درون کامپیوتر

در این بخش مقدمه ای را برای سازمان و کار درونی کامپیوترها فراهم می آوریم. مدل بکار رفته یک مدل عمومی است، ولی مفاهیم مورد استفاده قابل اعمال به همه کامپیوترها از جمله IBM , PS/2 و سازگار با آنهاست. قبل از آغاز این مبحث،مروری بر تعاریف برخی از اصطلاحات در کامپیوتر،مانند کیلو (k) ، مگا،گیگا، بایت، RAM, ROM و غیره مفید است.

بعضی اصطلاحات مهم

یکی از امکانات مهم یک کامپیوتر حافظه موجود در آن است. بنابراین اکنون اصطلاحات بکار رفته برای اندازه حافظه در IBM PC ها و سازگار با آنها را بیان می کنیم. از بحث قبل بیاد دارید که بیت یک رقم دودویی بود که می توانست مقدار ۰ یا ۱ داشته باشد. بایت یک مجموعه ۸ بیتی است. نیبل نصف یک بایت، یا ٤ بیت است. کلمه دو بایت یا ١٦ بیت می باشد. نمایش زیر به منظور نشان دادن اندازه نسبی این واحدها ارائه شده است. البته،آنها می توانند هر ترکیبی از صفرها و یک ها باشند.

بیت ۰

نیبل ۰۰۰۰

بایت ۰۰۰۰ ۰۰۰۰

یک کیلوبایت، $۲^{۱۰}$ بایت یا ۱۰۲٤ بایت است. اغلب از K برای بیان آن استفاده می شود. مثلاً برخی از فلاپی دیسک ها (یا دیسک نرم) ۳٥٦ k داده را نگه می دارند. یک مگابایت، یا ساده تر مگ، $۲^{۲۰}$ بایت است. این مقدار، کمی بیش از یک میلیون بایت است و مقدار دقیق آن ۱/۰٤۸/٥۷٦ می باشد. با گذری سریع در ظرفیت به گیگابایت یا $۲^{۳۰}$ بایت می رسیم (بیش از ۱ بیلیون یا میلیارد)، و یک ترابایت نیز $۲^{٤۰}$ بایت است (بیش از ۱ تریلیون). برای مثالی از چگونگی کاربرد آنها،فرض کنید که کامپیوتری دارای ۱٦ مگابایت حافظه باشد. این مقدار برابر با $۲^{۲۰}$ * ۱٦ یا $۲^{۲۰}$ * $۲^{٤}$ یا $۲^{۲٤}$ است. بنابراین ۱٦ مگابایت $۲^{۲٤}$ بایت می باشد.

در میکروکامپیوترها معمولا از دو نوع حافظه استفاده می شود که عبارتند از RAM ، که به معنی حافظه با دستیابی تصادفی است (گاهی هم حافظه خواندن / نوشتن نامیده می شود) و ROM که به معنی حافظه فقط خواندنی می باشد. RAM بوسیله کامپیوتر برای ذخیره سازی موقت برنامه های در حال اجرا مورد استفاده قرار می گیرد. این برنامه ها یا اطلاعات بعد از خاموش شدن کامپیوتر از بین می روند. به همین دلیل، RAM را گاهی حافظه فرار هم می خوانند. ROM برای برنامه ها و اطلاعات لازم در عملکرد کامپیوتر لازم است. اطلاعات در ROM دائمی است و قابل تعویض بوسیله کاربر نمی

باشد و پس از خاموش شدن کامپیوتر هم از بین نمی رود. بنابراین آن را حافظه غیرفرار گوییم.

سازمان درونی کامپیوترها

بخش عملیاتی هر کامپیوتر قابل تفکیک به سه قسمت است: CPU (واحد پردازش مرکزی)، حافظه و وسایل I/O (ورودی / خروجی) ، شکل ۹-۰ ملاحظه شود. نقش CPU اجرای (پردازش) اطلاعات ذخیره شده در حافظه است. عمل وسایل I/O همچون صفحه کلید، مانیتور تصویر (ویدئو) ، تهیه مفاهیم ارتباط و محاوره با CPU است. CPU از طریق رشته ای از سیم ها به نام گذرگاه به حافظه و I/O متصل است. گذرگاه داخل یک کامپیوتر، درست مثل گذرگاههای خیابانی که مردم رااز مکانی به مکانی دیگر هدایت می کنند، اطلاعات را از جایی به جای دیگر انتقال می دهند. در هر کامپیوتر سه نوع گذرگاه موجود است: گذرگاه آدرس، گذرگاه داده و گذرگاه کنترل.

به منظور شناسایی یک وسیله (حافظه یا I/O) توسط CPU ، باید آدرسی به آن تخصیص داد. آدرس اختصاص یافته به یک وسله مورد نظر باید منحصر به فرد باشد؛ یعنی دو وسیله مختلف مجاز به داشتن یک آدرس نیستند. CPU آدرس را روی گذرگاه آدرس قرار می دهد (البته به شکل دودویی) و مدار دیکد وسیله را می یابد. آنگاه CPU از گذرگاه داده برای بدست آوردن داده از وسیله یا ارسال داده به آن استفاده می نماید.

گذرگاههای کنترل برای تهیه سیگنال های خواندن و نوشتن در وسیله و مطلع ساختن آن از تصمیم CPU برای دریافت اطلاعات و یا ارسال اطلاعات به آن است. از سه گـذرگاه فوق، آدرس و داده، توانمندی یک CPU را نشان می دهند.

توضیحی بیشتر درباره گذرگاه داده

چون گذرگاههای داده برای انتقال اطلاعات به و یا از CPU بکار می رونـد، هـر چـه گذرگاههای داده بیشتر باشند، CPU بهتر است. اگر گذرگاههای داده را همچون خطـوط اتوبان تصور کنیم، واضح است که هر چه خطوط بیشتر باشند، مسیر بین CPU و وسایل بیرونی (مانند چاپگرها، ROM, RAM و غیره ، شکل ۱۰-۰ ملاحظه شود) بهتر خواهـد بود. اما افزایش در تعداد خطوط، هزینه ساخت را افزایش می دهد. گـذرگاههای حافظـه بیشتر،بمعنای CPU و کامپیوتر گرانتر می باشد. اندازه متوسط گـذرگاه داده در CPU هـا بین ۸ و ٦٤ متغیر است، کامپیوترهای اولیه ماننـد Apple2 از یـک گـذرگاه داده ۸ بیـت استفاده می کردند، در حالیکه سوپرکامپیوترهایی همچون Cray گذرگاه داده ٦٤ بیتـی را به کار می برند. گذرگاههای داده دو طرفه هستند، زیرا CPU از آنها به هنگام دریافـت و یا ارسال داده استفاده می کند. توان پردازش CPU به اندازه این گذرگاهها وابسته اسـت، زیرا یک گذرگاه ۸ بیتی هر بار قادر است ۱ بایت داده را بفرستد،ولی گذرگاه ١٦ بیتی، ۲ بایت را هر بار ارسال می کند که در نتیجه دو برابر سریعتر خواهد بود.

توضیحی بیشتر درباره گذرگاه آدرس

چون گذرگاه آدرس برای شناسایی وسیله و حافظه متصل به CPU بکار می رود، هر چه گذرگاههای آدرس بیشتر باشند، تعداد وسایلی که آدرس دهی می شوند بیشتر خواهند بود. به بیان دیگر تعداد گذرگاههای آدرس برای یک CPU ، تعداد مکان هایی را که با آن محاوره می کند افزایش می دهد. همواره تعداد مکان ها 2^x است که در آن x تعداد خطوط آدرس می باشد و ربطی به اندازه خطوط داده ندارد. مثلا یک CPU با ۱۶ خط آدرس می تواند ۶۵۵۳۶ ($۲^{۱۶}$) یا ۶۴ k حافظه را آدرس دهی کند. هر مکان حداکثر ۱ بایت داده دارد. به این دلیل است که غالباً تمام ریزپردازنده های همه منظور را بایت آدرس پذیر می نامند. برای مثالی دیگر، کامپیوتر IBM PC AT از یک CPU با ۲۴ خط آدرس و ۱۶ خط داده استفاده می کند. در این حالت،کل حافظه قابل دسترس،۱۶ مگابایت خواهد بود (مگابایت ۱۶=$۲^{۲۴}$) در این مثال $۲^{۲۴}$ مکان وجود دارد، و چون هر مکان یک بایت است،۱۶ مگابایت حافظه موجود خواهد بود. گذرگاه آدرس یک گذرگاه یک طرفه می باشد، و به این معنی است که CPU از گذرگاه آدرس فقط برای ارسال آدرس به خارج از خود استفاده می کند. بطور خلاصه: تعداد کل حافظه های آدرس‌پذیر بوسیله یک CPU همیشه برابر با ۲^x می باشد که در آن x تعداد بیت های آدرس است و ربطی به اندازه گذرگاه داده ندارد.

