاً یک نمایشگر LCD) را کنترل کرد.

استفاده از چاپگر بدون نیاز به کامپیوتر

در بخش قبل آموختیم که کامپیوتر چگونه برای چاپ یک رشته با چاپگر ارتباط برقرار می‌کند. حال فرض کنید می‌خواهید اطلاعات جمع‌آوری شده توسط یک دستگاه را چاپ کنید؛ مثلاً یک دستگاه کارت‌خوان که پس از خواندن بارکد روی کارت یک کارمند، باید برای او ژتون غذا چاپ کند، دستگاهی که باید دمای اتاق را هر ۱۰ ثانیه یک بار چاپ کند و .... آیا برای چاپ این اطلاعات، باید حتماً آنها را به کامپیوتر فرستاد و به کمک کامپیوتر این اطلاعات را چاپ نمود؟

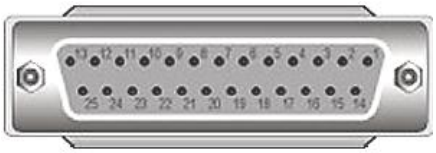
مطالبی که در بخش قبل آموختیم به ما می‌گویند در این موارد نیازی به استفاده از کامپیوتر نیست. کافی است یک میکروکنترلر را طوری برنامه‌ریزی کنید که برای چاپ یک رشته اطلاعات، دقیقاً رفتاری مشابه کامپیوتر با چاپگر داشته باشد. به این صورت بدون اینکه نیازی به کامپیوتر باشد می‌توانید اطلاعات مورد نظر خود را به کمک چاپگر چاپ کنید. به عنوان مثال، رشته ALI را به کمک میکروکنترلر ۸۰۵۱ چاپ می‌کنیم. برای این کار اتصالات صفحه بعد را بین

[1] Carriage Return

میکروکنترلر و چاپگر برقرار کنید. توجه کنید که پورت P0 میکروکنترلر برای تبادل داده با چاپگر، بیتهای پورت P1 برای ارسال سیگنالهای کنترلی و بیتهای پورت P2 برای دریافت سیگنالهای وضعیت چاپگر مورد استفاده قرار گرفته‌اند.

شماره پایه	عملکرد	پورت میکروکنترلر ۸۰۵۱	شماره پایه	عملکرد	پورت میکروکنترلر ۸۰۵۱
۱	-STROBE	P1.0	۱۰	-ACK	P2.0
۲	+DATA BIT 0	P0.0	۱۱	+BUSY	P2.1
۳	+DATA BIT 1	P0.1	۱۲	+PE	P2.2
۴	+DATA BIT 2	P0.2	۱۳	+SLCT OUT	P2.3
۵	+DATA BIT 3	P0.3	۱۴	-AUTO FEED	P1.1
۶	+DATA BIT 4	P0.4	۱۵	-ERROR	P2.4
۷	+DATA BIT 5	P0.5	۱۶	-INIT	P1.2
۸	+DATA BIT 6	P0.6	۱۷	-SLCT IN	P1.3
۹	+DATA BIT 7	P0.7	۱۸-۲۵	GND	زمین میکروکنترلر

دقت کنید سر کابل چاپگر چون باید داخل پورت موازی کامپیوتر قرار گیرد دارای ۲۵ میله است که شماره‌گذاری پایه‌های آن معکوس آینه‌ای پورت پشت کامپیوتر است (یعنی جای چپ و راست عوض می‌شود). اما چون شما می‌خواهید میکروکنترلر خود را به این کابل متصل کنید، باید از یک قطعه DB25 دارای حفره (مانند شکل مقابل) استفاده کنید. شماره‌گذاری پایه‌های این قطعه (که در جدول بالا آمده است) مانند پورت پشت کامپیوتر است.



برنامه صفحه بعد برای میکروکنترلر ۸۰۵۱ نوشته شده تا با فرض وجود اتصالات بالا، رشته ALI را به کمک چاپگر چاپ کند.

توجه کنید در این برنامه تنها از سیگنال BUSY برای کنترل وضعیت چاپگر استفاده شده است. می‌توانید از سیگنال ACK همزمان با BUSY استفاده کنید.

```
ORG     0
MOV     P2,#255     ; Define P2 as input (for reading status signals)
CLR     P1.3        ; Activating –SLCT IN
JNB     P2.3,$      ; Wait until +SLCT OUT is actiated by printer
CLR     P1.2            ; Activating -INIT
ACALL   DELAY       ; Wait
SETB    P1.2            ; Deactivating –INIT to make a pulse to reset the printer
JNB     P2.3,$      ; Wait until +SLCT OUT is actiated by printer
JB      P2.1,$      ; Wait until +BUSY is deactiated by printer

MOV     P0,#'A'     ; Put the ASCII code of 'A' on DATA lines
CLR         P1.0            ; Activating –STROBE
ACALL   DELAY               ; Wait
SETB        P1.0            ; Deactivating –STROBE to make a pulse
JB      P2.1,$      ; Wait until +BUSY is deactiated by print

MOV     P0,#'L'     ; Put the ASCII code of 'L' on DATA lines
CLR         P1.0            ; Activating –STROBE
ACALL   DELAY           ; Wait
SETB        P1.0            ; Deactivating –STROBE to make a pulse
JB      P2.1,$      ; Wait until +BUSY is deactiated by print

MOV     P0,#'I'     ; Put the ASCII code of 'I' on DATA lines
CLR         P1.0            ; Activating –STROBE
```

```
ACALL    DELAY              ; Wait
SETB          P1.0               ; Deactivating –STROBE to make a pulse
JB       P2.1,$          ; Wait until +BUSY is deactiated by print

JMP           $

DALEY:
MOV    R1,#50
AGAIN:
     NOP
DJNZ    R1, AGAIN
RET

END
```

پرسش) با استفاده از ACK به جای BUSY و اتصال آن به پایه وقفه میکروکنترلر می‌توانید برنامه حرفه‌ای‌تری بنویسید. برنامه بالا را با این فرض اصلاح کنید.

پرسش) در برنامه بالا فرض کرده‌ایم فرآیند چاپ بدون هیچ مشکلی انجام شود. برای حرفه‌ای‌تر شدن برنامه می‌توانید هنگامی که منتظر غیرفعال شدن سیگنال BUSY هستید، سیگنالهای PE و ERROR را نیز چک کنید و در صورت فعال شدن آنها واکنش مناسب را نشان دهید. به علاوه می‌توانید زمان انتظار برای غیرفعال شدن BUSY را نیز اندازه بگیرید و اگر بیش از حد عادی به طول انجامید پیغام خطا صادر کنید. برنامه میکروکنترلر را طوری اصلاح کنید که این موارد را نیز به انجام برساند.

دقت کنید مطالب گفته شده تنها در مورد چاپگرهای سوزنی (ماتریسی) صادق است. نمونه‌ای از کاربرد این نوع چاپگر پرصدا در بانکها برای چاپ اطلاعات روی قبوض آب و برق و ... است. اگر یک چاپگر سوزنی را به صورت گفته شده به میکروکنترلر ۸۰۵۱ متصل کنید و برنامه بالا را روی میکروکنترلر اجرا کنید، رشته ALI به چاپ می‌رسد. چاپگرهای جوهرافشان و لیزری دارای پردازنده‌های خاصی داخل خود هستند و برای تبادل اطلاعات با آنها باید از زبانهای برنامه‌نویسی مخصوص استفاده کرد.

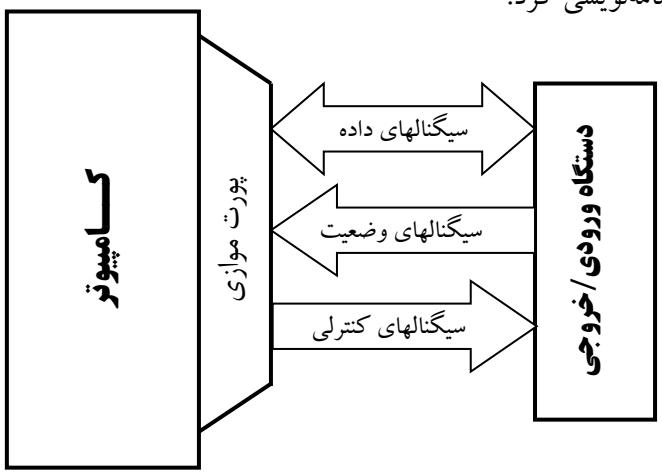
کنترل دستگاههای جانبی به کمک پورت موازی کامپیوتر

در بخش قبل آموختیم چگونه کامپیوتر با چاپگر رفتار می‌کند. آیا می‌توان دستگاهی دیگر غیر از چاپگر (مثلاً یک تابلو تبلیغات، یک روبات یا ...) را با پورت موازی کامپیوتر کنترل کرد؟

خوشبختانه پاسخ مثبت است. به کمک برنامه‌نویسی پورت موازی و با طراحی مناسب سخت‌افزار می‌توان به سادگی دستگاههای جانبی را به کمک این پورت کنترل کرد. جزییات این کار، موضوع این بخش است.

دیدیم که در پورت موازی کامپیوتر سه دسته سیگنال داده، وضعیت و کنترل وجود دارند. برای طراحی دستگاهی که به پورت موازی کامپیوتر متصل شود، از سیگنالهای داده برای ارتباط داده‌ای، از سیگنالهای کنترلی برای کنترل دستگاه و از سیگنالهای وضعیت برای بررسی وضعیت دستگاه استفاده می‌شود. شکل صفحه بعد این ارتباط را نشان می‌دهد.

متناظر با این سه دسته سیگنال، هر پورت موازی در کامپیوتر دارای سه ثبات داده[1] و وضعیت[2] و کنترل[3] است که بیتهای این ثباتها مرتبط با سیگنالهای متناظر در پورت موازی کامپیوتر هستند. از طریق برنامه‌ریزی این سه ثبات می‌توان پورت موازی کامپیوتر را برنامه‌نویسی کرد.



شکل ۶-۱-سیگنال‌های لازم برای کنترل دستگاه جانبی توسط کامپیوتر

جدول زیر بیتهای این سه ثبات را نشان می‌دهد:

ثبات داده			ثبات وضعیت			ثبات کنترلی		
عملکرد	Low	High	عملکرد	Low	High	عملکرد	Low	High
Data7	0	1	**Busy**	**Busy**	**Not Busy**	Not Used	-	-
Data6	0	1	Acknowledge	Ack	Nack	Not Used	-	-
Data5	0	1	Paper Status	Paper	No Paper	Direction	Output	Input
Data4	0	1	Selection Status	Not Selected	Selected	Interrupt Control	Interrupt Disabled	Interrupt Enabled
Data3	0	1	Error Status	Error	No Error	**Select**	**Not Selected**	**Selected**
Data2	0	1	IRQ	True	False	Initialize	Enable	Disable
Data1	0	1	Not Used	-	-	**Auto Feed**	**Disable**	**Enable**
Data0	0	1	Not Used	-	-	**Strobe**	**Disable**	**Enable**

همانطور که در جدول بالا می‌بینید:

سیگنالهای داده پورت موازی به بیتهای ثبات داده پورت موازی مربوط هستند؛ یعنی هرچه در این ثبات توسط کامپیوتر نوشته شود از طریق سیگنالهای داده پورت موازی به صورت سیگنالهای دیجیتال الکترونیکی قابل دسترسی است. به همین صورت اگر به شیوه‌ای که بعداً می‌بینیم، از پورت موازی به عنوان ورودی استفاده شود، برای خواندن عدد دیجیتالی که به سیگنالهای داده پورت موازی متصل شده کافی است ثبات داده پورت موازی را بخوانیم.

بیتهای ثبات وضعیت با سیگنالهای وضعیت پورت موازی متناظرند. در واقع کامپیوتر برای خواندن سیگنالهای وضعیت، ثبات وضعیت را می‌خواند. مثلاً اگر بیت شماره ۳ ثبات وضعیت صفر باشد، یعنی سیگنال $\overline{\text{ERROR}}$ پورت موازی توسط دستگاه خارجی (چاپگر یا دستگاه دیگر) فعال (صفر) شده است. اما سیگنال BUSY (که با حروف پررنگ‌تر مشخص شده است) داستانی متفاوت دارد؛ حالت بیت BUSY در ثبات وضعیت، عکس حالت سیگنال

[1] Data Register
[2] Status Register
[3] Control Register

BUSY در پورت موازی است؛ به بیان دیگر دستگاه خارجی برای غیرفعال کردن سیگنال فعال بالای BUSY باید آن را «صفر» کند؛ در این حال بیت ویژه BUSY در ثبات وضعیت «یک» خواهد بود. بنابراین کامپیوتر مرتباً ثبات وضعیت را می‌خواند و تا زمانی که **بیت** BUSY «صفر» است (**سیگنال** BUSY «یک» است) منتظر می‌ماند.

برای تغییر سیگنالهای کنترلی پورت موازی باید از ثبات کنترلی استفاده کنیم. به عنوان مثال برای فعال کردن سیگنال INIT کافی است بیت شماره ۲ این ثبات را صفر کنیم. سه بیت این ثبات که با حروف پررنگ‌تر مشخص شده‌اند، وضعیتی مانند سیگنال BUSY دارند و حالت بیتی آنها عکس حالت سیگنال معادل آنهاست. مثلاً برای فعال کردن سیگنال $\overline{\text{STROBE}}$ باید در بیت معادل آن در ثبات کنترلی «یک» بنویسیم؛ با این کار سیگنال فوق در پورت موازی «صفر» می‌شود.

بیت شماره ۵ ثبات کنترلی اهمیتی خاص دارد. اگر بخواهید از پورت موازی کامپیوتر که به صورت پیش‌فرض در حالت خروجی قرار دارد به عنوان ورودی استفاده کنید، باید این بیت را در ثبات کنترلی «یک» کنید.

بنابراین برای کنترل هر دستگاهی که به پورت موازی متصل است، باید با سیگنالهای این پورت به کمک ثباتهای ذکر شده کار کنیم. اما چگونه می‌توان به این ثباتها دسترسی داشت؟

آدرس ثباتهای پورت موازی

هر پورت موازی در کامپیوتر دارای یک آدرس پایه در فضای آدرس‌دهی ورودی/خروجی است که سه ثبات ذکر شده با شروع از این آدرس قرار می‌گیرند. اگر این آدرس را Base بنامیم، آدرس ثبات داده پورت موازی برابر Base، آدرس ثبات وضعیت پورت موازی برابر Base+1 و آدرس ثبات کنترل پورت موازی برابر Base+2 خواهد بود. مثلاً اگر آدرس پورت موازی کامپیوتر برابر 0378h[1] باشد (که در اکثر کامپیوترهای IBM چنین است)، آنگاه آدرس ثبات داده پورت موازی برابر 0378h، آدرس ثبات وضعیت پورت موازی برابر 0379h و آدرس ثبات کنترل پورت موازی برابر 037Ah خواهد بود. به بیان دیگر برای تبادل داده با پورت موازی با ثباتی به آدرس 0378h، برای خواندن سیگنالهای وضعیت پورت موازی با ثباتی به آدرس 0379h و برای ایجاد تغییر در سیگنالهای کنترلی پورت موازی با ثباتی به آدرس 037Ah سروکار داریم.

در کامپیوترهای امروزی اغلب یک پورت موازی وجود دارد که آدرس مبنای آن معمولاً 0378h است. در کامپیوترهای قدیمی‌تر به کمک کارت I/O امکان اضافه کردن پورتهای موازی بیشتری به کامپیوتر نیز وجود داشت. در صورت وجود دو پورت موازی، آدرس LPT1 و LPT2 به ترتیب برابر 0378h و 0278h یا 03BCh و 0378h بود. در انتهای این بخش چگونگی به دست آوردن آدرسهای مزبور در ویندوز را خواهیم دید.

به کمک BIOS نیز می‌توان آدرس مبنای ثباتهای ورودی/خروجی پورت موازی را دانست. در واقع این آدرسها از آفست 0008 به بعد سگمنت 0040 حافظه ذخیره می‌شود. جدول مقابل را ببینید.

پورت LPT	آدرس پایه ورودی/خروجی ثباتهای پورت
LPT1	0040:0008 – 0040:0009
LPT2	0040:000A – 0040:000B
LPT3	0040:000C – 0040:000D
LPT4	0040:000E – 0040:000F

[1] به یاد دارید که آدرسهای ورودی/خروجی در کامپیوتر ۱۶ بیتی هستند.

مثلاً دستور *D 0040:0008 L8* در نرم‌افزار Debug، آدرس پایه پورتهای موازی نصب شده در کامپیوتر را نشان می‌دهد. قبلاً با نرم‌افزار Debug و فرمانهای آن آشنا شدیم.

به کمک برنامه‌نویسی C نیز می‌توان آدرسهای ذخیره شده در این بخش حافظه را مشاهده کرد. برنامه زیر، آدرس پورت LPT1 و در صورت عدم نصب پورت مزبور، پیغامی را نمایش می‌دهد.

```
#include    <STDIO.H>
#include    <DOS.H>

void   main(){
       unsigned int far *fptr;
       fptr = (unsigned int far *) 0x00000408;
       if (*fptr > 0)
              printf("LPT1 I/O base address = %X",*fptr);
       else
              printf("LPT1 not found");
}
```

برنامه‌نویسی پورت موازی

برای این کار باید از ثباتهای داده و وضعیت و کنترل پورت موازی استفاده کنیم. سه تابع برای این کار پیشنهاد می‌کنیم:

ارسال داده‌ها به پورت موازی

```
   void  SendData(int  PortID, unsigned char  Data){
       outportb(PortID, Data);
   }
```

قبلاً با توابع outportb و outp برای ارسال یک بایت به پورتی با آدرس مشخص آشنا شدیم[۱]. برای استفاده از این توابع، شامل کردن فایلهای سرآیند مناسب را فراموش نکنید. مثلاً برای ارسال عدد 55h به پورت موازی از دستور SendData(0x0378,0x55); استفاده می‌کنیم. این دستور عدد 55h (01010101) را در ثبات داده پورت موازی می‌نویسد. با این کار عدد فوق روی سیگنالهای داده این پورت نمایش داده خواهد شد.

بررسی وضعیت یک بیت ثبات وضعیت

```
   int    Bit_Status(int   PortID, int BitNumber){
       unsigned char status, temp;
       status = inportb(PortID);
       temp = 1;
       temp = temp << BitNumber;
       if ((status & temp) > 0)
              return 1;
         else
         return 0;
   }
```

قبلاً با توابع inportb و inp برای دریافت یک بایت از پورتی با آدرس مشخص آشنا شدیم[۲]. برای استفاده از این توابع، شامل کردن فایلهای سرآیند مناسب را فراموش نکنید. تابع Bit_Status مقدار بیت شماره BitNumber ثباتی به

---

[۱] در فصل قبل نحوه استفاده از دستور OUT در زبان اسمبلی برای ارسال داده‌ها به یک پورت خروجی را مشاهده کردید.

[۲] در فصل قبل نحوه استفاده از دستور IN در زبان اسمبلی برای دریافت داده‌ها از یک پورت ورودی را مشاهده کردید.

آدرس PortID را برمی‌گرداند. به عنوان مثال، دستور printf("%d", Bit_Status(0x0379, 5)); مقدار بیت شماره ۵ ثبات وضعیت (بیت Paper Status) را نمایش می‌دهد. این کار چگونه انجام می‌شود؟

ابتدا مقدار پورت به کمک دستور inportb خوانده و در متغیر status ذخیره می‌شود. برای بررسی مقدار بیت شماره ۵ این متغیر، باید کاری کرد که مقدار تمام بیتهای آن غیر از بیت شماره ۵ صفر شود. دستور زیر این کار را انجام می‌دهد:

```
          D7  D6  D5  D4  D3  D2  D1  D0
AND
           0   0   1   0   0   0   0   0
        ---------------------------------
       0   0  D5   0   0   0   0   0
```

حال اگر مقدار حاصل AND برابر صفر باشد نشان‌دهنده این است که D5 نیز صفر بوده است. اگر حاصل مقداری غیر از صفر داشته باشد (۳۲) یعنی بیت D5 یک بوده است.

در تابع بالا متغیر temp با مقدار اولیه 1 تعریف شده است. برای بررسی بیت شماره BitNumber، مقدار temp به تعداد BitNumber به سمت چپ شیفت داده شده و حاصل AND آن با متغیر status بررسی می‌شود.

مثال) قطعه برنامه صفحه بعد، اعداد صفر تا ۲۵۵ را برای دستگاهی که به پورت موازی کامپیوتر متصل است، ارسال می‌کند. بعد از ارسال هر عدد، کامپیوتر باید منتظر بماند تا دستگاه مقابل سیگنال Busy را غیرفعال کند:

```
void main(void){
    for (unsigned char num = 0;num <= 255;num++){
            SendData(0x378,num);
            while (!Bit_Status(0x379,7));
    }
}
```

تغییر مقدار یک بیت ثبات کنترلی

```
void   Change_Bit(int   PortID, int BitNumber, int value){
    unsigned char In, temp, Out;
    In = inportb(PortID);

    temp = 1;
    temp = temp << BitNumber;

    if   (value == 1)
            Out = In | temp;

    else  // value = 0
            Out = In & (~temp);

    outportb(PortID, Out);
}
```

تابع فوق مقدار بیت شماره BitNumber ثباتی به آدرس PortID را بر اساس پارامتر value تغییر می‌دهد. مثلاً اجرای تابع Change_Bit(0x37A,5,1); باعث می‌شود بیت شماره ۵ ثبات کنترلی پورت موازی (بیت Direction)

یک شود. همانطور که قبلاً دیدیم، این کار باعث می‌شود پورت موازی در حالت ورودی عمل کند. برای صفر کردن این بیت می‌توان از دستور ;Change_Bit(0x37A, 5,0) استفاده کرد.

همانطور که در تابع فوق دیده می‌شود برای تغییر یک بیت از یک ثبات، ابتدا مقدار فعلی آن ثبات را می‌خوانیم، مقدار بیت مورد نظر را تغییر می‌دهیم و مجدداً مقدار جدید را در ثبات می‌نویسیم. با این کار فقط مقدار بیت مورد نظر ما تغییر می‌کند.

پرسش) با توجه به توضیحات تابع Bit_Status، راجع به نحوه عملکرد تابع Change_Bit توضیح دهید.

اتصال دستگاههای خروجی به پورت موازی

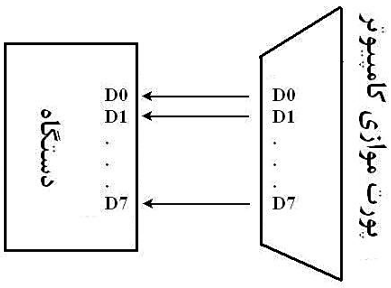
اکنون آماده هستیم نحوه اتصال دستگاههای خارجی (غیر از چاپگر) به پورت موازی و کنترل آنها را شرح دهیم. در حال حاضر فقط در مورد ارسال داده‌ها به دستگاه خارجی از طریق پورت موازی صحبت می‌کنیم و در ادامه به نحوه دریافت داده‌ها از این پورت نیز خواهیم پرداخت.

پیشتر دیدیم که برای ارسال داده‌های متوالی از فرستنده به گیرنده، به روشهای کنترل جریان داده نیازمندیم تا اطمینان حاصل شود حجم انتقال داده‌ها از فرستنده به گیرنده کنترل شده است و گیرنده فرصت پردازش آنها را دارد.

اکنون می‌خواهیم دستگاهی طراحی کنیم که از طریق پورت موازی یک بلوک اطلاعات (مثلاً ۱۰۰ بایت داده) برای آن ارسال شود. به عنوان مثال دستگاهی طراحی می‌کنیم که بایتهای داده را از پورت موازی کامپیوتر دریافت و در حافظه داخلی‌اش ذخیره کند. بدیهی است دستگاه برای ذخیره هر بایت نیاز به زمان دارد و کامپیوتر نباید بدون وقفه داده‌ها را برای آن ارسال کند.

بنابراین باید در برنامه‌ای که در کامپیوتر برای ارسال این داده‌ها به پورت موازی اجرا می‌شود، مکانیسمی برای کنترل جریان داده پیش‌بینی شود. قبلاً دو مکانیسم تأخیر و دست‌دهی را در مبحث کنترل جریان داده بررسی کردیم. نحوه طراحی دستگاه فوق با دو مکانیسم فوق را بررسی می‌کنیم.

کنترل جریان داده به شیوه تأخیر



در این روش، در برنامه کامپیوتری بین هر ارسال داده به پورت موازی مقداری تأخیر ایجاد می‌کنیم تا دستگاهی که به این پورت متصل است، فرصت کافی برای دریافت و ذخیره این اطلاعات را داشته باشد. برنامه زیر اعداد یک تا ۱۰۰ را با کنترل جریان داده به روش تأخیر برای پورت موازی ارسال می‌کند.

حلقه اصلی برنامه، یک بایت را به پورت موازی ارسال می‌کند و سپس یک ثانیه تأخیر ایجاد می‌کند تا دستگاهی که به این پورت متصل است فرصت دریافت و ذخیره این بایت را داشته باشد. سپس بایت بعدی را ارسال می‌کند.

```
#include    <STDIO.H>
#include    <STDLIB.H>
#include    <DOS.H>

void   SendData(int PortID, unsigned char Data){
   outportb(PortID, Data);
}
```

```
void   main(){

   SendData(0x0378, 0x55); // Start of Data Transfer
   delay(1000);

   for (unsigned char i = 1; i < 101; i++){
         SendData(0x0378, i);
         delay(1000);
   }

   SendData(0x0378, 0xAA); // End of Data Transfer
}
```

چرا در ابتدای ارتباط عدد 55h (01010101) و در انتها عدد AAh (10101010) برای دستگاه ارسال شده است؟ چون اطلاعات به روش تأخیر برای پورت ارسال می‌شود، دستگاه باید در فواصل زمانی مشخص (که برابر با تأخیر تعبیه شده در برنامه کامپیوتری است)، سیگنالهای داده پورت موازی را برای دریافت اطلاعات ارسالی بخواند. نکته مهم این است که در برنامه‌ریزی دستگاه، باید نحوه آغاز و پایان ارتباط به نحوی مشخص شود تا دستگاه بداند از چه زمانی دریافت اطلاعات را آغاز کند. در این ارتباط ما از این قرارداد استفاده کرده‌ایم که کامپیوتر در ابتدای ارتباط عدد 55h (01010101) و در انتها عدد AAh (10101010) را ارسال می‌کند. دستگاه نیز باید از همین قرارداد پیروی کند. توجه کنید اعداد فوق (55h و AAh) نشانه‌اند و نباید جزء داده‌های ارسالی باشند.

شبه‌کد برنامه دستگاه را ببینید.

```
Wait to receive 55h;// Start of communication
Delay (1500);       // Wait untill the first data is ready

Repeat {
   Receive new data from Parallel Port;

   If data = AAh then
         exit the loop; // End of communication
   else {
         Store Data;
         Delay (1000);
   }
}
```

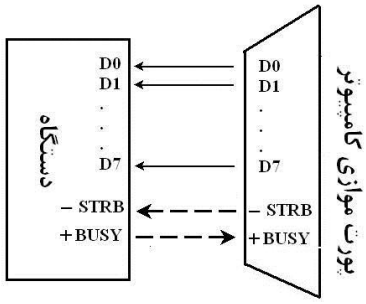
پرسش) چرا بعد از دریافت عدد نشانه 55h به میزان ۱۵۰۰ میلی‌ثانیه و در دفعات بعدی به میزان ۱۰۰۰ میلی‌ثانیه تأخیر ایجاد می‌شود؟

روش فوق همانطور که قبلاً دیدیم ساده، اما دارای مشکلاتی است. ممکن است دستگاه نیازی به یک ثانیه زمان برای ذخیره داده نداشته باشد که در این حالت زمان به هدر خواهد رفت. از طرف دیگر ممکن است به دلیل بروز مشکلی، دستگاه نتواند طی یک ثانیه بایت دریافتی را ذخیره کند؛ در این حال کامپیوتر بدون توجه به این مشکل بایت بعدی را ارسال خواهد کرد.

کنترل جریان داده به شیوه دست‌دهی

راه فنی‌تر و البته پیچیده‌تر کنترل جریان داده استفاده از سیگنالهای دست‌دهی است. در این روش دو سیگنال دست‌دهی به سیگنالهای داده اضافه می‌شوند.

برنامه کامپیوتری باید:



- ✓ داده را ارسال کند.
- ✓ یک پالس روی سیگنال STROBE ارسال کند تا دستگاهی را که به پورت موازی متصل است، از ارسال داده مطلع کند.
- ✓ منتظر بماند تا سیگنال BUSY از سوی دستگاه مقابل غیرفعال شود.
- ✓ داده بعدی را ارسال کند.
- ✓ .....

اکنون می‌خواهیم برنامه قبلی را با فرض اتصال سیگنالها به صورت شکل بالا بنویسیم. برنامه زیر، اعداد یک تا ۱۰۰ را با کنترل جریان داده به روش دست‌دهی برای پورت موازی ارسال می‌کند.

```
#include <STDIO.H>
#include <STDLIB.H>
#include <DOS.H>
void SendData(int   PortID, unsigned char   Data){
   outportb(PortID, Data);
}
int  Bit_Status(int  PortID, int BitNumber){
   unsigned char status, temp;
   status = inportb(PortID);
   temp = 1;
   temp = temp << BitNumber;
   if ((status & temp) > 1)
        return 1;
    else
       return 0;
}
void Change_Bit(int  PortID, int BitNumber, int value){
   unsigned char In, temp, Out;
   In = inportb(PortID);
   temp = 1;
   temp = temp << BitNumber;
   if   (value == 1)
         Out = In | temp;
   else  // value = 0
         Out = In & (~temp);
   outportb(PortID, Out);
}
void main(){

    unsigned char status;
    // Set STROBE bit to 0 (disabled)
    Change_Bit(0x037A, 0, 0);
    for (unsigned char i = 1 ; i < 101 ; i++){
          // Send data
         SendData(0x0378, i);
         // Send a pulse on STROBE pin
         // Set STROBE bit to 1 (Enabled)
         Change_Bit(0x037A, 0, 1);
         delay(1);
         // Set STROBE bit to 0 (disabled)
         Change_Bit(0x037A, 0, 0);
         //  wait while BUSY bit = 0 (active)
         while (Bit_Status(0x0379, 7) == 0);
   }
}
```

دستگاه متصل به پورت موازی باید طوری برنامه‌ریزی شده باشد که در ابتدا سیگنال BUSY را غیرفعال کند و سپس:

- ✓ منتظر دریافت پالس روی سیگنال STROBE بماند.
- ✓ با دریافت این پالس، سیگنال BUSY را فعال کند.
- ✓ سیگنالهای داده را بخواند و پردازش اطلاعات را آغاز کند.
- ✓ پس از پایان پردازش، سیگنال BUSY را غیر فعال کند.
- ✓ منتظر دریافت سیگنال STROBE بعدی بماند.
- ✓ ...

