فصل ۳

میکروکنترلر ۸۰۵۱،
سخت افزار و اصول برنامه نویسی

در فصل قبل ویژگیهای میکروکنترلرها بعنوان بخش پردازشگر سیستم های کنترلی و نیز معیارهای انتخاب یک میکروکنترلر را بررسی کردیم.
در این فصل با میکروکنترلر پایه این کتاب یعنی میکروکنترلر ۸۰۵۱، ویژگیهای سخت افزاری و اصول برنامه نویسی آن آشنا می شویم. در فصول آینده به جنبه های دیگر برنامه نویسی ۸۰۵۱ خواهیم پرداخت.

میکروکنترلر ۸۰۵۱

بیش از دو دهه از آغاز عرضه میکروکنترلر ۸۰۵۱ توسط شرکت اینتل به بازار می گذرد. تاکنون دهها میلیون تراشه ۸۰۵۱ در بازارهای سراسر جهان به فروش رفته است. دلیل این محبوبیت فوق العاده را علاوه بر *سادگی ساختار* و *استفاده آسان از آن*، می توان در *گستردگی طیف تولید کنندگان آن* یافت؛ در حال حاضر بیش از ۳۰ شرکت مختلف، گستره وسیعی از محصولات مبتنی بر ۸۰۵۱ اینتل را (بیش از ۴۰۰ میکروکنترلر مختلف) تولید می کنند.
شرکت اینتل اجازه تولید تراشه های مبتنی بر۸۰۵۱ را برای شرکتهای دیگر صادر کرده است؛ تنها شرط اینتل برای اعطای این مجوز، تطابق برنامه نویسی[1] محصولات آنها با ۸۰۵۱ ساخت اینتل است؛ یعنی برنامه ای که به زبان اسمبلی ۸۰۵۱ یا زبان C-51 (که زبان برنامه نویسی سطح بالای میکروکنترلر ۸۰۵۱ است) نوشته شده باشد، به شرط وجود امکانات سخت افزاری روی تمام نسخه های میکروکنترلرهای ۸۰۵۱ ساخت هر شرکت، قابل فهم است.
از مهم ترین تولید کنندگان فعلی میکروکنترلرهای مبتنی بر ۸۰۵۱ علاوه برشرکت اینتل، می توان به شرکت های Atmel، Dallas، Philips، Hynix، Infineon، Analog Devices، Wihbond، Silicon Laboratorios و TI اشاره کرد که محصولات Atmel متداول ترین میکروکنترلرهای مبتنی بر۸۰۵۱ در بازار کنونی هستند. یک برنامه به زبان مخصوص میکروکنترلر ۸۰۵۱ روی تمام محصولات مبتنی بر این میکروکنترلر اجرا می‌شود.

---

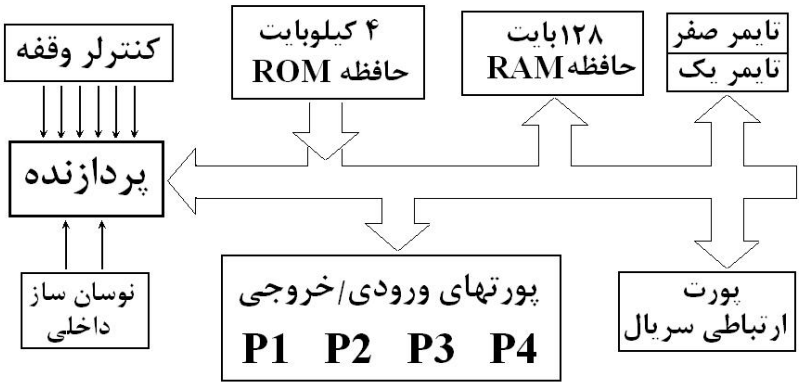
[1] ***Code Compatibilty***

میکروکنترلرهای مبتنی بر ۸۰۵۱ با وجود شباهت برنامه سازی و ساختار کلی سخت افزاری، در مشخصاتی نظیر نوع و مقدار حافظه ROM روی تراشه، مقدار حافظه RAM روی تراشه، سرعت میکروکنترلر، تعداد پورتهای ورودی/خروجی، ولتاژ تغذیه، تعداد پینها و نیز امکانات جانبی مانند تعداد تایمرها، تعداد منابع وقفه، وجود حافظه های E^2PROM روی تراشه، حمایت از پروتکلهای ارتباطی مانند سریال و غیره با هم تفاوت دارند که بنا به نیاز سیستم باید یکی از آنها را انتخاب کرد.

در ادامه فصل با ساختار پایه ۸۰۵۱ که در ابتدا توسط شرکت اینتل ساخته شد، آشنا می شویم و سپس به ویژگیهای سخت افزاری و اصول برنامه نویسی آن می پردازیم.

ساختار پایه ۸۰۵۱

آنچه شرکت اینتل در سال ۱۹۸۱ با نام تراشه MCS-51 بعنوان اولین میکروکنترلر ۸۰۵۱ به بازار عرضه کرد، دارای ساختاری به شکل زیر بود :



اجزاء داخلی میکروکنترلر ۸۰۵۱ پایه

در MCS-51 امکانات زیر وجود دارد :

- ۴ کیلوبایت حافظه ROM برای ذخیره برنامه کاربر
- ۱۲۸ بایت حافظه RAM برای ذخیره داده های موقت
- ۴ پورت ورودی/خروجی ۸ بیتی
- دو تایمر/شمارنده
- یک کنترلر ارتباط سریال
- یک کنترلر وقفه

۸۰۵۱ یک میکروکنترلر **۸ بیتی** است؛ یعنی پهنای گذرگاه داخلی، تعداد پینهای پورتهای ورودی/خروجی و نیز ثباتهای داخلی همگی ۸ بیتی (۱ بایتی) می باشند و پردازشهای حسابی، منطقی و ... نیز در قالبهای ۸ بیتی انجام می شود؛ داده های با اندازه بیش از ۸ بیت، باید به قسمتهای ۸ بیتی شکسته شوند تا قابل پردازش با ۸۰۵۱ باشند[1].

در اکثر میکروکنترلرهایی که توسط شرکتهای مختلف بر پایه ۸۰۵۱ ساخته می شود امکانات ۸۰۵۱ پایه (بعلاوه امکانات دیگر که توسط شرکت سازنده گنجانده می شود) وجود دارد. البته این امکانات بسیار متنوع هستند؛ مثلاً میکروکنترلر ۴۰ پایه AT89S52 دارای ۸ کیلوبایت حافظه ROM، ۲۵۶ بایت حافظه RAM و ۳ تایمر شمارنده است؛ در حالی که میکروکنترلر ۲۰ پایه AT89C1051 تنها یک کیلوبایت حافظه ROM، ۶۴ بایت حافظه RAM و ۲ پورت ورودی/خروجی دارد. بسته به نیازهای سیستم باید میکروکنترلری با امکانات مناسب را برگزید.

میکروکنترلر ۸۰۵۱ می‌تواند حداکثر تا ۶۴ کیلوبایت حافظه ROM داشته باشد که درون تراشه نوع پایه ۸۰۵۱، چهار کیلوبایت از این نوع حافظه وجود دارد. میکروکنترلرهای مختلف خانواده ۸۰۵۱، مقادیر مختلفی از حافظه ROM درون خود دارند. مثلاً میکروکنترلر DS89C450 دارای ۶۴ کیلوبایت حافظه ROM درون تراشه است.
حداکثر فضای RAM داخل تراشه نیز ۲۵۶ بایت است.
آشنایی با انواع میکروکنترلرهای ۸۰۵۱ در حال حاضر برای شما لزومی ندارد. هدف ما در این کتاب آشنا کردن شما با سخت افزار و برنامه نویسی ۸۰۵۱ نوع پایه است؛ با فراگیری آن می‌توانید به راحتی از هر محصول مبتنی بر ۸۰۵۱ استفاده کنید.
در این فصل یکی از محصولات شرکت Atmel به نام AT89S51 را به طور خاص معرفی و در ادامه کتاب به آن می پردازیم و معرفی محصولات دیگر Atmel و شرکتهای دیگر را در ضمیمه (ج.۱) پی می گیریم. با مطالعه فصول آینده و مراجعه به ضمیمه فوق می توانید برای طراحی هر سیستمی، مناسب ترین میکروکنترلر ۸۰۵۱ را انتخاب کنید.