CPU و ارتباط آن با RAM و ROM

در پردازش اطلاعات بوسیله CPU ، داده باید در RAM یا ROM ذخیره شود. وظیفه ROM در کامپیوترها ارائه اطلاعات ثابت و دائمی است. این اطلاعات عبارتند از :جداول برای الگوی کاراکترهای مورد نمایش روی صفحه مانیتور، یا برنامه هایی که در کامپیوتر نقش اساسی دارند، مانند برنامه هایی که کل RAM موجود در سیستم را می یابند و یا تست می کنند، و یا برنامه هایی که اطلاعات را روی مانیتور نمایش می دهند. بر عکس، RAM برای ذخیره اطلاعاتی بکار می رود که غیر دائمی و قابل تغییر با زمان می باشند، مانند انواع سیستم های عامل و بسته های کاربردی مثل بسته های پردازش کلمات و محاسبه مالیاتی. این برنامه ها را برای پردازش توسط CPU در RAM قرار داده می شوند. CPU اطلاعات مورد پردازش را از RAM یا ROM دریافت می نماید. در صورتیکه آن را در آنجا نباید شروع به جستجو در وسایل ذخیره سازی حجیم مانند دیسک می نماید، و سپس اطلاعات را به RAM منتقل می کند به این دلیل، گاهی RAM و ROM را حافظه اصلی می نامند و دیسک ها نیز حافظه ثانوی خوانده می شوند.

درون CPU ها

برنامه ذخیره شده در حافظه دستورالعمل هائی را برای CPU فراهم می سازد تا بر اساس آن عملی را انجام دهد. عمل می تواند یک جمع داده ساده همچون صورتحساب و یا کنترل یک ماشین مانند روبات باشد. برداشت این دستورات از حافظه و اجرای آنها بعهده CPU است. برای انجام اعمال برداشت و اجرا، تمام CPU ها مجهز به امکانات زیر هستند:

۱- قبل از هر چیز تعدادی ثبات در اختیار CPU قرار دارد. CPU از این ثبات ها برای ذخیره موقت اطلاعات استفاده می کند. اطلاعات می تواند دو مقدار مورد پردازش و یا آدرس مقدار مورد نظری باشد که باید از حافظه برداشت شود. ثبات های درون CPU می توانند ۸ بیت ، ۱٦ بیت، ۳۲ بیت و یا حتی ٦٤ بیت باشند. اندازه آنها به CPU بستگی دارد. بطور کلی هر چه ثبات ها بیشتر و بزرگتر باشند، CPU مناسب تر است. عیب ثبات های بیشتر و بزرگتر، گرانی CPU می باشد.

۲- CPU دارای بخشی بنام ALU (واحد حساب/ منطق) است. بخش ALU در CPU مسئول انجام اعمال حسابی مانند جمع، تفریق، ضرب و تقسیم، و اعمال منطقی مانند AND ، OR و NOT می باشد.

۳- هر CPU دارای یک شمارنده برنامه است. نقش شمارنده برنامه اشارهبه آدرس دستورالعمل بعدی برای اجرا است. با اجرای هر دستورالعمل، شمارنده برنامه افزایش یافته و به آدرس دستورالعمل بعدی برای اجرا اشاره خواهد کرد. در این اشاره، محتوای شمارنده برنامه روی گذرگاه آدرس قرار گرفته و دستورالعمل مورد نظر را یافته و آن را از مبدأ برداشت می کند. در IBM PC شمارنده برنامه را IP یا اشاره گر دستورالعمل می خوانند.

٤- نقش دیکدر دستورالعمل ، تفسیر دستور برداشت شده توسط CPU است. می توان دیکدر دستورالعمل را همانند یک فرهنگ لغت تصور کرد که مفهوم هر دستورالعمل را ذخیره نموده و CPU را در برداشت قدم های بعدی پس از دریافت دستورالعمل هدایت می کند. همانطور که فرهنگ لغت باتعریف هر چه بیشتر لغات نیاز به صفحات بیشتری دارد، CPU هم در درک دستورالعمل های بیشتر نیاز به ترانزیستورهای بیشتری خواهد داشت.

عملیات درونی کامپیوتر

برای نمایش برخی از مفاهیم مورد بحث فوق، تحلیل قدم به قدمی از پردازش یک CPU برای جمع سه عدد در زیر داده شده است. فرض کنید که یک CPU فرضی دارای چهارثبات با نام های D,C,B,A باشد. این پردازشگر دارای گذرگاه داده ۸ بیتی و

گذرگاه آدرس ۱۶ بیتی است. بنابراین CPU می تواند به حافظه هایی از ۰۰۰۰ تا FFFFH دسترسی داشته باشد (جمعاً ۱۰۰۰۰ H مکان) . عملی که CPU می خواهد انجام دهد عبارتست از قراردادن مقدار ۲۱ در ثبات A و سپس جمع ثبات A با مقادیر ۴۲H و ۱۲H فرض کنید که کد انتقال مقدار به ثبات A برابر (BOH) ۱۰۱۱۱۰۰۰ و کد جمع یک مقدار به ثبات A نیز (04H) ۰۱۰۰ ۰۰۰۰ باشد. مراحل لازم و کد اجرای آنها برابر زیر است:

اگر برنامه اجرایی فوق در مکان هایی از حافظه قرار گیرد که از ۱۴۰۰ H شروع می شود. محتوای هر مکان حافظه بقرارزیراست:

عملیاتی که CPU برای اجرای برنامه فوق طی می کند بقرار زیر است:

۱- شمارنده برنامه CPU می تواند مقداری بین ۰۰۰۰ و FFFFH داشته باشد. باید ۱۴۰۰ را در شمارنده برنامه نشاند تا آدرس اولین دستورالعمل برای اجرا مشخص گردد. پس از بارکردن شمارنده برنامه با آدرس اولین دستورالعمل، CPU آماده اجرا است.

۲- CPU ، ۱۴۰۰ H را روی گذرگاه آدرس قرار داده و آن را به خارج ارسال می دارد. مدار حافظه مکان را می یابد و در این هنگام CPU نیز سیگنال READ را فعال می نماید و به این ترتیب بایت مکان ۱۴۰۰H را از حافظه درخواست می کند. این موجب

می شود تا محتوای حافظه در مکان H ۱٤۰۰ ، که B0 است، روی گذرگاه قرار گیرد و به CPU انتقال یابد.

۳- CPU دستورالعمل B0 را به کمک مدار دیکد دستورالعمل، دیکد می کند. پس از یافتن تعریف دستورالعمل متوجه می شود که باید محتوی مکان حافظه بعدی را به ثبات A در داخل CPU بیاورد. بنابراین به مدار کنترل خود فرمان اجرای دقیق آن را صادر می نماید. وقتی که مقدار H ۲۱ را از مکان 1401 حافظه به درون آورد، دریچه های ورودی همه ثبات ها را بجز ثبات A ، می بندد. بنابراین مقدار H ۲۱ وقتی وارد CPU شود مستقیماً وارد ثبات A می گردد. پس از تکمیل یک دستورالعمل ، شمارنده برنامه به آدرس دستورالعمل بعدی برای اجرااشاره می کند، که در این حالت ۱٤۰۲ است. سپس آدرس H ۱٤۰۲ به روی گذرگاه آدرس ارسال می شود تا دستورالعمل بعدی برداشت شود.

٤- آنگاه از مکان H ۱٤۰۲ ، کد ۰٤ را بر می دارد. پس از دیکد کردن، CPU می فهمد که باید محتوای ثبات A را با بایتی که در آدرس بعدی قرار دارد (۱٤۰۳) جمع کند. پس از آوردن مقدار (در این حالت ٤۲H) به درون CPU ، مقدار درون ثبات A را همراه با این مقدار به ALU برای انجام جمع تحویل می دهد. سپس نتیجه جمع را از

خروجی ALU دریافت کرده و در ثبات A قرار می دهد. در این هنگام شمارنده برنامه برابر با ۱٤۰٤ H ، یعنی آدرس دستورالعمل بعدی می گردد.

۵- آدرس ۱٤۰٤ H روی گذرگاه آدرس قرار می گیرد و کد درون آن آدرس به داخل CPU آورده شده و سپس دیکد و اجرا می گردد. مجدداً این کد مقداری را به ثبات A می افزاید. شمارنده برنامه به ۱٤۰٦ H اصلاح می شود.