در رابطه دست‌دهی، به جای سیگنال BUSY می‌توان از سیگنال ACK هم استفاده کرد.

شبه‌کد برنامه دستگاه را در زیر می‌بینید:

```
Deactivate BUSY signal

Reapeat{
   Wait for a pulse on STROBE signal
   Activate BUSY signal
   Read DATA signals and process the data unit
   Inactivate BUSY signal
}
```

توجه کنید چون آغاز ارتباط با ارسال پالس روی STROBE مشخص می‌شود، مانند برنامه قبل نیازی نیست به کمک ارسال بایتهای خاص، آغاز ارتباط را معلوم کنیم. در صورت لزوم می‌توانید مانند قبل روشی برای پایان ارتباط به کار ببندید.

آیا می‌توان از سیگنالهای دیگر پورت موازی نیز در این ارتباط استفاده کرد؟

پرسش) برنامه کامپیوتری را طوری اصلاح کنید که در ابتدای کار یک پالس روی پایه INIT ارسال کند تا دستگاه مقابل reset شود. سیگنال INIT پورت موازی باید به پایه reset تراشه‌های دستگاه متصل شود.

پرسش) برنامه کامپیوتری را طوری اصلاح کنید که در حین انتظار برای غیرفعال شدن سیگنال BUSY، سیگنال ERROR را هم چک کند و در صورت فعال شدن آن، با صادر کردن پیغام مناسب از برنامه خارج شود. سیگنال ERROR پورت موازی را در دستگاه به یکی از پایه‌های تراشه کنترل‌کننده دستگاه متصل می‌کنیم. اصلاحات لازم را در برنامه تراشه فوق نیز طوری انجام دهید که در صورت بروز خطا در سیستم (مثلاً به طول انجامیدن بیش از حد ذخیره یک بایت) پایه فوق را فعال کند.

توجه کنید که سیگنالهای کنترلی پورت موازی برای استفاده در رابطه با دستگاههای دیگر باید با مقاومت 4.7 KΩ بالا کشیده[1] شوند.

[1] Pull-up

استفاده از پورت موازی به عنوان ورودی

در کامپیوترهای ابتدایی IBM، از پورت موازی تنها به هدف ارسال اطلاعات به چاپگر و به صورت خروجی استفاده می‌شد. این استاندارد به گونه‌ای که به زودی خواهیم دید SPP نامیده می‌شد. در این استاندارد خواندن اطلاعات از پورت موازی (استفاده از پورت موازی به عنوان پورت ورودی) مجاز نیست و باعث آسیب رسیدن به بورد اصلی کامپیوتر می‌شود.

در استانداردهای بعدی که EPP و ECP نامیده می‌شوند، خواندن از پورت موازی نیز تعریف شده است. برای این کار باید مراحل زیر انجام شوند:

- ✓ بیت Direction ثبات کنترلی باید «یک» شود.
- ✓ پینهای داده را با مقاومت 10 KΩ بالا بکشید.
- ✓ بین سیگنالهای پورت موازی و دستگاه خارجی از بافر استفاده کنید تا چنانچه دستگاه دچار مشکلی شد و جریان زیادی کشید، به پورت موازی و بورد اصلی کامپیوتر شما آسیب نرساند.
- ✓ در بعضی پورتهای موازی، بهتر است قبل از هر عملیات خواندن پورت، یک بایت 11111111 (FFh) برای این پورت ارسال کنید.

مثال کاربردی - قفل سخت‌افزاری

تاکنون راههای زیادی برای نصب قفل روی نرم‌افزارها و جلوگیری از کپی غیرمجاز آنها ارائه شده است که به جرأت می‌توان گفت اکثر آنها با ناکامی مواجه شده‌اند. این قفلهای نرم‌افزاری با وجود ملاحظات امنیتی بسیار تاکنون نتوانسته‌اند از دسترسی‌های غیرمجاز به نرم‌افزارها جلوگیری کنند.

یک راه مطمئن (و البته هزینه) برای جلوگیری از کپی غیرمجاز نرم‌افزارها استفاده از قفلهای سخت‌افزاری است. در این روش، تولیدکننده نرم‌افزار به همراه لوح فشرده حاوی نرم‌افزار، یک دستگاه کوچک به نام «قفل سخت‌افزاری» نیز ارائه می‌کند که در یکی از پورتهای کامپیوتر نصب می‌شود. نرم‌افزار فوق را فقط وقتی می‌توان مورد استفاده قرار داد که قفل مزبور داخل پورت کامپیوتر نصب شده باشد. با این روش، دیگر نمی‌توان کپی‌های غیرمجاز دیگری از نرم‌افزار فوق تهیه کرد.

کاربرد دیگر قفلهای سخت‌افزاری در سیستم‌های امنیتی است. فرض کنید پایگاه اطلاعاتی مهمی در یک سازمان وجود دارد که تنها مدیران ارشد آن سازمان باید به آن دسترسی داشته باشند. می‌توان به تمام اشخاص مجاز یک قفل سخت‌افزاری داد و نرم‌افزار را به گونه‌ای طراحی کرد که استفاده از آن منوط به نصب قفل فوق در پورت کامپیوتر باشد. به بیان دیگر، تنها هنگامی می‌توان به این پایگاه اطلاعاتی دسترسی داشت که قفل در پورت نصب شده باشد. برای بالاتر بردن امنیت سیستم، معمولاً از روشهای نرم‌افزاری در کنار قفل سخت‌افزاری استفاده می‌شود.

قفل سخت‌افزاری چگونه کار می‌کند؟

روش معمول در طراحی این قفلها این است که ارتباطی بین قفل و نرم‌افزار کامپیوتری قرارداد می‌شود که در صورت برقراری آن ارتباط دسترسی به نرم‌افزار انجام خواهد شد.

طراحی قفل سخت‌افزاری با پورت موازی کامپیوتر

مثلاً فرض کنید نرم‌افزار کامپیوتری را به گونه‌ای طراحی می‌کنیم که در ابتدای کار، پنج عدد تصادفی تولید و برای پورت موازی ارسال کند و منتظر بماند تا عددی از این پورت دریافت کند. سپس عدد دریافتی را با مجموع پنج عددی که ارسال کرده مقایسه می‌کند و در صورت تساوی، اجازه ادامه کار را صادر می‌کند.

قفل سخت‌افزاری که به پورت موازی کامپیوتر متصل است باید پنج عدد از این پورت دریافت و با هم جمع کند و نتیجه را برای پورت موازی ارسال کند.

با ترکیب این قفل و نرم‌افزار کامپیوتری، یک روش امنیتی برای دسترسی به سیستم ایجاد می‌شود. اگر قفل در پورت موازی باشد، این ارتباط به درستی صورت می‌گیرد؛ وگرنه عددی که نرم‌افزار کامپیوتری از پورت دریافت می‌کند برابر مجموع پنج عددی که ارسال کرده نیست و مجوز ادامه کار با نرم‌افزار صادر نمی‌شود.

برنامه صفحه بعد نمونه‌ای از استفاده از این روش را نشان می‌دهد.

```
#include    <STDIO.H>
#include    <STDLIB.H>
#include    <DOS.H>
void SendData(int PortID, unsigned char Data){
   outportb(PortID, Data);
}
void Change_Bit(int  PortID, int BitNumber, int value){
   unsigned char In, temp, Out;
   In = inportb(PortID);
   temp = 1 << BitNumber;
   if   (value == 1)  Out = In | temp;
   else  // value = 0
         Out = In & (~temp);
   outportb(PortID, Out);
}
void Check_Lock(int PortID){
   SendData(PortID, 0x55); // Start of Data Transfer
   delay(200);
   unsigned char Data[5], total = 0, sum;
   // Generate random numbers
   randomize();
   for (int i = 0; i < 5; i++){
         Data[i] = random(10);
         total = total + Data[i];
   }
   // Send random numbers to parallel port
   for (i = 0; i < 5; i++){
         SendData(PortID, Data[i]);
         delay(200);
   }
   Change_Bit(PortID+2, 5, 1); // Parallel Port as input
   delay(1000); // wait to receive the result from port
   sum = inport(PortID);
   Change_Bit(PortID+2, 5, 0); // Parallel Port as output
   if (total != sum){
         printf("Lock is not installed in parallel port");
         exit(0);
   }
   else   return;
}
void   main(){
   Check_Lock(0x0378);
   … // Continue to program
}
```

توجه داشته باشید همانطور که قبلاً گفته شد، این برنامه صرفاً تحت DOS یا ویندوز ۹۸ اجرا می‌شود. برای استفاده از این برنامه تحت ویندوزهای NT به بعد، باید از روشهای خاص دسترسی به پورتها استفاده کنید.

شبه‌کد برنامه قفل سخت‌افزاری به صورت زیر است:

```
Wait to receive 55h;// Start of communication
Delay (300);       // Wait untill the first data is ready

unsigned char sum = 0;

Repeat for 5 times{
   Receive new data from Parallel Port;
   Add new data to sum;

   Delay (200);
}

Send sum to parallel port;
```

پرسش) سیستم بالا به کمک روش کنترل جریان داده تأخیر طراحی شده است. برنامه‌ها را طوری اصلاح کنید که از روش دست‌دهی برای کنترل جریان داده استفاده کنند.

استفاده از پورت موازی به شیوه وقفه

بیت شماره ۴ ثبات کنترلی Interrupt Control نام دارد. اگر این بیت را در برنامه کامپیوتری خود «یک» کنید، وقفه پورت موازی فعال می‌شود. در این حالت، هنگامی که دستگاه خارجی یک لبه بالارونده روی پین ACK پورت موازی ایجاد کند، وقفه‌ای رخ می‌دهد، اجرای برنامه فعلی قطع و زیربرنامه ISR مربوط به آن وقفه اجرا می‌شود و سپس اجرای برنامه قبلی دنبال می‌شود.

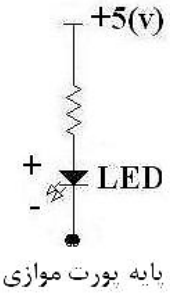
این روش برای جلوگیری از هدر رفتن وقت پردازنده برای کنترل رخ دادن یک اتفاق بسیار مفید است. به بیان دیگر، وقتی پردازنده باید در صورت روی دادن یک اتفاق خاص به اجرای زیربرنامه‌ای بپردازد، نیازی نیست مرتباً روی دادن آن اتفاق را کنترل کند. در روش وقفه به هنگام رخ دادن آن اتفاق، به پردازنده اطلاع داده می‌شود و پردازنده عملیات خود را رها کرده و زیربرنامه مورد نظر را اجرا می‌کند.

در پورت موازی معمولاً از IRQ5 یا IRQ7 استفاده می‌شود (هرچند احتمال استفاده از شماره‌های دیگر نیز وجود دارد). برای استفاده از پورت موازی به شیوه وقفه، منبع وقفه‌دهنده (که در اینجا یک دستگاه خارجی است) باید سیگنالی در حالت خروجی داشته باشد که در هنگام نیاز به جلب توجه پردازنده (مثلاً وقتی ارتفاع ماده مذاب در یک دیگ از حد مجاز زیاد می‌شود و باید به سرعت برنامه‌ای اجرا شود که ارتفاع را کم کند)، یک لبه بالارونده روی آن ایجاد کند. با اتصال این سیگنال به پایه ACK پورت موازی، در صورت وقوع لبه بالارونده وقفه (شماره ۵ یا ۷ یا شماره‌های دیگر) در کامپیوتر رخ داده و زیربرنامه ISR مربوط به آن وقفه اجرا می‌شود. تنها کار باقی مانده این است که زیربرنامه ISR فوق را به صورت مطلوبتان تغییر دهید.

تغییر ISR یک وقفه در کامپیوتر چندان پیچیده نیست و می‌توان آن را به کمک برنامه‌های اسمبلی یا سطح بالا (مانند C) انجام داد. در کتابهای زبان اسمبلی در بخش برنامه‌های مقیم در حافظه[1] می‌توانید شیوه این کار را فرا بگیرید. در منابع انتهای این فصل نیز برنامه‌های نمونه آورده شده است.

نکاتی در مورد استفاده از پورت موازی برای کنترل دستگاهها

- ✓ برد مطمئن ارسال داده‌ها به کمک روش موازی حدود ۳ متر است. در فواصل بیش از ۳ متر سیمهای کابل انتقال موازی روی یکدیگر ایجاد نویز کرده و باعث اختلال در ارسال صحیح داده‌ها می‌شوند. برای به کارگیری ارتباط موازی در فواصل زیاد باید اطلاعات در فواصل ۳ متری بافر شده و مجدداً ارسال شوند.
- ✓ برای کنترل جریان داده‌ها، استفاده از روش دست‌دهی راهکار حرفه‌ای‌تری نسبت به روش تأخیر است؛ اما این روش دو سیم به تعداد سیمهای زیاد ارتباط موازی اضافه کرده و هزینه ارتباط را بالاتر می‌برد.
- ✓ مقاومت خروجی پایه‌های پورت موازی برابر ۴۷۰ اهم و حداکثر جریان هر پایه ۱۰/۶ میلی‌آمپر است که در طراحی بخش واسط بین دستگاه و پورت موازی باید مورد توجه قرار گیرد. اگر بیش از مقدار مجاز از یکی از پایه‌های پورت موازی جریان کشیده شود، احتمال آسیب رسیدن به پورت زیاد است. حتی توصیه می‌شود برای اتصال LED به پایه‌های پورت موازی، LED را به صورت مقابل در گرایش معکوس ببندید تا جریان زیادی از پورت نکشد. بنابراین هرگز از پایه‌های پورت موازی به صورت مستقیم برای روشن کردن عناصری مانند رله، SCR، Triac، Tristor، سوییچهای الکترونیکی[2] و ... استفاده نکنید و حتماً واسطهایی مانند بافر، ترانزیستور، اوپتوکوپلر[3] و ... را به کار ببرید. این کار به ویژه در مورد مدارات قدرت که با جریانهای زیاد سروکار دارند برای پیشگیری از سوختن بورد اصلی کامپیوتر الزامی است.



- ✓ در برنامه‌هایی که دیدیم، از سیگنالهای وضعیت و کنترل پورت موازی برای عملیات دست‌دهی استفاده شد. طبیعی است که می‌توانید از سیگنالهای وضعیت برای دریافت و از سیگنالهای کنترلی برای ارسال داده‌ها نیز استفاده کنید.
- ✓ همانطور که قبلاً گفته شد، چون پورت موازی از ابتدا برای اتصال چاپگر به کامپیوتر استاندارد شده است، سیگنالهای این پورت نیز با نامهایی که در ارتباط کامپیوتر و چاپگر به کار می‌روند نامگذاری شده‌اند. اما در طراحی دستگاههای خارجی که باید به پورت موازی کامپیوتر متصل شوند، نیازی نیست از این سیگنالها با همان اسامی و معانی استفاده کنید. مثلاً سیگنال PE که در رابطه کامپیوتر با چاپگر برای اعلان خطای اتمام کاغذ توسط چاپگر به کار می‌رود، می‌تواند در رابطه کامپیوتر با دستگاه خارجی دیگر برای اعلام وضعیت یا خطایی مربوط به همان رابطه مورد استفاده قرار گیرد.

---

[1] Terminate & Stay Resident - TSR

[2] این سوییچها به عنوان جایگزینی برای رله‌های مکانیکی به کار می‌روند که از جمله آنها می‌توان به تراشه 4066B که با تکنولوژی CMOS ساخته شده یا CD74HC22106 اشاره کرد.

[3] Optocoupler قطعه‌ای است که ورودی و خروجی آن به کمک نور با یکدیگر در ارتباط هستند و از دید الکتریکی کاملاً مجزا می‌باشند. از جمله آنها می‌توان به CNY17 اشاره کرد.

استانداردهای پورت موازی

تاکنون استانداردهای مختلفی از پورت موازی معرفی شده است که در زیر به معرفی مختصر آنها می‌پردازیم:

استاندارد SPP[1] (ATI-Type یا ISA-Compatible): در سال ۱۹۸۱ و در اولین کامپیوترهای IBM به عنوان رابط چاپگر استانداردی به نام چاپگر سنترونیکس[2] یا EPSON FX-100 معرفی شد. مدارات داخلی پورت استاندارد SPP فقط برای خروج داده طراحی شده و استفاده از آن به عنوان ورودی باعث آسیب رساندن به آن می‌شود.

استاندارد PS/2: در سال ۱۹۸۷ معرفی شد و برخلاف پورت استاندارد SPP به عنوان ورودی نیز قابل استفاده است. این کار با «یک» کردن بیت Direction ثبات کنترلی پورت موازی انجام می‌شود.

در دو استاندارد ذکر شده، هر عملیات ورودی/خروجی در ۴ سیکل ساعت گذرگاه ISA انجام می‌شود.

استاندارد EPP[3]: این استاندارد در ۱۹۹۶ تحت عنوان IEEE-1284 ارایه شد و توسط شرکتهای Intel و Xircom و Zenith توسعه یافت. در این استاندارد، عملیات ورودی/خروجی در یک سیکل ساعت گذرگاه ISA انجام می‌شود که باعث سرعت بالاتر در تبادل داده (از 500 KBps تا 2 MBps) و نیز سرعت سوییچ بیشتر می‌شود. یکی از دلایل این سرعت بالا، تولید سیگنالهای کنترلی دست‌دهی به کمک مدارات سخت‌افزاری در پورت LPT است. در استانداردهای قبلی تولید این سیگنالها از وظایف نرم‌افزار کامپیوتر بود. در این استاندارد، تعداد ثباتهای پورت موازی به ۸ عدد افزایش یافته است.

استاندارد ECP[4]: این استاندارد تمام قابلیت‌های استاندارد EPP علاوه بر قابلیت‌های DMA و فشرده‌سازی تا حد $\frac{1}{64}$ را داراست و برای تبادل بلوکهای بزرگ داده کارایی خوبی دارد.

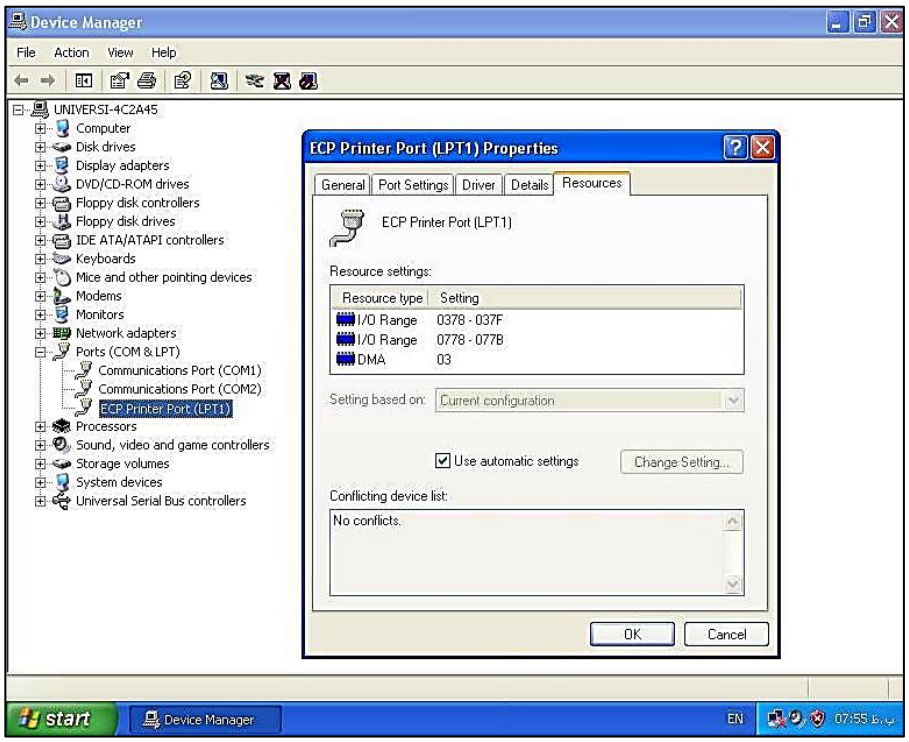
امروزه اکثر کامپیوترها از پورتهای موازی دارای استاندارد ECP استفاده می‌کنند. در ویندوز می‌توانید استاندارد پورت موازی کامپیوتر خود و نیز آدرس‌های تخصیص داده شده به ثباتهای ویژه پورت را ببینید. برای این کار:

- ✓ از Control Panel گزینه System را انتخاب کنید.
- ✓ به سربرگ Hardware بروید و گزینه Device Manager را انتخاب کنید.
- ✓ حال در قسمت Ports می‌توانید مشخصات پورتهای کامپیوتر خود را ببینید.

شکل ۶-۲، گزینه Ports مربوط به یک کامپیوتر نوعی را نشان می‌دهد. همانطور که در شکل بالا می‌بینید، پورت موازی این کامپیوتر از استاندارد ECP پیروی می‌کند. با کلیک راست روی نام پورت موازی و انتخاب گزینه Properties، کادر سمت راست ظاهر می‌شود که ویژگیهای پورت موازی را نشان می‌دهد. با انتخاب سربرگ Resources، منابع اختصاص داده شده به پورت موازی را می‌بینید (شکل بالا). به این پورت بازه آدرسهای ورودی/خروجی 0378 h – 037F h (مجموعاً ۸ ثبات) و نیز کانال شماره ۳ DMA اختصاص داده شده است.

---

[1] Standard Parallel Port
[2] Centronics
[3] Enhanced Parallel Port
[4] Extended Compatibilities Port



شکل ۶-۲-مشخصات پورت موازی یک کامپیوتر نوعی

توجه کنید آنچه در این فصل برای ارتباط با پورت موازی گفته شد، ساده‌ترین استاندارد این ارتباط را نشان می‌دهد که سازگار با تمام کامپیوترهاست. ارتباط با پورتهایی با استاندارد EPP و ECP به کمک ثباتهای اضافی و ویژگیهای پیشرفته‌تر این پورتها، شامل عملیاتی نسبتاً متفاوت است که طی آن عملکرد پینهای پورت موازی نیز تغییر می‌یابند. ارتباط با استاندارد ECP از استاندارد EPP پیچیده‌تر است؛ چرا که در آن لازم است وسیله‌ای که به پورت موازی متصل می‌شود نیز برنامه‌ریزی مفصل و هوشمندی داشته باشد. در حالی که در استاندارد EPP کنترل ارتباط کاملاً در دست کامپیوتر است. برای طولانی نشدن بحث، این شیوه ارتباطی را در اینجا بررسی نمی‌کنیم. اگر به مطالب بیشتری در این رابطه علاقه دارید، می‌توانید جزییات بیشتر را در منابع معرفی شده در انتهای فصل ببینید.

پروژه‌های کاربردی با پورت موازی کامپیوتر

۱) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید یک رقص نور روی هشت عدد LED که به پورت موازی متصل است ایجاد کند. برای این کار LEDها باید هر یک ثانیه یک بار به صورت یکی‌درمیان روشن شوند.

۲) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که محتویات ۸ عدد کلید دو حالته که به پورت موازی کامپیوتر متصل است را بخواند و روی نمایشگر کامپیوتر نشان دهد.

۳) تعداد LEDهای سیستم ۱ را با استفاده از سیگنالهای کنترلی به عنوان سیگنالهای داده خروجی، به ۱۲ عدد LED افزایش و اصلاحات لازم را در سخت‌افزار و نرم‌افزار انجام دهید.

٤) تعداد کلیدهای سیستم ۲ را با استفاده از سیگنالهای وضعیت به عنوان سیگنالهای داده ورودی، به ۱۳ عدد کلید افزایش و اصلاحات لازم را در سخت‌افزار و نرم‌افزار انجام دهید.

۵) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که محتویات هشت کلید دو حالته را از طریق پورت موازی بخواند، در برنامه کامپیوتری آن را منفی کند و حاصل را روی هشت LED متصل به پورت موازی نمایش دهد.

راهنمایی: چون پورت موازی تنها هشت پایه داده دارد که هم برای خواندن محتویات کلیدها و هم برای نمایش نتیجه روی LEDها به کار می‌رود، باید از تکنیکهایی که در فصل اول برای استفاده مشترک از گذرگاه داده آموختید استفاده کنید؛ یعنی تراشه نگهدار (Latch) را برای اتصال خروجی و تراشه بافر سه‌حالته را برای اتصال ورودی به کار ببرید. از سیگنالهای کنترلی برای فعال کردن این تراشه‌ها در زمان مناسب استفاده کنید.

٦) LCD یک نمایشگر ساده است که در دستگاههای بسیاری برای نمایش اطلاعات کاراکتری و رقمی مورد استفاده قرار می‌گیرد. نمونه‌ای از LCD را در شکل مقابل می‌بینید.

برای ارسال اطلاعات متوالی به LCD، باید از روشهای کنترل جریان داده بهره ببرید که در LCD هر دو روش قابل استفاده است. با مطالعه ساختار LCD، سیستمی شامل نرم‌افزار و سخت‌افزار طراحی کنید که با اجرای برنامه‌ای روی کامپیوتر، نامتان روی LCD متصل به پورت موازی نمایش داده شود. این سیستم را در عمل می‌توانید به راحتی بسازید.

۷) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که یک LED را با فرکانس ۱ هرتز روشن و خاموش کند.

۸) برنامه قبلی را به گونه‌ای تغییر دهید که فرکانس مورد نظر کاربر را در ابتدای برنامه از وی سؤال کند. بازه این فرکانس باید از ۱ هرتز تا ۱۰ کیلوهرتز متغیر باشد. توجه کنید در فرکانسهای بالاتر از ۵۰ هرتز، چشم متوجه خاموش و روشن شدن LED نمی‌شود و همیشه آن را روشن می‌بیند. برای دیدن این فرکانسهای بالا می‌توانید از اسیلوسکوپ استفاده کنید.

۹) سیستم ۷ را به گونه‌ای تغییر دهید که روی هشت پایه داده پورت موازی، هشت موج مربعی با فرکانسهای مختلف تولید کند. برای این کار باید از کوچکترین دوره تناوب به عنوان پایه دوره‌های تناوب دیگر استفاده کنید.

۱۰) سیستمی طراحی کنید که به کمک مبدل دیجیتال به آنالوگ (DAC) یک موج سینوسی با فرکانس دلخواه را روی یکی از پایه‌های داده پورت موازی ایجاد کند.

۱۱) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که هر یک ثانیه یک بار دمای اتاق را از یک سنسور دما خوانده و به کمک مبدل آنالوگ به دیجیتال (ADC) مقدار آن را به یک عدد دیجیتال تبدیل کند و روی نمایشگر کامپیوتر نشان دهد.

۱۲) سیستمی طراحی کنید که بدون نیاز به کامپیوتر، هر یک ثانیه یک بار دمای اتاق را به کمک چاپگر چاپ کند.

۱۳) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که به عنوان یک کارت اسیلوسکوپ عمل کند. سخت‌افزار این سیستم باید یک سیگنال آنالوگ را به کمک ADC به عدد دیجیتال تبدیل کرده و به پورت موازی کامپیوتر ارسال کند. برنامه کامپیوتری نیز باید منحنی تغییرات آن سیگنال را در نمایشگر کامپیوتر نشان دهد.

١٤) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که فرکانس یک موج مربعی که به کمک تراشه ۵۵۵ تولید و به پورت موازی کامپیوتر وارد می‌شود را روی نمایشگر کامپیوتر نشان دهد.

۱۵) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که جهت و سرعت چرخش یک موتور پله‌ای را به کمک برنامه کامپیوتری کنترل کند. با این سیستم می‌توانید روباتی بسازید که به کمک پورت موازی کامپیوتر کنترل شود.

۱۶) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که اعداد صفر تا ۹ را با فواصل زمانی یک ثانیه روی یک نمایشگر هفت‌قسمتی[1] نمایش دهد. از تراشه مبدل کد BCD به کد نمایشگر هفت قسمتی استفاده نکنید. برنامه خود را طوری بنویسید که نوع نمایشگر هفت‌قسمتی (آند مشترک یا کاتد مشترک) را در ابتدای برنامه از کاربر سؤال کند.

۱۷) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که اعداد 0.00 تا 9.99 را با فواصل زمانی ۵۰۰ میلی‌ثانیه روی سه عدد نمایشگر هفت‌قسمتی نمایش دهد. چون پورت موازی تنها دارای ۸ بیت داده است که باید به هر سه نمایشگر متصل شود، در هر لحظه باید یکی از آنها انتخاب شود. برای انتخاب نمایشگر مورد نظر از سیگنالهای کنترلی پورت موازی استفاده کنید.

۱۸) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که یک ولتاژ آنالوگ بین صفر تا ۵ ولت که به کمک یک پتانسیومتر تولید می‌شود را از طریق پورت موازی خوانده و مقدار این ولتاژ را با دقت دو رقم اعشار روی سه عدد نمایشگر هفت‌قسمتی که به پورت موازی متصلند، نمایش دهد.

۱۹) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که مقدار یک ولتاژ آنالوگ بین صفر تا ۵ ولت که به کمک یک پتانسیومتر تولید می‌شود را از طریق پورت موازی کنترل کند. چنانچه این ولتاژ از ۳ ولت بیشتر شد (به کمک تراشه‌های مقایسه‌کننده مانند LM311 می‌توانید این موضوع را نشان دهید)، باید از طریق وقفه به کامپیوتر اطلاع داده شده و پیغام خاصی روی نمایشگر کامپیوتر نشان داده شود.

۲۰) فرض کنید می‌خواهید یک فایل متنی را از کامپیوتر ۱ به کامپیوتر ۲ ارسال کنید. روشهایی مانند استفاده از شبکه‌های کامپیوتری، اینترنت و یا حافظه‌های قابل حمل (فلاپی، CD، فلاش و ...) اولین راههایی است که به ذهن می‌رسد.

یک راه ساده و کم‌خرج، اتصال مستقیم پورتهای دو کامپیوتر به یکدیگر و انتقال اطلاعات از این راه است. این شیوه به اصطلاح مودم پوچ[2] نامیده می‌شود؛ چون کامپیوترها تصور می‌کنند از طریق مودم به یکدیگر متصل هستند، در حالی که عملاً مودمی وجود ندارد و کامپیوترها فقط از طریق یک کابل ساده به یکدیگر متصل هستند. یکی از راههای ایجاد مودم پوچ، اتصال پورتهای موازی دو کامپیوتر به یکدیگر است.