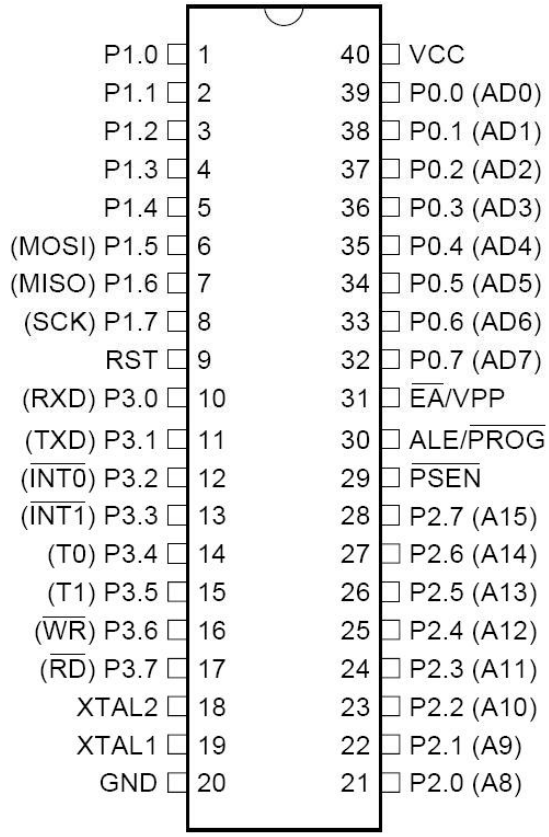
لازم به ذکر است که تولید میکروکنترلرهای مبتنی بر ۸۰۵۱ علیرغم رقابت شدید میکروکنترلرهای AVR (ساخت شرکت Atmel) و PIC (ساخت شرکت Microchip) هنوز ادامه دارد؛ میکروکنترلر DS80C400 که در سال ۲۰۰۳ توسط شرکت Dallas بر

[1] البته ۸۰۵۱ دستورات بیتی نیز دارد که به آن توانایی کار کردن با تک تک بیتهای یک پورت یا بعضی ثباتها را می دهد. فصل ۷ کتاب به دستورات بیتی اختصاص دارد.

پایه ۸۰۵۱ عرضه شد، مؤیدی بر این مطلب است. در این میکروکنترلر، پشته TCP/IP گنجانده شده است. سیستمهای مبتنی بر این میکروکنترلر[1] توانایی اتصال به شبکه و اینترنت و قابلیت های جالبی از جملهFTP Server، Telnet Server، ماشین مجازی Java و ... را نیز دارند.

تراشه AT89S51

تراشه فوق ساخت شرکت Atmel است و تمام امکانات ذکر شده ۸۰۵۱ پایه را داراست[2].



در آنچه از این پس گفته می شود، هرگاه به نام میکروکنترلر ۸۰۵۱ اشاره شد، منظور ساختار کلی ۸۰۵۱ است که در محصولات متداول آن مشترک می باشد و هرگاه به میکروکنترلر AT89S51 اشاره شد، منظور ذکر خصوصیات خاص این تراشه است. مجدداً یادآوری می کنیم که این کتاب برای آموختن میکروکنترلر ۸۰۵۱ نگاشته شده است و هدف از اشاره های خاص به AT89S51، نشان دادن خصوصیات یک نمونه متداول از طیف وسیع محصولات مبتنی بر ۸۰۵۱ است.

پینهای AT89S51

شکل بالا پینهای تراشه AT89S51 را نشان می دهد.

پینهای تغذیه

پینهای Vcc و GND مخصوص تأمین تغذیه تراشه هستند. در AT89S51، ولتاژ تغذیه (Vcc) می‌تواند در بازه ۴ ولت تا ۵/۵ ولت باشد.

پورتهای ورودی/خروجی[3]

[1] مانند DSTINIs400 که همزمان توسط شرکت Dallas طراحی و عرضه شد.

[2] در AT89S51 علاوه بر امکانات ۸۰۵۱ پایه، یک **تایمر سگ نگهبان (Watch-dog Timer)** نیز وجود دارد که در فصل ۹ به آن خواهیم پرداخت.

[3] ***Input/Output (I/O) ports***

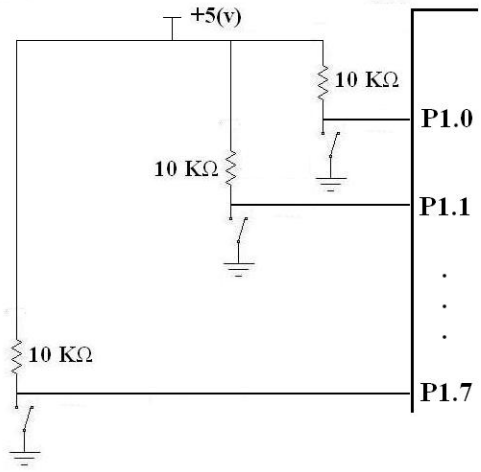
میکروکنترلر ۸۰۵۱ دارای ۴ پورت ورودی/خروجی ۸ بیتی به نامهای P0 (پایه‌های ۳۹ تا ۳۲)، P1 (پایه‌های ۱ تا ۸)، P2 (پایه‌های ۲۱ تا ۲۸) و P3 (پایه‌های ۱۰ تا ۱۷) است؛ در واقع داخل میکروکنترلر، چهار ثبات ۸ بیتی به نامهای P0، P1، P2 و P3 وجود دارند که بیتهای این ثباتها به پینهای چهار پورت ورودی/خروجی متصل هستند.
بعنوان مثال پینهای ۱ تا ۸ میکروکنترلر AT89S51، به پورت P1 اختصاص دارد. شماره گذاری بیتهای یک پورت، مثلاً پورت P1 به صورت زیر است :

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

پین P1.0، بیت شماره صفر پورت P1 (یا به بیان بهتر ثبات داخلی P1) است.
منظور از ورودی/خروجی بودن یک پورت این است که پینهای این پورت هم می توانند بعنوان ورودی میکروکنترلر و هم به عنوان خروجی آن مورد استفاده قرار گیرند. به بیان دیگر با استفاده از این پورتها، می‌توان بیتهای اطلاعاتی را در قالب سیگنالهای ولتاژی (ولتاژ حدود صفر برای «صفر منطقی» و ولتاژ حدود پنج ولت برای «یک منطقی») با میکروکنترلر مبادله کرد.

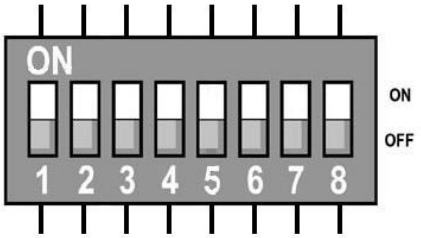
استفاده از پورت بعنوان ورودی

ساختار روبرو را در نظر بگیرید.

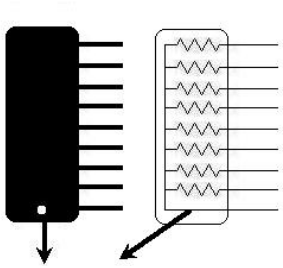


در اینجا پورت P1 بعنوان ورودی به کار می رود و با خواندن مقدار آن توسط برنامه میکروکنترلر، یک عدد ۸ بیتی که توسط ۸ کلید دوحالته تولید شده، وارد میکروکنترلر (در واقع وارد ثبات داخلی P1 میکروکنترلر) می شود.

پرسش) اگر کلیدهای اول و دوم باز و بقیه کلیدها بسته باشند، با خواندن مقدار P1، چه عددی وارد میکروکنترلر می شود ؟ در این حالت ولتاژی که روی پینهای P1.0 و P1.6 دیده می شود، چند ولت است ؟



DIP Switch



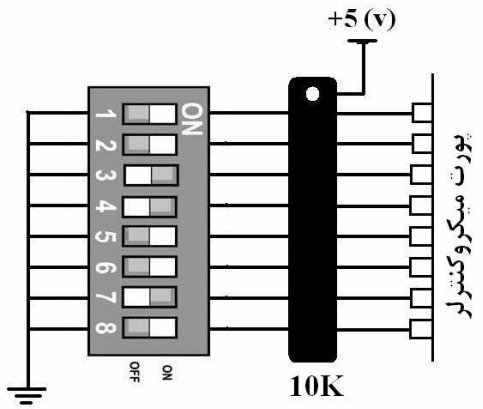
سر مشترک مقاومت‌ها

مقاومت آرایه ای
(SIP)

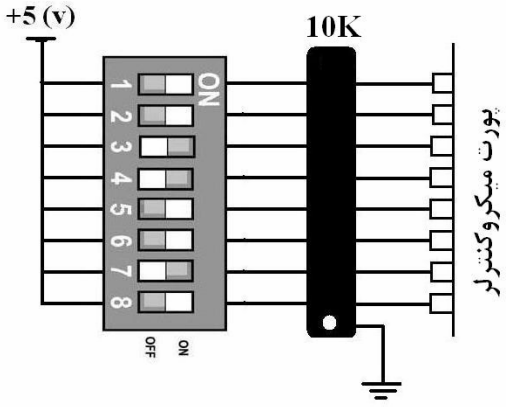
معمولاً برای اتصال ساختار فوق (۸ کلید دوحالته)، از کلیدهای کوچک کنار هم که در تعداد ۴ تایی یا ۸ تایی ساخته می شوند و به نام میکروسوییچ یا DIP Switch معروفند، استفاده می شود.