٦- نهایتاً، محتوای آدرس ۱٤۰٦ برداشت و اجرا می گردد. این دستورالعمل، HALT ، به CPU می فهماند تا افزایش شمارنده برنامه را متوقف نماید. در غیاب HALT ، CPU به اصلاح شمارنده برنامه ادامه داده و دستورالعمل ها را برداشت می نماید.

اکنون فرض کنید که آدرس ۱٤۰۳H به جای ٤۲ H ، حاوی ۰٤ باشد. CPU چگونه داده ۰٤ را برای جمع از کد ۰٤ تفکیک می کند؟ بخاطر آورید که برای این CPU ، کد ۰٤ به معنی انتقال یک مقدار به داخل ثبات A است، بنابراین CPU سعی بر دیکد مقدار بعدی نخواهد کرد، بلکه محتوای مکان حافظه بعدی را بدون توجه به مقدار آن بداخل ثبات A منتقل می سازد.

این فصل با بحثی در موردنقش و اهمیت میکروکنترلرها در زندگی روزمره آغاز می شود. در بخش ۱-۱ روال انتخاب یک میکروکنترلر، همراه با استفاده از آنها را مورد بحث قرار می دهیم. بخش ۲-۱ انواع اعضای خانواده ۸۰۵۱ ، همچون ۸۰۵۲ ، ۸۰۳۱ و

ویژگی های آنها را پوشش می دهد. بعلاوه انواع مختلف ۸۰۵۱ مانند ۸۷۵۱ ، AT ۸۹C۵۱ و DS۵۰۰۰ را مورد بحث قرار خواهیم داد.

میکروکنترلرها و پردازنده های درونی

در این بخش نیاز به میکروکنترلرها و مقایسه آنها با میکروپروسسورهای همه منظوره ای چون پنتیوم و دیگر میکروپروسسورها بحث شده است. ما به نقش میکروکنترلر در بازار نیز نگاه خواهیم کرد. بعلاوه، روالی را برای انتخاب یک میکروکنترلر نیز ارائه خواهیم داد.

میکروکنترلرها در برابر میکروپروسسورهای همه منظوره

تفاوت بین یک میکروپروسسور و یک میکروکنترلر چیست؟ منظور از یک میکروپروسسور (ریزپردازنده)، میکروپروسسورهایی از خانواده x۸۶ اینتل مثل ۸۰۸۶،۸۰۲۸۶ ، ۸۰۳۸۶ ، ۶۸۰۲۰ ، ۶۸۰۳۰ ، ۶۸۰۴۰ و یا خانواده هایی از این قبیل است. این میکروپروسسورها فاقد RAM ، ROM و پورت های I/O در درون خود تراشه هستند. با این دلیل به آنها میکروپروسسورهای همه منظور می گویند.

طراح سیستمی که از میکروپروسسور همه منظوره ای چون پنتیوم ، ۶۸۰۴۰ استفاده می کند باید در خارج از آن ROM, RAM ، پورت های I/O و تایمرها را اضافه نماید تا سیستمی قابل کار ساخته شود. گر چه افزایش ROM, RAM و پورت های I/O

موجب حجیم شدن و گرانتر شدن سیستم ها می گـردد، ولـی بـه قابلیـت انعطـاف آنهـا افزوده می شود. از جمله اینکه طراح می تواند روی مقـدار ROM, RAM پـورت هـای I/O بـر حسـب نـوع کاربردتصـمیم گیـری و اعمـال نظـر نمایـد. ایـن توانمنـدی در میکروکنترلرها امکان پذیر نیست. یک میکروکنترلر دارای یک CPU به همراه مقدار ثابتی از ROM, RAM ، پورت های I/O و تایمر در درون خـود مـی باشـد. بـه بیـان دیگـر، پروسسور، ROM, RAM پورت های I/O و تایمر همگی در یک تراشه جای داده شـده اند؛ بنابراین طراح نمی تواند یک حافظه، I/O یـا تـایمری را بـدون گسـترش لازم آن از بیرون اضافه کند. مقـدار ثابـت ROM ، RAM و مقـدار پـورت هـای تثبیـت شـده در میکروکنترلرها ، آنها را برای کاربردهایی که قیمت و محفظه در آنها بحرانی است ایده آل کرده است.

در بسیاری از کاربردها مثل کنترل از راه دور TV ،احتساب توان میکروپروسسورهایی چون ٤٨٦ یا حتی ٨٠٨٦ هم لزومی نـدارد. در بعضـی کاربردهـا، فضـای اشـغالی، تـوان مصرفی و قیمت هر واحد اهمیت بیشتری نسبت به توان محاسباتی دارد. ایـن کاربردهـا اغلب نیاز به بعضی عملیات I/O برای خواندن سیگنال هـا و روشـن و خـاموش کـردن بیت های معینی دارند. باین دلیل بعضی ، آنها را پروسسورهای "itty-bitty" می خوانند. مقالـه "Good things in small Packages Are Generating Big Product

Opportunities" که بوسیله Rick Grehan در مجله Byte شماره www. Byte 1994; com

september نوشته شده و در آن بحث جالبی از میکروکنترلرها ارائه شده مطالعه نمایید.

میکروکنترلرها و سیستم های تک منظوره

در مقالاتی که میکروپروسسورها مطرح می شوند، اغلب عبارت سیستم تک منظوره را ملاحظه می کنیم. میکروپروسسورها و میکروکنترلرها بطور گسترده ای در تولید سیستم های تک منظوره بکار می روند. یک محصول تک منظوره از یک میکروپروسسور (یا میکروکنترلر) برای انجام فقط و فقط یک کار استفاده می کند. یک چاپگر نمونه ای از یک سیستم تک منظوره است زیر پروسسور داخل آن فقط یک کار را انجام می دهد و آن این است که داده را بدست آورده و آن را چاپ کند. این کار را با یک PC مبتنی بر پنتیوم (مانند هر PC سازگار با IBM ٨٦ x) مقایسه نمایید. PC می تواند برای هر کاربردی مانند پردازشگرهای کلمات، مراکز چاپ، پایانه، لیست های بانک، بازی های ویدیوئی ، سرویس دهندهٔ شبکه و پایانه اینترنت مورد استفاده قرار گیرد، برای انواع کاربردها می توان به راحتی برنامه را در pC بار کرده وآن را اجرا کرد. البته دلیل قابلیت اجرای کارهای متنوع در PC این است که دارای حافظه RAM و سیستم عاملی است که

نرم افزار کاربردی را در RAM بار کرده و اجازه اجرای آن را به PC می دهد. در یک سیستم تک منظوره ، تنها یک نرم افزار کاربردی وجود دارد و معمولاً در ROM سوزانده می شود. یک PC x ۸٦ ممکن است به وسایل تک منظوره ای مانند صفحه کلید، چاپگر، مودم ، کنترل گر دیسک، کارت صدا، راه انداز CD-ROM ، ماوس و غیره متصل باشد. هر یک از این وسایل جانبی در داخل خود دارای یک میکروکنترلر برای انجام کار خاص می باشند. مثلاً در داخل هر ماوس یک میکروکنترلر وجود دارد که وظیفه اش یافتن مکان ماوس و ارسال آن به PC است. جدول ۱-۱ بعضی از محصولات تک منظوره را نشان می دهد.

۸۰۵۱ دارای دو شمارنده / تایمر است. آنها می توانند به عنوان تایمر (زمان سنج) برای تولید یک تأخیر زمانی یاشمارنده برای شمارش وقایع رخداده در خارج میکروکنترلر بکار روند. در این فصل نشان می دهیم که آنها چگونه برنامه ریزی شده و مورداستفاده قرار می گیرند. در بخش ۱-۹ چگونگی استفاده از تایمرها برای تولید تأخیرهای زمانی نشان داده شده است. در بخش ۲-۹ نشان می دهیم که چگونه در شمارش وقایع (یا پدیده های فیزیکی) مورد استفاده قرار می گیرند.

برنامه نویسی تایمرهای ۸۰۵۱

۸۰۵۱ دارای دو تایمر است: تایمر ۰ و تایمر ۱ این تایمرها می توانند به عنوان تایمر یا پدیده شمار بکار روند. در این بخش ما ابتدا ثبات های تایمر را بحث کرده و سپس چگونگی برنامه نویسی تایمرها را برای تولید تأخیرهای زمانی نشان خواهیم داد.