سیستمی شامل نرم‌افزار کامپیوتر فرستنده و کامپیوتر گیرنده و سخت‌افزار طراحی کنید که با اتصال پورتهای موازی دو کامپیوتر و اجرای برنامه‌ها روی دو کامپیوتر، ارسال فایل از فرستنده به گیرنده میسر شود.

منابع این فصل

[۱] "اصول کامل راه‌اندازی و کنترل دستگاههای جانبی توسط کامپیوتر"، محسن شکیبافر، انتشارات نص، ۱۳۸۴.

[۲] "گذرگاهها و درگاههای کامپیوترهای شخصی"، عبدالمجید منصوریان‌فر و اصغر کریمی، انتشارات دانش‌پژوهان برین – ارکان، ۱۳۸۳

[۳] "دیجیتال پایه و طراحی مدارهای واسط"، شیرزاد شهریاری، انتشارات پرتونگار، ۱۳۸۵.

[1] Seven Segments
[2] Null Modem

فصل هفتم

ارتباط سریال

آموختیم که در ارتباط سریال، بین فرستنده و گیرنده تنها یک سیم داده (و یک سیم زمین) وجود دارد. فرستنده بیتهای داده را به صورت پشت‌سرهم (سریال) روی این سیم قرار می‌دهد و در سمت گیرنده نیز بیتها یک‌به‌یک دریافت شده و کنار هم قرار می‌گیرند. با توجه به اینکه برای ارتباط سریال تنها یک جفت سیم ضروری است، این نوع ارتباط برای تجهیزات مخابراتی و ارتباطات راه دور کاربرد وسیعی دارد. مثلاً ارتباطات در شبکه (با کابل CAT-5 در فواصل چند متری) و اینترنت (خط تلفن در فواصل چند کیلومتری) از این نوع ارتباط استفاده می‌کنند (تصور کنید اگر ارتباطات اینترنتی با اتصال موازی انجام می‌شد، تمام معادن مس دنیا هم برای تأمین این همه سیم کافی نبود!). کنترل از راه دور تلویزیون یا دزدگیر، اتصال صفحه کلید به کامپیوتر و پورت سریال و USB مثالهایی از ارتباط سریال هستند.

در پروتکل ارتباط سریال چند موضوع مهم باید مورد توجه قرار بگیرند:

- ✓ نحوه هماهنگی سرعت ارسال و دریافت داده‌ها بین فرستنده و گیرنده
- ✓ نحوه قاب‌بندی[1] اطلاعات (نحوه مشخص کردن آغاز و پایان واحد اطلاعاتی، طول واحد اطلاعاتی و ...)
- ✓ مکانیسم کشف خطا
- ✓ نحوه کنترل جریان داده

در مورد هرکدام از موارد فوق، شرح مختصری مفید است.

هماهنگی سرعت بین فرستنده و گیرنده

ارتباط سریال را می‌توان مانند نوار نقاله‌ای در نظر گرفت که یک کارگر در یک سمت آن، جعبه‌هایی را روی نقاله قرار می‌دهد و کارگری در سمت دیگر باید جعبه‌ها را از روز نقاله بردارد. بدیهی است سرعت کار هر دو کارگر باید یکسان باشد؛ اگر کارگر اول با سرعت بیشتری کار کند، کارگر دوم فرصت برداشتن جعبه‌ها از روی نقاله را پیدا نمی‌کند. اگر سرعت کارگر اول کمتر باشد، کارگر دوم که سریعتر کار می‌کند بارها به نقاله مراجعه می‌کند بدون اینکه جعبه جدیدی رسیده باشد.

در ارتباط سریال، چون فرستنده داده‌ها را بیت‌به‌بیت روی خط می‌گذارد و گیرنده نیز بیتها را یکی‌یکی از روی خط برمی‌دارد، برای یک ارتباط بی‌نقص باید سرعت فرستنده و گیرنده یکسان باشد؛ وگرنه داده‌های ارسالی به درستی در گیرنده دریافت نمی‌شوند. بنابراین یکی از موارد مهم در برقراری یک ارتباط سریال، توافق بر سر سرعت ارتباط بین فرستنده و گیرنده است.

هنگامی که با مودم Dial-up به اینترنت متصل می‌شوید، سروصدایی در ابتدای ارتباط از مودم به گوش می‌رسد و سپس ارتباط برقرار می‌شود. این سروصدای مذاکره ابتدایی بین مودم کامپیوتر شما و مودم شرکت تأمین‌کننده اینترنت شما (ISP) است که بخشی از آن مربوط به توافق بر سر سرعت ارتباط است.

[1] Framing

سرعت ارتباط سریال با واحد بیت بر ثانیه (bps[1]) سنجیده می‌شود و نرخ بیتی[2] نامیده می‌شود[3]؛ مثلاً هنگامی که بعد از اتصال به اینترنت، عبارت 49 Kbps روی مانیتور کامپیوتر نشان داده می‌شود، نشان‌دهنده این است که هم‌اکنون بین مودم کامپیوتر شما و مودم ISP یک ارتباط سریال با سرعت ۴۹۰۰۰ بیت در ثانیه برقرار است[4]. حداکثر سرعت مودمهای Dial-up برابر 56 Kbps است که در عمل از 52 Kbps فراتر نمی‌رود. پورت سریال کامپیوتر از سرعتهای مختلفی مانند ۱۱۰، ۳۰۰، ۲۴۰۰، ۹۶۰۰ و ... تا ۹۲۱۶۰۰ بیت در ثانیه حمایت می‌کند.

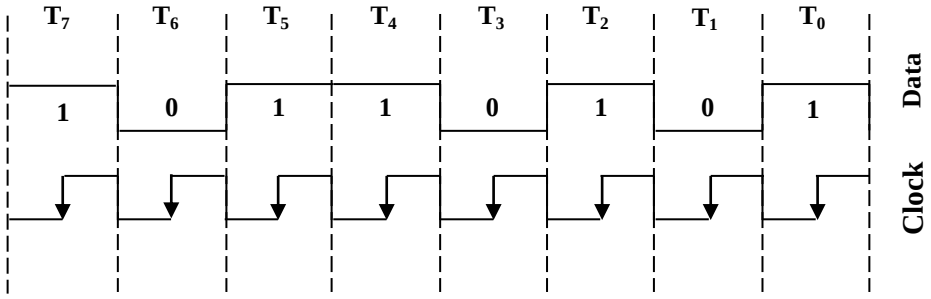
بر اساس نحوه هماهنگی بین فرستنده و گیرنده بر سر سرعت انتقال داده‌ها، دو نوع ارتباط سریال همگام[5] و غیرهمگام[6] تعریف می‌شود.

ارتباط سریال همگام

مثال قبلی در مورد نوار نقاله و دو کارگر را در نظر بگیرید. دیدیم سرعت کارگر اول در قرار دادن جعبه‌ها باید با سرعت کارگر دوم در برداشتن جعبه‌ها یکسان باشد.

یک روش برای همگام کردن دو کارگر این است که کارگر اول چند لحظه پس از قرار دادن جعبه روی نقاله، زنگی را به صدا درآورد تا کارگر دوم را متوجه کند. کارگر دوم نیز با شنیدن صدای زنگ متوجه می‌شود که وقت برداشتن جعبه فرا رسیده است.

در روش ارتباط سریال همگام، فرستنده به همراه داده‌ها، یک سیگنال ساعت (CLK) نیز ارسال می‌کند. در مواقعی که گیرنده باید خط را بخواند، فرستنده یک لبه بالارونده روی سیگنال ساعت ارسال می‌کند. گیرنده به سیگنال ساعت توجه می‌کند و هرگاه متوجه لبه بالارونده‌ای روی آن شد، خط داده‌ها را خوانده و بیتی که روی آن است را ثبت می‌کند. شکل ۷-۱ این نوع ارتباط را نشان می‌دهد.

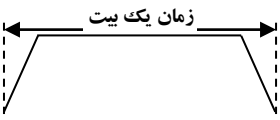


شکل ۷-۱-ارتباط سریال همگام

پیشتر در بورد اصلی کامپیوتر یک پورت ۲۵ پین سریال وجود داشت که از این روش برای ارتباط استفاده می‌کرد. ارتباط صفحه کلید با کامپیوتر نیز از این نوع است. در حال حاضر دو استاندارد ارتباط سریال همگام به نامهای BISYNC و SDLC وجود دارد.

---

[1] Bits Per Second
[2] Bit Rate
[3] گاهی در کتابهای مدارهای واسط از این نرخ به عنوان نرخ باود (Baud Rate) نیز یاد می‌شود. نرخ باود در واقع تعداد تغییرات سیگنال داده در ثانیه است که لزوماً در مخابرات با تعداد بیتهای ارسال یکسان نیست. اما در بحث ما، چون تنها دو ولتاژ وجود دارد که هرکدام نشان‌دهنده یک بیت است، تعداد تغییرات در ثانیه با تعداد بیت در ثانیه یکسان است. بنابراین در این بحث می‌توان نرخ بیتی و نرخ باود را یکسان در نظر گرفت.
[4] در مخابرات منظور از کیلو ، ۱۰۰۰ است. تنها در اندازه‌گیری حافظه کیلو برابر ۱۰۲۴ در نظر گرفته می‌شود.
[5] Synchronous
[6] Asynchronous



همانطور که در شکل ۷-۱ دیده می‌شود، فرستنده لبه بالارونده را در «وسط» زمانی که بیت روی خط است روی سیگنال ساعت قرار می‌دهد. شکل مقابل دلیل این موضوع را نشان می‌دهد. همانطور که می‌بینید، بیشترین پایداری ولتاژ بیت روی خط داده در «وسط» زمان قرار داشتن بیت روی خط داده است.

ارتباط سریال غیرهمگام

در این نوع ارتباط، دیگر خط CLK بین فرستنده و گیرنده وجود ندارد؛ بنابراین تنها راهی که برای تطابق سرعت بین فرستنده و گیرنده می‌ماند این است که **قبل از آغاز ارتباط**، نرخ داده بین فرستنده و گیرنده مورد توافق قرار گیرد. این نوع ارتباط سریال کاربردهای فراوانی دارد. پورت COM کامپیوترهای فعلی که ۹ پین دارد، پورت سریال میکروکنترلرها، ارتباط دستگاه کنترل از راه دور با تلویزیون و ... از جمله این کاربردها هستند.

قاب‌بندی داده‌ها در ارتباط سریال

چون در ارتباط سریال داده‌ها به صورت بیت‌به‌بیت پشت سر هم روی سیم ارتباطی ارسال می‌شوند، باید ابتدا و انتهای یک «مجموعه داده» مشخص باشد. به اصطلاح هر مجموعه داده باید «قاب»[1] شده و سپس روی خط داده قرار بگیرد.

قاب‌بندی در ارتباط سریال همگام

در ارتباط سریال همگام، چون زمان خواندن هر بیت اطلاعات از روی خط داده دقیقاً مشخص است، فرستنده می‌تواند هر تعداد بیت را پشت‌سرهم ارسال کند (خواهیم دید که در ارتباط غیرهمگام این‌گونه نیست). به بیان دیگر طول بسته اطلاعاتی می‌تواند بزرگ باشد.

ابتدا و انتهای قاب یا بسته اطلاعاتی با الگوهای خاصی که به ترتیب Preamble و Postamble نام دارند مشخص می‌شود. در واقع گیرنده مرتباً خط داده را می‌خواند تا الگوی Preamble را دریافت کند. سپس تا وقتی که الگوی Postamble را دریافت نکرده خط را می‌خواند و اطلاعات روی آن را ثبت می‌کند.

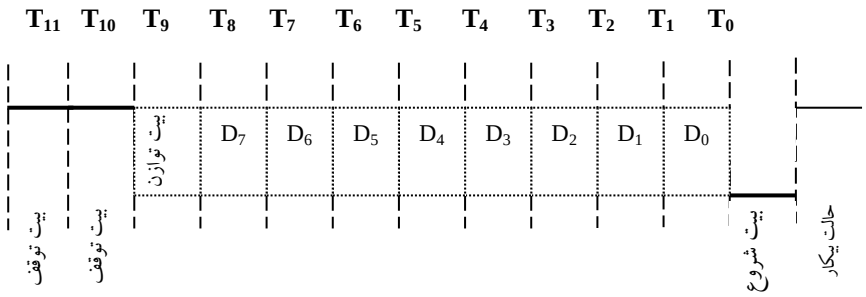
در بعضی طرحهای ارتباط سریال همگام، بعد از الگوی Preamble تعداد بایتهای اطلاعاتی گنجانده شده در بسته توسط فرستنده به گیرنده ابلاغ می‌شود. گیرنده نیز بعد از دریافت Preamble و اطلاعات کنترلی ابتدای بسته، به همان تعدادی که ذکر شده بایتهای اطلاعاتی را از خط می‌خواند.

قاب‌بندی در ارتباط سریال غیرهمگام

همانطور که ذکر شد، در ارتباط غیرهمگام سیگنال ساعت وجود ندارد و نرخ ارسال داده‌ها باید قبل از آغاز ارتباط بین فرستنده و گیرنده مورد توافق قرار گیرد. به همین لحاظ قاب ارتباط سریال غیرهمگام با قاب سریال همگام متفاوت است.

شکل زیر یک قاب داده در ارتباط سریال غیرهمگام را نشان می‌دهد:

[1] Frame



شکل ۷-۲-ارتباط سریال غیرهمگام

خط ارتباطی سریال غیرهمگام در حالت عادی (بیکار) حاوی «یک» است. هنگامی که فرستنده می‌خواهد ارسال اطلاعات را آغاز کند، باید به نحوی گیرنده را مطلع کند. به همین لحاظ در ابتدای قاب داده، یک بیت «صفر» به نام بیت شروع[1] ارسال می‌شود تا گیرنده از تغییر «یک» به «صفر» متوجه آغاز ارسال اطلاعات شود.

به دنبال بیت شروع، تعدادی بیت داده با شروع از کم‌ارزشترین بیت (LSB[2]) ارسال می‌گردد. این تعداد می‌تواند از ۵ بیت (صفحه‌کلیدهای کوچک غیراسکی) تا ۸ بیت (سیستمهای مهمول انتقال داده) باشد. بیت بعدی، بیت توازن[3] برای کشف خطا است که می‌تواند از نوع توازن زوج یا فرد باشد. قاب می‌تواند حاوی بیت توازن نباشد.

در انتهای بسته دو بیت «یک» توسط فرستنده ارسال می‌شود که به بیت توقف[4] معروفند. ارسال این بیتهای توقف به دو هدف انجام می‌شود: اول اینکه به گیرنده فرصت کافی برای پردازش بسته دریافتی داده داده شود. هدف دیگر این است که خط را به حالت عادی (حالت بیکار که حاوی «یک» است) برگرداند تا آغاز بسته بعدی به کمک بیت شروع که «صفر» است بتواند مشخص شود. تعداد بیتهای توقف می‌تواند ۱ یا ۱/۵ یا ۲ بیت باشد که معمولاً به تعداد بیتهای داده وابسته است. در سیستم‌های قدیمی‌تر که کند بودند از دو بیت توقف استفاده می‌شد. در سیستم‌های امروزی استفاده از یک بیت توقف رایج است.

نکته تأمل‌برانگیز در ساختار قاب ارتباط سریال غیرهمگام، تعداد بیتهای داده است که به حداکثر هشت بیت محدود شده است (البته این تعداد می‌تواند از ۵ تا ۸ بیت متغیر باشد). چون در ارتباط غیرهمگام سیگنال ساعت وجود ندارد، فرستنده و گیرنده باید بر سر یک سرعت به توافق برسند و گیرنده با فرض آن سرعت خط داده را بخواند.

فرض کنید سرعت توافق‌شده ۹۶۰۰ بیت در ثانیه است. این بدان معناست که فرستنده در هر ثانیه ۹۶۰۰ بیت ارسال می‌کند؛ بنابراین زمان ارسال هر بیت $\frac{1}{9600}$ ثانیه است. وقتی گیرنده بیت شروع را دریافت می‌کند، متوجه آغاز یک بسته اطلاعاتی می‌شود. پس از سپری شدن زمانی که بیت شروع روی خط قرار دارد ($\frac{1}{9600}$ ثانیه)، گیرنده باید بیتهای بعدی را به موقع (یعنی در «وسط» زمانی که بیت روی خط قرار دارد) از روی خط بخواند. بنابراین گیرنده پس از سپری شدن زمانی که بیت شروع روی خط قرار دارد ($\frac{1}{9600}$ ثانیه)، به اندازه نصف زمان یک بیت ($\frac{1}{9600\times 2}$ ثانیه) صبر

[1] Start Bit
[2] Least Significant Bit
[3] Parity
[4] Stop Bit

می‌کند و سپس بیت D_0 را می‌خواند، $\frac{1}{9600}$ ثانیه بعد بیت D_1 را می‌خواند، $\frac{1}{9600}$ ثانیه بعد بیت D_2 را می‌خواند و ... تا سرانجام به انتهای بسته برسد.

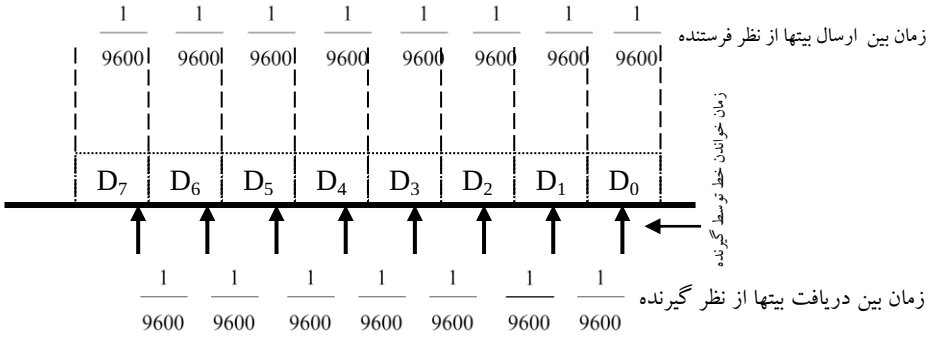
اگر همه چیز طبق برنامه بالا پیش برود، باید فرستنده در هر ثانیه ۹۶۰۰ بیت ارسال و گیرنده این ۹۶۰۰ بیت را به درستی دریافت کند. پس چرا طول قسمت داده قاب اطلاعاتی غیرهمگام محدود به ۸ بیت شده است؟

مشکل اینجاست که همه چیز طبق برنامه جلو نمی‌رود!

فرستنده «با ساعت خودش» یک بیت می‌فرستد، $\frac{1}{9600}$ ثانیه صبر می‌کند، بیت بعدی را می‌فرستد، $\frac{1}{9600}$ ثانیه صبر می‌کند، بیت بعدی را می‌فرستد، $\frac{1}{9600}$ ثانیه صبر می‌کند و ...

گیرنده «با ساعت خودش» یک بیت از روی خط می‌خواند، $\frac{1}{9600}$ ثانیه صبر می‌کند، بیت بعدی را از روی خط می‌خواند، $\frac{1}{9600}$ ثانیه صبر می‌کند، بیت بعدی را از روی خط می‌خواند، $\frac{1}{9600}$ ثانیه صبر می‌کند و ...

مسأله همین جاست که ساعت فرستنده با ساعت گیرنده یکسان نیست. در واقع در سیستم‌های مبتنی بر پردازنده برای تولید پالس ساعت از اجزاء الکترونیکی طبیعی (مانند کریستال) استفاده می‌شود که پیدا کردن دو جزء کاملاً یکسان در طبیعت تقریباً غیرممکن است. به همین لحاظ زمان $\frac{1}{9600}$ ثانیه در فرستنده با زمان $\frac{1}{9600}$ ثانیه در گیرنده متفاوت است و همین اختلاف زمان باعث ایجاد مشکل می‌شود. شکل ۷-۳ نشان می‌دهد اگر ساعت گیرنده سریعتر از ساعت فرستنده کار کند چه مشکلی ایجاد می‌شود:



شکل ۷-۳-مشکل عدم تطابق زمانی در ارتباط سریال غیرهمگام

گیرنده و فرستنده توافق کرده‌اند زمان بین بیتها $\frac{1}{9600}$ ثانیه باشد و هرکدام بر اساس ساعت خود این زمان را اندازه می‌گیرند. همانطور که می‌بینید بعد از خواندن هر بیت، لحظه خواندن خط توسط گیرنده کمی زودتر اتفاق می‌افتد. بعد از دریافت ۸ بیت اطلاعات، لحظه خواندن خط توسط گیرنده از «وسط» زمانی که بیت روی خط دارد فاصله زیادی پیدا کرده و اگر تعداد بیتهای داده بیشتر شود امکان دارد گیرنده زمانی خط را بخواند که هنوز بیت روی خط نیامده است.

پرسش) با رسم شکل توضیح دهید اگر ساعت گیرنده کندتر از ساعت فرستنده کار کند چه مشکلی ایجاد می‌شود؟

بنابرآنچه گفته شد، طول بیتهای داده بسته اطلاعاتی در ارتباط غیرهمگام محدود به ۸ بیت است و در ادامه ممکن است فرستنده و گیرنده از همگامی خارج شوند.

ممکن این سؤال برای شما پیش بیاید که چرا این مشکل در ارتباط همگام وجود ندارد؟

پاسخ واضح است. در ارتباط همگام، فرستنده به کمک سیگنال ساعت که به همراه داده‌ها ارسال می‌کند به گیرنده اعلام می‌کند که چه زمانی باید خط را بخواند. به همین جهت فرستنده و گیرنده هیچگاه از همگامی خارج نمی‌شوند. اما در ارتباط غیرهمگام چون سرعت قبل از ارتباط مورد توافق قرار می‌گیرد و سپس فرستنده و گیرنده هرکدام طبق ساعت خودشان عمل می‌کنند، به علت عدم تطابق کامل بین ساعت فرستنده و ساعت گیرنده، امکان دارد بعد از ارسال تعدادی بیت فرستنده و گیرنده از همگامی خارج شوند. به همین لحاظ، تعداد بیتهای داده به حداکثر ۸ بیت محدود شده است. در بسته اطلاعاتی بعدی، به کمک بیت شروع همگامی مجدداً برقرار می‌شود.

مقایسه بین ارتباطات همگام و غیرهمگام

هرچند هم‌اکنون بیشتر ارتباطات سریال از نوع غیرهمگام هستند، اما مقایسه‌ای بین ارتباطات همگام و غیرهمگام مفید است:

- ✓ در ارتباط همگام طول بسته اطلاعاتی محدود نیست (مگر اینکه محدودیتهایی از ناحیه لایه‌های بالاتر ارتباطی اعمال شده باشد).
- ✓ در ارتباط همگام، فرستنده در حین ارتباط می‌تواند با تغییر سیگنال ساعت، سرعت ارتباط را تغییر دهد (هرچند این کار معمول نیست). اما در ارتباط غیرهمگام سرعت باید از ابتدا بین فرستنده و گیرنده مورد توافق قرار بگیرد.
- ✓ سربار[1] اطلاعات در ارتباط غیرهمگام بالا است. در این نوع ارتباط برای ارسال ۸ بیت اطلاعات باید یک قاب ۱۲ بیتی ارسال شود که ۴ بیت آن سربار است. بنابراین تنها از $\frac{8}{12} = 66\%$ پهنای باند خط استفاده مفید می‌شود. در ارتباط همگام تعداد بیتهای داده بیشتر است و به کارایی بیشتری در استفاده از خط ارتباطی می‌انجامد.
- ✓ در ارتباط غیرهمگام نیازی به سیگنال ساعت (CLK) نیست که این موضوع در کاهش هزینه ارتباط مؤثر است. به علاوه در ارتباطات بیسیم (مانند دستگاه کنترل از راه دور تلویزیون که از سیگنالهای مادون قرمز استفاده می‌کند یا دزدگیر رادیویی) ارسال دو سیگنال داده و ساعت منطقی به نظر نمی‌رسد.

کنترل جریان داده

پیشتر دیدیم فرستنده باید حجم اطلاعات ارسالی را کنترل کند تا به گیرنده فرصت پردازش اطلاعات بدهد. این موضوع را کنترل جریان داده نامیدیم.

آیا کنترل جریان داده بحثی مجزا از تطابق سرعت بین فرستنده و گیرنده است؟ به بیان دیگر، اگر سرعت فرستنده و گیرنده با هم تطابق داشته باشد، باز هم نگرانی در مورد کنترل جریان داده وجود دارد؟

پاسخ مثبت است. برای مثال فرض کنید شخصی اطلاعاتی برای شما می‌خواند تا یادداشت کنید. ممکن است آن شخص آنقدر تند صحبت کند که شما اصلاً متوجه حرفهای او نشوید. این مشکل **عدم تطابق سرعت بین فرستنده و گیرنده** است. اما در حالتی دیگر ممکن است آن شخص طوری صحبت کند که شما متوجه حرفهای او بشوید، اما آنقدر اطلاعات را به سرعت بخواند که شما موفق به یادداشت کردن اطلاعات نشوید. این مشکل **عدم کنترل صحیح**

[1] Overhead

جریان داده است. بنابراین در ارتباط سریال، هم تطابق سرعت بین فرستنده و گیرنده و هم راهکاری برای کنترل جریان داده لازم است.

در ارتباطات سریال غیرهمگام دو نوع کنترل جریان داده وجود دارد. یکی روش سخت‌افزاری است که پس از معرفی درگاه سریال کامپیوتر به آن خواهیم پرداخت.

روش دیگر کنترل جریان داده به شیوه نرم‌افزاری است. در این روش، هرگاه بافر گیرنده در شرف پر شدن باشد و نتواند اطلاعات جدید را ثبت کند، یک کاراکتر ویژه به نام Xoff[۱] برای فرستنده ارسال می‌کند. فرستنده با دریافت این کاراکتر، ارسال اطلاعات را متوقف می‌کند و صبر می‌کند تا گیرنده آمادگی خویش برای دریافت اطلاعات جدید را با ارسال کاراکتری ویژه به نام Xon[۲] اعلام کند. این روش کنترل جریان داده به Xon/Xoff موسوم است.

مشخصات ارتباط غیرهمگام

به صورت خلاصه در ارتباط غیرهمگام باید مشخصات زیر بین فرستنده و گیرنده مورد توافق قرار بگیرد:

- ✓ نرخ داده (مثلاً ۹۶۰۰ یا ۱۹۲۰۰ بیت در ثانیه)
- ✓ تعداد بیت‌های داده (۵ یا ۶ یا ۷ یا ۸ بیت)
- ✓ نوع توازن به کار رفته (زوج، فرد، بدون توزان، همیشه «یک»، همیشه «صفر»)
- ✓ تعداد بیتهای توقف (۱ یا ۱/۵ یا ۲ بیت)
- ✓ روش کنترل جریان داده (سخت‌افزاری، Xon/Xoff یا بدون کنترل جریان داده)

تراشه‌های ارتباط سریال

به لحاظ پیچیدگی‌هایی که در مدیریت ارتباط سریال وجود دارد، تراشه‌هایی وجود دارند که عمل تبدیل داده‌های موازی به سریال و بالعکس را انجام می‌دهند. این تراشه‌ها قابل برنامه‌ریزی هستند و می‌توانند ارتباطات سریال با مشخصات مختلف را حمایت کنند.

یک دسته از این تراشه‌ها UART[۳] نامیده می‌شوند که برای ارتباطات غیرهمگام به کار می‌روند. از جمله این تراشه‌ها می‌توان به 8250 یا 16450 اشاره کرد. اکثر میکروکنترلرها یک یا دو کنترل‌کننده UART داخل خود دارند.

دسته دیگر از این تراشه‌ها USART[۴] نام دارند و برای مدیریت هر دو نوع ارتباط همگام و غیرهمگام به کار می‌روند که تراشه 8251 از جمله آنهاست.

در گذشته، کامپیوترها از هر دو نوع ارتباط همگام و غیرهمگام حمایت می‌کردند. اما در بوردهای کامپیوتر امروزی، تنها یک نوع پورت سریال یافته می‌شود که ویژه ارتباط غیرهمگام است. برای مدیریت این ارتباط به یک تراشه UART نیاز است. مودمهای خارجی که به پورت سریال متصل می‌شوند این تراشه را دارند؛ اما مودمهای ارزان‌قیمت فاقد این تراشه هستند و پردازنده کامپیوتر مدیریت ارتباط سریال را انجام می‌دهد که باعث کاهش سرعت ارتباط می‌شود (این

---

[۱] کاراکتر اسکی DC3 که با فشردن کلیدهای Ctrl+S تولید می‌شود.

[۲] کاراکتر اسکی DC1 که با فشردن کلیدهای Ctrl+Q تولید می‌شود.

[3] Universal Asynchronous Reciever/Transmitter

[4] Universal Synchronous/ Asynchronous Reciever/Transmitter

مودمها در اصطلاح WinModem یا مودمهای نرم‌افزاری نامیده می‌شوند). در کامپیوترهای امروزی تراشه UART در تراشه SUPER I/O یا South Bridge که تراشه اصلی ارتباطی بورد اصلی کامپیوتر است، گنجانده می‌شود. جدول زیر ویژگیهای تراشه‌های UART گوناگونی که در بوردهای کامپیوتر استفاده می‌شوند را نشان می‌دهد.

نوع UART	حداکثر سرعت	طول بافر	نوع سیستم
۸۲۵۰	بالای 9600 bps	–	(XT)۸۰۸۸
۸۲۵۰A	بالای 9600 bps	–	(XT)۸۰۸۸
۸۲۵۰B	بالای 9600 bps	–	۸۰۸۸ و ۸۰۲۸۶
۱۶۴۵۰	بالای 19200 bps	۱ بایت	۸۰۳۸۶ و ۸۰۴۸۶
۱۶۵۵۰	بالای 115200 bps	۱۶ بایت	۸۰۳۸۶ و ۸۰۴۸۶ و پنتیوم
۱۶۶۵۰	بالای 230000 bps	۳۲ بایت	کارتهای خاص I/O و ترمینالهای داخلی ISDN
۱۶۷۵۰	بالای 460000 bps	۶۴ بایت	کارتهای خاص I/O و ترمینالهای داخلی ISDN
۱۶۹۵۰	بالای 921.6 kbps	۱۲۸ بایت	کارتهای خاص I/O و ترمینالهای داخلی ISDN

سه تراشه آخر از استانداردی به نام ESP[1] (پورت سریال با سرعت بالا) پیروی می‌کنند.

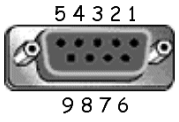
در ذیل به مشخصات پورت سریال غیرهمگام کامپیوتر و نحوه تبادل اطلاعات با آن می‌پردازیم.