همچنین برای خودداری از به کار بردن ۸ مقاومت مجزا که باعث شلوغی مدار می شود، از مقاومتهای آرایه ای که به نام SIP معروفند، استفاده می شود که هم جای کمتری اشغال می کنند و هم اتصال آنها ساده تر است.

پرسش) در شکل زیر، با اجرای دستور خواندن چه عددی وارد میکروکنترلر می‌شود؟ (پین بالایی از همه پرارزش‌تر است)

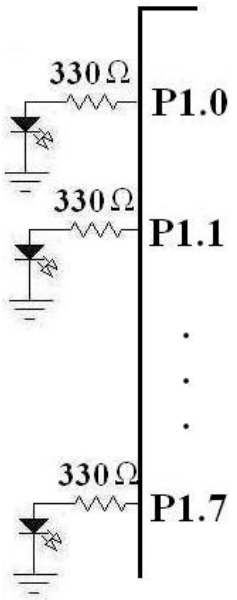


پرسش) در شکل زیر، با اجرای دستور خواندن چه عددی وارد میکروکنترلر می‌شود؟ (پین بالایی از همه پرارزش‌تر است). تفاوت این مدار با ساختار بالا چیست؟



استفاده از پورت بعنوان خروجی

حال ساختار روبرو را در نظر بگیرید.



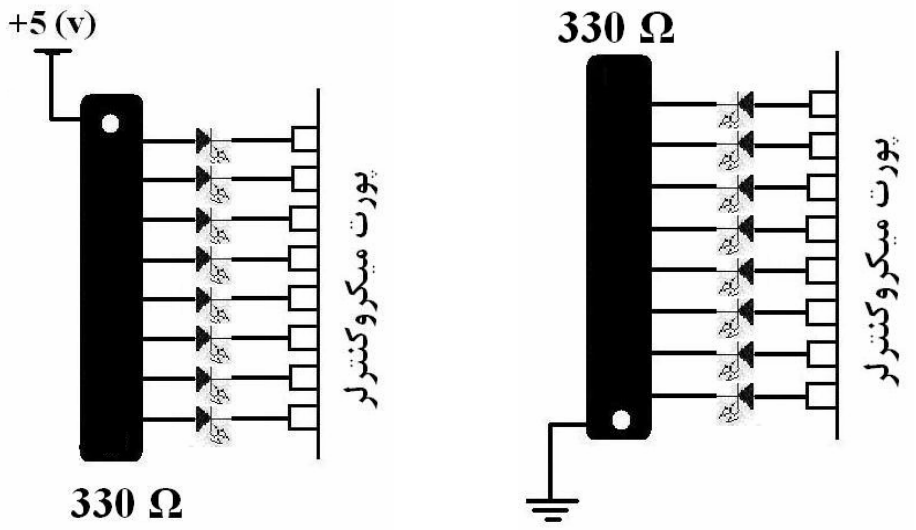
در این حال پورت P1 بعنوان خروجی به کار می رود و عددی که توسط میکروکنترلر در آن (در واقع در ثبات داخلی P1) نوشته می شود، روی LED ها ظاهر می شود.

پرسش) اگر برنامه میکروکنترلر عدد ۱۰۱۱۰۰۰۱ را در پورت P1 بنویسد، کدام LED ها روشن و کدامیک خاموش می شوند ؟ ولتاژی که روی هر پین وجود دارد را مشخص کنید.

پرسش) اگر بخواهیم LED خاموش نشان دهنده "یک" و LED روشن نشان دهنده "صفر" باشد، چه تغییری در ساختار LED ها باید داده شود ؟

در اینجا نیز می‌توان از مقاومتهای آرایه‌ای ۳۳۰ اهم استفاده کرد.

پرسش) دو ساختار زیر را بررسی کنید و عملکرد آنها را توضیح دهید.



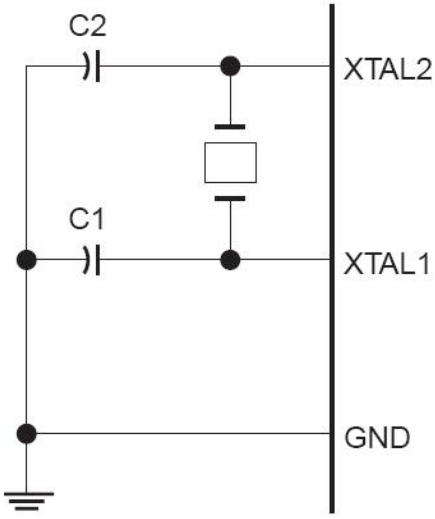
هر چهار پورت میکروکنترلر می توانند به صورت ورودی یا خروجی مورد استفاده قرار گیرند. دربخشهای بعدی نحوه خواندن یک پورت یا نوشتن در آن را خواهید آموخت.

پینهای ورودی پالس ساعت

گفتیم که پردازنده ها و میکروکنترلرها مانند مدارات منطقی ترتیبی کار می کنند و برای هماهنگ شدن فعالیتهای داخلی نیاز به یک پالس همزمانی دارند.

در ۸۰۵۱ می توان این پالس همزمانی را توسط یک مدار خارجی تولید کرد و به پین XTAL1 متصل کرد. در این حال پین XTAL2 بدون اتصال باقی می ماند.

راه ساده تر استفاده از نوسان ساز داخلی ۸۰۵۱ است. داخل تراشه ۸۰۵۱، یک مدار نوسان ساز ناقص وجود دارد که برای تکمیل آن باید چند جزء خارجی به پایه های XTAL1 و XTAL2 متصل شود. این اجزاء شامل یک کریستال و دو خازن هستند.
سرعت کاری میکروکنترلر به فرکانس کاری کریستال بستگی دارد. این فرکانس می‌تواند مقادیر مختلفی داشته باشد که معمولترین آنها 11.0592 MHz، 12 MHz، 16 MHz و 24 MHz است.



در شکل روبرو نحوه اتصال کریستال به میکروکنترلر را می بینید.

خازنهای C1 و C2 که خازنهای bypass نامیده می شوند، پایه های XTAL1 و XTAL2 را به زمین متصل می کنند. مقدار معمول این خازنها 30 PF است.
هنگامی که مدار نوسان ساز به درستی کار کند، فرکانس کاری تراشه با اسیلوسکوپ روی پایه XTAL2 قابل مشاهده است.

هرچه فرکانس کریستال بالاتر باشد، سرعت اجرای برنامه میکروکنترلر نیز بالاتر است. البته گاهی به دلایل فنی ناچار به استفاده از یک کریستال خاص هستیم؛ مثلاً اگر بخواهیم بین کامپیوتر و میکروکنترلر ارتباط سریال برقرار کنیم، باید از کریستال 11.0592 MHz استفاده کنیم.
حداکثر فرکانس کریستال قابل اتصال به AT89S51، 33 MHz است. حداکثر فرکانس کاری یک مدل میکروکنترلر، بخشی از نام آن است؛ مثلاً نام کاملی که روی تراشه AT89S51 دیده می شود، اطلاعات زیر را بیان می کند :

منظور از بسته بندی DIP[1]، شکل مستطیلی تراشه است که پینها در دو طرف آن قرار دارند. این تراشه با بسته بندی PLCC[2] و TQFP[3] هم عرضه می شود که شکل مربعی شکل دارند و پینها در چهار طرف آن قرار می گیرند.