ثبات های اساسی تایمر

هر دو تایمر ۰ و ۱ شانزده بیت عرض دارند. چون ۸۰۵۱ ساختار ۸ بیتی دارد، هر تایمر شانزده بیتی با دو ثبات مختلف بایت پایین و بایت بالا دستیابی می شود. هر تایمر بطور جداگانه بحث شده است.

ثبات های تایمر ۰

ثبات ۱۶ بیتی تایمر ۰ بصورت بایت بالا و بایت پایین دستیابی می شوند. ثبات بایت پایین را TL0 (بایت پایین تایمر ۰) و ثبات بایت بالا را TH0 (بایت بالا تایمر ۰) می خوانند. این ثبات ها مثل هر ثبات دیگری مانند R2,R1,R0,B,A و غیره قابل دسترسی هستند. مثلا دستور MOV TL0,#4FH ، مقدار 4FH را به TL0 منتقل می سازد، که بایت پایین تایمر 0 است. این ثبات ها مانند دیگر ثبات ها قابل خواندن هم هستند. مثلاً "MOV R5, TH0" , TH0 را در R5 ذخیره می کند.

ثبات های تامیر ۱

تایمر ۱ هم ۱۶ بیتی است و ۱۶ بیت آن به صورت دو بایت به دو نیم شده است. که هر یک را TL1 (بایت پایین تایمر ۱) و TH1 (بایت بالای تایمر ۱) می نامند. این ثبات ها به روشی مشابه با ثبات های تایمر ۰ قابل دستیابی اند.

ثبات TMOD (مد تایمر)

هر دو تایمر ۰ و ۱ از یک ثبات به نام TMOD برای تنظیم انواع مدهای عملیاتی تایمر استفاده می کنند. TMOD یک ثبات ۸ بیت است که در آن ٤ بیت پایین تر برای تایمر ۰ و ٤ بیت بالاتر برای تایمر ۱ کنار نهاده شده است. در هر حال، دو بیت پایین تر برای تنظیم تایمر و دو بیت بالاتر برای مشخص کردن عملیات بکار می روند. این انتخاب ها در زیر بحث شده اند.

M1, M0

M1, M0 برای انتخاب مد تایمر است. همانطور که در شکل ۳-۹ دیده می شود،سه مد ۰ و ۱ و ۲ وجود دارد. مد ۰ یک تایمر ۱۳ بیت، مد ۱ یک تایمر ۱۶ بیت و مد ۲ یک تایمر ۸ بیت است. ما عمدتاً بر مدهای ۱ و ۲ تأکید خواهیم کرد زیرا بیشتر از دیگری مورد استفاده اند. بزودی مشخصه این مدها را بیان خواهیم کرد. این کار بعد از توصیف بقیه ثبات TMOD خواهد بود.

C/T (تایمر / ساعت)

این بیت در ثبات TMOD برای تصمیم گیری انتخاب تایمر به عنوان مولد تأخیر و یا شمارنده وقایع بکار می رود. اگر C/T = ۰ باشد از آن به عنوان تایمر برای تولید زمان تأخیر استفاده می شود. منبع ساعت برای تأخیر فرکانس کریستال ۸۰۵۱ است. این بخش با این انتخاب به بحث ادامه خواهد داد. استفاده از تایمر به عنوان پدیده شمار در بخش بعد بحث شده است.

تولید زمان تأخیر طولانی

همانطور که در مثال های فوق تا کنون دیدیم، سایز تأخیر زمانی به دو فاکتور وابسته است (الف) فرکانس کریستال، و (ب) به ثبات ۱۶ بیتی تایمر در مد ۱ ، هر دو فاکتور فوق خارج از کنترل ۸۰۵۱ بوسیله برناهم نویس است. قبلاً دیدم که بزرگترین زمان قابل دسترسی با صفر کردن TH و TL بدست می آید. چرا این مقدار کافی نیست؟مثال ۱۳-۹ چگونگی دستیابی به تأخیرهای زمانی بزرگ را نشان می دهد.

با استفاده از ماشین حساب ویندوز، TL و TH را پیدا کنید.

ماشین حساب علمی در ویندوز میکروسافت یک ابزار ساده ای برای یافتن مقادیر TL و TH است. فرض کنید می خواهیم مقادیر TH و TL را برای تأخیر زمانی ۳۵۰۰۰ پالس ۱/۰۸۵ µs پیدا کنیم. در زیر مراحل محاسبه آمده است:

۱- ماشین حساب علمی را در MS Windows بالا آورده و دهدهی را انتخاب کنید.

۲- ۳۵۰۰۰ را وارد کنید.

۳- مبنای ۱۶ را انتخاب نمایید. این موجب تبدیل ۳۵۰۰۰ به B8H ۸۸ می شود.

٤- - / + را برای ۳۵۰۰۰- دهدهی (7748H) اختیار کنید.

۵- دو رقم کم ارزشتر (٤۸) از مقدار فوق به TL و (۷۷) برای TH است. ما همه F های سمت را صرفنظر می کنیم زیرا داده ۱۶ بیت است.

مد ۰

مد ۰ دقیقاً مثل مد ۱ است. با این تفاوت که بجای ۱۶ بیت، یک تایمر ۱۳ بیت است. شمارنده ۱۳ بیتی می تواند از ۰۰۰۰ تا ۱FFF را در TH-TL نگه دارد. بنابراین وقتی تایمر به ۱FFFH می رسد به ۰۰۰۰ بازگشته TF را بالا می برد.

برنامه نویسی مد ۲

در زیر مشخصات و عملکرد مد ۲ تشریح شده است.

۱- این یک تایمر ۸ بیت است، بنابراین فقط اعداد ۰۰ تا FFH را برای بارشدن TH از ثبات تایمر اجازه می دهد.

۲- پس از بارشدن TH با مقدار ۸ بیتی،۸۰۵۱ یک کپی از آن را به TL می دهد. سپس تایمر باید آغاز به کار کند. این کار بوسیله دستور "SETB TRO" برای تایمر ۰ و "SETB TRI" برای تایمر ۱ انجام می شود. این قسمت دقیقاً مثل مد ۱ می باشد.

۳- پس از شروع تایمر، با افزایش ثبات TL شروع به شمارش می کند. شمارش تا رسیدن به FFH ادامه دارد. وقتی که از FFH به ۰۰ باز گردد، پرچم TF را بالا می برد. اگر از تایمر ۰ استفاده شود، TF0 بالا می رود و اگر تایمر ۱ مورد استفاده قرار گیرد، TF1 بالا خواهد رفت.

٤- وقتی که ثبات TL از FFH به ۰ می رود TF برابر ۱ می شود و TL با مقدار اولیه بار شده در TH بطور خودکار مجدداً بار می گردد. برای تکرار فرآیند باید TF را فقط پاک کرده اجازه بدهیم بدون هر نیازی به بار شدن مقدار اولیه بوسیله برنامه نویس، کار ادامه یابد. این قابلیت، بر خلاف مد ۱ که در آن برنامه نویس باید بار شدن مجدد TH و TL را انجام دهد مد ۲ را بیک امکان خودکار با بار شدن اتوماتیک تبدیل کرده است.

باید تأکید کرد که مد ۲ یک تایمر ۸ بیت است. با این وجود، قابلیت بار شدن خودکار را دارد. در بار شدن خودکار،TH با مقدار اولیه بار می شود و سپس یک کپی از آن به TL داده می شود. بار کردن TH را تغییر نمی دهد، بلکه همچنان مقدار اولیه را حفظ م

یکند. این مد کاربردهای متعددی دارد از جمله آنها تنظیم میزان باود است کـه در فصـل ۱۰ خواهیم دید.

مراحل برنامه نویسی در مد ۲

برای تولید یک تأخیر زمانی با مد ۲ تایمر، مراحل زیر را اجرا کنید.

۱- مقدار ثبات TOMD را بار کنید تا مشخص شود کدام تایمر بکار می رود، و کدام مد تایمر انتخاب شده است.

۲- ثبات TH را با مقدار اولیه شمارش بار کنید.

۳- تایمر را شروع کنید.

٤- پرچم تایمر (TF) را با دستور "JNB TFx, target" ردیابی کنید تا ببینید آیـا بـالا رفته است. اگر TF بالا بود از حلقه خارج شوید.

۵- پرچم TF را پاک کنید.

٦- به مرحله ٤ بازگردید چون مد ۲ بطور خودکار بار می شود.