پورت سریال کامپیوتر

اگر به پشت بورد اصلی کامپیوتر خود نگاه کنید، یک پورت با ۹ پایه مانند شکل مقابل می‌بینید که به پورت سریال، پورت COM[2] یا در اصطلاح فنی الکترونیک به پورت DB9 معروف است. به کمک این پورت، می‌توان یک ارتباط سریال غیرهمگام با مشخصات قابل تغییر داشت[3].

در ادامه به بررسی سیگنالهای پورت سریال کامپیوتر می‌پردازیم. با شناخت این سیگنالها دو مطلب را خواهیم آموخت؛ اول چگونگی عملکرد مودمهای خارجی[4] در تعامل با کامپیوتر و دوم نحوه کنترل یک دستگاه جانبی به کمک پورت سریال کامپیوتر.

پیش از توضیح عملکرد پایه‌ها، توضیحی راجع به نحوه شماره‌گذاری پایه‌ها لازم است. پورتی که پشت کامپیوتر مشاهده می‌کنید، دارای ۹ **میله** است که سیگنالها را در اختیار کاربر قرار می‌دهد[5]. شماره‌گذاری این حفره‌ها به صورت مقابل است:



معمولاً برای استفاده از این پورت، قطعه‌ای به کار می‌رود که به جای میله دارای **حفره** است و طوری روی پورت سریال کامپیوتر نصب می‌شود که میله‌ها درون حفره‌ها قرار می‌گیرند[6]. این قطعه

---

[1] Enhanced Serial Port

[2] COMmunication – این نام به دلیل کاربرد این پورت در ارتباطات و مخابرات به آن داده شده است.

[3] پیشتر در بورد اصلی کامپیوتر یک پورت ۲۵ پین سریال وجود داشت که از روش همگام برای ارتباط استفاده می‌کرد. امروزه این نوع پورت را بیشتر در دستگاههای صنعتی می‌توان یافت.

[4] External

[5] این نوع پورت معمولاً به Plug یا Male معروف است.

[6] این نوع پورت معمولاً به Socket یا Female معروف است.

نیز به نام DB9 معروف است. شکل مقابل را ببینید. سر کابل مودم خارجی (یا دستگاههای دیگر مانند اسکنر که به پورت سریال متصل می‌شوند) نیز چنین شکلی دارد.

دقت کنید که در پورت اصلی کامپیوتر، پایه بالا سمت چپ و در قطعه دومی که روی پورت نصب می شود، پایه بالا سمت راست شماره یک است. چون وقتی قطعه بالا روی پورت سریال نصب می‌شود، سمت چپ آن با سمت راست پورت مقابل هم قرار می گیرد. پشت این قطعه در محل هر میله، جایی برای لحیم کردن سیم تعبیه شده تا به راحتی بتوانید از سیگنالهای پورت سریال استفاده کنید. شکل بالا را ببینید.

استاندارد ارتباط سریال کامپیوتر

در سال ۱۹۶۰ سازمان صنایع الکترونیک (EIA[1]) برای استاندارد کردن ارتباط سریال بین وسایل مختلف (از جمله کامپیوترها) پروتکلی پیشنهاد کرد که به EIA232 یا RS232 معروف شد[2]. بخش سخت‌افزاری این پروتکل شامل سیگنالهای داده و کنترلی است که در ادامه با آشنا خواهیم شد.

ولتاژهای منطقی در RS232

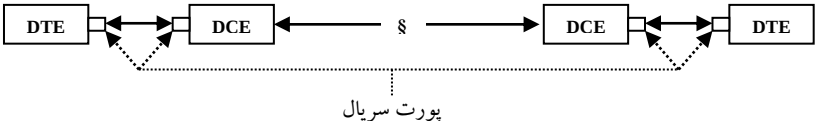
قبلاً آموختیم که اگر ولتاژی بین صفر تا ۰/۸ ولت باشد از آن به «صفر منطقی» و اگر بین ۳ تا ۵ ولت باشد از آن به «یک منطقی» تعبیر می‌شود[3]. چون RS232 قبل از TTL استاندارد شد، سطوح ولتاژ منطقی آن با TTL متفاوت است. در RS232 (که پورت سریال کامپیوتر نمونه‌ای از کاربرد آن است)، ولتاژی بین ۳- ولت تا ۲۵- ولت به عنوان «یک» و ولتاژی بین ۳+ ولت تا ۲۵+ ولت به عنوان «صفر» منطقی در نظر گرفته می‌شود.

توجه کنید چون معمولاً تراشه‌هایی با استاندارد TTL به پورت سریال کامپیوتر متصل می‌شود (مانند تراشه‌های مدیریت ارتباط سریال یا میکروکنترلرها)، به تراشه‌ای نیاز داریم که ولتاژهای RS232 را به TTL و بالعکس تبدیل کند. این تراشه‌ها Line Driver نامیده می‌شوند که MAX232 و MAX233 نمونه‌ای از آن هستند.

یکی از مزایای پورت سریال این است که برای دستگاههای کم‌مصرف که به آن متصل شوند، می‌تواند تغذیه مورد نیاز را تأمین کند. جریان خروجی پورت سریال کامپیوتر بین ۱۰ تا ۲۰ میلی آمپر و مقاومت ورودی ۱۰ کیلو اهم است.

سیگنالهای RS232

چون پورت سریال از ابتدا برای اتصال مودم خارجی به کامپیوتر استاندارد شد، سیگنالهای این پورت نیز با نامهایی که در ارتباط کامپیوتر و مودم به کار می‌روند نامگذاری شده‌اند. برای فهم این سیگنالها ابتدا به نحوه برقراری ارتباط بین دو کامپیوتر به کمک مودم خارجی می‌پردازیم. شکل ۷-۴ این ارتباط را نشان می‌دهد.



شکل ۷-۴-ارتباط سریال به کمک مودم

[1] Electronics Industries Association

[2] کامپیوترهای اپل مکینتاش (که دارای پردازنده موتورولا هستند) دارای رابط سریال با استانداردی به نام RS422 هستند.

[3] این منطق ولتاژ موسوم به TTL است. در منطق‌های ولتاژی دیگر مانند CMOS، ولتاژهای دیگری به «صفر» و «یک» منطقی نسبت داده می‌شوند.

منظور از DTE[1] هر دستگاهی است که بخواهد اطلاعاتی ارسال یا دریافت کند. کامپیوتر یا میکروکنترلر مثالی از DTE هستند.

DCE[2] دستگاهی است که وظیفه آن دریافت داده‌ها از DTE و ارسال آن روی خط داده یا دریافت داده‌ها از روی خط انتقال ارائه آن به DTE است. مودم مثال خوبی از DCE است.

برای ارسال داده‌ها از DTE اولی به DTE دوم روند زیر دنبال می‌شود:

- ✓ ارسال داده‌ها از DTE اول به DCE اول
- ✓ ارسال داده‌ها از DCE اول به DCE دوم
- ✓ ارسال داده‌ها از DCE دوم به DTE دوم

اکنون ببینیم پورت سریال چگونه به عنوان رابط بین DTE و DCE عمل می‌کند. جدول زیر سیگنالهای پورت سریال کامپیوتر را نشان می‌دهد:

شماره پایه	عملکرد	جهت سیگنال (از دید کامپیوتر)	نوع سیگنال
۱	**Data Carrier Detect (DCD)**	ورودی	کنترل
۲	**Received Data (RxD)**	ورودی	داده
۳	**Transmitted Data (TxD)**	خروجی	داده
۴	**Data Terminal Ready (DTR)**	خروجی	کنترل
۵	**Ground (Gnd)**	–	–
۶	**Data Set Ready (DSR)**	ورودی	کنترل
۷	**Request To Send (RTS)**	خروجی	کنترل
۸	**Clear To Send (CTS)**	ورودی	کنترل
۹	**Ring Indicator (RI)**	ورودی	کنترل

سیگنالهای فوق را می‌توان به سه دسته کلی تقسیم کرد.

الف) سیگنالهای داده

پایه RxD

کامپیوتر از طریق این پایه، اطلاعات را به صورت سریال از مودم دریافت می‌کند.

پایه TxD

کامپیوتر از طریق این پایه، اطلاعات را به صورت سریال برای مودم ارسال می‌کند.

پایه Gnd

زمین پورت سریال است که باید با زمین DCE مشترک شود.

[1] Data Terminal Equipment
[2] Data Communication Equipment

ب) سیگنالهای اعلام آمادگی ارتباط

پایه DCD

وقتی مودم با مودم طرف مقابل ارتباط برقرار می‌کند و سیگنال حامل[1] معتبر روی خط کشف می‌کند، از طریق این پایه مراتب را به کامپیوتر سمت خودش اطلاع می‌دهد.

پایه DTR

وقتی کامپیوتر آماده ارتباط با مودم است این پایه را فعال می‌کند.

پایه DSR

وقتی مودم آماده ارتباط با کامپیوتر است این پایه را فعال می‌کند.

پایه RI

وقتی مودم سیگنال زنگ (Ring) را روی خط ارتباطی با مودم دیگر دریافت می‌کند، از طریق این پایه به کامپیوتر اطلاع می‌دهد تا به خط جواب بدهد.

ج) سیگنالهای دست‌دهی برای کنترل جریان داده

پایه RTS

هنگامی که کامپیوتر می خواهد برای مودم یک بایت بفرستد، با فعال کردن این پایه از مودم درخواست اجازه می‌کند.

پایه CTS

وقتی مودم سیگنال RTS را از کامپیوتر دریافت کرد، در صورت آمادگی برای دریافت داده، سیگنال CTS را برای کامپیوتر ارسال می‌کند.

پرسش) روش کنترل جریان داده به روش سخت‌افزاری (Hardware Flow Control) به کمک سیگنالهای RTS و CTS را توضیح دهید.

توجه کنید در توضیحات بالا کامپیوتر مثالی از DTE و مودم مثالی از DCE است. اصول ارتباط هر DTE با هر DCE طبق استاندارد RS232 مانند آنچه گفته شد می‌باشد.

به صورت خلاصه می‌توان گفت برای ارتباط بین دو DTE به کمک DCE:

- ✓ به کمک سیگنالهای DTR و DSR و DCD و RI ارتباط برقرار می‌شود.
- ✓ به کمک سیگنالهای RxD و TxD داده‌ها رد و بدل می‌شوند.
- ✓ به کمک سیگنالهای RTS و CTS کنترل جریان داده بین DTE و DCE انجام می‌شود.

[1] Carrier

مودم پوچ (Null Modem)

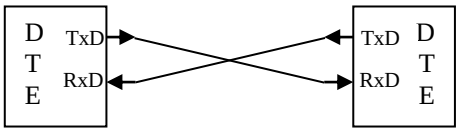
در بخش قبل دیدیم چگونه می‌توان دو DTE را به کمک دو DCE به هم متصل کرد. حال فرض کنید می‌خواهیم دو DTE (مثلاً دو کامپیوتر، دو میکروکنترلر یا یک کامپیوتر و یک میکروکنترلر) را به یکدیگر متصل کنیم، اما DCE (مثلاً مودم) در اختیار نداریم. در این حالت، باید با توجه به عملکرد سیگنالهای RS232، طوری بین پورتهای سریال دو DTE سیم‌بندی کنیم که تصور کنند از طریق DCE بهم متصل هستند، در حالی که عملاً DCE بین آنها وجود ندارد. به این اتصال، مودم پوچ گفته می‌شود.

سیم‌بندی بین پورتهای سریال برای شبیه‌سازی وجود DCE چگونه باید باشد؟

به عنوان مثال، DTE اول می‌خواهد برای DTE دوم اطلاعات ارسال کند. اگر DCE وجود داشته باشد:

- ✓ DTE اول از طریق خط TxD خودش اطلاعات را برای DCE خودش ارسال می‌کند
- ✓ DCE اطلاعات را برای DCE مقابل ارسال می‌کند.
- ✓ DCE مقابل اطلاعات را برای DTE دوم می‌فرستد که از طریق سیگنال RxD پورت سریال DTE دریافت می‌شود.

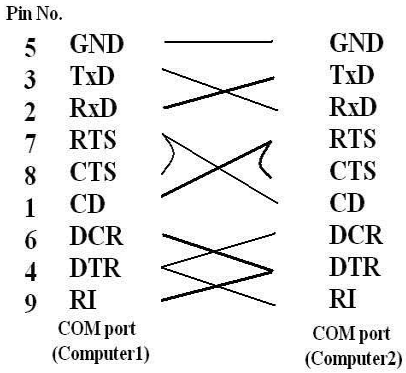
حال که DCE وجود ندارد، بدیهی است که باید خط RxD هر DTE به خط TxD در DTE مقابل و بالعکس متصل شود.



شکل مقابل ساده‌ترین نوع مودم پوچ را نشان می‌دهد. بدیهی است زمین هر دو DTE باید مشترک باشد؛ به همین لحاظ سیگنال Gnd دو پورت سریال به هم متصل می‌شود.

تکلیف سیگنالهای دست‌دهی چیست؟

گفتیم هنگامی که DTE می‌خواهد برای DCE یک بایت بفرستد، با فعال کردن پایه RTS از وی درخواست اجازه می‌کند. وقتی DCE سیگنال RTS را از DTE دریافت کرد، در صورت آمادگی برای دریافت داده، سیگنال CTS را برای DTE ارسال می‌کند. حال که DCE وجود ندارد که DTE بخواهد از آن اجازه بگیرد، یعنی این اجازه باید همیشه داده شود، به همین لحاظ در ساختار مودم پوچ، RTS هر DTE به CTS آن متصل می‌شود. یعنی هرگاه DTE برای اجازه گرفتن از DCE سیگنال RTS را فعال کند، بلافاصله سیگنال CTS خودش فعال می‌شود و DTE تصور می‌کند که اجازه از سوی DCE صادر شده است.



شکل مقابل ساختار کامل مودم پوچ برای ارتباط بین پورت سریال دو کامپیوتر را نشان می‌دهد.

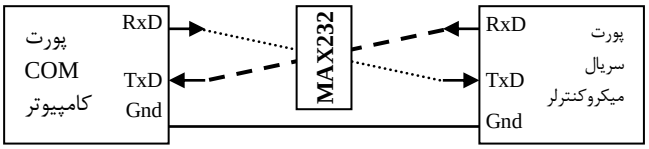
کابل CAT-5 که در سیم‌کشی شبکه LAN معمول است، برای برقراری این ارتباط مناسب می‌باشد. چون ۴ زوج سیم و یک سیم بدنه (شیلد) دارد که می‌توان هر ۹ اتصال را به کمک آن برقرار کرد. به کمک دو قطعه DB9 که روی پورت سریال دو کامپیوتر نصب شوند و مقداری سیم CAT-5

می‌توان ارتباط مودم پوچ را بین دو کامپیوتر برقرار کرد. در این حالت به کمک ارتباط نرم‌افزاری مناسب (که بعدها خواهیم دید) می‌توان ارتباط داده‌ای و حتی شبکه بین دو کامپیوتر برقرار کرد.

پرسش) با بررسی ساختار کامل مودم پوچ، دلیل سیم‌بندی‌های ذکر شده را توضیح دهید.

اتصال میکروکنترلر به کامپیوتر

اکثر میکروکنترلرها دارای کنترل‌کننده ارتباط سریال غیرهمگام (UART) هستند. برای ارتباط بین میکروکنترلر و کامپیوتر می‌توان از ساختار مودم پوچ به صورت زیر استفاده کرد:



شکل ۷-۵-ارتباط میکروکنترلر با کامپیوتر به کمک مودم پوچ

شرح برنامه میکروکنترلر برای ارتباط سریال به دلیل تنوع میکروکنترلرها در حوصله بحث ما نیست. تنها به یاد داشته باشید که مشخصات ارتباط غیرهمگام (نرخ داده، تعداد بیت‌های داده، نوع توازن به کار رفته، تعداد بیتهای توقف، روش کنترل جریان داده) در کامپیوتر و میکروکنترلر باید یکسان باشد.

نرم‌افزار کامپیوتر برای ارتباط با پورت سریال

اکنون اتصال سخت‌افزاری بین کامپیوتر و DTE دیگری (کامپیوتر دیگر یا میکروکنترلر یا ...) برقرار شده است. می‌خواهیم در کامپیوتر برنامه‌ای بنویسیم که ارتباط سریال را با DTE مقابل برقرار کند؛ یعنی اطلاعات را به صورت سریال ارسال و دریافت کند.

برای برقراری ارتباط نرم‌افزاری در ابتدا باید ارتباط سریال پیکربندی شود؛ یعنی نرخ داده، تعداد بیت‌های داده، نوع توازن به کار رفته، تعداد بیتهای توقف و روش کنترل جریان داده مشخص شود. این مشخصات عیناً در نرم‌افزار DTE طرف مقابل برای پیکربندی پورت سریال آن نیز باید به کار برود.

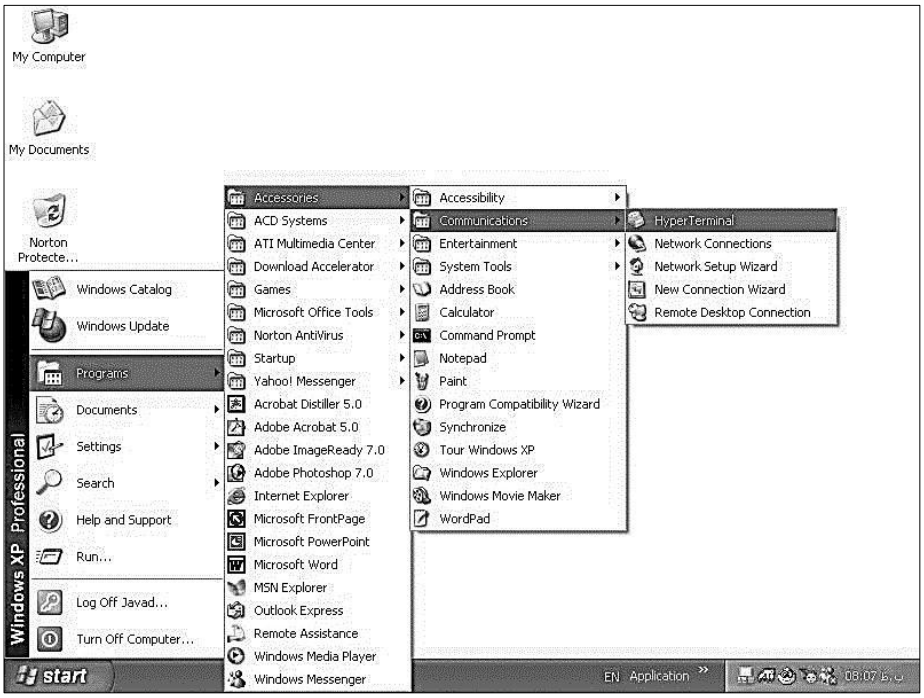
نرم‌افزار ارتباط سریال در کامپیوتر می‌تواند به کمک زبانهای سطح بالا مانند C یا Delphi یا Visual C++ یا ... نوشته شود که در ادامه مثالهایی در DOS و ویندوز خواهیم دید.

اما راه ساده‌تری نیز وجود دارد؛ ویندوز دارای نرم‌افزاری به نام HyperTerminal است که می‌تواند ارتباطهای مختلف از جمله ارتباط سریال را به راحتی برقرار کند. ابتدا با این نرم‌افزار آشنا می‌شویم.

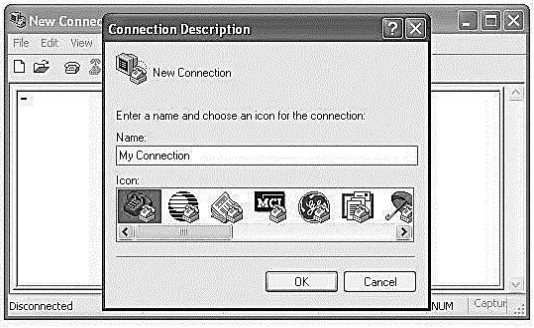
نرم‌افزار HyperTerminal

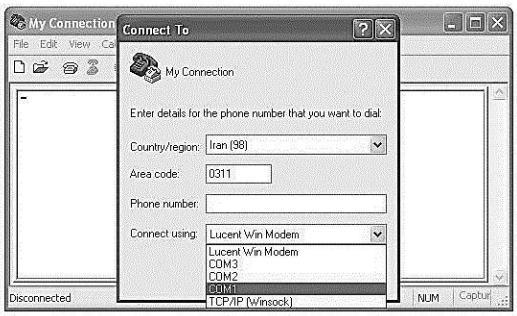
این نرم‌افزار در نسخه‌های مختلف ویندوز وجود دارد. برای استفاده از آن (در ویندوز XP) به ترتیب زیر عمل کنید:

۱) از منوی Start ویندوز به مسیر Programs/Accessories/Communications بروید و گزینه HyperTerminal را انتخاب کنید (شکل ۷-۶).



شکل ۷-۶- ورود به محیط HyperTerminal





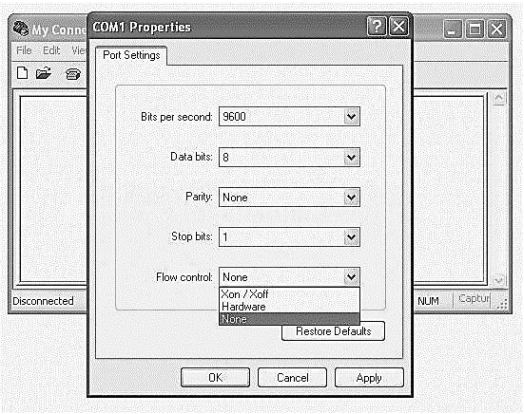
۲) مرحله بعد مانند شکل مقابل است که در آن باید نام ارتباط را انتخاب کنید. در این شکل نام My Connection برای ارتباط انتخاب شده است.

۳) در مرحله بعد باید وسیله ارتباطی را انتخاب کنید. این وسیله ارتباطی می‌تواند کارت شبکه، کارت مودم یا پورتهای سریال باشد. پورت COM1 را انتخاب کنید. توجه کنید که کامپیوتر دارای یک یا دو پورت سریال سخت‌افزاری (COM1 و احتمالاً COM2) است که برای اتصال تجهیزات خارجی به کار می‌روند. به علاوه تعدادی پورت سریال داخلی (COM3، COM4 و ...) نیز وجود دارد که برای اتصال کارت مودمهای داخلی استفاده می‌شوند. در بعضی متون به این پورتهای سریال، پورتهای مجازی[1] گفته می‌شود.

۴) در مرحله بعد باید ویژگیهای ارتباط سریال را تنظیم کنید. در ارتباط مقابل نرخ داده برابر ۹۶۰۰ بیت در ثانیه، تعداد بیتهای داده برابر ۸ بیت، بدون بیت توازن، با یک بیت توقف انتخاب شده است. توجه کنید که پورت سریال کامپیوتر از نرخهای ارتباط داده محدودی مانند ۱۱۰، ۳۰۰، ۶۰۰، ۱۲۰۰، ۲۴۰۰، ۴۸۰۰، ۹۶۰۰ و ۱۹۲۰۰ بیت در ثانیه حمایت می‌کند و دستگاه متصل به این پورت باید نرخ داده مورد نظرش را این لیست انتخاب کند[2].

[1] Virtual

[2] دلیل استفاده از کریستال بسیار دقیق 11.0592MHz برای میکروکنترلر ۸۰۵۱ هنگامی که می‌خواهد با پورت سریال کامپیوتر ارتباط برقرار کند همین موضوع است.



برای کنترل جریان داده می‌توان از روشهای سخت‌افزاری یا نرم‌افزاری (Xon/Xoff) استفاده کرد. اگر DTE مقابل از یکی از این روشها حمایت کند (مانند میکروکنترلر AWC86 که از روش سخت‌افزاری یا RTS/CTS حمایت می‌کند)، می‌توان روش مناسب را برگزید. اما برای ارتباطات ساده، لازم به استفاده از کنترل جریان داده نیست و باید گزینه None را در قسمت Flow Control برگزید.

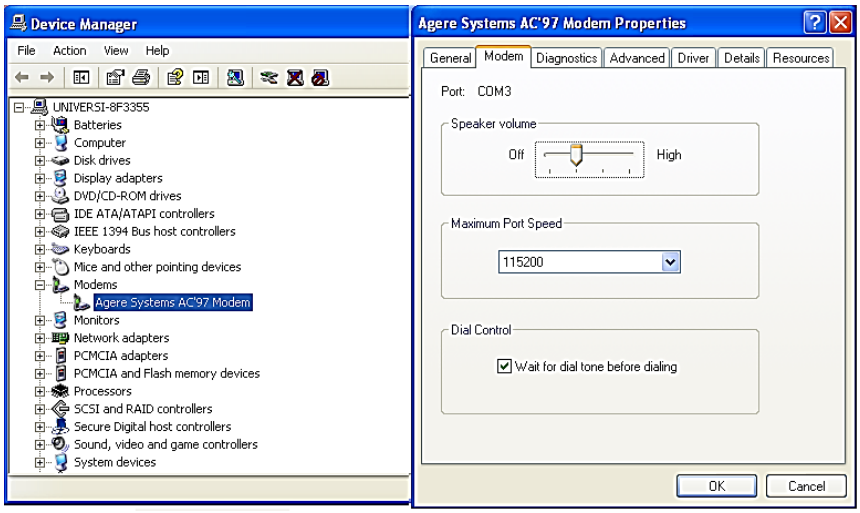
پس از این مراحل، وارد محیط HyperTerminal می‌شوید. از این پس هرچه در این محیط تایپ کنید، با مشخصات تنظیم‌شده از پورت سریال کامپیوتر به بیرون فرستاده می‌شود. توجه کنید آنچه از پورت بیرون فرستاده می‌شود، کد اسکی کاراکترهایی است که تایپ می‌کنید. به علاوه آنچه تایپ می‌کنید در محیط HyperTerminal نشان داده نمی‌شود.

از سوی دیگر، هر اطلاعاتی به صورت سریال با مشخصات تنظیم‌شده به پورت سریال کامپیوتر برسد، در محیط HyperTerminal نمایش داده می‌شود. نحوه نمایش این اطلاعات در HyperTerminal قابل تنظیم است. در حالت پیش‌فرض، هر عددی دریافت شود، کاراکتری که کد اسکی آن برابر آن عدد است در محیط HyperTerminal نشان داده می‌شود.

اگر پورت COM دو کامپیوتر را به کمک کابل مودم پوچ (که ساختار آن پیشتر گفته شد) به هم وصل کنید، هرچه در محیط HyperTerminal یک کامپیوتر تایپ کنید در محیط HyperTerminal کامپیوتر دیگر دیده می‌شود. به علاوه به کمک منوی Transfer می‌توانید بین دو کامپیوتر فایل نیز رد و بدل کنید.

اتصال مودم

همانگونه که گفته شد، کامپیوتر دارای تعدادی پورت COM داخلی (COM3 و COM4 و ...) است که برای اتصال مجازی کارتهای مودم داخلی به پورت سریال استفاده می‌شوند. با انتخاب گزینه Modem در Device Manager می‌توانید ببینید مودم کامپیوتر شما به کدام پورت COM مربوط شده است.

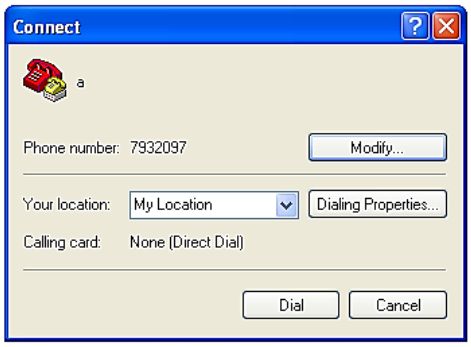


شکل ۷-۷-مودم مجازی

ارتباط دو کامپیوتر به کمک خط تلفن و HyperTerminal

حتماً برای شما پیش آمده که نیاز به فایلی داشته باشید که دوستتان باید آن را از روی کامپیوترش برای شما بفرستد. ساده‌ترین راه این مسأله چیست؟ خواهید گفت اینترنت! حق با شماست. اما اگر شما یا دوستتان به اینترنت دسترسی نداشته باشید چه باید کرد؟ نگران نباشید! به کمک HyperTerminal و مودم، می‌توانید با استفاده از خطوط تلفن اطلاعات را با کامپیوتر دوستتان مبادله کنید.

برای این کار، یکی از دو کامپیوتر باید به عنوان تماس‌گیرنده و دیگری به عنوان پذیرنده تماس مشخص شود. فرض کنید شما تماس‌گیرنده باشید. برای این کار مانند آنچه قبلاً گفته شد، وارد محیط HyperTermianl شوید، نامی برای ارتباط انتخاب کنید و به مرحله بعد بروید.



در این مرحله، از ابزارهای مختلف ارتباطی، مودم را انتخاب کنید و در کادرهای بالا، شماره و کد تلفن منزل دوستتان را وارد کنید. با انتخاب گزینه OK به مرحله بعد بروید. اگر اکنون گزینه Dial را انتخاب کنید، کامپیوتر شما به کمک مودم با شماره تلفنی که مشخص کرده‌اید (در مثال بالا ۷۹۳۲۰۹۷) تماس می‌گیرد و می‌خواهد با مودم مقابل تماس ارتباطی برقرار کند.

قبل از این کار، باید دوست شما در کامپیوتر منزل خودش همین مراحل را طی کند و با رسیدن به مرحله بالا، با انتخاب گزینه Cancel وارد محیط HyperTerminal شود. در ادامه، دوست شما باید از منوی Call گزینه Wait for Call را انتخاب کند تا مودم کامپیوتر منتظر تماس تلفنی شما بماند.

اکنون شما باید گزینه Dial را در HyperTerminal کامپیوتر خودتان انتخاب کنید. وقتی مودم کامپیوتر شما با تلفن منزل دوستتان تماس می‌گیرد، زنگ تلفن منزل او به صدا درمی‌آید. اگر کسی در منزل او گوشی تلفن را بردارد، جز سر و صدای گوشخراش مودم کامپیوتر شما چیزی نمی‌شنود! اگر کسی به تلفن جواب ندهد، مودم کامپیوتر دوستتان که منتظر تماس از سوی مودم کامپیوتر شماست گوشی را برمی‌دارد. در این حالت همان سر و صداهایی که هنگام اتصال به اینترنت از مودم کامپیوترتان می‌شنوید به گوش می‌رسد و سپس ارتباط بین دو کامپیوتر به کمک خط تلفن و مودمها برقرار می‌شود.

از این به بعد هرچه شما در HyperTerminal کامپیوتر خودتان تایپ کنید در محیط HyperTerminal کامپیوتر دوستتان دیده می‌شود و عکس این موضوع نیز برقرار است. در ضمن با انتخاب گزینه Transfer می‌توانید برای یکدیگر فایل ارسال کنید.