حرف "I" نشان دهنده این است که تراشه فوق دارای خواصی برای کار در محیطهای صنعتی (مانند تحمل دمای بالا) است. حرف "C" نشان دهنده تجاری[4] بودن تراشه (برای سیستمهای آموزشی) و حرف "M" نشان دهنده نظامی[5] بودن تراشه (برای کار در محیطهای سخت) است.

پس در هنگام خرید یک میکروکنترلر به نام کامل آن دقت کنید.

پین ورودی RST

پین RST یک پین ورودی فعال بالاست که هرگاه برای مدت کافی به (v) 5+ متصل شود، باعث Reset شدن میکروکنترلر می شود. Reset شدن به معنای بازگشت میکروکنترلر به حالت ابتدایی موقع روشن شدن است؛ با این کار ثباتهای داخلی میکروکنترلر مقادیر پیش فرض کارخانه سازنده را پیدا می کنند[6] و میکروکنترلر اجرای برنامه کاربر را مجدداً از ابتدا آغاز می کند.

Reset هنگامی انجام می شود که می خواهیم برنامه ذخیره شده در حافظه میکروکنترلر را دوباره از اول اجرا کنیم. دلیل این کار می تواند قفل شدن پردازندهٔ میکروکنترلر در نقطه ای از برنامه باشد یا اینکه می‌خواهیم برنامه مجدداً از ابتدا اجرا شود.

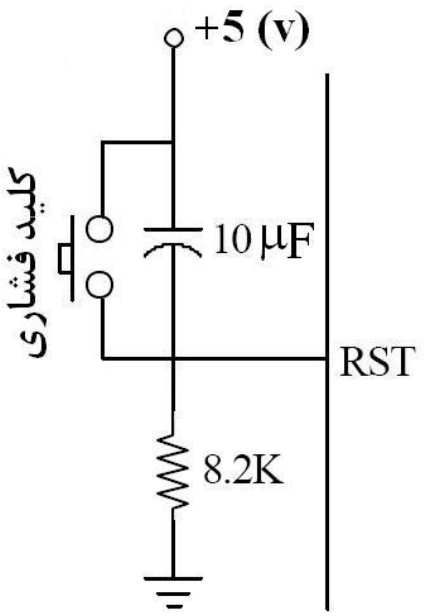
برای Reset شدن مناسب میکروکنترلر، پین RST باید بیش از مدت زمان 24/f (f فرکانس کریستال است) به (v) 5+ متصل باشد. مثلاً اگر از کریستال 12 MHz استفاده کنیم، این پین باید بیش از 2 µs به (v) 5+ متصل باشد.

---

[1] ***Dual Inline Package***
[2] ***Plastic Leaded Chip Carrier***
[3] ***Thin Quad Flat Package***
[4] ***Commercial***
[5] ***Military***

[6] این مقادیر **Reset Value** یا **Wake up Value** نامیده می شوند.

برای اینکه مطمئن شویم علاوه بر هنگامی که در حین عملکرد سیستم می‌خواهیم میکروکنترلر را Reset کنیم، هنگام اتصال ولتاژ تغذیه (v) 5+ به میکروکنترلر نیز عمل Reset انجام و کار میکروکنترلر به طور مناسب آغاز می‌شود، معمولاً پین Reset به مداری به شکل روبرو متصل می شود.



این مدار Auto Power-on Reset نام دارد. هنگام وصل شدن ولتاژ (v) 5+ به سیستم، خازن اتصال کوتاه است و پین RST به (v) 5+ متصل و فعال می شود. پس از مدت زمانی که به مقادیر خازن و مقاومت بستگی دارد، خازن شارژ شده و اتصال بین پین RST و (v) 5+ قطع شده و این پین به زمین متصل و غیرفعال می شود تا میکروکنترلر کار عادی خود را از سر بگیرد.

هرگاه نیاز به Reset کردن میکروکنترلر بود، با فشردن کلید فشاری، پین RST به (v) 5+ متصل شده و خازن اتصال کوتاه و تخلیه می شود. با رها کردن کلید، خازن مجدداً شارژ شده و ارتباط پین RST با (v) 5+ را قطع می کند و میکروکنترلر انجام کار را از سرمی گیرد.

مقادیر 10 µF برای خازن و 8.2 KΩ برای مقاومت به نحوی انتخاب شده اند که در هنگام اتصال (v) 5+ به مدار، مدت زمان لازم جهت شارژ شدن خازن به اندازه ای باشد که عمل Reset به درستی انجام شود. همیشه مدار Reset را با همین عناصر ببندید.

پین ورودی $\overline{EA}$ / VPP

به این پین، ۳ ولتاژ مختلف می تواند متصل شود :

- **ولتاژ صفر (زمین)** : اگر این پین به زمین متصل شود، میکروکنترلر به جای ROM داخلی اش از ROM خارجی استفاده می کند. نام EA[1] به همین موضوع اشاره دارد.
- **ولتاژ (v) 5+** : برای کار میکروکنترلر در حالت عادی، این پین باید به ولتاژ (v) 5+ متصل شود.
- **ولتاژ (v) 12+** : هنگام برنامه ریزی ROM داخلی میکروکنترلر به صورت موازی، این پین به ولتاژ (v) 12+ (ولتاژ برنامه ریزی یا Vpp) باید متصل شود[1].

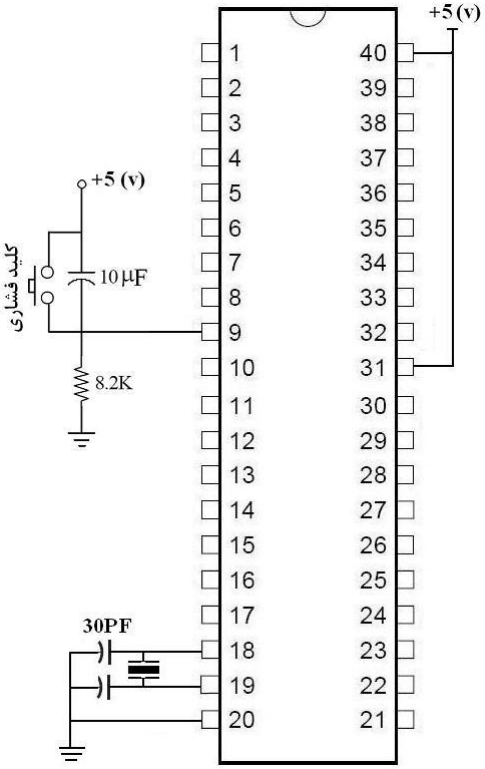
[1] *External Access*

پین $\overline{\text{ALE}}/\overline{\text{PROG}}$

این پین هنگام برنامه ریزی میکروکنترلر به صورت موازی، به زمین متصل می شود (نام PROG به همین مطلب اشاره دارد).

وقتی میکروکنترلر به درستی کار می کند، یک پالس مربعی با فرکانس بالا روی این پین ارسال می‌کند که آن را پالس ALE می نامیم. این پالس با اسیلوسکوپ قابل مشاهده است. وجود این پالس نشان‌دهنده عملکرد صحیح سیستم از دید سخت‌افزاری است.

مدار پایه ۸۰۵۱



برای اینکه میکروکنترلر ۸۰۵۱ صرفنظر از برنامه و سخت افزارهای جانبی به درستی کار کند، باید چند اتصال لازم به میکروکنترلر انجام شود که مجموع آنها را مدار پایه ۸۰۵۱ می نامیم. این مدار را در شکل روبرو می بینید.

اگر مدار فوق به طور صحیح بسته شده باشد و تمام اجزاء سالم باشند، پالس ALE را با اسیلوسکوپ می توانید مشاهده کنید. عدم وجود این پالس نشان دهنده وجود اشکال در مدار پایه ۸۰۵۱ است؛ در این صورت سیستم کار نخواهد کرد.

پین خروجی $\overline{\text{PSEN}}$[۲]

این پین توسط میکروکنترلر برای فعال کردن حافظه ROM خارجی در هنگام عدم استفاده از ROM داخلی مورد استفاده قرار می گیرد.

کاربرد دوگانه پورتها

تعدادی از پینها که در پرانتز نام دیگری نیز جلو آنها ذکر شده است، پینهای **دو منظوره**[۳] نام دارند و می توانند به دو منظور مورد استفاده قرار گیرند.

---

[1] میکروکنترلر AT89S51 قابلیت برنامه‌ریزی به صورت سریال را نیز دارد.