برنامه ریزی شمارنده

در آخرین بخش از تایمر/ شمارنده ۸۰۵۱ برای تولید تأخیرهای زمانی استفاده کردیم. این تایمرها برای شمارش پدیده های رخداده در خارج ۸۰۵۱ هـم اسـتفاده مـی شـوند. استفاده از تایمر/ شمارنده به عنوان یک پدیده شمار، هر چه در بخش قبل راجع به تایمر

گفته شد، در برنامه ریزی آن به عنوان شمارنده، بجز منبع فرکانس،صادق است. وقتی که تایمر / شمارنده به عنوان تایمر (زمان سنج) بکار رود، کریستال ۸۰۵۱ به عنوان منبع فرکانس بکار می رود. با این وجود وقتی که به عنوان شمارنده مورد استفاده قرار گیرد، پالس بیرون ۸۰۵۱ ثبات های TH و TL را افزایش می دهد. در مد شمارنده، ثبات های TH و TL و TMOD همان هایی هستند که در بخش قبل بحث شده؛ آنها حتی می توانند نام یکسانی داشته باشند. مدهای تایمر هم یکسانند.

بیت C/T در ثبات TMOD

در بخش قبل دیدیم که بیت C/T در ثبات TMOD در مورد منبع ساعت برای تایمر تصمیم می گیرد. اگر C/T=0 باشد، تایمر پالس ها را از کریستال بدست می آورد. در مقابل، وقتی C/T=1 است، تایمر به عنوان شمارنده بکار رفته و پالس ها را از بیرون ۸۰۵۱ بدست می آورد. بنابراین در C/T=1 شمارنده با ورود پالس ها به پایه های ١٤ و ۱۵ رو به بالا می شمارد. این پایه ها T0 (ورودی تایمر ۰) و T1 (ورودی تایمر ۱) نامیده می شوند. توجه کنید که این دو پایه متعلق به پورت ۳ هستند. در تایمر ۰ ، وقتی C/T=1 است، پایه P3.4 پالس ساعت را فراهم می نماید و بازاء هر پالس وارده از پایه، شمارنده رو به بالا می شمارد. بطور مشابه، برای تایمر ۱ ، وقتی C/T=1 باشد هر پالس ساعت وارده از پایه P3.5 موجب بالا رفتن شمارنده می گردد.

ثبات TCON

در مثال هایی که تا کنون ارائه شدند کاربرد پرچم های TRO و TR1 در روشن و خاموش کردن تایمرها را ملاحظه کردیم. این بیت ها بخشی از ثباتی ۸ بیتی به نام TCON (کنترل تایمر) است. همانطور که در جدول ۲-۹ ملاحظه شد چهاربیت بالا برای ذخیره بیت های TF و TR در دو تایمر ۰ و ۱ بکار رفتند. ٤ بیت پایین تر برای کنترل بیت های وقفه اختصاص یافته است که در فصل ۱۱ در مورد آنها صحبت خواهیم کرد. متذکر می شویم که ثبات TCON یک ثبات آدرس پذیر بیتی است. در عوض بکارگیری دستوراتی چون "STEB TCON" "CLR TR1 , TR1" می توانیم بترتیب از "SETB TCON.6" استفاده کنیم. جدول ۲-۹ جایگزینی بعضی از دستوراتی را که تاکنون دیده ایم نشان می دهد.

وقفه های ۸۰۵۱

در این بخش ابتدا تفاوت بین سرکشی و وقفه ها بررسی گردیده و سپس انواع وقفه های ۸۰۵۱ توصیف شده اند.

وقفه ها در برابر سرکشی

یک میکروکنترلر می تواند چندین دستگاه را سرویس بدهد. برای انجام آن دو راه وجود دارد: روش وقفه و سرکشی. در روش وقفه ، هر وقت وسیله ای به سرویس نیاز

داشته باشد، با ارسال یک سیگنال وقفه میکروکنترلر را مطلع می سازد. میکروکنترلر پس از دریافت سیگنال وقفه،هر کاری را کنار گذاشته و به وسیله سرویس می دهد. برنامه متعلق به وقفه ، روال سرویس وقفه (ISR)، یا اداره کننده وقفه نامیده می شود. در روش سرکشی، میکروکنترلر مرتباً وضعیت وسیله مورد نظر را ردیابی می کند. وقتی که شرایط ایجاب می کند،سرویس را انجام می دهد. پس از آن به سراغ وسیله دیگر می رود، تا اینکه همه را سرویس دهد. گرچه سرکشی قادر است وضعیت چندین دستگاه را ردیابی کرده و تحت شرایطی آنها را سرویس دهد،ولی از نظر کاربرد میکروکنترلر، یک روش کارآیی نیست. مزیت وقفه این است که میکروکنترلر می تواند چندین وسیله را سرویس دهد (البته نه بطور همزمان) هر وسیله قادر است توجه میکروکنترلر را بر حسب اولویت منتسب به خود جلب کند.در روش سرکشی امکان تخصیص اولویت وجود ندارد زیرا همه وسایل را بر حسب روش تخصیص دوره ای (rond-robin fasion) چک کند. مهمتر اینکه،در روش وقفه میکروکنترلر قادر است تا یک وسیله و تقاضای آن را برای سرویس نادیده بگیرد. مجدداً این کار در سرکشی میسر نیست. مهمترین دلیل در ارجحیت وقفه این است که روش سرکشی بخش قابل توجهی از زمان میکروکنترلر را با سرکشی وسایل و دستگاههای که نیاز به سرویس ندارند تلف می کند. به بیان دیگر برای خودداری از کند شدن عملکرد میکروکنترلر، وقفه ها بکار می روند. مثلا در بحث

تایمرها که در فصل ۹ انجام شد ما از دستور "JNB TF, target" استفاده کردیم، و آنقدر صبر کردیم تا وقت تایمر به سر برسد. در آن مثال ضمن انتظار قادر به انجام هیچ کار دیگری نبودیم. این همان وقت تلف شده ای است که می توانستیم طی آن کارهای مفید بسیاری انجام دهیم. در وضعیت تایمر،اگر از روش وقفه استفاده کنیم، میکروکنترلر می تواند مشغول به کار دیگری باشد و وقتی پرچم TF بالا برود به میکروکنترلر در هر صورت وقفه بدهد.

روال سرویس وقفه

بازای هر وقفه، باید روال سرویس وقفه (ISR) یا اداره کننده وقفه ای وجود داشته باشد. وقتی که وقفه ای رخ می دهد، میکروکنترلر روال سرویس وقفه را اجرا می کند. بازاء هر وقفه باید مکان ثابتی در حافظه تعریف شود تا آدرس ISR را نگهدارد. مجموعه حافظه های کنار گذاشته شده برای نگهداری آدرس های ISR ، جدول بردار وقفه نام دارد.

مراحل اجرای یک وقفه

پس از فعال شدن وقفه، میکروکنترلر وارد مراحل زیر می شود.

۱- اجرای دستور جاری را پایان می دهد و آدرس دستور بعدی (PC) را در پشته ذخیره می کند.

۲- وضعیت جاری همه وقفه های درونی را نیز ذخیره می نماید (نه در پشته)

۳- به مکان معینی از حافظه به نام جدول بردار وقفه که آدرس روال سرویس وقفه را نگه می دارد، پرش می کند.

٤- میکروکنترلر آدرس ISR را از جدول بردار وقفه بدست آورده و به آن پرش می نماید. آنگاه شروع به اجرای زیر روال سرویس وقفه می کند تا به آخرین دستور که RET1 است برسد.

۵- پس از اجرای دستور RET1 ، میکروکنترلر به مکانی که در آن وقفه را دریافت کرده بود، باز می گردد ابتدا، آدرس شمارنده برنامه را از پشته با برداشت از دو بایت بالای پشته بازیافت می کند و به PC می فرستد. سپس شروع به اجرای برنامه از آن آدرس می نماید.

به حساسیت نقش پشته در مرحله ۵ توجه کنید. باین دلیل باید در دستکاری محتوای پشته در ISR دقت شود، خصوصاً در ISR همچون فراخوانی زیر روال (CALL) ، تعداد درج ها (PUSH) و بازیافت ها (POP) باید مساوی باشند.

شش وقفه در ۸۰۵۱

در واقع پنج وقفه برای کاربران ۸۰۵۱ وجود دارد، ولی بسیاری از برگه های اطلاعات بوجود شش وقفه از جمله بازنشانی (ری ست) اشاره می کنند. شش وقفه موجود در ۸۰۵۱ بشرح زیر اختصاص یافته اند.

۱- بازنشانی وقتی که پایه بازنشانی فعال شود، ۸۰۵۱ به آدرس ۰۰۰۰ پرش می کند. این همان بازنشانی مورد بحث در فصل ٤ است.