استفاده از پورت سریال در زبانهای سطح بالا

نرم‌افزار HeyperTerminal تنها برای برقرار کردن ارتباط‌های داده‌ای ساده به کمک ابزارهای ارتباطی کامپیوتر به کار می‌رود. چنانچه بخواهید یک ارتباط هوشمند برقرار کنید (مثلاً نمودار تغییرات دمای اتاق در طول یک شبانه‌روز را روی کامپیوتر مشاهده کنید یا یک روبات را به کمک کامپیوتر کنترل کنید)، دیگر استفاده از HyperTerminal به کار نمی‌آید و باید به کمک برنامه‌ای که در کامپیوتر می‌نویسید این کار را انجام دهید.

اکثر زبانهای سطح بالا از ارتباط با پورت سریال حمایت می‌کنند. برای نوشتن برنامه ارتباط سریال در هر زبانی باید موارد زیر مشخص شود:

- ✓ چگونه پورت سریال باید پیکربندی شود؟ به بیان دیگر ویژگیهای ارتباط سریال مانند نرخ داده، تعداد بیت‌های داده، نوع توازن به کار رفته، تعداد بیتهای توقف و روش کنترل جریان داده باید به چه صورتی مشخص می‌گردد؟
- ✓ ارسال داده به پورت سریال به چه صورت انجام می‌شود؟
- ✓ چگونه می‌توان از دریافت کاراکتر توسط پورت سریال مطمئن شد؟
- ✓ دریافت داده از پورت سریال به چه صورت انجام می‌شود؟

در ادامه به سؤالات بالا در زبان C پاسخ می‌دهیم. توجه داشته باشید که همانطور که قبلاً گفته شد، داستان ارتباط با پورتهای ورودی/خروجی کامپیوتر (از جمله پورت سریال) در DOS و ویندوز متفاوت است. آنچه در ادامه می‌آید، برقراری ارتباط در C تحت DOS را نشان می‌دهد. برای برقراری ارتباط سریال در زبانهای تحت ویندوز باید به شیوه‌ای متفاوت عمل کنید که در بخش برقرار ارتباط با پورتها در ویندوز، به نحوه ارسال و دریافت داده‌ها در زبان Visual C++ خواهیم پرداخت. گفتنی است این مشکل در HyperTerminal وجود ندارد.

ثباتهای پورت سریال

مانند پورت موازی که دارای یک مجموعه ثبات است، هر پورت سریال کامپیوتر نیز یک مجموعه‌ای متشکل از هشت ثبات برای مدیریت ارتباط سریال دارد. آدرس پایه این ثباتها برای پورتهای COM مختلف را در جدول مقابل می‌بینید.

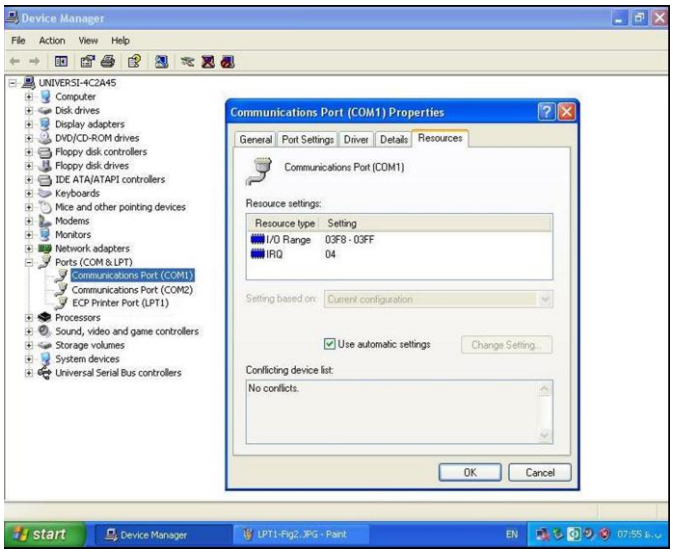
پورت COM	آدرس پورت
COM1	03F8-03FF
COM2	02F8-02FF
COM3	03E8-03EF
COM4	02E8-02EF

در ویندوز می‌توانید پورتهای سریال سخت‌افزاری کامپیوتر خود و نیز آدرس‌های تخصیص داده شده به ثباتهای ویژه پورت را ببینید. برای این کار:

- ✓ از Control Panel گزینه System را انتخاب کنید.
- ✓ به سربرگ Hardware بروید و گزینه Device Manager را انتخاب کنید.

حال در قسمت Ports می‌توانید مشخصات پورتهای کامپیوتر خود را ببینید. شکل ۷-۸، گزینه Ports مربوط به یک کامپیوتر نوعی را نشان می‌دهد.

همانطور که دیده می‌شود، هشت ثبات با شروع از 03F8 h به پورت COM1 اختصاص داده شده است.



شکل ۷-۸- مشخصات یک پورت کامپیوتر

به کمک BIOS نیز می‌توان آدرس مبنای ثباتهای ورودی/خروجی پورت سریال را دانست. در واقع این آدرسها از آفست 0000 به بعد سگمنت 0040 حافظه ذخیره می‌شود. جدول مقابل را ببینید.

پورت COM	آدرس پایه ورودی/خروجی ثباتهای پورت
COM1	0040:0000 – 0040:0001
COM2	0040:0002 – 0040:0003
COM3	0040:0004 – 0040:0005
COM4	0040:0006 – 0040:0007

مثلاً دستور *D 0040:0000 L8* در نرم‌افزار Debug، آدرس پایه پورتهای سریال نصب شده در کامپیوتر را نشان می‌دهد. قبلاً با نرم‌افزار Debug و فرمانهای آن آشنا شدیم.

به کمک برنامه‌نویسی C نیز می‌توان آدرسهای ذخیره شده در این بخش حافظه را مشاهده کرد. برنامه زیر، آدرس پورت COM1 و در صورت عدم نصب پورت مزبور، پیغامی را نمایش می‌دهد.

```
#include    <STDIO.H>
#include    <DOS.H>

void   main(){
    unsigned int far *fptr;
    fptr = (unsigned int far *) 0x00000400;
    if (*fptr > 0)
          printf("COM1 I/O base address = %X",*fptr);
    else
          printf("COM1 not found");
}
```

جدول زیر ثباتهای پورت سریال و آدرس آنها نسبت به آدرس مبنا (Base) را نشان می‌دهد:

آدرس	عملکرد	خواندنی یا نوشتنی	نام	بیت DLAB
Base+0	بافر ارسال اطلاعات	نوشتنی	–	0
	بافر دریافت اطلاعات	خواندنی	–	0
	قسمت پایین نرخ داده	خواندنی/نوشتنی	–	1
Base+1	فعال‌کننده وقفه	خواندنی/نوشتنی	IER	0
	قسمت پایین نرخ داده	خواندنی/نوشتنی	–	1
Base+2	شناسایی وقفه	خواندنی	IIR	–
	کنترل‌کننده FIFO	نوشتنی	FCR	–
Base+3	کنترل‌کننده قالب اطلاعات	خواندنی/نوشتنی	LCR	–
Base+4	کنترل‌کننده مودم	خواندنی/نوشتنی	MCR	–
Base+5	نمایشگر وضعیت خط	خواندنی	LSR	–
Base+6	نمایشگر وضعیت مودم	خواندنی	MSR	–
Base+7	ثبات حافظه پنهان	خواندنی/نوشتنی	–	–

مثلاً اگر آدرس مبنای پورت سریال COM1 برابر 03F8 h باشد، در برنامه کامپیوتر برای ارسال داده به این پورت و نیز دریافت داده از آن باید از ثبات واقع در آدرس 03F8 h استفاده کنیم (به خط اول جدول توجه کنید). با نوشتن داده ۸ بیتی در این ثبات، تراشه UART آن را به صورت مناسب قالب‌بندی کرده و از پورت سریال به بیرون ارسال می‌کند. وقتی نیز بسته‌ای توسط پورت سریال دریافت می‌شود، تراشه UART داده را از دل آن بسته بیرون کشیده و در ثبات فوق قرار می‌دهد. توجه داشته باشید که داده‌های ارسالی و دریافتی در بافرهای جداگانه‌ای از نوع FIFO[1] ذخیره می‌شوند.

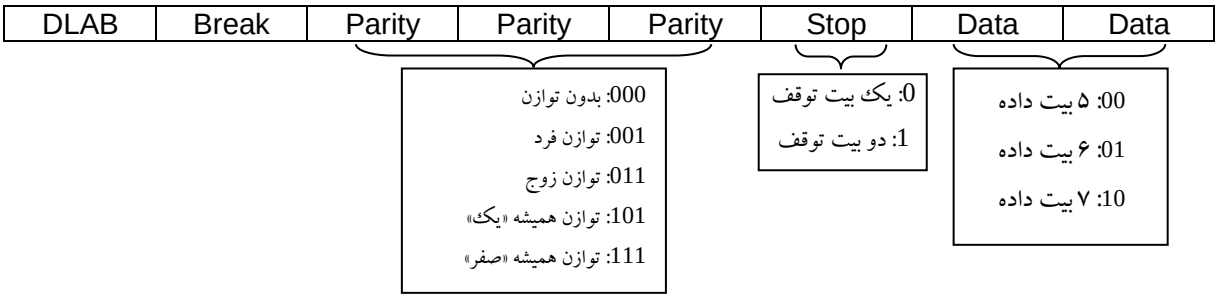
[1] First In First Out – مانند صف

با سایر ثباتها در ادامه آشنا می‌شویم.

پیکربندی ارتباط سریال

گفتیم پیش از آغاز ارتباط، ویژگیهای ارتباط سریال مانند نرخ داده، تعداد بیت‌های داده، نوع توازن به کار رفته و تعداد بیتهای توقف باید مشخص شود. برای مشخص کردن نرخ داده باید عدد ویژه‌ای (وابسته به نرخ داده) را در ثباتهای اول و دوم قرار بدهید. در این حالت باید بیت DLAB (بیت شماره ۷ ثبات کنترل‌کننده قالب اطلاعات) یک باشد.

برای تعیین ویژگیهای دیگر ارتباط سریال باید بیتهای ثبات کنترل‌کننده قالب اطلاعات مقداردهی شوند:



با «یک» شدن بیت قطع (Break)، فرستنده ارسال اطلاعات را قطع می‌کند.

پیکربندی پورت سریال در C (اسمبلی)

در سیستم‌عامل DOS، وقفه 14H برای ارتباط با پورت سریال تعریف شده است که شامل توابعی برای پیکربندی پورت سریال و ارسال و دریافت داده با این پورت است. برای پیکربندی پورت سریال در زبانهای تحت DOS مانند اسمبلی یا C تحت DOS، از تابع شماره ۴ این وقفه استفاده می‌کنیم. با مقداردهی ثباتهای مختلف پردازنده و اجرای این تابع، می‌توان بردون نیاز به درگیر شدن با ثباتهای پورت سریال به پیکربندی دلخواه برای ارتباط سریال دست یافت.

جزییات این تابع در ذیل آمده است:

AH = 04

AL = 00H (توقف)، 01H (بدون توقف)

DX = شماره پورت (COM1 = 0, COM2 = 1,….)

BH = 00H (بدون توازن), 01H (توازن فرد), 02H (توازن زوج), 03H (توازن فرد پایدار) , 04H (توازن زوج پایدار)

BL = 00H (۱ بیت توقف), 01H (۱/۵ بیت توقف برای کلمه ۵ بیتی), 02H (۲ بیت توقف برای کلمه های بزرگتر از ۵ بیت)

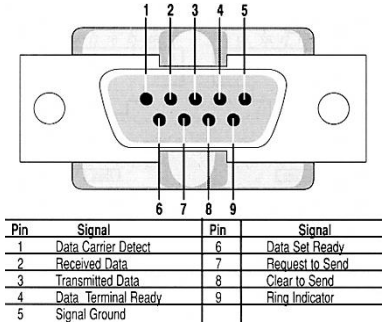
CH = 00H (کلمه ۵ بیتی), 01H (کلمه ۶ بیتی), 02H (کلمه ۷ بیتی), 03H (کلمه ۸ بیتی)

CL = 00H (110 bps), 01 (150 bps), 02 (300 bps), 03 (600 bps), 04 (1200 bps), 05 (2400 bps), 06 (4800 bps), 07 (9600 bps), 08 (19200 bps)

تابع صفحه بعد در زبان C پورت سریال را با نرخ داده ۹۶۰۰ بیت در ثانیه، کلمه ۸ بیتی، یک بیت توقف و بدون استفاده از بیت توازن پیکربندی می‌کند.

```
void InitialSerialPort(void){
      asm {
        mov   ah,04h
        mov   al,00h
        mov   bh,00h  // No Parity
        mov   bl,00h  // 1 Stop Bit
        mov   ch,03h  // 8 bit data
        mov   cl,07h  // 9600 bps
        mov   dx,00h  // port COM1
        int   14h
      }
   }
```

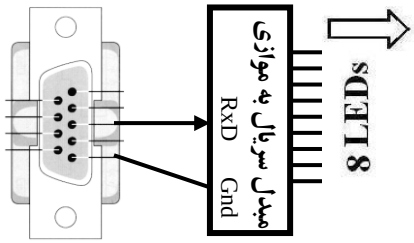
ارسال داده به پورت سریال



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

برای ارسال داده به پورت سریال، مشابه با آنچه قبلاً دیدیم از تابع outportb استفاده می‌شود. البته تابع شماره ۱ وقفه 14h نیز برای این منظور می‌تواند به کار رود. در ادامه یک مثال عملی را بررسی می‌کنیم.

پایه‌های پورت سریال را قبلاً دیدیم. شکل روبرو خلاصه‌ای از آنچه گفتیم و باید در مثالهای طراحی سیستم به کار ببریم را نشان می‌دهد.



فرض کنید می‌خواهیم سیستمی طراحی کنیم که اعداد صفر تا ۲۵۵ را روی هشت عدد LED نمایش دهد. برای این کار، از پورت سریال کامپیوتر به صورت مقابل استفاده می‌کنیم.

همانطور که می‌بینید، پایه TxD پورت سریال کامپیوتر به پایه RxD مبدل سریال به موازی (مثلاً یک میکروکنترلر) متصل شده است. این مبدل، اطلاعات را به صورت سریال از کامپیوتر دریافت و پس از تبدیل به اطلاعات موازی روی هشت LED نمایش می‌دهد.

برنامه را در زیر می‌بینید.

```
#include <STDIO.H>
#include <CONIO.H>
#include <STDLIB.H>

void InitialSerialPort(void){
   asm {
     mov   ah,04h
     mov   al,00h
     mov   bh,00h
     mov   bl,00h
     mov   ch,03h
     mov   cl,07h  // 9600 bps
     mov   dx,00h
     int   14h
   }
}
void main(void){
  unsigned char data = 0;
  InitialSerialPort();
  while (!kbhit()){
      outportb(0x3F8,data);
      delay(500);
      if (data == 255)
    data = 0;
      else
    data++;
```

```
    }
}
```

این برنامه، اعداد صفر (00000000) تا ۲۵۵ (11111111) را با تأخیر ۵۰۰ میلی‌ثانیه به پورت سریال کامپیوتر ارسال می‌کند. پس از رسیدن متغیر خروجی به ۲۵۵، مجدداً مقدار آن صفر می‌شود. برنامه با فشار یک کلید به پایان می‌رسد.

پرسش) برنامه را به گونه‌ای اصلاح کنید که LEDهای زوج و فرد را به ترتیب و با تأخیر یک ثانیه روشن و خاموش کند و یک رقص نور ایجاد کند. برنامه باید با فشار یک کلید، همه LEDها را خاموش کرده و به پایان برسد.

دریافت داده از پورت سریال

فرض کنید دستگاهی به پورت سریال کامپیوتر متصل است که اطلاعات را به صورت سریال برای کامپیوتر ارسال می‌کند. می‌خواهیم در کامپیوتر برنامه‌ای بنویسیم که این اطلاعات را دریافت کند.

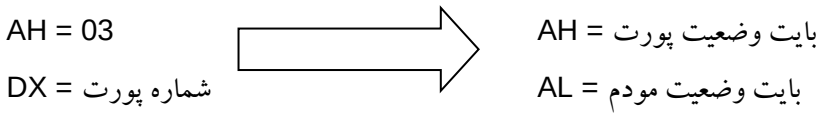
به نظر می‌رسد دستور inportb کافی باشد:

```
data = inportb(0x3F8);
```

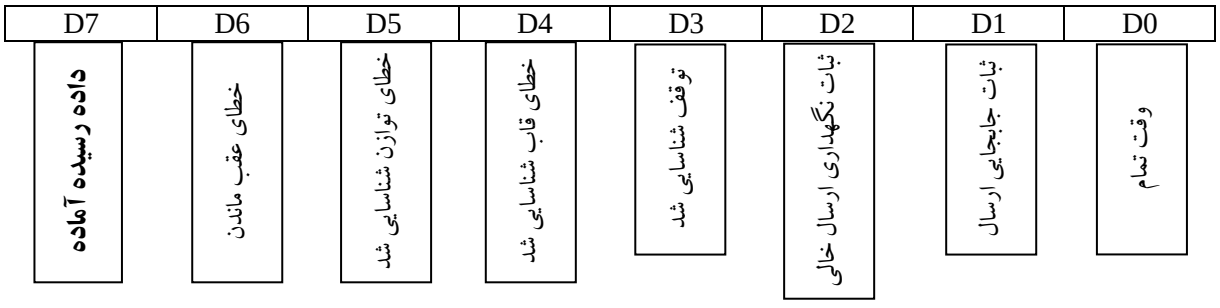
این دستور غلط نیست. یعنی اطلاعات را از پورت COM1 (به آدرس 03F8 h) می‌خواند و در متغیر data ذخیره می‌کند. اما به کار بردن این دستور به تنهایی صحیح نیست. بلکه:

باید ابتدا از دریافت داده توسط پورت سریال مطمئن شوید و سپس داده فوق را بخوانید.

تابع ۳ وقفه 14 h برای خواندن وضعیت پورت به کار می‌رود:



بعد از اجرای این تابع، بایت وضعیت پورت سریال در ثبات AH قرار می‌گیرد که بیتهای آن به شرح زیر هستند:



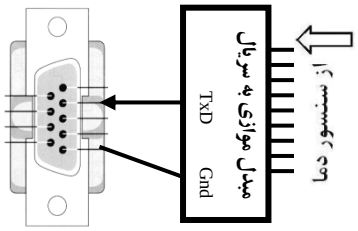
به بیت شماره ۷ دقت کنید. وقتی تابع ۳ وقفه 14 h اجرا شود، با بررسی بیت شماره ۷ ثبات AH می‌توان دانست آیا داده‌ای دریافت شده است یا نه؟ اگر پورت سریال داده‌ای دریافت کرده باشد، این بیت «یک» و در غیر این صورت «صفر» خواهد بود. تابع مقابل بیت دریافت داده را بررسی می‌کند و در صورت آماده بودن داده ارزش منطقی صحیح («یک») و در غیر این صورت ارزش

```
int RecvData(){
  unsigned char status;
  asm {
    mov   ah,03h
    mov   dx,00h
    int   14h
    mov  status,ah
  }
   status = status & 128;
   if (status == 0) return 0;
   else return 1;
}
```

منطقی غلط («صفر») را باز می‌گرداند. این تابع، بایت وضعیت صفحه کلید که در ثبات AH قرار دارد را در متغیر status کپی می‌کند. سپس این م تغیر را با عدد ۱۲۸ (1000000) AND می‌کند. با این کار، چنانچه بیت شماره ۷ متغیر فوق «صفر» باشد، حاصل AND صفر بوده و مقدار «صفر» (ارزش منطقی نادرست) به عنوان خروجی تابع برگردانده می‌شود. در غیر این صورت حاصل AND غیر صفر است و مقدار «یک» (ارزش منطقی درست) به عنوان خروجی به تابع برگردانده می‌شود. بنابراین اگر خروجی تابع RecvData درست باشد یعنی داده‌ای دریافت شده و آماده خوانده شدن است. اکنون به کمک تابع inportb می‌توانید داده را بخوانید. تابع شماره ۲ وقفه 14h نیز برای این منظور قابل استفاده است.

اکنون می‌خواهیم سیستمی طراحی کنیم که دمای اتاق را هر یک ثانیه یک بار خوانده و روی نمایشگر کامپیوتر نمایش دهد. برای این کار، باید از سنسور دما استفاده شود. خروجی سنسور دما که یک کمیت آنالوگ است به کمک تراشه ADC[1] به کمیتی دیجیتال تبدیل می‌شود.

مثلاً با استفاده از یک ADC هشت بیتی (مانند تراشه ADC804)، خروجی آنالوگ سنسور دما به یک عدد دیجیتال ۸ بیتی تبدیل می‌شود[2]. اکنون برای ارسال این عدد هشت بیتی (که در خروجی تراشه ADC به صورت موازی قرار دارد) به پورت سریال کامپیوتر، به یک مبدل سریال به موازی (مثلاً میکروکنترلر) نیاز داریم. همانطور که در شکل مقابل می‌بینید، پایه TxD (ارسال) مبدل موازی به سریال به پایه RxD (دریافت) پورت سریال کامپیوتر متصل و زمینهای دو طرف مشترک شده است. اکنون باید در کامپیوتر برنامه‌ای بنویسیم که اطلاعات را از پورت سریال دریافت کرده و نمایش دهد. برنامه زیر را ببینید.




```
#include <STDIO.H>
#include <CONIO.H>
#include <STDLIB.H>
void InitialSerialPort(void){
   asm {
     mov   ah,04h
     mov   al,00h
     mov   bh,00h
     mov   bl,00h
     mov   ch,03h
     mov   cl,07h   // 9600 bps
     mov   dx,00h
     int   14h
   }
}
int RecvData(){
  unsigned char status;
  asm {
    mov   ah,03h
    mov   dx,00h
    int   14h
    mov  status,ah
  }

status = status & 128;
   if (status == 0)
    return 0;
   else
```

[1] Analog to Digital Converter

[2] بعضی از میکروکنترلرها خود دارای ADC داخلی هستند.

```
    return 1;
}
void main(void){
    unsigned char data;
    InitialSerialPort();
    while (1){
    while (RecvData() == 0 && !kbhit());
    if (kbhit()) exit(0);
    data = inportb(0x3F8);
    printf("%X\n",data);
    }
}
```

به این خط برنامه توجه کنید:

```
while (RecvData() == 0 && !kbhit());
```

تا زمانی که کاراکتری دریافت نشده (خروجی تابع RecvData صفر است) و نیز کاربر کلیدی را فشار نداده باشد، برنامه در همین نقطه متوقف می‌ماند. با نقض شدن یکی از این دو شرط، برنامه از این خط عبور کرده و ادامه می‌دهد. در خط بعدی، بررسی می‌شود که آیا در اثر فشرده شدن کلیدی از خط قبلی عبور کرده‌ایم؟ اگر اینگونه باشد، برنامه به پایان می‌رسد. در غیر این صورت، به کمک تابع inportb داده دریافتی خوانده شده و در دستگاه شانزده‌تایی روی نمایشگر کامپیوتر نشان داده می‌شود. به صورت خلاصه، برنامه بالا تا زمانی که کاربر کلیدی را فشار نداده باشد، اطلاعات دریافتی از پورت سریال را خوانده و روی نمایشگر کامپیوتر نشان می‌دهد.

پرسش) برنامه بالا را به گونه‌ای اصلاح کنید که قبل از خروج، تعداد بایتهای دریافتی را نمایش دهد.

در زبان C، تابع bioscom که در فایل سرآیند bios.h قرار دارد، برای پیکربندی ارتباط سریال، ارسال و دریافت داده‌های سریال و نیز بررسی وضعیت ارتباط سریال به کار می‌رود.

در دو مثال طراحی سیستم، سخن از مبدل سریال به موازی و مبدل موازی به سریال گفته شده است. اکثر میکروکنترلرها دارای پورت سریال هستند و می‌توانند نقش این دو مبدل را ایفا کنند.

با مطالبی که در این بخش آموختید، می‌توانید برنامه‌هایی بنویسید که به کمک پورت سریال اطلاعاتی به خارج کامپیوتر ارسال یا از پورت سریال داده‌هایی دریافت کنند.

همانگونه که قبلاً گفته شد، ارتباط سریال در زبانهای تحت ویندوز به گونه‌ای متفاوت انجام می‌شود. در بخش ارتباط با پورتهای کامپیوتر تحت ویندوز، برنامه‌هایی به زبان Visual C++ نوشته شده است که به کمک آنها می‌توانید برنامه‌نویسی پورت سریال در ویندوز را به راحتی انجام دهید.

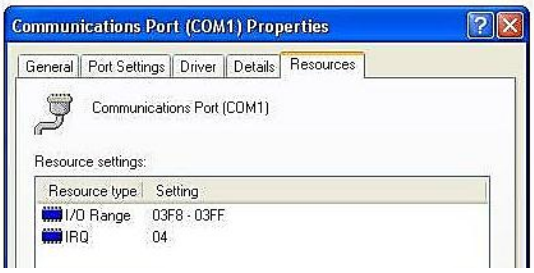
استفاده از پورت سریال به شیوه وقفه

فرض کنید می‌خواهیم در کامپیوتر برنامه‌ای بنویسیم که داده‌هایی را از پورت سریال کامپیوتر خوانده و نمایش دهد. یک راهکار این است که مرتباً بررسی کنیم آیا داده‌ای در پورت سریال دریافت شده یا خیر و در صورت مثبت بودن پاسخ، آن را بخوانیم و نمایش دهیم (روش سرشماری یا Polling). این راه ساده است؛ اما وقت را بیهوده هدر می‌دهد. برنامه در هر ثانیه هزاران بار پورت سریال را کنترل می‌کند که آیا داده‌ای در آن دریافت شده است یا خیر؟ که به

خصوص در مورد ارتباطاتی که حجم انتقال داده‌ها در آن زیاد نیست به صرفه نمی‌باشد. حداکثر سرعت ارتباط در این روش بدون از بین رفتن داده‌ها 34.7 Kbps است.

راه دیگر این است که از پورت سریال در حالت وقفه استفاده کنیم. در این حالت، وقتی داده‌ای توسط پورت سریال دریافت شود وقفه‌ای در کامپیوتر رخ می‌دهد. می‌توانید با بازنویسی زیربرنامه پاسخ به آن وقفه (ISR)، واکنش لازم در قبال دریافت آن داده (مثلاً نمایش آن در کامپیوتر) را نشان دهید. با این کار دیگر نیازی نیست مرتباً پورت سریال را بررسی کنید؛ بلکه هرگاه داده‌ای دریافت شد، وقفه‌ای رخ خواهد داد. با این روش حتی در کامپیوترهای قدیمی سرعت به راحتی تا 115.2 Kbps بالا می‌رود.

تغییر ISR یک وقفه در کامپیوتر چندان پیچیده نیست و می‌توان آن را به کمک برنامه‌های اسمبلی یا سطح بالا (مانند C) انجام داد. در کتابهای زبان اسمبلی در بخش برنامه‌های مقیم در حافظه[1] می‌توانید شیوه این کار را فرا بگیرید. برای این کار باید بدانید کامپیوتر کدام شماره وقفه را به پورت سریال اختصاص داده است. در شکل مقابل که مربوط به یک کامپیوتر نوعی است و قبلاً شیوه مشاهده آن در ویندوز را دیدیم، وقفه IRQ4 (INT 0C h) به پورت سریال اختصاص داده شده است.



وقفه IRQ 4 به پورتهای COM1 و COM3 و وقفه IRQ 3 به پورتهای COM2 و COM4 اختصاص دارد.

اما چگونه باید از سیستم خواست که با دریافت یک کاراکتر اعلام وقفه کند؟ مقداردهی ثبات فعال‌کننده وقفه (از مجموعه ثباتهای پورت سریال) این کار را انجام می‌دهد. بیتهای این ثبات را ببینید:

0	0	0	0	IN	ER	TBE	RxRDY

اگر بیت RxRDY «یک» باشد، دریافت داده باعث ایجاد وقفه می‌شود.

اگر بیت TBE «یک» باشد، خالی شدن بافر ارسال داده باعث روی دادن وقفه می‌شود.

بیت ER در رابطه با وقفه وضعیت خط و بیت IN در رابطه با وقفه وضعیت مودم عمل می‌کند.

دستور outportb(0x3F8+1,1); وقفه دریافت داده را فعال می‌کند. از این پس، با دریافت داده توسط پورت سریال وقفه‌ای رخ می‌دهد.

شبکه مودم پوچ

همانطور که قبلاً دیدیم، با اتصال پورتهای سریال دو کامپیوتر به یکدیگر به کمک یک کابل مودم پوچ (که ساختار کامل آن را دیدید و البته عملاً در بیشتر اوقات، تنها اتصالات TxD، RxD، RTS و CTS کافی است) و با استفاده از نرم‌افزار HyperTerminal، می‌توانید ارتباط داده‌ای بین دو کامپیوتر برقرار کنید و حتی فایل بین دو کامپیوتر رد و بدل نمایید.

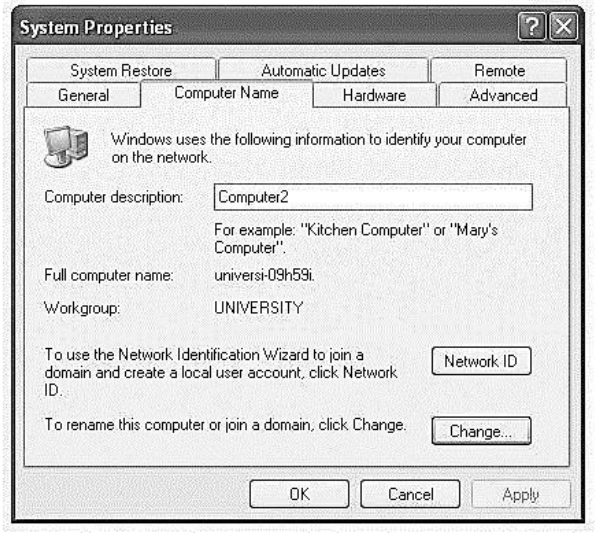
[1] Terminate & Stay Resident - TSR

در ویندوز ۲۰۰۰ یا XP، قابلیت ایجاد شبکه بین دو کامپیوتر به کمک اتصال پورتهای سریال دو کامپیوتر وجود دارد. در ادامه، با نحوه ایجاد این شبکه در ویندوز آشنا می‌شویم. برای ایجاد این شبکه، یکی از کامپیوترها باید میزبان[1] (پذیرنده اتصال) و دیگری مهمان[2] (برقرارکننده اتصال) باشد.