[2] ***Program Store ENable***

[3] ***Dual Purpose***

اتصال ROM خارجی به میکروکنترلر

در بعضی محصولات ۸۰۵۱ که ROM داخلی ندارند (مانند ۸۰۳۱) یا در مواقعی که حجم ROM داخلی میکروکنترلر برای برنامه ما کافی نیست، برنامه را در یک تراشه ROM جداگانه ذخیره کرده و آن را به میکروکنترلر متصل می کنیم. در این حال پورتهای P0 و P2 از حالت پورت ورودی/خروجی خارج شده و برای آدرس دهی و تبادل داده با ROM خارجی به کار می روند. پینهای EA/Vpp، ALE/PROG، PSEN، WR (P3.6) و RD (P3.7) دیگر سیگنالهای لازم برای این ارتباط را فراهم می سازند.
در ضمیمه (ج.۲) با نحوه اتصال ROM خارجی به میکروکنترلر ۸۰۵۱ آشنا خواهید شد.

پورت P3

تمام پینهای پورت P3 علاوه بر وظیفه ورودی/خروجی، دارای کاربرد دوم نیز هستند که در زیر آنها را به طور مختصر معرفی می کنیم و در فصول مربوط به طور مفصل به آنها خواهیم پرداخت.

RxD (P3.0) و TxD (P3.1) : این دو پین برای ارتباط سریال میکروکنترلر با جهان خارج به کار می رود. فصل ۱۰ کتاب به این مبحث اختصاص دارد.

INT0 (P3.2) و INT1 (P3.3) : این دو پین برای سرویس دادن به وقفه های سخت افزاری استفاده می شوند. وقفه ها موضوع فصل ۸ این کتاب هستند.

T0 (P3.4) و T1 (P3.5) : این دو پین در رابطه با تایمر/شمارنده ها به کار می روند که در فصل ۹ کتاب به آنها خواهیم پرداخت.

پورت P1

پینهای MOSI (P1.5)، MISO (P1.6) و SCK (P1.7) برای برنامه ریزی میکروکنترلر به صورت سریال به کار می روند.

اکنون پینهای میکروکنترلر ۸۰۵۱ را به طور کلی شناخته ایم و آماده ایم تا با نوشتن یک برنامه، اولین سیستم مبتنی بر ۸۰۵۱ را طراحی کنیم.

مدل برنامه نویسی میکروکنترلر ۸۰۵۱

برای آموختن مدل برنامه نویسی ۸۰۵۱ (یا هر میکروکنترلر و یا پردازنده دیگر) آشنایی با دو موضوع لازم است :

۱) معماری داخلی میکروکنترلر به ویژه ثباتها و حافظه ها تا بدانیم برای برنامه نویسی و استفاده از امکانات میکروکنترلر، چه ابزاری در اختیار داریم.

۲) قواعد زبان برنامه نویسی میکروکنترلر تا بدانیم چگونه می توانیم از امکانات میکروکنترلر استفاده کنیم.

ابتدا به معماری داخلی ۸۰۵۱ می پردازیم.

ثباتهای ۸۰۵۱

چون ۸۰۵۱ یک میکروکنترلر ۸ بیتی است، تمام ثباتهای داخلی آن نیز ۸ بیتی هستند.

ثباتهای همه‌منظوره[1]

ثباتهایی هستند که می‌توانند برای انجام هر عملیاتی در میکروکنترلر مورد استفاده قرار گیرند. دو ثبات همه منظوره مهم ۸۰۵۱، ثباتهای **A** و **B** هستند؛ ثبات A، ثبات انباره[2] نام دارد که در بسیاری از دستورات منطقی و محاسباتی به کار می رود. ثبات B نیز در برنامه نویسی نقش یک برگه یادداشت موقت را دارد.

در ۸۰۵۱ پایه، ۱۲۸ بایت حافظه RAM وجود دارد که برای دستکاری و ذخیره موقت داده ها بعنوان برگه یادداشت به کار می روند.

۳۲ بایت ابتدایی حافظه RAM *تحت عنوان ثبات*[3] برای برنامه نویسی مورد استفاده قرار می‌گیرند. این ۳۲ بایت در ۴ **بانک** ۸ بایتی (هر بانک از ۸ بایت تشکیل شده است) قرار دارند که نقشه آن را در روبرو می بینید :

بانک صفر		بانک یک		بانک دو		بانک سه	
0	R0	8	R0	16	R0	24	R0
1	R1	9	R1	17	R1	25	R1
2	R2	10	R2	18	R2	26	R2
3	R3	11	R3	19	R3	27	R3
4	R4	12	R4	20	R4	28	R4
5	R5	13	R5	21	R5	29	R5
6	R6	14	R6	22	R6	30	R6
7	R7	15	R7	23	R7	31	R7

[1] ***General Function Registers***
[2] ***Accumulator***
[3] ***Register***

هر خانه یک بانک ۸ بیتی است.

همانطور که می بینید خانه (بایت) اول هر بانک، R0 نام دارد! بـه طریـق مـشابه، خانـه هـای متناظر بانکها نام مشابه دارند. وقتی در دستوری به ثبات R0 مراجعه می شود، منظـور کـدام یک از ۴ ثبات R0 است؟

این موضوع بستگی به این دارد که در حال حاضر از کدام بانک استفاده مـی کنـیم؛ در ۸۰۵۱ دو **بیت انتخاب بانک**[1] به نامهای **RS0** و **RS1** وجود دارند که انتخاب بانک توسط آنهـا بـه صورت زیر انجام می شود :

RS1	RS0	بانک انتخاب شده
0	0	0
0	1	1
1	0	2
1	1	3

مثلاً وقتی RS0 = 0 و RS1 = 1 باشد، منظور از ثبات R0، *ثبات R0 بانک ۲* است.

چون هنگام روشن شدن ۸۰۵۱ داریم : RS1 = RS0 = 0 ، بانک پیش فـرض، بانـک صـفر است. نحوه مقداردهی بیتهای انتخاب بانک را به زودی خواهیم دید.

پس یک راه استفاده از ثباتهـای RAM، مقـداردهی بـه بیتهـای RS0 و RS1 و اسـتفاده از نامهای R0،R1، ... و R7 است.

راه دیگر مراجعه به این ثباتها، استفاده از آدرس آنهاست که در شکل صـفحه قبـل دیـده مـی شود؛ مثلاً می توان در یک دستور بـه ثبـات شـماره ۱۷ حافظـه RAM مراجعـه کـرد یـا بـا مقداردهی بیتهای انتخاب بانک به صورت RS0 = 0 و RS1 = 1 (یعنی انتخاب بانک ۲) بـه ثبات R1 مراجعه نمود. به بیان دیگـر *ثبـات R1 از بانـک ۲* همـان *خانـه شـماره ۱۷ حافظـه RAM* است.

پرسش) مراجعه به

الف) ثبات شماره ۴ حافظه RAM

ب) ثبات شماره ۳۰ حافظه RAM

معادل مراجعه به کدام ثبات از کدام بانک است ؟

---

[1] ***Register Switch***

ثبات پرچم یا کلمه حالت برنامه (PSW[1])

ثبات مهم دیگر، ثبات **PSW** است که در واقع ثبات **پرچم** میکروکنترلر ۸۰۵۱ به شمار می‌رود. با بیتهای این ثبات که در تصمیم گیریهای شرطی مورد استفاده قرار می گیرند به طور مختصر در انتهای این فصل و به طور مفصل در ادامه کتاب آشنا می شوید.

۸۰۵۱ دارای مجموعه متنوعی از ثباتهاست که هرکدام کاربرد خاصی دارند و در ادامه کتاب به تناسب مباحث، با آنها آشنا خواهیم شد.

در هنگام Reset شدن میکروکنترلر، هرکدام از ثباتهای آن مقدار پیش‌فرض کارخانه سازنده را به خود می‌گیرند که به اصطلاح Wake-up Value نامیده می‌شود. جدول زیر این مقادیر را برای بعضی ثباتهای ۸۰۵۱ نشان می‌دهد.

ثبات	Wake up Value (hex)
PC	0000
A	00
B	00
PSW	00
P0-P3	FF
SP	07
DPTR	0000
خانه‌های RAM	00

قواعد برنامه نویسی میکروکنترلر ۸۰۵۱

نحوه نوشتن برنامه های کنترلی با میکروکنترلر ۸۰۵۱ و دستورات آن، موضوع اصلی مباحث ادامه کتاب است. در اینجا می خواهیم چند دستور ساده ۸۰۵۱، قالب کلی برنامه های ۸۰۵۱ و نحوه ترجمه برنامه ها به کد ماشین ۸۰۵۱ را فرابگیریم.