۲- دو وقفه برای تایمرها کنار گذاشته شده است: یکی برای تایمر ۰ و دیگری برای تایمر ۱ مکان های حافظه 000BH و 001BH در جدول بردار وقفه، به ترتیب متعلق به تایمر ۰ و ۱ هستند.

۳- دو وقفه برای وقفه های سخت افزاری بیرونی کنار گذاشته شده اند. پایه های شماره 12(P3.2) و 13 (P3.3) در پورت ۳ به ترتیب برای وقفه های سخت افزاری INT1,INT0 می باشند. این وقفه ها ، EX2,EX1 هم خوانده می شوند. مکان های 0003H و 0013H در جدول بردار وقفه به ترتیب INT1,INT0 تخصیص یافته اند.

٤- تبادل داده سریال دارای وقفه ای است که متعلق به ارسال و دریافت می باشد. مکان 0023H از جدول بردار وقفه به این وقفه متعلق است.

توجه کنید که در جدول ۱-۱۱ تعداد محدودی بایت برای هر وقفه کنار نهاده شده است. مثلاً برای INT0 یعنی وقفه ۰ سخت افزار بیرونی ، کلاً ۸ بایت از مکان ۰۰۰۳ تا ۰۰۰A در نظر گرفته شده است. بطور مشابه ۸ بایت از ۰۰۰ BH تا ۰۰۱۲H برای TF0 ، وقفه تایمر ۰ کنار گذاشته شده است. اگر روال سرویس برای وقفه معینی آنقدر کوتاه است تا در فضای اختصاصی جای گیرد، آن را در جدول بردار قرار دهید تا به آدرس ISR اشاره کند.

در این حال، بقیه بایت های متعلق به وقفه بلا استفاده می مانند. در سه بخش بعدی، مثال های متعددی از برنامه نویسی وقفه که این مفاهیم را روشن کنند، خواهیم دید.

از جدول ۱-۱۱ دیده می شود که تنها سه بایت از فضای ROM به پایه بازنشانی اختصاص یافته است. این مکان ها عبارتند از ۰ و ۱ و ۲ آدرس مکان ۳ متعلق است به وقفه ۰ سخت افزاری بیرونی . باین دلیل در برنامه به عنوان اولین دستور، LJMP را قرار می دهیم و پردازنده را از جدول بردار وقفه طبق شکل ۱-۱۱ دور می کنیم. در بخش بعد، خواهیم دید که چگونه این مطالب در قالب مثال هایی پیاده می شوند.

فعال سازی و غیر فعال سازی وقفه

پس از بازنشانی، همه وقفه ها غیر فعال می شوند، باین معنی که اگر هر کدام فعال شوند هیچیک بوسیله میکروکنترلر پاسخ داده نمی شوند. برای اینکه میکروکنترلر به آنها پاسخ دهد، باید وقفه ها را با نرم افزار فعال کرد. ثباتی به نام فعال ساز وقفه ، IE ، مسئول این تواناسازی و ناتوان کردن وقفه هاست. شکل ۲-۱۱ ثبات IE را نشان می دهد. توجه کنید که IE یک ثبات آدرس پذیر بیتی است.

در شکل ۲-۱۱ توجه کنید که بیت D7 در ثبات EA,IE (تمام فعال ساز) خوانده شده است. برای فعال شدن بقیه ثبات، این بیت باید ۱ شود. D6 استفاده نشده است. D۵ بوسیله ۸۰۵۲ بکار می رود. بیت D٤ برای وقفه سریال است و به همین ترتیب.

مراحل فعال سازی یک وقفه

برای فعال کردن یک وقفه، مراحل زیر اجرا می گردد.

۱- بیت D۷ از ثبات IE (EA) باید به سطح بالا برده شود تا بقیه ثبات فعال گردد.

۲- اگر EA=1 باشد، وقفه ها فعال شده و هنگامی که بیت های مربوطه به هر وقفه در IE فعال گردد به آن وقفه پاسخ داده خواهد شد. اگر EA=0 باشد، به هیچ وقفه ای پاسخ داده نمی شود حتی اگر بیت های مربوطه در IE در سطح بالا باشند. برای درک این نکته مهم به مثال ۱-۱۱ توجه فرمایید.

برنامه‌نویسی وقفه‌های تایمر

در فصل ۹ چگونگی استفاده از تایمر ۰ و تایمر ۱ را با روش سرکشی بحث کردیم. در این بخش از وقفه‌ها برای برنامه‌ریزی تایمرهای ۸۰۵۱ استفاده می‌کنیم.

غلتیدن پرچم تایمر و وقفه

در فصل ۹ بیان شد که، بهنگام رفتن تایمر از بالاترین شمارش ممکن به اولین شمارش، پرچم تایمر TF بالا می‌رود. در آن فصل، دیدیم که چگونه TF را با دستور JNB TF، TARGET ردیابی کنیم. در سرکشی به TF مجبوریم آنقدر صبر کنیم تا TF بالا برود. مشکل موجود در این روش این است که میکروکنترلر مادامی که منتظر بالا رفتن TF است گره خورده و کار دیگری نمی‌تواند انجام دهد. استفاده از وقفه این مشکل را حل می‌کند و از گره خوردن پیشروی آن جلوگیری می‌نماید. اگر وقفه تایمر در ثبات IE فعال شود، هر وقت تایمر از بالاترین مقدار به پایین‌تر مقدار بغلط (به ۰ برسد)، TF بالا می‌رود، و در نتیجه میکروکنترلر از هر کاری که در حال انجام آن باشد دست کشیده و به جدول بردار وقفه برای سرویس‌دهی ISR پرش می‌کند. به این ترتیب. میکروکنترلر می‌تواند تا زمانی که تایمر به ۰ برسد مشغول کار دیگری باشد.

به نکات زیر در برنامه مثال ۲-۱۱ توجه کنید.

۱- باید از بکارگیری فضای حافظه متعلق به جدول بردار وقفه خودداری کنیم. بنابراین همه کدهای مقدار دهی اولیه را از حافظه 30H به بعد قرار می‌دهیم. دستور LJMP اولین دستوری است که 8051 به هنگام روشن شدن اجرا می‌کند. LJMP همیشه کنترل‌گر را از جدول بردار وقفه دور می‌کند.

۲- ISR برای تایمر ۰ در مکان 000BH قرار دارد و به اندازه کافی برای جای دادن فضای آن متعلق به این وقفه، کوچک است.

۳- ما وقفه تایمر ۰ را با "MOV IE,# 10000010B" در برنامه اصلی "MAIN" فعال می‌کنیم.

٤- مادامی که داده P0 بداخل آورده شده و مرتباً به P1 صادر می‌شود، هنگام صفر شدن تایمر (تغییر از پایه ۰)، پرچم TF0 بالا می‌رود و میکروکنترلر از حلقه "BACK" خارج شده به 0000BH برای اجرا ISR مربوط به تایمر ۰ خواهد رفت.

۵- توجه کنید که در ISR تایمر ۰ به دستور "CLR TF0" قبل از دستور RET1 نیازی نیست. دلیل این است که 8051 پرچم TF را ضمن پوش به جدول بردار وقفه پاک می‌کند.

۸۰۵۱ دارای دو وقفه سخت‌افزاری بیرونی است. پایه 13(P3.2) از ۸۰۵۱ که برای NOT0 و NIT1 در نظر گرفته شده‌اند، برای وقفه‌های سخت‌افزاری بکار رفته‌اند. با فعال شدن این پایه‌ها، ۸۰۵۱ تحت هر وضعیتی وقفه یافته و به جدول بردار وقفه برای اجرای روال سرویس وقفه پرش می‌کند. در این بخش این دو وقفه سخت‌افزاری بیرونی برای ۸۰۵۱ را با مثال‌هایی مطالعه خواهیم کرد.

وقفه‌های بیرونی INT0 و INT1

در ۸۰۵۱ تنها دو وقفه سخت‌افزاری وجود دارد: NIT0 و INT1. این وقفه‌ها بترتیب روی پایه‌های P3.2 و P3.3 از پورت ۳ قرار دارند. مکان‌های جدول بردار وقفه برای INT0 و INT1 بترتیب عبارتند از 0003H و 0013H. همانطور که در بخش ۱-۱۱ بیان شد آنها بوسیله ثبات IE فعال یا غیرفعال می‌شوند.

این فعال شدن به چه صورت است؟ برای وقفه‌های سخت‌افزاری بیرونی دو سطح فعال شدن وجود دارد: (۱) حساس به سطح، و (۲) حساس به لبه. اجازه بدهید تا به هر یک نگاه کنیم. ابتدا ببینیم چگونه وقفه حساس به سطح کار می‌کند.