پیکربندی نرم افزاری کامپیوترهای میزبان و مهمان :

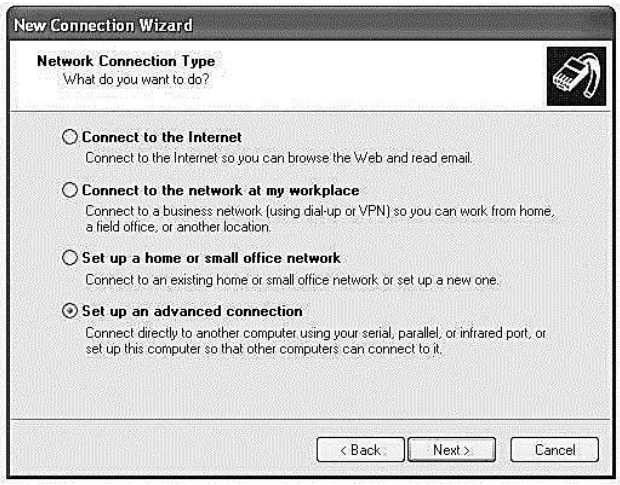
پیش از پیکربندی سیستمها مطمئن شوید که هر دو کامپیوتر به یک گروه کاری (مثلاً به نام University) متعلق هستند. با مراجعه به Control Panel/System/Computer Name از این موضوع مطمئن شوید و در صورت لزوم نام گروههای کاری دو سیستم را تغییر دهید.

شکل زیر گروه کاری کامپیوتر میزبان (Computer2) را نشان می دهد.



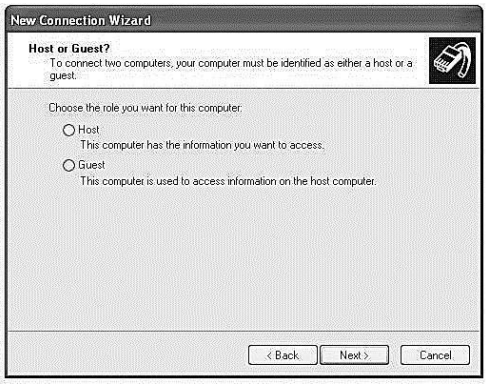
الف) پیکربندی مشترک در هر دو سیستم

- ✓ با مراجعه به Control Panel/Network Connection/Create a new connection کار را آغاز کنید.
- ✓ نوع شبکه را انتخاب کنید:



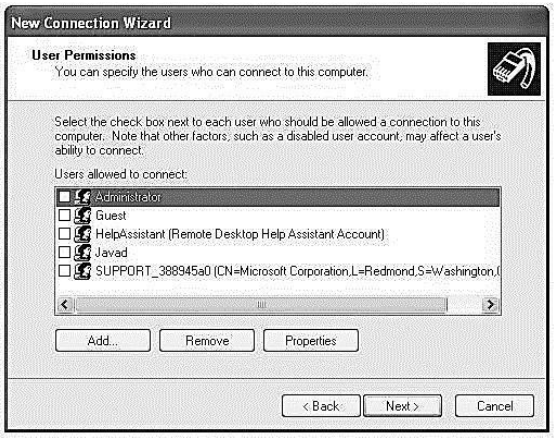
[1] Host
[2] Guest

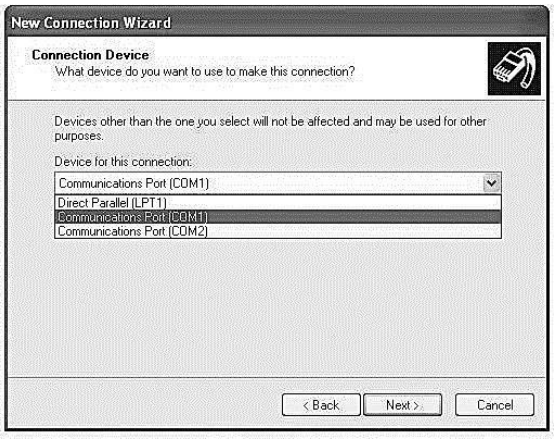
✓ نوع ارتباط را انتخاب کنید :



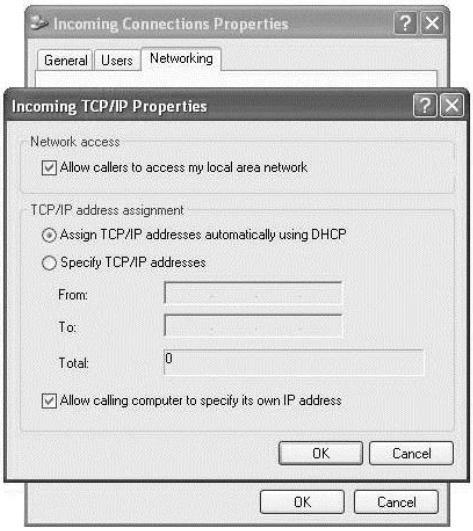
ب) ادامه پیکربندی نرم افزاری در کامپیوتر میزبان (Computer2) با انتخاب گزینه Host :

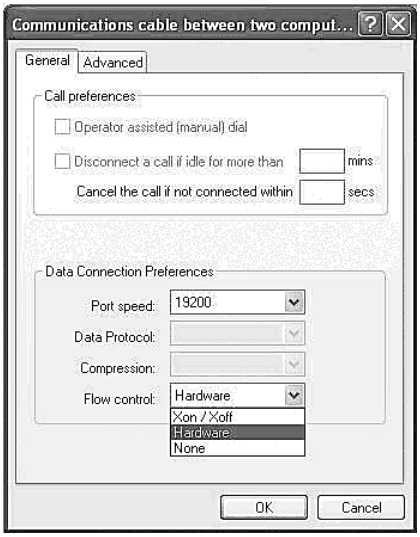
۱- پورت ارتباطی و سطوح دسترسی کاربران را مشخص کنید :

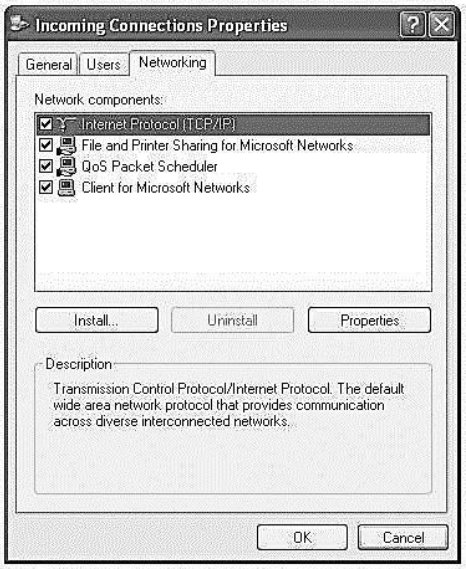


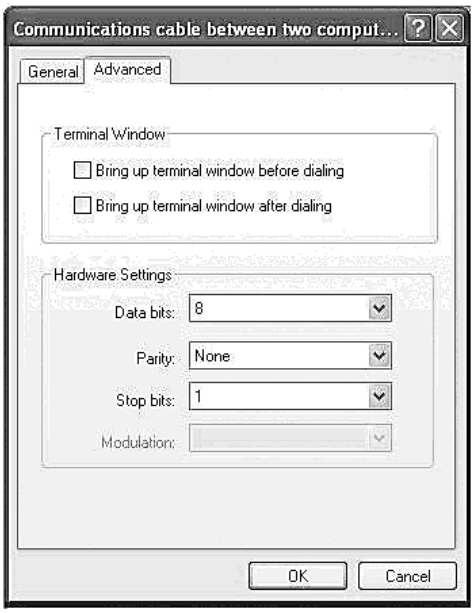


۳- اکنون پیکربندی کامپیوتر میزبان پایان یافته است و در قسمت Network Connections گزینه ای به نام Incoming Connections ساخته شده است. با کلیک راست روی این گزینه و انتخاب Properties یا کلیک دوبل روی خود گزینه، می توانید مشخصات ارتباط (از جمله سرعت که باید در دو طرف یکسان باشد) را تغییر دهید:



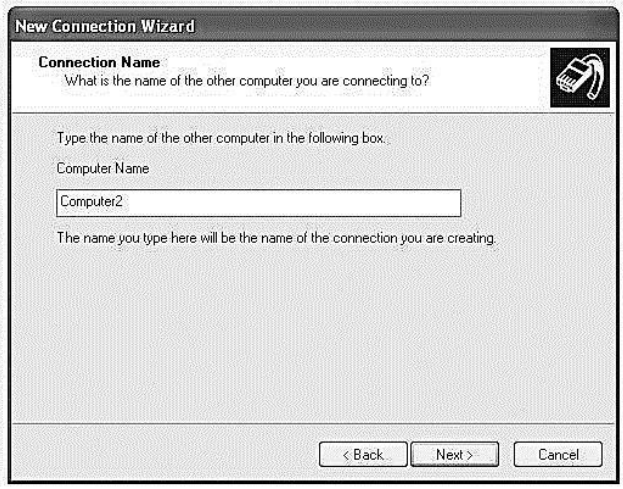




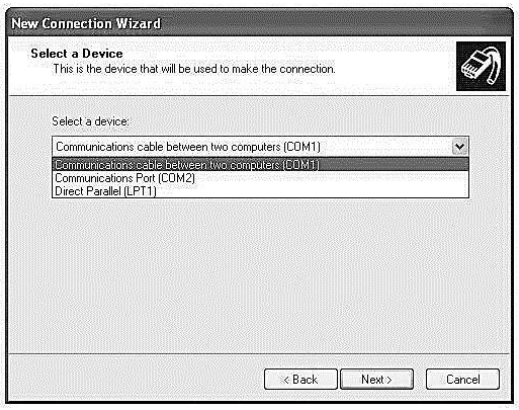


ب) ادامه پیکربندی نرم افزاری در کامپیوتر میهمان (Computer1) با انتخاب گزینه Guest :

۱ – نام کامپیوتر دیگر را وارد کنید :



۲– نوع ارتباط را مشخص کنید :



اکنون پیکربندی سیستم میهمان نیز پایان یافته است. یک گزینه با نام Computer2 در قسمت Network Connections سیستم میهمان ایجاد شده است. با کلیک روی آن و وارد کردن User Name و Password سیستم

میزبان، می توانید به آن متصل شوید وبا استفاده از آدرس IP یـا نـام آن، از تمـام منابع به اشتراک گذاشته شده آن استفاده کنید. مشخصات ارتباط (از جمله سرعت که باید در دو طرف یکسان باشد) با کلیک روی Properties قابل تنظیم است.

پروژه‌های کاربردی با پورت سریال کامپیوتر

۱) سیستمی شامل سخت‌افزار و نرم‌افزار طراحـی کنیـد کـه فـایلی را از یـک کامپیوتر به کامپیوتر دیگر ارسال کند. برنامه‌نویسی کامپیوترها را خودتان انجام دهید.

۲) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید یک رقص نور روی هشت عـدد LED ایجـاد کنـد. بـرای ایـن کار LEDها باید هر یک ثانیه یک بار به صورت یکی‌درمیان روشن شوند. LEDها به یـک میکروکنترلـر و میکروکنترلـر به پورت سریال کامپیوتر متصل است.

۳) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که محتویات ۸ عدد کلید دو حالته را بخوانـد و روی نمایشـگر کامپیوتر نشان دهد. کلیدها به یک میکروکنترلر و میکروکنترلر به پورت سریال کامپیوتر متصل است.

٤) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که جهت و سرعت چـرخش یـک موتـور پلـه‌ای را بـه کمـک برنامه کامپیوتری کنترل کند. با این سیستم می‌توانید روباتی بسازید که به کمک پورت سریال کامپیوتر کنترل شود.

۵) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که اعداد صفر تا ۹ را بـا فواصـل زمـانی یـک ثانیـه روی یـک نمایشگر هفت‌قسمتی[1] نمایش دهد. از تراشه مبدل کد BCD به کد نمایشگر هفت قسمتی اسـتفاده نکنیـد. برنامـه خـود را طوری بنویسید که نوع نمایشگر هفت‌قسمتی (آند مشترک یا کاتد مشترک) را در ابتدای برنامه از کاربر سؤال کند.

٦) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که یک ولتاژ آنالوگ بین صفر تا ۵ ولت کـه بـه کمـک یـک پتانسیومتر تولید می شود را از طریق پورت سـریال خوانـده و مقـدار ایـن ولتـاژ را بـا دقـت دو رقـم اعشـار روی نمایشـگر کامپیوتر نشان دهد.

۷) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که یک ولتاژ آنالوگ بین صفر تا ۵ ولت کـه بـه کمـک یـک پتانسیومتر تولید می شود را از طریق پورت سـریال خوانـده و مقـدار ایـن ولتـاژ را بـا دقـت دو رقـم اعشـار روی سـه عـدد نمایشگر هفت‌قسمتی که به پورت موازی متصلند، نمایش دهد.

۸) برنامه‌ای بنویسید که یک ارتباط همگام با پالس ساعتی که فرکانس آن از طریق برنامـه قابـل تنظـیم باشـد را روی دو پایه پورت موازی ایجاد کند.

۹) سیستمی شامل سخت‌افزار و نرم‌افزار طراحی کنید که یک فایل را از طریق امواج مادون قرمز ارسال کند.

[1] Seven Segment

منابع این فصل

[1] "The 80x86 IBM PC & Compatible Computers", Muhammed Ali Mazidi and et al, Prentice Hall,2000.

[2] "PC Interfaces under Windows", Burkhard Kaink and Hans-Joachim Berndt, Elektor Electronics Publications, 2002.

(این کتاب با ترجمه کیهان حدادشرق توسط انتشارات نشر علوم به چاپ رسیده است)

[۳] "اصول کامل راه‌اندازی و کنترل دستگاههای جانبی توسط کامپیوتر"، محسن شکیبافر، انتشارات نص، ۱۳۸۴.

[۴] "گذرگاهها و درگاههای کامپیوترهای شخصی"، عبدالمجید منصوریان‌فر و اصغر کریمی، انتشارات دانش‌پژوهان برین - ارکان، ۱۳۸۳

[۵] "دیجیتال پایه و طراحی مدارهای واسط"، شیرزاد شهریاری، انتشارات پرتونگار، ۱۳۸۵.

[۶] "مرجع علمی-کاربردی سخت‌افزار"، شیرزاد شهریاری، انتشارات جهاد دانشگاهی مشهد، ۱۳۸۴.

زنگ تفریح!

مشخصات اولین کامپیوتر شخصی IBM یعنی PC-XT را در زیر مشاهده می‌کنید. مقایسه آن با مشخصات کامپیوترهای امروزی خالی از لطف نیست!

NAME (۸	*PC XT - Model 5160* (۹
MANUFACTURER	IBM
TYPE	Professional Computer
ORIGIN	U.S.A.
YEAR	1983
END OF PRODUCTION	Unknown
BUILT IN LANGUAGE	Microsoft BASIC
KEYBOARD	Full-stroke keyboard with numeric keypad and function keys 84 or 101 keys
CPU	Intel 8088
SPEED	4.77 MHz
COPROCESSOR	Socket for a 8087 math co-processor
RAM	From 64k to 640k, depending on models
ROM	64 kb
TEXT MODES	80 x 24 / 40 x 24
GRAPHIC MODES	CGA modes : 320 x 200 / 640 x 200
COLORS	16
SOUND	Tone Generator
I/O PORTS	eight internal slots (five 8 bit ISA), RS232c, Centronics
BUILT IN MEDIA	One 5.25" FDD, 360k (3.5" on later models) 10Mb or 20Mb hard-disk
OS	MS DOS
POWER SUPPLY	PSU built-in
PERIPHERALS	Numerous IBM and third-parties expansion cards, i.e. the QuadRam 512 KB RAM card
PRICE	$8000 (Complete version with 640 KB RAM, 10 MB HDD, colour display)

فصل هشتم
برنامه‌نویسی پورت‌های کامپیوتر تحت ویندوز

در کامپیوتری که با سیستم‌عامل DOS یا ویندوز ۹۵ یا ۹۸ یا Millenium (Me) کار می‌کند، برنامه‌های تبادل اطلاعات با درگاههای کامپیوتر به درستی اجرا می‌شوند. اما در نسخه‌های بعدی ویندوز (NT، 2000، XP، Vista و ...) داستان ارتباط با درگاههای ورودی/خروجی متفاوت است؛ در این سیستم‌عامل‌ها، برنامه‌های کاربر در مد کاربردی[1] و توابع سیستمی (از جمله توابع دسترسی به درگاههای ورودی/خروجی) در مد هسته سیستم‌عامل[2] اجرا می‌شوند. این کار برای بالا بردن امنیت سیستم‌عامل و جلوگیری از حملات مخرب انجام می‌شود و آزادی عملی که کاربران سیستم‌عامل‌های قبل از ویندوز NT برای اجرای توابع سیستمی داشتند را از آنان می‌گیرد. بنابراین اگر برنامه‌ای برای تبادل مستقیم اطلاعات با درگاههای ورودی/خروجی بنویسید (حتی اگر از زبانهای تحت ویندوز مانند Visual C++ و Delphi استفاده کنید)، در ویندوزهای NT به بعد در اجرای صحیح برنامه ناکام خواهید ماند. برای تصحیح این مشکل، باید از برنامه‌هایی مانند PortTalk یا کتابخانه پویای inpout32 استفاده کنید که به صورت واسط بین مد کاربردی و مد هسته عمل می‌کنند؛ به بیان دیگر درخواستهای کاربر برای دسترسی به درگاهها را دریافت و به هسته سیستم‌عامل ارائه کرده و پاسخ را از هسته گرفته و به کاربر می‌دهند.

در ادامه دو روش برای این کار مورد بررسی قرار می‌گیرند. در روش اول، از توابع کتابخانه‌ای آماده که در فایلهای inpout32.dll و inpout32.lib وجود دارند استفاده می‌کنیم و در روش دوم بسته نرم‌افزاری PortTalk را مورد بررسی قرار می‌دهیم. این بسته دارای یک فایل اجرایی به نام AllowIO.exe است که ابزاری ساده برای در دسترس قرار دادن درگاههای ورودی/خروجی عرضه می‌کند. پس از بررسی این ابزار، با الهام از برنامه PortTalk توابعی سیستمی خواهیم نوشت که عملیات دسترسی به درگاه را انجام می‌دهند. با قرار دادن این توابع در یک فایل سرآیند و استفاده از آن در برنامه‌های خود می‌توانید به درگاههای ورودی/خروجی دسترسی داشته باشید.

برای فهم مطالبی که در ادامه می‌آید، نیازی به دانستن مفاهیم پیشرفته برنامه‌نویسی تحت ویندوز نیست و به سادگی می‌توانید از آنها حتی در برنامه‌های کنسول[3] که شباهت زیادی به برنامه‌های تحت DOS دارند، استفاده کنید.

کتابخانه inpout32.dll

فایل inpout32.dll یک کتابخانه پویاست که به کمک توابع آن می‌توان از درگاههای ورودی/خروجی کامپیوتر استفاده کرد. با مراجعه به سایت http://www.logix4u.net فایلهای inpout32.dll و inpout32.lib را دانلود کنید. در این کتابخانه دو تابع به نامهای Inp32 و Out32 تعریف شده که الگوی استفاده از آنها دقیقاً مانند توابع inportb و outportb زبان C است که قبلاً معرفی شدند.

از این کتابخانه‌ها می‌توان در تمام زبانهای تحت ویندوز استفاده کرد. در ادامه، مراحل به‌کارگیری این کتابخانه‌ها را در محیط Visual C++ تشریح می‌کنیم.

---
[1] Application Mode
[2] Kernel Mode
[3] Console

- ✓ ابتدا به کمک بسته Visual Studio یک برنامه دلخواه برای ارتباط با درگاه بنویسید. این برنامه می‌تواند مبتنی بر MFC یا API یا Console باشد. از توابع Inp32 و Out32 برای ارتباط با درگاهها استفاده کنید.
- ✓ دو فایل فوق را در شاخه پروژه خود کپی کنید.
- ✓ از منوی Project گزینه Setting را انتخاب کنید. به سربرگ Link بروید و نام فایل کتابخانه‌ای inpout32.lib را به قسمت object/library اضافه کنید.
- ✓ دو خط زیر را به ابتدای فایل اصلی برنامه خود (بعد از راهنماهای اولیه[1]) اضافه کنید:

```
short      _stdcall      Inp32(short      portaddr);
void  _stdcall Out32(short portaddr, short datum);
```

برای نمونه، برنامه ارسال داده‌ها به کارت XT خروجی را (که قبلاً در زبان C تحت DOS نوشته بودیم) بازنویسی می‌کنیم.

```
#include " stdafx.h "
#include < conio.h >
#include < windows.h >

short _stdcall Inp32(short portaddr);
void _stdcall Out32(short portaddr, short datum);


void main(void){

     short data = 0x55;

     while (!kbhit()){

          Out32(0x031B , data);
          data = ~data;
          Sleep(500);
     }
}
```

با اجرای این دستورات در مد کنسول محیط Visual Studio می‌توانید به نتایجی مشابه قبل دست یابید و نیز آن را به دلخواه خود تغییر دهید؛ مثلاً اگر به جای آدرس 031Bh، از آدرس درگاه موازی (0378h) استفاده کنید و هشت LED به پایه‌های داده این درگاه متصل کنید، چشمک زدن آنها را خواهید دید.

به همین روش می‌توانید از توابع Inp32 و Out32 در برنامه‌های دیگر تحت ویندوز استفاده کنید. برای تمرین، برنامه‌ای بنویسید که خروجی یک A/D که دمای اتاق را به عنوان ورودی از حسگر دما دریافت کرده را روی نمایشگر کامپیوتر نشان دهد.

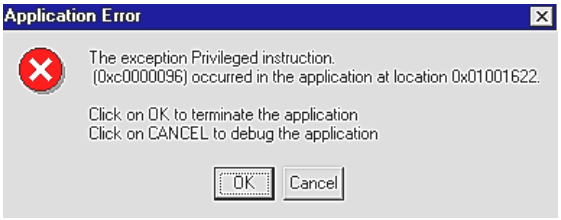
نرم‌افزار PortTalk

این نرم‌افزار که از طریق سایت http://www.beyondlogic.org قابل دسترسی است، برای استفاده از درگاههای ورودی/خروجی در برنامه‌های تحت ویندوز به کار می‌رود. برای توضیح شیوه کاری این نرم‌افزار، نگاهی دقیقتر به دلایل محدودیت دسترسی به درگاههای ورودی/خروجی کامپیوتر در ویندوزهای NT به بعد می‌اندازیم. برای فهم این مطالب مد محافظت‌شده پردازنده‌های ۸۰۲۸۶ و ۸۰۳۸۶ را تا حدی بشناسید.

[1] pre processor directives

ویندوز NT با استفاده از قابلیتهای پردازنده در مد محافظت‌شده[1] محدودیتهایی برای برنامه‌های کاربردی در دسترسی به درگاههای ورودی/خروجی کامپیوتر به وجود می‌آورد. این دسترسی تابع دو مفهوم بیتهای IOPL[2] و نقشه بیتی اجازه دسترسی به درگاههای ورودی/خروجی (IOPM)[3] است.

بیتهای IOPL: دو بیت IOPL در ثبات EFLAGS پردازنده وجود دارند که چهار سطح دسترسی به ورودی/خروجی را ایجاد می‌کنند که سطح صفر بالاترین اولویت است. اگر برنامه‌ای که سطح دسترسی تعریف شده آن پایینتر از سطح تعریف شده در IOPL باشد برای دسترسی به درگاهها تلاش کند، وقفه شماره ۱۳[4] رخ می‌دهد و در ویندوز پیغامی به شکل زیر ظاهر می‌شود.



هر برنامه‌ای که می‌خواهد به درگاههای ورودی/خروجی دسترسی داشته باشد باید سطح دسترسی تعریف شده توسط IOPL را تا سطح اولویت برنامه پایین بیاورد.

نقشه بیتی دسترسی به درگاه: هر فرآیند[5] که در ویندوز اجرا می‌شود، دارای یک سگمنت حالت فرآیند (TSS[6]) است که اطلاعات کامل مربوط به آن فرآیند را نگهداری می‌کند. از TSS برای تعویض بین فرآیندها در سیستمهای چندوظیفه[7] استفاده می‌شود. هر TSS حاوی یک نقشه بیتی اجازه دسترسی به درگاههای ورودی/خروجی است که IOPM[8] نامیده می‌شود. به ازای هرکدام از 2^{16} (۶۵۵۳۶) درگاه ورودی/خروجی کامپیوتر (که به کمک ۱۶ بیت آدرس‌دهی می‌شود)، یک بیت در این نقشه وجود دارد که اگر «یک» باشد، دستور دسترسی به درگاه با یک خطای استثنای[9] «نقض اولویت[10]» روبرو می‌شود و اگر «صفر» باشد دسترسی به درگاه صورت خواهد گرفت.

در ویندوز NT، تنها دو سطح دسترسی (از چهار سطح دسترسی) به کار می‌رود که سطح صفر (بالاترین اولویت) و سطح سه (پایین‌ترین اولویت) می‌باشند. برنامه‌های کاربر در سطح سه و برنامه‌های راه‌انداز وسایل[11] و هسته سیستم‌عامل در سطح صفر (معروف به حلقه صفر[12]) اجرا می‌شوند. به همین دلیل برنامه‌های کاربر از حق کافی برای دسترسی به درگاههای ورودی/خروجی برخوردار نیستند و این دسترسی فقط باید توسط برنامه‌های سطح صفر انجام شود. تمام برنامه‌های مد کاربر باید از طریق یک برنامه راه‌انداز وسیله که در سطح صفر اجرا شده به درگاههای ورودی/خروجی دست یابند.

---

[1] Protected Mode
[2] I/O Privileged Level
[3] I/O Permission bitMap
[4] General Protection Error
[5] Task - Process
[6] Task State Segment
[7] Multi-Task
[8] Input/Output Permission Map
[9] Privilege Exception
[10] Perivileged Instruction Violation
[11] Device Driver
[12] Ring 0

هنگامی که یک دستور ورودی/خروجی اجرا می‌شود، پردازنده ابتدا بررسی می‌کند آیا فرآیند درخواست‌کننده، مجوز لازم برای دسترسی به درگاههای ورودی/خروجی را دارد یا خیر؟ در صورت مثبت بودن پاسخ، دستور اجرا خواهد شد. در غیر این صورت، نقشه بیتی اجازه دسترسی به درگاهها در TSS آن برنامه بررسی و به کمک آن مشخص می‌شود آیا اجازه دسترسی به درگاه ورودی/خروجی مورد نظر داده شده است یا خیر؟

دو راه برای حل مشکل دسترسی به درگاهها در ویندوزهای NT به بعد وجود دارد. روش اول، نوشتن یک برنامه راه‌انداز وسیله است که در سطح صفر اجرا شده و روند دسترسی به درگاههای ورودی/خروجی را به نفع شما تغییر دهد. تبادل اطلاعات بین برنامه کاربر و برنامه راه‌انداز به کمک توابع کنترلی ورودی/خروجی (IOCTL) انجام شده و برنامه راه‌انداز مسئول اجرای دستورات ورودی/خروجی شما خواهد بود.

راه دیگر، اصلاح نقشه بیتی دسترسی به درگاهها در TSS فرآیند است تا مجوزهای لازم برای دسترسی به درگاههای مشخص به آن برنامه داده شود. این راه چندان مناسب نیست، اما به شما این امکان را می‌دهد که برنامه‌هایی که از قبل (مثلاً تحت DOS) نوشته‌اید را بدون تغییر و تنها به کمک یک برنامه که مجوزهای دسترسی را تغییر می‌دهد، در ویندوزهای NT به بعد اجرا کنید (برنامه AllowIO.exe که در ادامه بررسی می‌شود همین کار را انجام می‌دهد).

استفاده از برنامه‌های راه‌انداز مانند PortTalk می‌تواند به ناکارآمدی از نظر زمانی بیانجامد. هربار یک تابع IOCTL برای خواندن از درگاه ورودی یا نوشتن در درگاه خروجی اجرا می‌شود، پردازنده باید از سطح سه به سطح صفر برود، تابع را اجرا کند و مجدداً به سطح سه بازگردد. به این جهت باید در نوشتن برنامه‌هایی که نیاز به تبادل سریع اطلاعات با درگاهها ورودی/خروجی دارند، ملاحظات زمانی را مد نظر قرار دهید.

برنامه AllowIO.exe

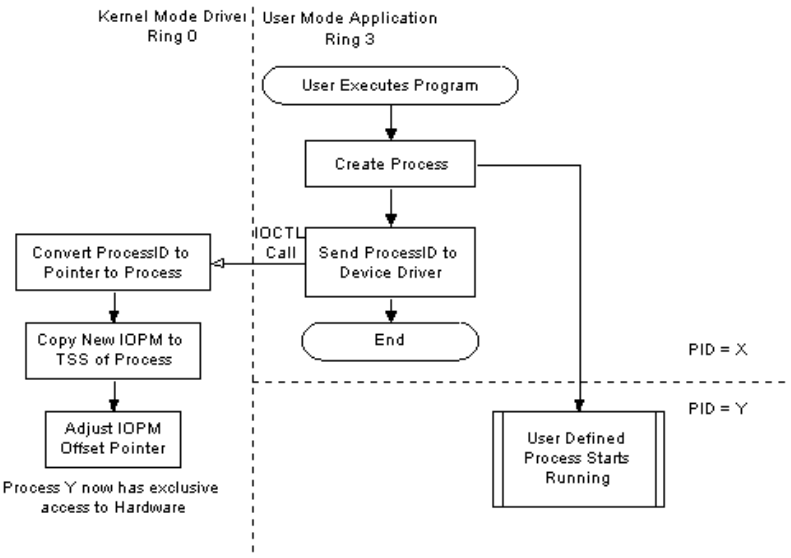
به کمک این برنامه می‌توانید برنامه‌هایی که در DOS یا ویندوزهای قبل از NT (برنامه‌های ۱۶ بیتی) به درگاهها دسترسی داشته‌اند را بدون مشکل در ویندوزهای NT به بعد اجرا کنید. برنامه‌های ۱۶ بیتی ویندوز یا برنامه‌های DOS، در ویندوزهای NT به بعد به کمک ماشین مجازی[1] (که در ویندوز عمل می‌کند) اجرا می‌شوند. ماشین مجازی، عملیات ورودی/خروجی را متوقف کرده و آنها را به یک مرجع رسیدگی ورودی/خروجی ارجاع می‌دهد. اگر ماشین مجازی DOS (VDM[2]) دارای مجوز لازم برای دسترسی به درگاههای ورودی/خروجی باشد، عملیات ورودی/خروجی را قطع نکرده و از نظر ملاحظات زمانی بهتر عمل می‌کند.