در این کتاب برای نوشتن برنامه ها، از زبان اسمبلی ۸۰۵۱ استفاده می کنیم. با وجود ساده نبودن برنامه نویسی به زبان اسمبلی در ابتدای کار، این زبان می تواند شما را به خوبی با ساختار ثباتها، حافظه ها و دیگر ویژگیهای سخت افزاری میکروکنترلر آشنا کند. بعلاوه کد ماشین برنامه ای که به زبان اسمبلی نوشته شده باشد، کوتاهتر از کد ماشین برنامه ای با

[1] ***Program Status Word***

عملکرد مشابه است که به زبان سطح بالا نوشته شده است و به همین دلیل سریعتر اجرا می شود.

در فصل آخر کتاب تعدادی از مثالهای کتاب به زبان سطح بالای ۸۰۵۱ یعنی C-51 که مشابه زبان C است، بازنویسی شده تا علاوه بر فراگیری این زبان بتوانید برنامه نویسی به زبان اسمبلی و یک زبان سطح بالا را با هم مقایسه کنید.

برای آشنایی با نحوه طراحی سیستم با میکروکنترلر ۸۰۵۱، برنامه نویسی و نیز قالب کلی برنامه های آن، یک سیستم را برای مثال بررسی می کنیم. پیش از آن باید چند دستور را فرا بگیریم.

آشنایی با دو دستور مهم

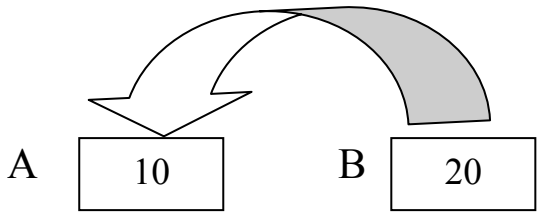
دستور MOV

این دستور برای انتقال داده ها در ۸۰۵۱ مورد استفاده قرار می گیرد و شکل کلی آن به صورت زیر است :

MOV مبدأ , مقصد

این دستور، *یک نسخه از داده موجود در "مبدأ" را در "مقصد"* ***کپی*** *می کند.* بنابراین محتویات مبدأ تغییری نمی کند.

مثلاً فرض کنید مقدار ثبات A برابر ۱۰ و مقدار ثبات B برابر ۲۰ است؛ دستور MOV A,B یک نسخه از داده موجود در ثبات B (مبدأ) یعنی ۲۰ را در ثبات A کپی می کند؛ یعنی مقدار جدید ثبات A بعد از اجرای این دستور، ۲۰ است و مقدار B نیز تغییر نمی‌کند[1].



قبل از اجرای دستور

بعد از اجرای دستور

[1] توجه کنید که دستور MOV برخلاف نامش، عمل انتقال را انجام نمی دهد؛ بلکه داده ها را **کپی** می کند . پس محتویات مبدأ تغییری نمی‌کند.

پرسش) محتویات ثباتهای A و B پس از اجرای دستور MOV B,A چه مقدار خواهد بود ؟

دستورات زیر چند مثال دیگر از کاربرد دستور MOV را نشان می دهند :

```
MOV   A , R0          MOV   B , R6
MOV   R4 , A          MOV   R1 , B
```

مثال) دستوراتی بنویسید که مقادیر موجود در ثباتهای A و B را با هم عوض کند.
دستورات زیر را در نظر بگیرید :

```
MOV   A , B
MOV   B , A
```

آیا این دو دستور عمل تعویض را به درستی انجام می دهند ؟
پاسخ منفی است؛ فرض کنید مقدار ثبات A برابر ۴۰ و مقدار ثبات B برابر ۵ است؛ با اجرای دستور اول، محتویات B (۵) در ثبات A کپی می شود. یعنی بعد از اجرای این دستور هر دو ثبات A و B دارای مقدار ۵ هستند و محتویات اولیه A (۴۰) که باید در B کپی شود از بین می رود!

پرسش) نشان دهید که دو دستور زیر نیز عمل تعویض محتویات ثباتهای A و B را به درستی انجام نمی دهند :

```
MOV   B,A
MOV   A,B
```

پرسش) نشان دهید دستورات زیر عمل تعویض محتویات ثباتهای A و B را به درستی انجام می دهند[1] :

```
MOV   R0,A
MOV   A,B
MOV   B,R0
```

مقداردهی ثباتها با دستور MOV

برای ذخیره یک عدد ثابت در یک ثبات از دستور MOV به شکل زیر استفاده می شود :

```
MOV   عدد ثابت# , مقصد
```

[1] دستور ***XCH   A,B*** نیز همین عمل را انجام می دهد.

مثلاً دستور MOV A,#10 عدد ۱۰ را در ثبات A کپی می کند.

پرسش) معنای دستورات زیر را بیان کنید :

MOV B,#50 MOV R2,#43

پرسش) با توجه به ۸ بیتی بودن ثبات های ۸۰۵۱، نشان دهید حداکثر عدد دهدهی که می تواند در یک ثبات ۸۰۵۱ ذخیره شود، عدد ۲۵۵ است؛ مثلاً دستور MOV A,#300 نادرست است.

در تمام مثالهای بالا، از اعداد ثابت در *دستگاه دهدهی* استفاده کردیم. اعداد ثابت می توانند در *دستگاههای دودویی و ۱۶تایی* نیز نوشته شوند. به مثالهای زیر توجه کنید:

مثال) دستور MOV A,#10010001**B** عدد دودویی ۱۰۰۱۰۰۰۱ (معادل ۱۴۵ دهدهی) را در ثبات A کپی می کند. علامت ***B*** بعد از عدد ثابت نشان دهنده بیان آن عدد در قالب دودویی (***B***inary) است.

پرسش) دو دستور زیر چه تفاوتی دارند ؟

MOV B,#11 MOV B,#11B

مثال) دستور MOV R2,#2EH عدد ۱۶تایی 2E (معادل ۴۶ دهدهی) را در ثبات R2 کپی می کند. برای نمایش یک عدد ثابت در قالب ۱۶تایی، بعد از عدد ثابت از علامت ***H*** (***H***exadecimal) استفاده می کنیم. طبق قواعد زبان اسمبلی، یک عدد ۱۶تایی حتماً باید با یک رقم شروع شود؛ پس چنانچه رقم سمت چپ عدد ۱۶تایی یکی از حروف A، B، C، D، E یا F باشد، قبل از آن یک رقم صفر قرار می دهیم. مثلاً دستور MOV A,#F4H غلط است و به جای آن باید از دستور MOV A,#0F4H استفاده کرد.

مثال) دستور MOV R6 , #159D یا دستور MOV R6 , #159 عدد ۱۵۹ دهدهی را در ثبات R6 قرار می‌دهد. اگر بعد از عدد ثابت حرف ***D*** (***D***ecimal) نوشته شود یا هیچ حرفی نوشته نشود، عدد ذکر شده یک عدد دهدهی عادی است.

پرسش) نشان دهید دستورات زیر دارای یک نتیجه هستند :

```
MOV   A,#255D
MOV   A,#255
MOV   A,#0FFH
MOV   A,#11111111B
```

پرسش) معادل دهدهی و دودویی دستورات زیر را بنویسید :

```
MOV   B,#11H                    MOV   B,#0BH
```

پرسش) معادل ۱۶تایی و دهدهی دستور MOV R5,#110B را بنویسید.

پرسش) برای ذخیره اعداد زیر دستورات دهدهی، دودویی و ۱۶تایی بنویسید :

الف) صفر ب) ۹ ج) ۱۶ د) ۱۷

مراجعه به حافظه RAM با دستور MOV

برای مراجعه به یکی از خانه های حافظه RAM علاوه بر نام ثبات (مثلاً R6)، می توان از شماره آن خانه به عنوان یکی از عملوندهای دستور MOV استفاده کرد؛ مثلاً دستور MOV A,12 ، محتویات خانه شماره ۱۲ حافظه RAM (که ثبات R4 از بانک ۱ است) را در ثبات A کپی می کند.

توجه کنید که این دستور را به صورت MOV A,0CH یا MOV A,1100B نیز می‌توان نوشت.