وقفه حساس به سطح

در مد حساس به سطح، معمولاً پایه‌های INT0 و INT1 در سطح بالا هستند (درست مثل پایه‌ای پورت I/O) و اگر یک سیگنال سطح پایین به آنها اعمال شود وقفه را فعال خواهد کرد. سپس میکروکنترلر هر کاری را که به آن مشغول باشد رها کرده و به جدول بردار وقفه برای سرویس وقفه پرش می‌نماید. به این نوع وقفه، وقفه حساس به سطح، یا فعال شده با سطح می‌گویند که مد پیش فرض ۸۰۵۱ بعد از بازنشانی است. سیگنال سطح پایین در پایه INT قبل از اجرای آخرین دستور در روال سرویس وقفه، یعنی RETI، باید حذف شود؛ در غیر اینصورت وقفه دیگری رخ خواهد داد. به بیان دیگر، اگر سیگنال وقفه سطح پایین قبل از اتمام ISR حذف نشود، ۸۰۵۱ وقفه دیگری را تصور کرده و برای اجرای مجدد، به جدول بردار وقفه، پرش خواهد کرد. به مثال ۵-۱۱ توجه کنید.

در این برنامه میکروکنترلر در حلقه HERE به طور پیوسته می‌چرخد. هر وقت که سوئیچ INT1 (پایه P3.3) فعال شود میکروکنترلر از حلقه خارج شده و به بردار 0013H پرش خواهد کرد. ISR مربوط به LED, INT1 را روشن می‌کند، و برای مدتی روشن نگه می‌دارد، و قبل از بازگشت آن را خاموش می‌کند. اگر در زمانی که دستور RETI اجرا می‌شود پایه INT1 هنوز پایین باشد، میکروکنترلر مجدداً وقفه را تکرار

می‌نماید. بنابراین برای خاتمه دادن به این مسئله، باید INT1 قبل از اجرای RETI به سطح بالا بازگردانده شود.

نمونه‌برداری وقفه حساس به سطح پایین

پایه‌های P3.2 و P3.3 برای I/O معمولی به کار می‌رود، مگر اینکه بیت‌های INT0 و INT1 در ثبات IE فعال شوند. پس از فعال شدن وقفه‌های سخت افزاری در ثبات IE، کنترل‌گر از پایه INTn در ازاء هر سیکل ماشین برای سیگنال سطح پایین شروع به نمونه‌برداری می‌کند. طبق مستندات سازنده، پایه باید تا شروع اجرای ISR پایین نگهداشته شود. "اگر پایه INTn قبل از شروع اجرای ISR به منطق بالا بازگردد، وقفه‌ای رخ نخواهد داد. با این وجود، بدلیل فعال شدن وقفه با توجه به سطح پایین سیگنال وقفه، باید قبل از اجرای دستور RETI ، آن را به سطح بالا آورد. مجدداً طبق مستندات یکی دیگر از سازندگان اگر پایه INTn پس از اجرای دستور RETI در ISR، همچنان در سطح پایین باقی بماند، وقفه دیگری پس از اجرای یک دستور رخ خواهد داد. "

بنابراین به منظور اطمینان از فعال شدن وقفه سخت افزاری در ایه INTn ، مطمئن شوید که طول سیگنال سطح پایین حدود چهار برابر سیکل ماشین است و بشتر نیت. دلیل این است که وقفه حساس به سطح لچ نیست.

بنابراین پایه باید تا شروع اجرای ISR پایین بماند.

وقفه‌های حساس به لبه

همانطور که قبلاً بیان شد، پس از بازنشانی 8051، وقفه‌های INT0 و INT1 حساس به سطح خواهد شد. برای تبدیل آنها به نوع حساس به لبه، باید بیت های ثبات TCON را برنامه‌نویسی کنیم. ثبات TCON، همراه با دیگر چیزها، بیت های پرچم IT1,IT0 که تعیین کننده وقفه حساس به سطح یا لبه هستند، را نگه می‌دارد. IT1, IT0 به ترتیب بیت‌های D0 و D2 از ثبات TCOn هستند. به آنها TCON.0 و TCON.2 هم می‌گویند زیرا ثبات TCON از نوع آدرس پذیر بیتی است. پس از بازنشانی، هر دو TCON.0 (ITO) و TCON.2 (IT1) در منطق 0 هستند، به این معنی که وقفه‌های سخت افزاری پایه‌های INT0 و ITN1 به سطح حساس می‌باشند. با بالا بردن بیت‌های TCON.0 و TCON.2 که با دستوراتی چون " SET B TCON.0 " و " SET B TCON.2 " انجام می‌شود، وقفه‌های سخت افزاری بیرونی INT0 و ITN1، سیگنال‌های حساس به لبه خواهند شد. مثلاً، دستور " SET B TCON.2 " آنچه را که **وقفه حساس به لبه** است در INT1 بوجود می‌آورد. در این حالت هنگام با گذر سطح بالا به پایین در سیگنال اعمال شده به پایه P3.3، میکروکنترلر وقفه یافته و بمکان 0013H در جدول بردار وقفه برای سرویس ISR خواهد پرید. البته فرض بر این است که بیت وقفه در ثبات IE فعال شده است.

اکنون به مثال ٦-١١ توجه کنید. تنها اختلاف این برنامه با برنامه مثال٥-١١ این است که در اولین خط MAIN ، دستور "SET B TCON.2" موجب می‌شود. تا UNT1 یک وقفه حساسبه لبه باشد. وقتی که لبه پایین رونده سیگنال به پایهITN1 اعمال شد، LED یک لحظه روشن می‌شود. دوره روشن ماندن LED به تأخیر زمانی درون IER برای INT1 بستگی دارد. برای روشن کردن مجدد LED ، یک پالس بالا به پایین دیگر باید به پایه P3.3 اعمال گردد. این برخلاف مثال ٥-١١ است. در مثال ٥-١١ با توجه به طبیعت حساسیت به سطح وقفه، مادامی که ITN1 در سطح پایین نگه داشته شد، LED در وضعیت روشن بود. ولی در این مثال برای روشن کردن مجدد LED ، پالس ITN1 باید به بالا برگدانده شده و سپس به سطح پایین برده شود تا یک لبه پایین رونده برای فعال کردن وقفه انجام گردد.

نمونه‌برداری وقفه حساس به لبه

قبل از پایان این بخش، لازم است باین سؤال پاسخ دهیم که نمونه‌برداری از وقفه حساس به لبه با چه فاصله زمانی اتفاق می‌افتد. در وقفه‌های حساس به لبه، منبع خارجی باید حداقل برای یک سیکل ماشین در سطح بالا و سپس برای یک سیکل ماشین دیگر پایین نگهداشته شود تا میکروکنترلر قادر به دیدن آن گردد.

لبه پایین رونده بوسیله 8051 لـچ شـده و بوسـیله قبـات TCON نگهـداری مـی‌شـود. بیت‌های TCON.1 و TCON.3 لبه پـایین رونـده لـج شـده را بترتیـب بـاری INT0 و ITN1 نگه می‌دارند. TCON.1 و TCON.3 را IE0 و IE1 هم می‌خوانند، شـکل ٦-١١، آنها همچون پرچم های وقفه در حال سرویس عمل می‌کنند. وقتـی کـه پـرچم وقفـه در حال سرویس بال برود، باین معنی است که به دنیای بیرون اعـلام شـود، وقفـه در حـال سرویس است و تا پایان سرویس به این پایه وقفه INTn پاسخی داده نخواهد شـد. ایـن درست مانند سیگنال مشغولی بر روی تلفنی است که در حال حاضـر مشـغول باشـد. بـا توجه به بیت‌های IT0 و IT1 در ثبات TCON ، بر دو نکته باید تأکید کرد.

١- اولین نکته این است که وقتی ISR ها پایان یافتند (یعنی دستور RETI اجـرا شـد)، این بیت‌ها (TCON.1 و TCON.3) پاک می‌شوند و باین معنی است کـه وقفـه پایـان یافته و 8051 آماده پاسخگویی به وقفه دیگری در آن پایه است.

برای تشخیص وقفه دیگر، پایه باید به سطح بالا بازگردد و مجدداً به پایین بیاید تا بعنوان یک وقفه حساس به لبه شناخته شود.