برای تغییر IOPM یک فرآیند ، ابتدا باید شماره شناسایی (ID) فرآیند را بدانید. اگر فرآیند را خودتان ایجاد کنید، به سادگی می‌توانید شماره شناسایی آن را به برنامه راه‌انداز رد کنید. برنامه کاربردی کوچکی می‌تواند این کار را انجام دهد که نام برنامه شما را بگیرد، با اجرای آن یک فرآیند ایجاد کند و شماره شناسایی آن را به شما ارائه کند[3]. شکل زیر مراحل ایجاد مجوزهای دسترسی به درگاهها را نشان می‌دهد.

---

[1] Virtual Machines
[2] Virtual DOS Machine

[3] راه دیگر استفاده از تکنیکهای Callback است که در اینجا به آن نمی‌پردازیم.



هنگامی که اجرای یک برنامه ۳۲ بیتی ویندوز (برنامه‌ای که تحت ویندوزهای NT به بعد نوشته شده) به کمک تابع سیستمی CreateProcess() آغاز می‌شود، شماره شناسایی فرآیند ایجاد شده و به کمک یک تابع IOCTL به برنامه راه‌انداز ارسال می‌شود.

برنامه‌های تحت DOS شماره فرآیند ندارند و تحت ماشین مجازی DOS ویندوز NT (NTVDM) اجرا می‌شوند که یک زیرسیستم حفاظت‌شده برای شبیه‌سازی DOS است. هنگامی که یک برنامه تحت DOS در ویندوز NT فراخوانی می‌شود، شماره شناسایی NTVDM را به خود می‌گیرد.

هنگامی که برنامه راه‌انداز شماره شناسایی فرآیند را بداند، اشاره‌گر به فرآیند را یافته و مجوزهای دسترسی به ورودی/خروجی (IOPM) را در آن اصلاح می‌کند.

استفاده از برنامه AllowIO.exe بسیار ساده است؛ فرض کنید برنامه‌ای به نام TestPort.exe تحت DOS نوشته‌اید که قبلاً با درگاه موازی به شماره 0378h کار می‌کرده و اکنون در ویندوز NT به مشکل برخورده است. مراحل زیر را دنبال کنید:

- ✓ بسته نرم‌افزاری PortTalk را از سایت http://beyondlogic.org دانلود و بازگشایی کنید.
- ✓ فایل PortTalk.sys را در شاخه System32\Driver ویندوز خود کپی کنید. این فایل، برنامه راه‌اندازی است که باید در سطح اولویت صفر اجرا شود.
- ✓ برنامه porttalk.reg را اجرا کنید تا اصلاحات لازم در رجیستری ویندوز انجام شود.
- ✓ سیستم خود را خاموش و مجدداً روشن کنید.
- ✓ فایل اجرایی AllowIO.exe را در مسیری که فایل اجرایی برنامه‌تان قرار دارد (مثلاً C:\MyProject)، کپی کنید.
- ✓ وارد محیط DOS ویندوز شوید و به مسیر فوق بروید و برنامه خود را به صورت زیر اجرا کنید:

```
C:\MyProject\> AllowIO.exe  TestPort.exe  0x0378
```

با اجرای این دستور، برنامه شما اجرا شده و بدون ایجاد هیچ مشکلی با درگاه موازی ارتباط برقرار می‌کند. عددی که مقابل دستور ذکر می‌شود محدوده درگاههای مورد نیاز را مشخص می‌کند؛ در حالت کلی اگر عدد N را مقابل دستور بیاورید ، برنامه AllowIO.exe دسترسی به درگاهها در محدوده آدرس N تا N+8 را فراهم می‌آورد. اگر به جای آدرس درگاه مورد نظر، از /a استفاده کنید، برنامه شما می‌تواند به همه درگاههای ورودی/خروجی دسترسی داشته باشد. تنها باید مراقب باشید که از این برنامه تحت نظارت کامل خودتان استفاده شود.

در شاخه checked از پوشه PortTalk، نسخه دیگری از برنامه راه‌انداز porttalk.sys وجود دارد که اطلاعات debug در آن نهفته است و به همین لحاظ در اجرا سرعت پایین‌تری دارد. اگر مایل به استفاده از اطلاعات debug (مانند سرریز بافر و ...) هستید می‌توانید به جای برنامه راه‌اندازی که در شاخه اصلی پوشه PortTalk قرار دارد، از این برنامه استفاده کنید. با این کار می‌توانید پیغامهای debug را به کمک نرم‌افزارهای مناسب[1] مشاهده کنید.

استفاده از PortTalk در برنامه‌های تحت ویندوز

با اینکه برنامه AllowIO.exe ابزار لازم برای اجرای برنامه‌های دسترسی به درگاههای ورودی/خروجی را فراهم می‌آورد، اما شیوه فراخوانی آن برای استفاده در برنامه‌هایی که کاربر به کمک محیطهای تحت ویندوز می‌نویسد، مناسب نیست.

در ادامه، توابع سیستمی نرم‌افزار PortTalk را معرفی و به اختصار توضیحاتی راجع به آنها ارائه می‌کنیم. با نوشتن این توابع در یک فایل سرآیند[2] (مثلاً PortTalk.h) و شامل کردن آن در برنامه‌های خود، می‌توانید به راحتی از این توابع برای دسترسی به درگاههای ورودی/خروجی استفاده کنید. این توابع از سرآیندهای winioctl.h و windows.h استفاده می‌کنند. شامل کردن آنها را در سرآیند PortTalk.h فراموش نکنید.

تعاریف زیر را در سرآیند PortTalk.h انجام دهید:

```
#define   PORTTALK_TYPE   40000 /* 32768-65535 are reserved for customers */

// The IOCTL function codes from 0x800 to 0xFFF are for customer use.

#define   IOCTL_IOPM_RESTRICT_ALL_ACCESS \
   CTL_CODE(PORTTALK_TYPE,0x900,METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_IOPM_ALLOW_EXCUSIVE_ACCESS \
   CTL_CODE(PORTTALK_TYPE, 0x901, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_SET_IOPM \
   CTL_CODE(PORTTALK_TYPE, 0x902, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_ENABLE_IOPM_ON_PROCESSID \
   CTL_CODE(PORTTALK_TYPE, 0x903, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_READ_PORT_UCHAR \
   CTL_CODE(PORTTALK_TYPE, 0x904, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

---

[1] مانند نرم‌افزار System Internals DebugView که از سایت http://www.sysinternals.com قابل دانلود است.

[2] Header

```
#define IOCTL_WRITE_PORT_UCHAR \
  CTL_CODE(PORTTALK_TYPE, 0x905, METHOD_BUFFERED, FILE_ANY_ACCESS)

HANDLE   PortTalk_Handle;       /* Handle for PortTalk Driver */
```

از دستگیره[1] PortTalk_Handle برای دسترسی به درگاه استفاده می‌شود.

در ابتدای کار باید نرم‌افزار راه‌انداز PortTalk نصب شده باشد. به شیوه‌ای که قبلاً برای استفاده از ALLOWIO شرح داده شد عمل کنید و توابع زیر را در ابتدای برنامه خود برای نصب و راه‌اندازی PortTalk فراخوانی کنید.

```
void InstallPortTalkDriver(void)
{
    SC_HANDLE  SchSCManager;
    SC_HANDLE  schService;
    DWORD      err;
    CHAR       DriverFileName[80];

/*
Get Current Directory. Assumes PortTalk.SYS driver is in this directory. Doesn't
detect if file exists, nor if file is on removable media - if this is the case then
when windows next boots, the driver will fail to load and a error entry is made in
the event viewer to reflect this.
Get System Directory. This should be something like c:\windows\system32 or
c:\winnt\system32 with a Maximum Character lenght of 20. As we have a buffer of 80
bytes and a string of 24 bytes to append, we can go for a max of 55 bytes.
*/

   if (!GetSystemDirectory(DriverFileName, 55)){
          printf("PortTalk: Failed to get System Directory.");
          printf("Is System Directory Path > 55 Characters?\n");
          printf("PortTalk: Please manually copy driver to your");
          printf("system32/driver directory.\n");
   }

    /* Append our Driver Name */
    lstrcat(DriverFileName,"\\Drivers\\PortTalk.sys");
    printf("PortTalk: Copying driver to %s\n",DriverFileName);

    /* Copy Driver to System32/drivers directory.
    This fails if the file doesn't exist. */

    if (!CopyFile("PortTalk.sys", DriverFileName, FALSE)){
          printf("PortTalk: Failed to copy driver to");
      printf("%s\n",DriverFileName);
      printf("PortTalk: Please manually copy driver to your");
      printf("system32/driver directory.\n");
    }

    /* Open Handle to Service Control Manager */
    SchSCManager = OpenSCManager (
          NULL,/*machine (NULL == local)*/
          NULL,/*database             (NULL             ==             default)*/
          SC_MANAGER_ALL_ACCESS);/*access required*/

/*
Create Service/Driver - This adds the appropriate registry keys in
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services - It doesn't care if the
driver exists, or if the path is correct.
*/

    schService = CreateService (
              SchSCManager,/* SCManager database */
```

[1] Handle

```
                    "PortTalk",/* name of service */
                    "PortTalk",/* name to display */
                    SERVICE_ALL_ACCESS,/* desired access */
                    SERVICE_KERNEL_DRIVER,/* service type */
                    SERVICE_DEMAND_START,/* start type */
                    SERVICE_ERROR_NORMAL,/* error control type */
                    "System32\\Drivers\\PortTalk.sys",
                   /* service's binary */
                    NULL,/* no load ordering group */
                    NULL,/* no tag identifier */
                    NULL,/* no dependencies */
                 NULL,/* LocalSystem account */
                    NULL/* no password */);

    if (schService == NULL) {
         err = GetLastError();
         if (err == ERROR_SERVICE_EXISTS){
                 printf("PortTalk: Driver already exists.");
          printf("No action taken.\n");
         }
         else {
          printf("PortTalk: Unknown error while creating");
          printf("Service.\n");
      }
    }
    else
       printf("PortTalk: Driver successfully installed.\n");

    /* Close Handle to Service Control Manager */
    CloseServiceHandle (schService);
}

//*************************************************************

unsigned char StartPortTalkDriver(void)
{
    SC_HANDLE  SchSCManager;
    SC_HANDLE  schService;
    BOOL       ret;
    DWORD      err;

    /* Open Handle to Service Control Manager */
    SchSCManager = OpenSCManager (
                NULL,/* machine (NULL == local)*/
                   NULL,/*database (NULL == default)*/
                   SC_MANAGER_ALL_ACCESS);/* access required */

    if (SchSCManager == NULL)
      if (GetLastError() == ERROR_ACCESS_DENIED) {
       /*
       We do not have enough rights to open the SCM, therefore we must be a poor
    user with only user rights.
       */
 printf("PortTalk: You do not have rights to access");
 printf("the Service Control Manager and\n");
 printf("PortTalk: the PortTalk driver is not installed");
 printf("or started. Please ask \n");
 printf("PortTalk: your administrator to install the driver");
 printf("on your behalf.\n");
return(0);
}

    do {
         /* Open a Handle to the PortTalk Service Database */
schService = OpenService(
SchSCManager,/* handle to service control manager database */
"PortTalk",/*pointer to name of service to start */
SERVICE_ALL_ACCESS);
```

```
   /* type of access to service */

            if (schService == NULL)
               switch (GetLastError()){
                   case ERROR_ACCESS_DENIED:
   printf("PortTalk: You do not have rights ");
   printf("to the PortTalk service database\n");
        return(0);
   case ERROR_INVALID_NAME:
                  printf("PortTalk: The specified service name ");
              printf("is invalid.\n");
                     return(0);
   case ERROR_SERVICE_DOES_NOT_EXIST:
       printf("PortTalk: The PortTalk driver does");
       printf("not exist. Installing driver.\n");
       printf("PortTalk: This can take up to 30 seconds");
       printf("on some machines . .\n");
              InstallPortTalkDriver();
              break;
    }
   } while (schService == NULL);

   /*
   Start the PortTalk Driver. Errors will occur here if PortTalk.SYS file doesn't
exist
   */

       ret = StartService (schService,/* service identifier */
                           0, /* number of arguments */
                           NULL); /* pointer to arguments */

       if (ret)
   printf("PortTalk: The PortTalk driver has been successfully started.\n");
       else {
           err = GetLastError();
           if (err == ERROR_SERVICE_ALREADY_RUNNING)
   printf("PortTalk: The PortTalk driver is already running.\n");
           else {
      printf("PortTalk: Unknown error while starting PortTalk");
          printf("driver service.\n");
             printf("PortTalk: Does PortTalk.SYS exist in your");
          printf("\\System32\\Drivers Directory?\n");
             return(0);
           }
       }

       /* Close handle to Service Control Manager */
       CloseServiceHandle (schService);
       return(TRUE);
   }
```

تابع OpenPortTalk برای باز کردن درگاه ورودی/خروجی

```
   unsigned char OpenPortTalk(void)
   {
   /* Open PortTalk Driver. If we cannot open it, try installing and starting it */
       PortTalk_Handle = CreateFile("\\\\.\\PortTalk",
                                    GENERIC_READ,
                                    0,
                                    NULL,
                                    OPEN_EXISTING,
                                    FILE_ATTRIBUTE_NORMAL,
                                    NULL);

       if(PortTalk_Handle == INVALID_HANDLE_VALUE) {
               /* Start or Install PortTalk Driver */
               StartPortTalkDriver();
               /* Then try to open once more, before failing */
```

```
            PortTalk_Handle = CreateFile("\\\\.\\PortTalk",
                                         GENERIC_READ,
                                         0,
                                         NULL,
                                         OPEN_EXISTING,
                                        FILE_ATTRIBUTE_NORMAL,
                                         NULL);

               if(PortTalk_Handle == INVALID_HANDLE_VALUE) {
              printf("PortTalk: Couldn't access PortTalk Driver,");
        printf("Please ensure driver is loaded.\n\n");
                       return -1;
               }
                    else
                          return 1;
      }
     else
           return 1;
      }
}
```

از تابع سیستمی CreateFile برای باز کردن ارتباط با درگاه استفاده شده است. چنانچه این ارتباط به درستی برقرار نشود، به کمک تابع StartPortTalkDriver ابتدا راه‌انداز نصب شده و مجدداً تلاش برای ارتباط با درگاه آغاز می‌شود.

تابع ClosePortTalk برای بستن درگاه ورودی/خروجی

برای بستن ارتباط از تابع زیر استفاده می‌کنیم:

```
void ClosePortTalk(void)
{
    CloseHandle(PortTalk_Handle);
}
```

توجه کنید که دستگیره PortTalk_Handle را در فایل سرآیند PortTalk.h به صورت عمومی تعریف کرده‌ایم و به همین لحاظ نیازی نیست به صورت پارامتر یا مقدار خروجی با توابع ارتباط برقرار کنند.

تابع outportb برای نوشتن در درگاه خروجی

```
void outportb(unsigned short PortAddress, unsigned char byte)
{
    unsigned int error;
    DWORD BytesReturned;
    unsigned char Buffer[3];
    unsigned short * pBuffer;
    pBuffer = (unsigned short *)&Buffer[0];
    *pBuffer = PortAddress;
    Buffer[2] = byte;

    error = DeviceIoControl(PortTalk_Handle,
                            IOCTL_WRITE_PORT_UCHAR,
                            &Buffer,
                            3,
                            NULL,
                            0,
                            &BytesReturned,
                            NULL);

    if (!error){
       printf("Error occured during outportb while talking to PortTalk");
    printf("driver %d\n",GetLastError());
    }
}
```

تابع outportb برای خواندن از درگاه ورودی

```
unsigned char inportb(unsigned short PortAddress)
{
    unsigned int error;
    DWORD BytesReturned;
    unsigned char Buffer[3];
    unsigned short * pBuffer;
    pBuffer = (unsigned short *)&Buffer;
    *pBuffer = PortAddress;

    error = DeviceIoControl(PortTalk_Handle,
                            IOCTL_READ_PORT_UCHAR,
                            &Buffer,
                            2,
                            &Buffer,
                            1,
                            &BytesReturned,
                            NULL);

    if (!error){
   printf("Error occured during inportb while talking to PortTalk");
   printf("driver %d\n",GetLastError());
}
    return(Buffer[0]);
}
```

همانطور که می‌بینید، توابع فوق با فرض استفاده از برنامه‌های مد کنسول نوشته شده‌اند. اگر مایل به نوشتن برنامه‌های تحت ویندوز هستید می‌توانید دستورات مد کنسول (مثلاً printf) را با توابع ویندوز (مثلاً MessageBox) عوض کنید.

برنامه ارسال اطلاعات به کارت XT خروجی را بازنویسی می‌کنیم. برنامه زیر را ببینید.

```
#include "stdafx.h"
#include "PortTalk.h"
#include <conio.h>
#include <windows.h>

int main(int argc, char* argv[])
{
    unsigned char data = 0x55;

    OpenPortTalk();

    while(!kbhit()){
   outportb(0x031B, data);
   data = ~data;
   Sleep(500);
     }

    ClosePortTalk();

   return 0;
}
```

ارتباط با درگاه سریال

ارتباط با درگاه سریال کامپیوتر دارای تشریفات مفصلتری نسبت به درگاههای دیگر است. در این بخش، به نحوه استفاده از توابع ویندوز برای ارتباط با درگاه سریال می‌پردازیم. با تغییرات کمی می‌توانید همین مفاهیم را در مورد ارتباط با درگاههای ورودی/خروجی دیگر به کار ببرید.

همانطور که قبلاً در مبحث ارتباط سریال بیان شد، یک بسته اطلاعاتی در ارتباط سریال غیرهمگام شامل بیت آغاز، بیتهای اطلاعاتی، بیت توازن و بیت توقف است. برای بنا نهادن یک ارتباط سریال صحیح، باید ساختار بسته اطلاعاتی و مشخصات دیگر ارتباط (مانند سرعت و نحوه کنترل جریان داده) بین فرستنده و گیرنده یکسان باشد.

از دیدگاه نرم‌افزار، در یک ارتباط سریال مراحل زیر وجود دارند:

- ✓ گشودن درگاه سریال
- ✓ تنظیم ویژگیهای ارتباط سریال
- ✓ تنظیم سرریزهای زمانی[1]
- ✓ خواندن اطلاعات از درگاه سریال یا نوشتن در آن
- ✓ بستن درگاه سریال

برای هرکدام از اعمالی که در بالا ذکر شد، تابعی می‌نویسیم که عملیات مورد نظر را انجام دهد. در توابع ویندوز با درگاههای ورودی/خروجی به صورت فایل برخورد می‌شود. برای باز کردن درگاه از تابع CreateFile، برای خواندن و نوشتن به ترتیب از توابع ReadFile و WriteFile و برای بستن آن از تابع CloseFile استفاده می‌شود.

توابعی که در ادامه بررسی می‌شود را در یک فایل سرآیند به نام COMPort.h بنویسید و در برنامه‌های ارتباطی سریال آن را شامل کنید. فایل windows.h را در این فایل شامل کنید. تعریف زیر را به صورت عمومی در این فایل انجام دهید:

```
HANDLE hComm;
```

متغیر hComm دستگیره‌ای برای دسترسی به درگاه مورد نظر است که در توابع بعدی مورد استفاده قرار می‌گیرد. به عبارتی می‌توان این دستگیره را نام منطقی فایل در برنامه دانست.

تابع گشودن درگاه سریال

```
BOOL OpenPort(char *Port)
{
      char   portname[20];
      strcpy(portname," //./");
      strcat(portname,Port);

      hComm = CreateFile(portname,
                                 GENERIC_READ | GENERIC_WRITE,
                        0,
                        0,
                                OPEN_EXISTING,
                        0,
                        0);
      if (hComm == INVALID_HANDLE_VALUE){
                  printf("\nCreateFile Error Code = %d", GetLastError());
            return FALSE;
      }
      else
            return TRUE;
}
```

اگر درگاه به درستی باز شود، خروجی این تابع TRUE و در غیر این صورت FALSE است. پارامتر ورودی این تابع، نام درگاه سریال مورد نظر است؛ مثلاً فراخوانی تابع به صورت ;("OpenPort("COM1"، درگاه COM1 را گشوده

[1] Time-outs

و دستگیره آن را در متغیر hComm قرار می‌دهد. از تابع CreateFile برای باز کردن درگاه استفاده شده است. از پارامتر دوم این تابع مشخص است که درگاه هم برای خواندن و هم برای نوشتن گشوده شده است.

برای باز کردن درگاه می‌توان به دو روش «با همپوشانی» (OVERLAPPED) و «بدون همپوشانی» (NON-OVERLAPPED) عمل کرد. در این نوشته (با مقداردهی پارامترهای CreateFile) از روش بدون همپوشانی استفاده شده است. برای اطلاعات بیشتر راجع به این موضوع به مستندات MSDN مراجعه کنید.

اگر در انجام تابع CreateFile خطایی رخ دهد، این تابع به جای برگرداندن یک دستگیره معتبر، وقوع خطایی را گزارش می‌کند. تابع GetLastError، کد آخرین خطایی که رخ داده است را ارائه می‌کند. به کمک مستندات MSDN می‌توانید جزییات این خطا را ببینید. در ادامه از توابعی متعلق به ویندوز استفاده می‌کنیم که هرکدام یک خروجی منطقی دارند که در صورت موفقیت TRUE و در غیر این صورت FALSE می‌باشد. اگر خروجی توابع فوق (که در متغیر RetValue قرار می‌گیرند) FALSE باشد به کمک تابع GetLastError کد خطای رخ داده شده نمایش داده می‌شود.

بعضی از خطاهای مهم عبارتند از:

- **خطای شماره ۲ (ERROR_FILE_NOT_FOUND):** هنگامی رخ می‌دهد که برای باز کردن درگاهی تلاش شود که در سیستم موجود نیست. مثلاً اگر درگاه com3 در سیستم موجود نباشد و با تابع CreateFile سعی کنید آن را باز کنید، این خطا گزارش می‌شود.
- **خطای شماره ۵ (ERROR_ACCESS_DENIED):** اگر برای باز کردن درگاهی تلاش کنید که قبلاً توسط برنامه دیگری باز شده این خطا رخ می‌دهد.
- **خطای شماره ٦ (ERROR_INVALID_HANDLE):** اگر در باز کردن درگاه موفق نباشید و دستگیره برگردانده شده توسط تابع CreateFile نامعتبر باشد، تلاش برای استفاده از این دستگیره در توابع دیگر باعث وقوع این خطا می‌شود.
- **خطای شماره ۹۹۵ (ERROR_OPERATION_ABORTED):** نشانه وقوع خطا در عملیات خواندن یا نوشتن است.

تابع پیکربندی درگاه سریال

```
BOOL ConfigurePort(DWORD BaudRate, BYTE ByteSize,
                   DWORD fParity, BYTE Parity, BYTE StopBits)
{
    BOOL   RetValue;
    DCB    m_dcb;

    RetValue = GetCommState(hComm, &m_dcb);
    if (RetValue == FALSE)
    {
        printf("\nGetCommState Error Code = %d", GetLastError());
     CloseHandle(hComm);
       return FALSE;
    }
    m_dcb.BaudRate =BaudRate;
    m_dcb.ByteSize = ByteSize;
    m_dcb.Parity =Parity ;
    m_dcb.StopBits =StopBits;
```

```
        m_dcb.fBinary=TRUE;
        m_dcb.fDsrSensitivity=FALSE;
        m_dcb.fParity=fParity;
        m_dcb.fOutX=FALSE;
        m_dcb.fInX= FALSE;
        m_dcb.fNull= FALSE;
        m_dcb.fAbortOnError=TRUE;
        m_dcb.fOutxCtsFlow=FALSE;
        m_dcb.fOutxDsrFlow= FALSE;
        m_dcb.fDtrControl=DTR_CONTROL_DISABLE;
        m_dcb.fDsrSensitivity= FALSE;
        m_dcb.fRtsControl=RTS_CONTROL_DISABLE;
        m_dcb.fOutxCtsFlow= FALSE;
        m_dcb.fOutxCtsFlow= FALSE;

        RetValue = SetCommState(hComm, &m_dcb);
        if(RetValue == FALSE)
        {
            printf("\nSetCommState Error Code = %d", GetLastError());
         CloseHandle(hComm);
         return FALSE;
        }
        return TRUE;
    }
```

در ویندوز ساختاری به نام DCB تعریف شده که برای پیکربندی ارتباط سریال از آن استفاده می‌شود. متغیری به نام m_dcb از نوع ساختار DCB تعریف می‌کنیم. تابع GetCommState از توابع ویندوز است که پیکربندی فعلی درگاهی که دستگیره آن ذکر شده را در متغیر m_dcb ذخیره می‌کند. این کار برای حفظ مشخصات پیش‌فرض درگاهها که مایل به تغییر آنها نیستیم مفید است. اکنون برای پیکربندی درگاه مورد نظر باید فیلدهای متغیر ساختاری m_dcb را مقداردهی کنیم. شرح این فیلدها و مقادیری که می‌توانند به خود بگیرند و معانی آنها به تفصیل در مستندات MSDN موجود است. در اینجا تنها به چهار پارامتر ورودی تابع ConfigurePort که چهار فیلد مهم متغیر m_dcb را مقداردهی می‌کنند می‌پردازیم:

- ✓ **پارامتر BaudRate:** به کمک این پارامتر می‌توان سرعت ارتباط سریال را تنظیم کرد. این سرعت باید مطابق با سرعت ارتباطی دستگاهی باشد که به درگاه سریال متصل است. مثلاً برای سرعت ۹۶۰۰ بیت در ثانیه باید این پارمتر را برابر 9600 یا CBR_9600 مقداردهی کنید. مقادیر معتبر برای درگاه سریال کامپیوتر عبارتند از:

CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200, CBR_128000, CBR_256000

- ✓ **پارامتر ByteSize:** این پارامتر، تعداد بیتهای داده در بسته اطلاعاتی را نشان می‌دهد که می‌تواند از ۴ تا ۸ باشد.
- ✓ **پارامتر fParity:** اگر مقدار این پارامتر TRUE باشد، بررسی توازن انجام خواهد شد.
- ✓ **پارامتر Parity:** این پارامتر نوع توازن مورد استفاده را مشخص می‌کند. مقادیر قابل قبول عبارتند از EVENPARITY (توازن زوج)، ODDPARITY (توازن فرد)، NOPARITY (بدون توازن)، MARKPARITY (توازن همیشه «یک»)، SPACEPARITY (توازن همیشه «صفر»)
- ✓ **پارامتر StopBits:** برای مشخص کردن تعداد بیتهای توقف در بسته اطلاعاتی از این پارامتر استفاده می‌کنیم. مقادیر معتبر عبارتند از:

ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS

به نکته مهمی توجه کنید. تابع ConfigurePort از مکانیسمی برای کنترل جریان داده استفاده نمی‌کند. به بیان دیگر، نه RTS/CTS و نه XON/XOFF را کنترل نمی‌کند. اگر مایل به کنترل جریان داده هستید، با انجام اصلاحات روی سخت‌افزار و نیز فیلدهای دیگر متغیر m_dcb می‌توانید این کار را انجام دهید.

پس از تنظیم فیلدهای متغیر m_dcb، به کمک تابع SetCommState این متغیر را به درگاهی که دستگیره آن در تابع ذکر شده نسبت می‌دهیم.

مثالی از فراخوانی تابع ConfigurePort به صورت زیر است:

```
ConfigurePort(CBR_9600, 8, TRUE, EVENPARITY , ONESTOPBIT )
```

پیکربندی درگاه سریال با سرعت ۹۶۰۰ بیت در ثانیه، ۸ بیت داده، توازن زوج و یک بیت توقف انجام می‌شود. دستگاه مخاطب درگاه سریال نیز باید از همین پیکربندی استفاده کند. اگر خروجی تابع TRUE باشد، پیکربندی به درستی انجام شده است.

تنظیم ملاحظات زمانی

توابع GetCommTimeouts و SetCommTimeouts و ساختاری به نام COMMTIMEOUTS تنظیم مسایل زمانی ارتباط سریال در برنامه‌نویسی ویندوز را انجام می‌دهند.

```
BOOL  SetCommunicationTimeouts(DWORD ReadIntervalTimeout,
                               DWORD ReadTotalTimeoutMultiplier,
                               DWORD ReadTotalTimeoutConstant,
                               DWORD WriteTotalTimeoutMultiplier,
                               DWORD WriteTotalTimeoutConstant)
{

    BOOL          RetValue;
    COMMTIMEOUTS  m_CommTimeouts;

    RetValue = GetCommTimeouts (hComm, &m_CommTimeouts);
    if(RetValue == FALSE){
        printf("\nGetCommTimeouts Error Code = %d", GetLastError());
     CloseHandle(hComm);
         return FALSE;
    }

    m_CommTimeouts.ReadIntervalTimeout =ReadIntervalTimeout;
    m_CommTimeouts.ReadTotalTimeoutConstant=
                     ReadTotalTimeoutConstant;
    m_CommTimeouts.ReadTotalTimeoutMultiplier=
                     ReadTotalTimeoutMultiplier;
    m_CommTimeouts.WriteTotalTimeoutConstant=
                   WriteTotalTimeoutConstant;
    m_CommTimeouts.WriteTotalTimeoutMultiplier=
                   WriteTotalTimeoutMultiplier;

    RetValue = SetCommTimeouts (hComm, &m_CommTimeouts);

    if(RetValue == FALSE)
    {
        printf("\nSetCommTimeouts Error Code = %d",    GetLastError());
     CloseHandle(hComm);
     return FALSE;
    }
    return TRUE;
}
```

متغیری از نوع ساختار COMMTIMEOUTS به نام m_CommTimeouts تعریف شده که فیلدهای آن برای تنظیم مسایل زمانی ارتباط سریال به کار می‌روند. به کمک تابع GetCommTimeouts، مشخصات فعلی ارتباطی که دستگیره آن مشخص شده در متغیر m_CommTimeouts ذخیره می‌شود و پس از تغییرات لازم به کمک تابع SetCommTimeouts ویژگیهای تنظیم شده را به درگاهی که دستگیره آن را مشخص کرده‌ایم، نسبت می‌دهیم.

پارامترهای ورودی تابع SetCommunicationTimeouts را در زیر بررسی می‌کنیم:

- ✓ **پارامتر ReadIntervalTimeout:** ماکزیمم زمان مجاز (برحسب میلی‌ثانیه) بین دریافت دو کاراکتر متوالی را مشخص می‌کند. اگر زمان سپری شده از این مقدار بیشتر شود، تابع ReadFile که برای خواندن درگاه به کار می‌رود، عملیات خواندن را پایان‌یافته فرض کرده و به تابع فراخواننده بازمی‌گردد. اگر مقدار این پارامتر را صفر تعیین کنید، به معنای استفاده نکردن از این مشخصه ارتباط سریال است.
- ✓ **پارامتر ReadTotalTimeoutConstant:** یک مقدار ثابت بر حسب میلی ثانیه است که برای محاسبه مجموع مدت زمان بین اعمال خواندن به کار می‌رود. برای هر عمل خواندن، مقدار این ثابت به حاصلضرب تعداد بایتهای درخواستی در پارامتر ReadTotalTimeoutMultiplier اضافه می‌شود و مجموع زمانی که اصولاً عمل خواندن باید به طول بیانجامد را معلوم می‌کند. اگر زمان سپری شده از این مقدار بیشتر شود، تابع ReadFile که برای خواندن درگاه به کار می‌رود، عملیات خواندن را پایان‌یافته فرض کرده و به تابع فراخواننده بازمی‌گردد. اگر دو پارامتر ذکر شده را صفر کنید، از استفاده از مجموع زمان بین اعمال خواندن صرف‌نظر کرده‌اید.
- ✓ **پارامتر ReadTotalTimeoutMultiplier:** ماکزیمم زمان انتقال هر بایت بر حسب میلی‌ثانیه به کمک این پارامتر مشخص می‌شود. تعداد بایتهای درخواستی در عمل خواندن در این پارامتر ضرب و با پارامتر ReadTotalTimeoutConstant جمع می‌شود تا مجموع زمان مجاز عملیات خواندن مشخص شود.

پارامترهای ReadTotalTimeoutConstant و ReadTotalTimeoutMultiplier مانند پارامترهای بالا در عملیات نوشتن عمل می‌کنند.

مثلاً فراخوانی تابع فوق به صورت ;SetCommunicationTimeouts(10,500,0,0,0) مدت زمان ماکزیمم عملیات خواندن برای یک بلوک از داده‌ها را ۵۰۰ میلی‌ثانیه تعیین می‌کند. خروجی TRUE به معنای صحت انجام عملیات پیکربندی زمانی ارتباط سریال است.

تابع نوشتن در درگاه سریال

همانطور که قبلاً ذکر شد، برای نوشتن در درگاه سریال از تابع نوشتن در فایل استفاده می‌شود.