پرسش) نشان دهید با فرض استفاده از بانک صفر، با اجرای دو دستور زیر عدد ۳۰ در ثبات A قرار می گیرد :

```
MOV   R5,#30
MOV   A,5
```

پرسش) تفاوت دستورات زیر چیست ؟

```
MOV   20,#10
MOV   20,10
MOV   #20,10    (این دستور غلط است. چرا ؟)
```

دقت داشته باشید که هنگام استفاده از اعداد ثابت، علامت # را فراموش نکنید؛ چون دستور شما معنای کاملاً متفاوتی پیدا می کند؛ مثلاً دستور MOV A,#10 به معنای *کپی کردن* ***عدد ۱۰*** *در ثبات* A و دستور MOV A,10 به معنای *کپی کردن* ***محتویات خانه شماره ۱۰ حافظه RAM*** *(که ممکن است هر عددی باشد) در ثبات* A است.

دستور ADD

این دستور برای جمع محتویات دو ثبات یا جمع یک ثبات با یک عدد ثابت به شکل زیر به کار می رود :

ADD A,مبدأ ⟹ A ⟵ A + مبدأ

این دستور، مقدار موجود در مبدأ را با مقدار فعلی ثبات A جمع کرده و نتیجه را در ثبات A ذخیره می کند.

بعنوان مثال دستورات زیر را درنظربگیرید :

```
MOV  A,#20
ADD  A,#10
```

دستور اول عدد ۲۰ را در ثبات A کپی می کند. دستور دوم مقدار فعلی A (۲۰) را با عدد ۱۰ جمع کرده و حاصل (۳۰) را در A قرار می دهد.

پرسش) عدد ذخیره شده در ثبات A بعد از اجرای هر مجموعه دستورات را مشخص کنید :

```
MOV  A,#43
MOV  B,#0AAH
ADD  A,B
```

```
MOV  R0,#11101000B
MOV  A,#11B
ADD  A,R0
```

```
MOV  31,#99
MOV  A,#56
ADD  A,31
```

```
MOV  A,#200
ADD  A,#2BH
```

از دستورات بالا چند نکته مشخص می شود :

۱) یکی از عملوندهای جمع حتماً باید در ثبات A باشد. مثلاً دستور ADD B,#60 برای جمع محتویات ثبات B با عدد ۶۰ و ذخیره نتیجه در ثبات B نادرست است و به جای آن باید از یکی از مجموعه دستورات زیر استفاده کرد :

```
MOV   A,#60          MOV   A,B
ADD   A,B            ADD   A,#60
MOV   B,A            MOV   B,A
```

۲) پاسخ عمل جمع فقط می تواند در ثبات A ذخیره شود.

۳) عملوندهای جمع که به صورت عدد ثابت باشند، می توانند در هر دستگاهی بیان شوند.

یک نکته مهم نیز راجع به دستور جمع وجود دارد و آن احتمال سرریز[1] است.
می دانیم بزرگترین عددی که می توان در یک ثبات ۸ بیتی ذخیره کرد، عدد ۲۵۵ است.
مجموعه دستورات

```
MOV   A,#100
ADD   A,#200
```

باعث می شوند حاصل جمع، عددی بزرگتر از ۲۵۵ شود که در ثبات ۸ بیتی A جا نمی گیرد. بنابراین آنچه پس از این دستورات در A ذخیره شده، یک نتیجه نادرست است. اصطلاحاً گفته می شود که در عمل جمع **سرریز** اتفاق افتاده است.
مشکل سرریز یکی از مشکلات مهم برنامه نویسی است که باید مراقب آن باشیم[2]؛ به خصوص اینکه اسمبلر متوجه این خطا نمی شود و تنها از نادرست بودن نتایج برنامه، وجود آن مشخص می شود.

اکنون با فراگرفتن دستورات MOV و ADD آماده ایم تا اولین مثال طراحی سیستم را بررسی کنیم.

مثال طراحی سیستم ۱) می خواهیم یک سیستم کامل با استفاده از ۸۰۵۱ طراحی کنیم که دو عدد ۸ بیتی را بخواند، با هم جمع کند، حاصل را با عدد ۱۰ جمع کند و نتیجه را نمایش دهد.

طراحی یک سیستم مبتنی بر میکروکنترلر را در فصل ۲ بررسی کردیم. به طور کلی سه مرحله مهم در طراحی سیستم با میکروکنترلر ۸۰۵۱ وجود دارد :

---

[1] ***Overflow***

[2] خواهید دید که دو بیت ثبات پرچم (PSW) به نامهای CY و OV برای اعلان سرریز به کار می روند. پس از یک عمل جمع با بررسی این دو بیت می توان به وقوع سرریز پی برد.

۱) طراحی راه حل : در این مرحله با توجه به صورت مسأله و امکانات، یک راه حل کامل برای حل مسأله شامل سخت افزار و نرم افزار (برنامه) طراحی می شود.

۲) طراحی سخت افزار : در این مرحله پس از بستن مدار پایه ۸۰۵۱ و با توجه به راه حل طراحی شده، سخت افزارهای لازم را به مدار اضافه می کنیم.

۳) طراحی نرم افزار و برنامه ریزی میکروکنترلر : در این مرحله برنامه سیستم بر اساس راه حل طراحی شده و نحوه اتصال سخت افزارها، نوشته شده و به کد ماشین ۸۰۵۱ ترجمه می شود و در نهایت در حافظه ROM میکروکنترلر ذخیره خواهد شد.

برای طراحی سیستم های مفصل و پیچیده، حتماً قبل از اتصال سخت افزار و نوشتن برنامه، ابتدا باید طراحی راه حل انجام شود تا تمام نکات طراحی قبل از پیاده سازی در نظر گرفته شود و خطاهای نهایی سیستم کاهش یابد. در این مورد در فصل ۱۱ سخن خواهیم گفت.
فعلاً در سیستمهای ساده ای که بررسی می کنیم، طراحی راه حل چندان پیچیده نیست، به همین دلیل برای روانتر شدن بحث، فقط دو مرحله طراحی سخت افزار و برنامه نویسی را در نظر می گیریم.
این دو مرحله را در مورد سیستمی که بعنوان مثال مطرح کردیم انجام می دهیم.

طراحی سخت افزار

همانطور که گفته شد، برای طراحی سخت افزاری هر سیستمی ابتدا باید مدار پایه ۸۰۵۱ (شکل صفحه ۵۴) بسته شود و سپس برحسب نیازهای سیستم، سخت افزارهای لازم را به مدار پایه اضافه کنیم.
نیازهای این سیستم چیست ؟

- دو عدد ۸ بیتی باید وارد میکروکنترلر شوند.
- یک عدد ۸ بیتی بعنوان نتیجه باید نمایش داده شود.

طبیعتاً باید برای ورود اطلاعات به میکروکنترلر و خروج اطلاعات از آن از پورتهای ورودی/خروجی استفاده کنیم. برای این کار از چهار پورت ورودی/خروجی میکروکنترلر، دو پورت را بعنوان ورودی و یک پورت را بعنوان خروجی انتخاب می کنیم. این انتخاب کاملاً اختیاری است. در این سیستم پورتهای P0 و P2 را بعنوان ورودی و پورت P1 را بعنوان خروجی میکروکنترلر در نظر می گیریم. بنابراین برنامه ما باید محتویات پورتهای ورودی P0 و P2 را بخواند و حاصل P0 + P2 + 10 را به پورت خروجی P1 ارسال کند.