٢- نکته دوم این است که در حین اجرای روال سرویس وقفه، پایه INTn بدون توجه به اینکه چند بار گذر بالا به پایین را انجام می‌دهد، چشم پوشی می گردد. در واقع یکـی از وظایف دستور RETI این است که بیـت مربوطـه را در ثبـات TCON (TCON.1 یـا

TCON.3) پاک کند. ایـن بـدان معنـی اسـت روال سـرویس در حـال اجـرا نسـیت و سرویس دهی پایان یافته است. بـاین دلیـل TCON.1 و TCON.3 در ثبـات TCON را **پرچم‌های در حال سروی وقفه** می‌خوانند. هر وقت که در پایه INT لبـه پـایین رونـده تشخیص داده شود، پرچم در **حال سرویس بالا** خواهد رفـت و درهنگـام اجـرای ISR همچنان باقی می‌ماند. این پرچم فقط بوسیله RETI، که آخرین دستور ISR اسـت پـاک می‌شود. باین دلیل نیازی بـرای اسـتفاده از دسـتوری مثـل " CLR TCON.1 (" CLR TCON.3 برای INT1) قبل از RETI در ISR مربوطه نیست. همـانطور کـه در بخـش بعدی خواهیم دید. این مطلب در وقفه سریال صادق نیست.

IT0 و IT1

TCON.0 و TCON.2 بترتیب IT0 و IT1 خوانده می‌شوند. این دو بیت مدهای سـطح پایین یا حساس به لبه را در وقفه‌های سخت افـزاری بیرونـی پایـه‌هـای INT0 و ITN1 تنظیم می کنند. هر دوی آنها پس از بازنشانی 0 می‌شوند، که بدینوسیله از نـوع حسـاس به سطح پایین خواهند شد. برنامه‌نویس می‌تواند هر یک از آنها را به وقفه سخت‌افـزاری بیرونی حساس به لبه تبدیل نماید. در یک سیستم مفروض مبتنی بر ۸۰۵۱ ، یکبار که در ۱ یا 0 تنظیم شوند، دیگر تغییر نمی‌یابند زیرا طراح آن را بعنوان وقفه حساس بـه لبـه یـا سطح تثبیت نموده است.

IE0 و IE1

TCON.1 و TCON.3 بترتیب IE0 و IE1 خوانده می‌شوند. این دو بیت بوسیله ۸۰۵۱ فقط برای ردیابی وقفه حساس به لبه بکار می‌روند، به بان دیگر اگر IT0 و IT1 هر دو 0 باشند، وقفه‌های سخت‌افزاری هر دو حساس به سطح پایین هستند. این مطلب برای IE0 و IE1 هرگز چنین نیست. بیت‌های IE0 و IE1 بوسیله ۸۰۵۱ فقط برای لچ گذر لبه بالا به پایین در پایه‌های INT0 و ITN1 بکار می‌روند. بعد از پالس‌گذار لبه در پایه INT0 (یا ITN1)، ۸۰۵۱ در ثبات TCON بیت IEx را علامت می‌زند (به سطح بالا می‌برد)، به بردار مربوطه در جدول بردار وقفه می پرد، و شروع به اجرای ISR می‌نماید. در حین اجرای ISR، هیچ تغییر سطح یا پالس H به L در INT0 (یا INT1) تشخیص داده نمی‌شود، و بنابراین از وقع هر وقفه‌ای در وقفه جلوگیری می‌نماید. تنها اجرای دستور RETI در پایان ISR بیت IEx را پاک کرده و مشخص می‌کند که پالس جدید H به L (بالا به پایین) مجدداً وقفه را فعال خواهد کرد. از این بحث می‌توان دید که بیت‌های IE0 و IE1 در درون بوسیله ۸۰۵۱ بکار می‌ورد تا مشخص شود ایا وقفه‌ای در حال عمل است یا خیر. به بیان دیگر برنامه‌نویس کاری با این بیت‌ها ندارد زیرا آنها مخصوص استفاده داخلی هستند.

TR0 و TR1

این بیت ها، همان D4 (TCON.4) و D6 (TCON.6) از ثبات TCON هستند. ما این بیت‌ها را دفصل ۹ معرفی کردیم. آنها برای شروع و توقف تایمرهای 0 و ۱ بکار می‌رفتند. گرچه ما از دستوری چون " SETB TRx " یا CLR TRx استفاده کردیم، ولی می‌توانستیم دستورات "SETB TCON.4 " و " CLR TCON.4" هم استفاده نمائیم زیرا TCON یک ثبات آدرس پذیر بیتی است.

TF0 و TF1

این بیت‌ها همان D5 (TCON.5) و D7 (TCON.7) از ثبات TCON می‌باشند. آنها بترتیب بوسیله تایمرهای 0 و ۱ برای مشخص کردن عبور از 0 بکار می‌رفتند. گرچه ما از JNB Tfx , target و " CLR Trx" استفاده کردیم، ولی می توانستیم دستوراتی مانند TCON.5 JNB target و " CLR TCON.5 " را هم بکار ببریم زیرا TCON یک ثبات آدرس پذیر بیتی است.

قبل از پایان این بخش به لیست همه پرچم های وقفه در جدول ۲-۱۱ توجه کنید. در حالی که ثبات TCON چهار پرچم وقفه را نگه می‌دارد، در ۸۰۵۱ ، ثبات SCON پرچم های RI و TI را داراست.

اولویت وقفه در ۸۰۵۱

عنوان مورد بحث بعدی این است که اگر دو وقفه بطور همزمان فعال شوند، چه خواهد شد؟ کدام یک از این دو ابتدا پاسخ داده می شود. اولویت وقفه تیتر اصلی در این بخش است.

اولویت وقفه بعد از بازنشانی

وقتی که ۸۰۵۱ روشن شود، اولویت ها بر طبق جدول ۳-۱۱ منتسب می شوند. مثلاً از جدول ۳-۱۱ می بینیم که اگر وقفه سخت افزاری بیرونی ۰ و ۱ بطور همزمان فعال شوند، ابتدا به وقفه 0(INT0) پاسخ داده می شود. پس از سرویس دهی INT0 ، نوبت به سرویس دهی INT1 می رسد زیرا INT1 دارای اولویت پایین تری است. در واقعیت، مراتب اولویت در جدول چیزی جز رشته سرکشی درونی نیست که در آن ۸۰۵۱ وقفه ها برحسب ترتیب لیست شده در جدول ۳-۱۱ سرکشی می نماید و بر طبق آن پاسخ می دهد.

تنظیم اولویت وقفه با ثبات IP

می توانیم با بر هم زدن ترتیب جدول ۳-۱۱ اولویت بالاتر را به هر یک از وقفه ها بدهیم. این کار با برنامه نویسی ثباتی بنام IP (اولویت وقفه) انجام می شود . شکل ۸-

۱۱ بیت های ثبات IP را نشان می دهد. پس از بازنشانده شدن، ثبات IP تماماً ۰ خواهد بود، و بنابراین ترتیب اولویت مطابق جدول ۱۲-۱۱ توجه کنید.

نکته دیگری که باید روشن شود این است که دو یا چند بیت اولویت وقفه در ثبات IP در سطح بالا قرار گیرند. در این حالت، ضمن اینکه این وقفه ها اولویت بالاتری از دیگران دارند، بر طبق ترتیبشان درجدول ۳-۱۱ خواهند بود. مثال ۱۳-۱۱ ملاحظه شود.

وقفه در درون وقفه

اگر ۸۰۵۱ در حال اجرای ISR متعلق به یک وقفه باشد و وقفه دیگری فعال شود چه خواهد شد؟ در این حالت یک وقفه با اولویت بالاتر می تواند اولویت پایین را وقفه دهد. این، وقفه در درون وقفه است. در ۸۰۵۱، یک وقفه پایین رتبه می تواند با وقفه بالاتر وقفه یابد و نه با وقفه پایین تر. گرچه همه وقفه ها لچ شده و در درون نگهداری می شوند، هیچ وقفه اولویت پایین تر نمی تواند توجه فوری CPU تا اتمام سرویس های وقفه های بالا رتبه در ۸۰۵۱ را به خود جلب کند.

راه اندازی وقفه با نرم افزار

مواردی وجود دارد که نیاز به تست ISR بروش شبیه سازی است. این کار را می توان با دستورات ساده ای انجام داده و وقفه ها را در بالا تنظیم کنیم و بنابراین ۸۰۵۱ به جدول بردار وقفه پرش نماید. مثلاً، اگر بیت IE برای تایمر ۱ را ۱ کنیم، دستوری مثل

"SETB TF1" ۸۰۵۱ را در هر کاری که باشد وقفه داده و آنرا به پرش به جدول بردار وقفه وا می دارد. ما می توانیم وقفه ای را با یک دستور تولید کنیم که این خود موجب بالا رفتن پرچم وقفه خواهد شد.