```
BOOL WriteByte(BYTE bybyte)
{
    DWORD iBytesWritten=0;
    BOOL  RetValue;

    RetValue = WriteFile(hComm,&bybyte,1,&iBytesWritten,NULL);

    if(RetValue == FALSE){
        printf("\nWriteByte Error Code = %d",GetLastError());
        return FALSE;
    }
    else
        return TRUE;
```

```
}
```

بایتی که می‌خواهیم به خروجی ارسال شود را به عنوان پارامتر ورودی به تابع ارسال می‌کنیم. پارامترهای تابع WriteFile نیز جالب توجه هستند. پارامتر اول دستگیره درگاه مورد نظر، پارامتر دوم اشاره‌گر به ابتدای بافر حاوی اطلاعاتی که می‌خواهیم نوشته شود و پارامتر سوم تعداد بایتهای مورد نظر برای نوشتن است. پارامتر چهارم بعد از اجرای دستور WriteFile تعداد بایتهایی که با موفقیت نوشته شده را در خود نگه می‌دارد. نکته جالب در مورد این تابع این است که لزومی به ارسال بایتهای اطلاعاتی به صورت تک‌تک در برنامه نیست و می‌توان با قرار دادن آنها در یک بافر، کل اطلاعات را با یک دستور WriteFile به درگاه فرستاد (که البته توسط خود سیستم تک‌تک به درگاه ارسال خواهد شد). تابعی که در بالا نوشته شده تنها یک بایت را که به صورت پارامتر برای آن ارسال شده، در خروجی می‌نویسد. اگر عملیات تابع به صورت صحیح انجام شود، خروجی آن TRUE خواهد بود.

تابع خواندن اطلاعات از درگاه سریال

همانگونه که از بخش ارتباط سریال به خاطر دارید، برای ارسال اطلاعات به درگاه سریال کافی است اطلاعات مورد نظر در این درگاه نوشته شود. اما برای دریافت اطلاعات از درگاه سریال داستان کمی متفاوت است. برای این کار باید ابتدا از دریافت کاراکتر مطمئن شویم و سپس درگاه را بخوانیم.

تابع زیر برای اطمینان از دریافت کاراکتر نوشته شده است. اگر خروجی تابع TRUE باشد یعنی کاراکتر دریافت شده است:

```
BOOL RecvData(void)
{

    DWORD   dwErrorFlags;
    COMSTAT ComStat;
    BOOL  RetValue;

    RetValue = ClearCommError(hComm,&dwErrorFlags,&ComStat);
    if (RetValue == FALSE){
        printf("\nClearCommError Error Code = %d", GetLastError());
     return FALSE;
    }

    if  (ComStat.cbInQue == 0)
        return FALSE;
    else
        return TRUE;
}
```

ساختار COMSTAT در ویندوز می‌تواند مشخصات ارتباطی یک وسیله را نگهداری کند. متغیر ComStat که از این نوع تعریف شده با تابع ClearCommError مقداردهی می‌شود و فیلدهای آن اطلاعاتی مختلفی در مورد ارتباط سریال را نگهداری می‌کنند که ما از فیلد cbInQue استفاده کرده‌ایم. این فیلد حاوی تعداد بایتهایی است که دریافت شده، اما هنوز به کمک تابع ReadFile خوانده نشده است.

حال به تابع خواندن اطلاعات از درگاه سریال می‌پردازیم:

```
BOOL ReadByte(BYTE *resp)
{
    BYTE rx;
    *resp = 0;
    BOOL  RetValue;

    DWORD dwBytesRead = 0;
```

```
        RetValue = ReadFile(hComm, &rx, 1, &dwBytesRead, NULL);

        if (RetValue == TRUE)
        {
            if (dwBytesRead == 1)
            {
                *resp = rx;
             return TRUE;
          }
        }
        printf("\nReadByte Error Code = %d",GetLastError());
        return FALSE;
    }
```

پس از آنکه به کمک تابع RecvData از دریافت کاراکتر مطمئن شدیم، به کمک تابع ReadByte آن را می‌خوانیم. این تابع مانند قبل، از تابع خواندن فایل (ReadFile) برای دریافت اطلاعات از درگاه استفاده می‌کند. پارامتر چهارم تابع ReadFile تعداد بایتهای دریافتی را پس از اجرای تابع فوق نشان می‌دهد که اگر مقدار آن برابر یک باشد، یعنی یک بایت دریافت شده و بایت فوق که در پارامتر دوم تابع قرار گرفته مورد استفاده قرار می‌گیرد. خروجی TRUE نشان‌دهنده صحت دریافت اطلاعات است.

بستن درگاه سریال

تابع زیر درگاه ارتباطی را می‌بندد:

```
void ClosePort()
{
    BOOL  RetValue;
    RetValue = CloseHandle(hComm);
    if (RetValue == FALSE)
     printf("\nReadByte Error Code = %d",GetLastError());
}
```

توابع فوق را در سرآیندی به نام ComPort.h نوشته و در برنامه‌های بعدی از آنها استفاده می‌کنیم. مانند قبل، توابع فوق با فرض استفاده از برنامه‌های مد کنسول نوشته شده‌اند. اگر مایل به نوشتن برنامه‌های تحت ویندوز هستید می‌توانید دستورات مد کنسول (مثلاً printf) را با توابع ویندوز (مثلاً MessageBox) عوض کنید.

راه بهتر، نوشتن این توابع به صورت یک کلاس است که استفاده از آنها در برنامه‌های شیءگرا و تغییر و به‌روز کردن آنها ساده‌تر باشد. به کمک برنامه‌نویسی MFC یا API حتی می‌توانید برنامه‌های حرفه‌ای مبتنی بر پیغام[1] بنویسید.

مثالی از ارسال داده‌ها به درگاه سریال کامپیوتر

برنامه زیر را در محیط کنسول Visual Studio بنویسید و اجرا کنید. در سوی مقابل، باید سخت‌افزاری وجود داشته باشد (مثلاً یک میکروکنترلر) که اطلاعات را به صورت سریال (با همین پیکربندی) از درگاه کامپیوتر دریافت کند و روی یکی از درگاههای میکروکنترلر نمایش دهد. اگر به این درگاه میکروکنترلر هشت عدد LED متصل کنید، باید شمارش اعداد صفر تا ۱۵ را به صورت متوالی روی LEDها مشاهده کنید.

```
#include "stdafx.h"
#include <windows.h>
#include <ConIO.h>
#include "ComPort.h"
```

[1] Message-Based

```
int main(int argc, char* argv[]){

  OpenPort("COM1");
  ConfigurePort(CBR_9600,8,TRUE,NOPARITY,ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);
//**************************
  BYTE data = 0;

  while (!kbhit()){

      WriteByte(data);
      Sleep(500);

      if (data == 15)
       data = 0;
      else
       data++;
  }
//*************************
  ClosePort();
  return 0;
}
```

برای تست این برنامه راه ساده‌تری نیز وجود دارد. اگر بورد اصلی کامپیوتر شما دارای دو درگاه COM باشد، می‌توانید به کمک یک کابل Null Modem (که در بخش ارتباط سریال از آن سخن گفتیم)، دو درگاه سریال را به هم متصل کنید. اگر در این حالت دو پنجره HyperTerminal باز کنید و یکی را به درگاه COM1 و دیگری را به درگاه COM2 نسبت داده و با پیکربندی یکسان تنظیم کنید، هرآنچه روی یکی از دو پنجره تایپ کنید در پنجره دیگر دیده خواهد شد.

برای تست برنامه بالا، پنجره HyperTerminal که به درگاه COM1 متصل است را ببندید تا تنها پنجره منتسب به درگاه COM2 باز باشد. اگر چنین نکنید، برنامه بالا نمی‌تواند درگاه COM1 را باز کند و خطای شماره ۵ (ERROR_ACCESS_DENIED) رخ می‌دهد.

حال اگر برنامه فوق را اجرا کنید، داده‌هایی که برنامه شما از طریق درگاه COM1 ارسال می‌کند را روی پنجره HyperTerminal متعلق به درگاه COM2 مشاهده خواهید کرد. البته چون Hyperterminal اعدادی که دریافت می‌کند را به فرمت ASCII نمایش می‌دهد، اطلاعات قابل فهمی روی پنجره HyperTerminal نمی‌بینید. برای حل این مشکل، در قسمتی از برنامه بالا که بین دو خط ستاره مشخص شده، عدد 0 را با کاراکتر 'A' و عدد 15 را با کاراکتر 'Z' جایگزین کنید. با این کار تا زمانی که برنامه ادامه دارد، حروف 'A' تا 'Z' را روی پنجره HyperTerminal خواهید دید.

اگر بورد اصلی کامپیوتر شما تنها یک درگاه COM داشته باشد، به ناچار باید از دو کامپیوتر استفاده کنید؛ روی یکی برنامه خودتان و روی دیگری برنامه HyperTerminal را اجرا کنید. در هر دو طرف از درگاه COM1 استفاده کنید.

نکته جالب دیگر راجع به برنامه بالا این است که همانطور که در توضیح دستور WriteFile گفته شد، نیازی به ارسال داده‌ها به صورت تک‌به‌تک نیست و می‌توان کل آنها را با هم ارسال کرد. برنامه زیر، نسخه اصلاح شده برنامه بالا است:

```
#include "stdafx.h"
#include <windows.h>
#include <ConIO.h>
#include "ComPort.h"
```

```
int main(int argc, char* argv[]){

  OpenPort("COM1");
  ConfigurePort(CBR_9600,8,TRUE,NOPARITY,ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);

  BYTE data[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  DWORD num;

  WriteFile(hComm,data,16,&num,NULL);

  printf("\n Numberof written bytes = %d\n",num);
  ClosePort();
  return 0;
}
```

دستور WriteFile، ۱۶ بایت (پارامتر سوم) از اطلاعات آرایه data (پارامتر دوم) را به درگاهی که دستگیره آن در پارامتر اول مشخص شده ارسال می‌کند. سیستم‌عامل پس از ارسال داده‌ها، تعداد واقعی بایتهایی که توانسته ارسال کند را در متغیر num ذخیره می‌کند. با اجرای این برنامه، هر ۱۶ بایت اطلاعات پشت سر هم ارسال می‌شوند. طبیعی است که سخت‌افزار مقابل هم باید طوری برنامه‌ریزی شود که بتواند این رشته بایت (که بدون استفاده از روشهای کنترل جریان داده ارسال می‌شوند) را دریافت کند. برای تست این برنامه به شیوه‌ای که قبلاً گفته شد، می‌توانید دو درگاه COM را به هم متصل کنید. اگر آرایه data را با مقادیر کاراکتری مقداردهی کنید، کل رشته ارسالی را در پنجره HyperTerminal گیرنده مشاهده خواهید کرد.

مثالی از دریافت داده‌ها از درگاه سریال کامپیوتر

برنامه زیر را در محیط کنسول Visual Studio بنویسید و اجرا کنید. در سوی مقابل، باید سخت‌افزاری وجود داشته باشد (مثلاً یک میکروکنترلر) که اطلاعاتی را به صورت سریال (با همین پیکربندی) بایت به بایت برای کامپیوتر ارسال کند. برنامه مقابل منتظر می‌ماند تا یک بایت اطلاعات دریافت کند و سپس آن را به فرمت شانزده‌تایی نمایش می‌دهد. اجرای برنامه با فشار یک کلید به پایان می‌رسد.

```
#include "stdafx.h"
#include <ConIO.h>
#include "ComPort.h"

int main(int argc, char* argv[]){

  OpenPort("com1");
  ConfigurePort(CBR_9600,8,TRUE,NOPARITY,ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);

  BYTE data = 0;

  while (1){

      while (!RecvData() && !kbhit());
     if (kbhit())
           break;
     else{
           ReadByte(&data);
           printf("%X",data);
         }
  }
  ClosePort();
  return 0;
}
```

می‌توانید به شیوه قبل با اتصال دو درگاه COM کامپیوتر به هم، این برنامه را تست کنید. در این حال، برنامه بالا هرچه را در برنامه HyperTerminal تایپ کنید، در دستگاه شانزده‌تایی نمایش خواهد داد.

حال برنامه مقابل را ببینید. در این برنامه، ابتدا منتظر می‌مانیم تا اطلاعاتی دریافت شود و سپس به کمک تابع ReadFile، ۱۶ بایت اطلاعات را به صورت متوالی خواهیم خواند. اگر این برنامه و برنامه ارسال داده‌ها به کمک WriteFile (که قبلاً گفته شد) به شیوه اتصال درگاههای COM تست شود، شانزده بایتی که برنامه فرستنده ارسال می‌کند توسط برنامه گیرنده دریافت و نمایش داده خواهد شد. فراموش نکنید که ابتدا برنامه گیرنده و سپس برنامه فرستنده را اجرا کنید.

```
   #include "stdafx.h"
   #include <ConIO.h>
   #include "ComPort.h"

   int main(int argc, char* argv[]){

     OpenPort("com1");
     ConfigurePort(CBR_9600,
                   8,
                   TRUE,
                   NOPARITY,
                   ONESTOPBIT);
     SetCommunicationTimeouts(10,0,500,0,0);

     BYTE data[16];
     while(!RecvData());      //Wait      for
reception
     DWORD num = 0;
     ReadFile(hComm,data,16,&num,NULL);
     for (int i = 0;i < num;i++)
        printf("%c",data[i]);
     getch();
     ClosePort();
     return 0;
```

منابع این فصل

- http://www.beyondlogic.org/porttalk/porttalk.htm
- Microsoft Windows NT Device Driver Kit
- Microsoft Win32 SDK
- Intel Architecture Developer's Manual - Basic Architecture, Order Number 243190.
- Intel Architecture Developer's Manual - Instruction Set Reference Manual, Order Number 243191.
- Intel Architecture Developer's Manual - System Programming Guide, Order Number 243192.
- http://www.codeproject.com/KB/system/cserialcom.aspx

پیوست – خلاصه‌ای از کاتالوگ پردازنده ۸۰۸۸

intel

8088
8-BIT HMOS MICROPROCESSOR
8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with 8086 CPU
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Two Clock Rates:
  — 5 MHz for 8088
  — 8 MHz for 8088-2
- Available in EXPRESS
  — Standard Temperature Range
  — Extended Temperature Range

The Intel 8088 is a high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS-II), and packaged in a 40-pin CERDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with 8086 software and 8080/8085 hardware and peripherals.

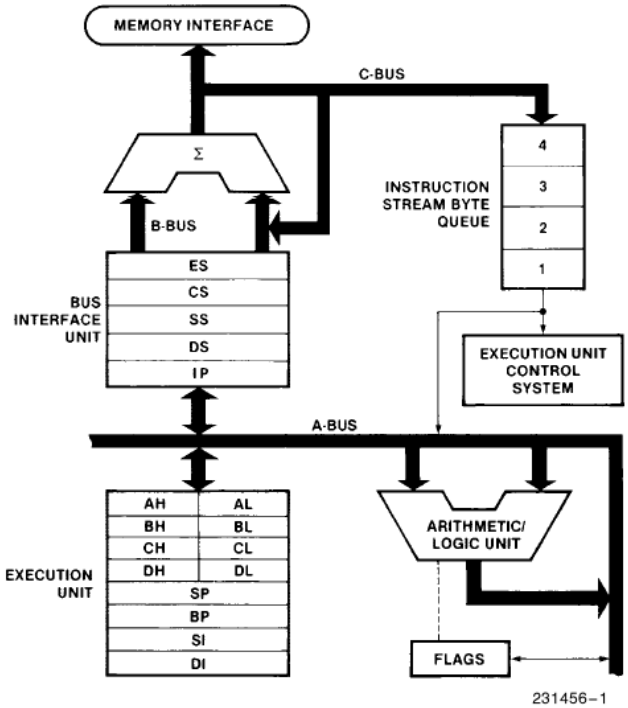


Figure 1. 8088 CPU Functional Block Diagram

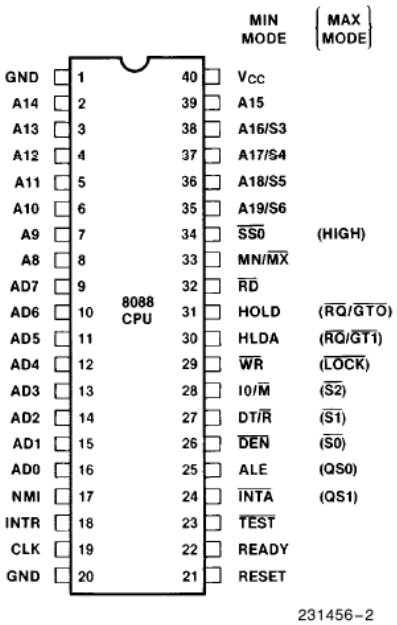


Figure 2. 8088 Pin Configuration

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$	30, 31	I/O	**REQUEST/GRANT:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (See Figure 8): 1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 8088 (pulse 1). 2. During a T4 or TI clock cycle, a pulse one clock wide from the 8088 to the requesting master (pulse 2), indicates that the 8088 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released. 3. A pulse one CLK wide from the requesting master indicates to the 8088 (pulse 3) that the "hold" request is about to end and that the 8088 can reclaim the local bus at the next CLK. The CPU then enters T4. Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met: 1. Request occurs on or before T2. 2. Current cycle is not the low bit of a word. 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active (LOW). The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state off in "hold acknowledge".
QS1, QS0	24, 25	O	**QUEUE STATUS:** provide status to allow external tracking of the internal 8088 instruction queue. The queue status is valid during the CLK cycle after which the queue operation is performed.

QS1	QS0	Characteristics
0(LOW)	0	No Operation
0	1	First Byte of Opcode from Queue
1(HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

—	34	O	Pin 34 is always high in the maximum mode.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD, HLDA	31, 30	I, O	**HOLD:** indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or Ti clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. HOLD and HLDA have internal pull-up resistors. Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.
$\overline{SSO}$	34	O	**STATUS LINE:** is logically equivalent to $\overline{S0}$ in the maximum mode. The combination of $\overline{SSO}$, IO/$\overline{M}$ and DT/$\overline{R}$ allows the system to completely decode the current bus cycle status.

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SSO}$	Characteristics
1(HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O Port
1	1	0	Write I/O Port
1	1	1	Halt
0(LOW)	0	0	Code Access
0	0	1	Read Memory
0	1	0	Write Memory
0	1	1	Passive

The following pin function descriptions are for the 8088/8288 system in maximum mode (i.e., MN/$\overline{MX}$ = GND). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	26–28	O	**STATUS:** is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 8288 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 and Tw is used to indicate the end of a bus cycle. These signals float to 3-state OFF during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Characteristics
0(LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1(HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NMI	17	I	**NON-MASKABLE INTERRUPT:** is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**CLOCK:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** is the +5V ±10% power supply pin.
GND	1, 20		**GND:** are the ground pins.
MN/$\overline{MX}$	33	I	**MINIMUM/MAXIMUM:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8088 minimum mode (i.e., MN/$\overline{MX}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
IO/$\overline{M}$	28	O	**STATUS LINE:** is an inverted maximum mode $\overline{S2}$. It is used to distinguish a memory access from an I/O access. IO/$\overline{M}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O = HIGH, M = LOW). IO/$\overline{M}$ floats to 3-state OFF in local bus "hold acknowledge".
$\overline{WR}$	29	O	**WRITE:** strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/$\overline{M}$ signal. WR is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
$\overline{INTA}$	24	O	**INTA:** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.
ALE	25	O	**ADDRESS LATCH ENABLE:** is provided by the processor to latch the address into an address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**DATA TRANSMIT/RECEIVE:** is needed in a minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/$\overline{R}$ is equivalent to $\overline{S1}$ in the maximum mode, and its timing is the same as for IO/$\overline{M}$ (T = HIGH, R = LOW). This signal floats to 3-state OFF in local "hold acknowledge".
$\overline{DEN}$	26	O	**DATA ENABLE:** is provided as an output enable for the data bus transceiver in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access, and for $\overline{INTA}$ cycles. For a read or $\overline{INTA}$ cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. $\overline{DEN}$ floats to 3-state OFF during local bus "hold acknowledge".

Table 1. Pin Description

The following pin function descriptions are for 8088 systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 8088 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD7–AD0	9–16	I/O	**ADDRESS DATA BUS:** These lines constitute the time multiplexed memory/IO address (T1) and data (T2, T3, Tw, T4) bus. These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A15–A8	2–8, 39	O	**ADDRESS BUS:** These lines provide address bits 8 through 15 for the entire bus cycle (T1–T4). These lines do not have to be latched by ALE to remain valid. A15–A8 are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A19/S6, A18/S5, A17/S4, A16/S3	35–38	O	**ADDRESS/STATUS:** During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown. This information indicates which segment register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge".

S4	S3	Characteristics
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
S6 is 0 (LOW)		

$\overline{RD}$	32	O	**READ:** Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/$\overline{M}$ pin or S2. This signal is used to read devices which reside on the 8088 local bus. $\overline{RD}$ is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 8088 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8088 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.
INTR	18	I	**INTERRUPT REQUEST:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{TEST}$	23	I	**$\overline{TEST}$:** input is examined by the "wait for test" instruction. If the $\overline{TEST}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

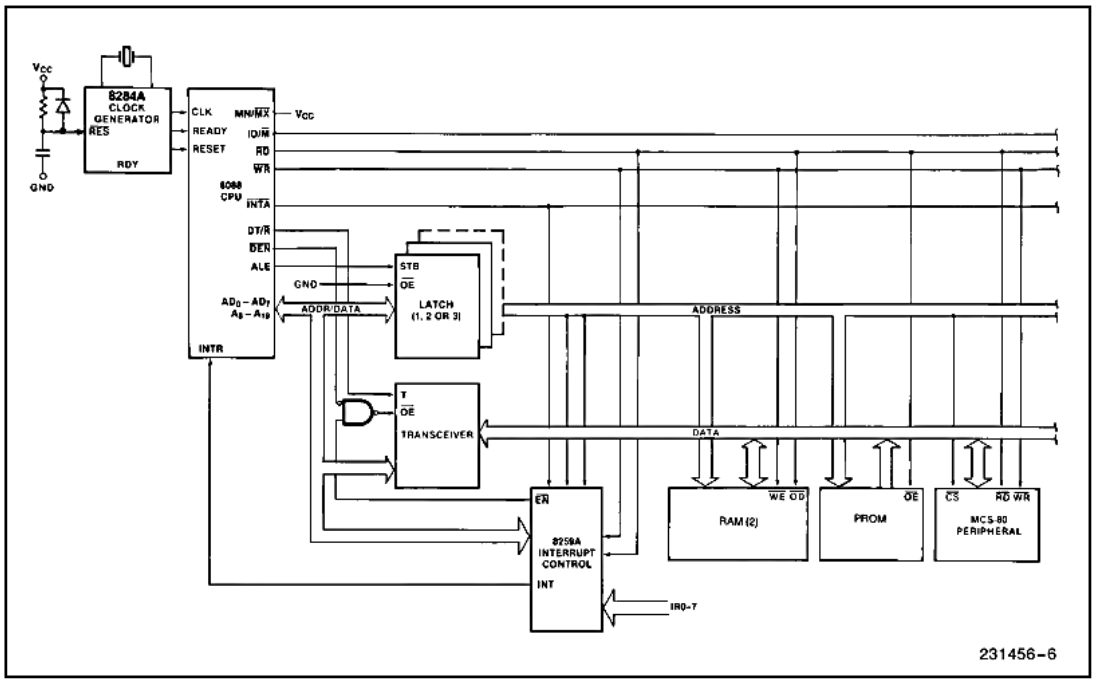


Figure 6. Demultiplexed Bus Configuration

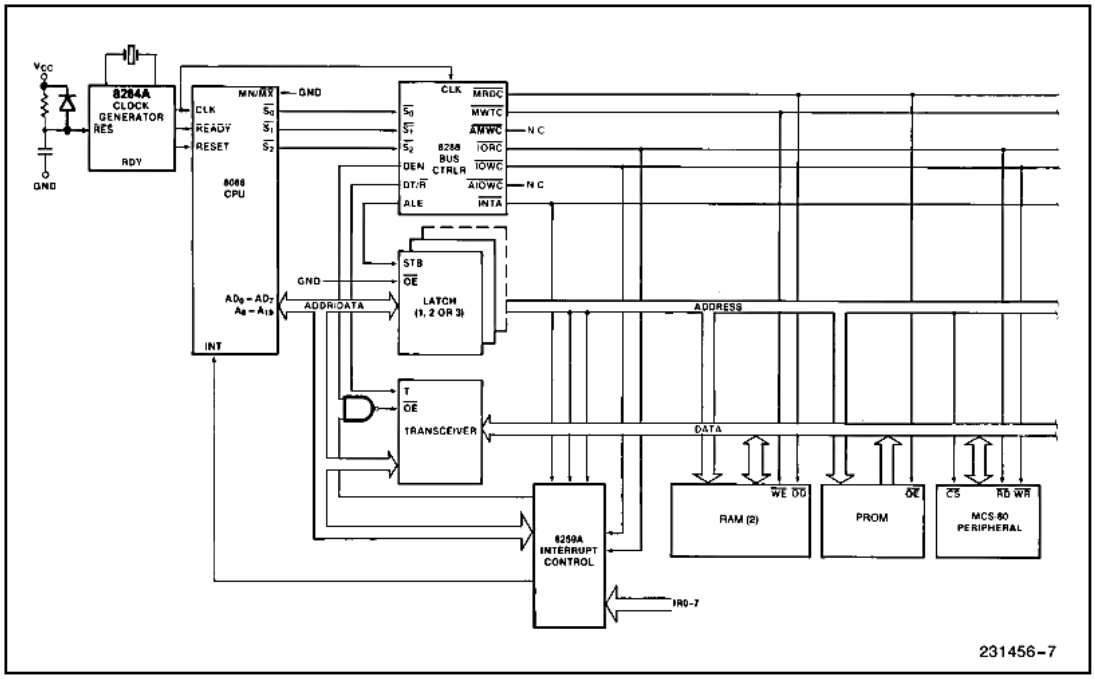


Figure 7. Fully Buffered System Using Bus Controller

کتاب‌شناسی

- "The 80x86 IBM PC & Compatible Computers", Muhammed Ali Mazidi and et al, Prentice Hall,2000.

کتاب فوق با نام مدارهای واسط توسط دکتر قدرت سپیدنام به فارسی ترجمه شده است.

- "اصول کامل راه‌اندازی و کنترل دستگاههای جانبی توسط کامپیوتر"، محسن شکیبافر، انتشارات نص، ۱۳۸۴.
- "مرجع علمی-کاربردی سخت‌افزار"، شیرزاد شهریاری، انتشارات جهاد دانشگاهی مشهد، ۱۳۸۴.
- "اسلاتهای توسعه و طراحی کارتها"، شیرزاد شهریاری، انتشارات جهاد دانشگاهی مشهد، ۱۳۷۸.
- "دیجیتال پایه و طراحی مدارهای واسط"، شیرزاد شهریاری، انتشارات پرتو نگار، ۱۳۸۵.
- "PC Interfaces under Windows", Burkhard Kaink and Hans-Joachim Berndt, Elektor Electronics Publications, 2002.
- "گذرگاهها و درگاههای کامپیوترهای شخصی"، عبدالمجید منصوریان‌فر و اصغر کریمی، انتشارات دانش‌پژوهان برین – ارکان، ۱۳۸۳
- "IBM personal computer XT, Technical Reference", Volume 1,2, IBM, 1983.
- "IBM personal computer AT, Technical Reference", IBM, 1984.
- "The Intel Microprocessors", Barry B. Brey, Prentice Hall, 2006.
- "آشنایی با قطعات سخت‌افزار کامپیوتر و نحوه کار آنها"، محمدرضا گرمخورانی، انتشارات نوپردازان، ۱۳۸۳
- "طراحی، برنامه نویسی و ربط دهی خانواده 8086/88"، جان افن بک، ترجمه دکتر علی پیروی، انتشارات آستان مقدس، ۱۳۷۷.
- "مدارهای واسطه"، تامپکینز – وبستر، ترجمه دکتر علی پیروی، انتشارات آستان مقدس، ۱۳۸۲.
- "مبانی میکروپروسسورها و مدارهای واسطه"، دکتر سید محمد احدی، انتشارات دانشگاه امیرکبیر، ۱۳۸۲.
- "اصول طراحی سیستمهای میکروپروسسوری"، دکتر سید احمد معتمدی، چاپخانه علمی و فرهنگی، ۱۳۷۷.
- "سیستمهای میکروپروسسوری ۱۶ و ۳۲ بیتی"، دکتر سید احمد معتمدی، چاپخانه علمی و فرهنگی، ۱۳۷۷.

PC INTERFACING

By : Javad Rasti

FREE EBOOK