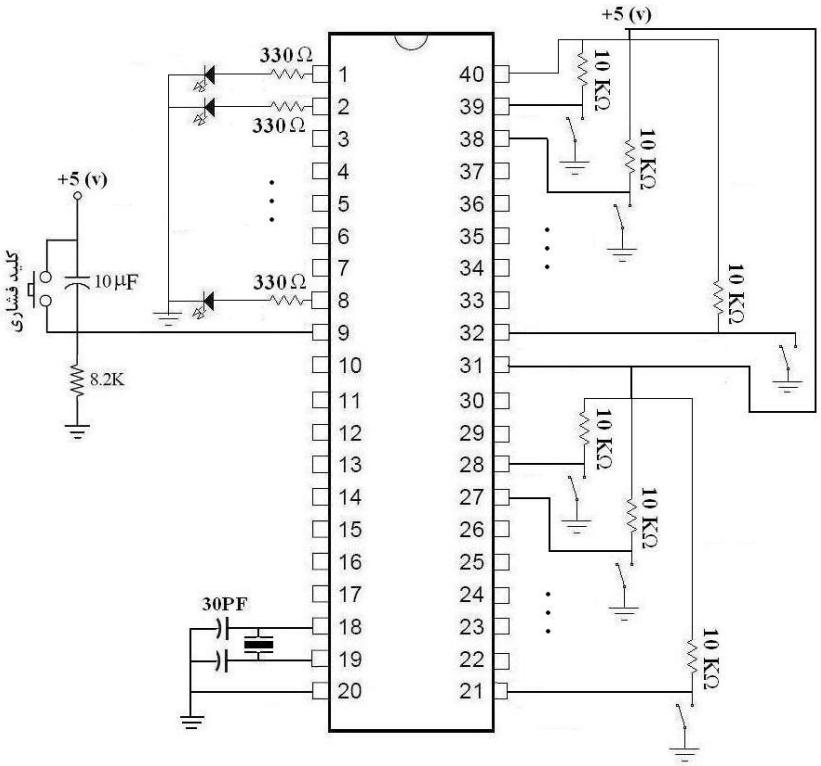
به صورت خلاصه:

P1←— P0 + P2 + 10

دو عدد ۸ بیتی ورودی می توانند به هر صورتی ایجاد شوند یا حتی ممکن است خروجی بخش دیگری از سیستم باشند. در اینجا برای ایجاد یک عدد ۸ بیتی، از ساختار شکل صفحه ۴۷ استفاده می کنیم. باز و بسته بودن کلیدها، عدد مورد نظر را به صورت دودویی مشخص می کنند. چون دو عدد ۸ بیتی باید خوانده شود، به دو مجموعه ۸ تایی کلید دوحالته (یا یک مقاومت آرایه‌ای ۱۰ کیلواهم و یک DIP SWITCH هشت‌تایی) نیاز داریم که یکی از این مجموعه کلیدها را به پورت P0 و دیگری را به پورت P2 متصل می کنیم.

نتیجه را نیز می توان به سادگی با مشاهده ولتاژ پینهای پورت خروجی P1 مشاهده کرد (ولتاژ حدود صفر به معنای "صفر منطقی" و ولتاژ حدود ۵ ولت به معنای "یک منطقی" است).

در اینجا برای نمایش نتیجه (که یک عدد ۸ بیتی است) از ساختار شکل صفحه ۴۸ استفاده می کنیم. خاموش و روشن بودن LED ها نتیجه عملیات را به صورت دودویی بیان می کنند. برای نمایش نتیجه که یک عدد ۸ بیتی است، ۸ عدد LED را به همراه مقاومت‌های لازم به پورت P1 متصل می کنیم.



سخت افزار این سیستم به شکل مقابل است:

از این به بعد هرگاه اشاره به خواندن یا نمایش عدد داشتیم، منظور ساختار شکل بالا است. البته این کار به منظور ساده تر شدن مثالها انجام می شود و هیچ الزامی نیست که اعداد حتماً توسط کلیدهای دوحالته ایجاد شوند یا روی LED ها نمایش داده شوند.

باز تأکید می کنیم پورتهای ورودی/خروجی می توانند به دلخواه تغییر کنند؛ مثلاً می توان پورتهای P0 و P1 را به عنوان ورودی و پورت P2 را به عنوان خروجی در نظر گرفت؛ در این صورت کلیدها به پورتهای P0 و P1 و LED ها به پورت P2 متصل می شوند.

طراحی نرم افزار (برنامه نویسی)

پس از طراحی سخت افزار نوبت به نوشتن برنامه سیستم می رسد. این برنامه باید محتویات پورتهای P0 و P2 (یعنی اعداد دودویی ۸ بیتی که با باز و بسته شدن کلیدهای دوحالته ایجاد شده اند) را بخواند و حاصل P0 + P2 + 10 را به پورت P1 ارسال کند (که روی LED ها نمایش داده می شود).

برنامه سیستم را در زیر می بینید :

```
ORG  0

MOV  P0,#0FFH
MOV  P2,#255

MOV  A,P0
ADD  A,P2
ADD  A,#10
MOV  P1,A

HERE :
SJMP  HERE

END
```

این برنامه قالب کلی برنامه های ۸۰۵۱ را نشان می دهد.

عباراتی که در یک برنامه اسمبلی نوشته می‌شوند دو دسته‌اند:

۱) دستورات (Instructions) که فرمانهایی هستند که باید اجرا شوند و اسمبلر آنها را به کد ماشین تبدیل می‌کند. جدول صفحه بعد، مجموعه دستورات اسمبلی میکروکنترلر ۸۰۵۱ را نشان می‌دهد:

۲) رهنمودها (Directive) که دستور اجرایی نیستند و تنها برای راهنمایی اسمبلر به کار می‌روند. کلمات ORG و END و EQU از جمله این رهنمودها هستند.

در زیر شرح برنامه بالا را بررسی می‌کنیم.

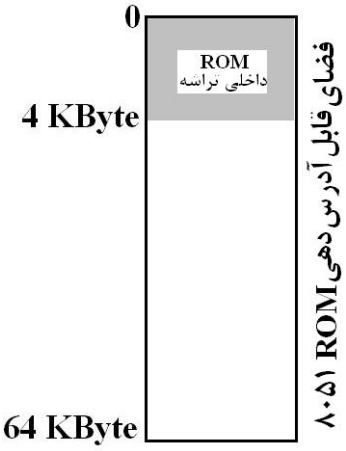
ACALL	Absolute call	ADD	Add	ADDC	Add with carry
AJMP	Absolute jump	ANL	Logical and	CJNE	Compare & jump if not equal
CLR	Clear	CPL	Complement	DA	Decimal adjust
DEC	Decrement	DIV	Divide	DJNZ	Decrement & jump if not zero
INC	Increment	JB	Jump if bit set	JBC	Jump & clear bit if bit set
JC	Jump if carry set	JMP	Jump	JNB	Jump if bit not set
JNC	Jump if carry not set	JNZ	Jump if accum. not zero	JZ	Jump if accumulator zero
LCALL	Long call	LJMP	Long jump	MOV	Move
MOVC	Move code	MOVX	Move external	MUL	Multiply
NOP	No operation	ORL	Inclusive or	POP	Pop stack
PUSH	Push stack	RET	Return	RETI	Return from interrupt
RL	Rotate left	RLC	Rotate left thru carry	RR	Rotate right
RRC	Rotate right thru carry	SETB	Set bit	SJMP	Short jump
SUBB	Subtract with borrow	SWAP	Swap nibbles	XCH	Exchange bytes
XCHD	Exchange digits	XRL	Exclusive or	CALL	Generic call

ORG 0 [1]

در ۸۰۵۱ دو ثبات ۸ بیتی به نامهای PCL و PCH وجود دارند که در کنار هم یک ثبات ۱۶ بیتی به نام PC که همان ثبات شمارنده برنامه است را تشکیل می دهند.

همانطور که قبلاً گفته شد، مقدار ثبات شمارنده برنامه در هر لحظه برابر شماره دستوری است که باید اجرا شود.

هنگامی که ۸۰۵۱ روشن (یا Reset) می شود، مقدار PC آن برابر صفر می شود[2]؛ *یعنی اولین دستوری که ۸۰۵۱ اجرا می کند، دستور شماره صفر است که در خانه شماره صفر حافظه ROM ذخیره شده است*. به بیان دیگر ۸۰۵۱ برای واکشی و اجرای اولین دستور، به سراغ خانه شماره صفر حافظه ROM می رود. بنابراین برنامه ای که می نویسیم باید با شروع از خانه شماره صفر حافظه ROM ذخیره شود.



ORG 0 به اسمبلر اعلام می کند که دستورات برنامه را به نحوی به زبان ماشین ترجمه کند که دستگاه برنامه ریز میکروکنترلر آنها را از خانه شماره صفر حافظه ROM میکروکنترلر به بعد ذخیره کند. اگر به جای ORG 0 مثلاً از ORG 50 استفاده کنیم، هنگام برنامه ریزی میکروکنترلر،

---

[1] ORG مخفف کلمه **Origin** به معنای "خاستگاه" و "سرچشمه" است.

[2] به اصطلاح گفته می شود : ***PC Wake-up Value = 0***.

برنامه از خانه شماره ۵۰ به بعد حافظه ROM ذخیره می شود.

چون PC در ۸۰۵۱ در ابتدای روشن (یا Reset) شدن سیستم برابر صفر می شود، همیشه اولین خط برنامه های اسمبلی ۸۰۵۱، 0 ORG است.

پرسش) الف) گفتیم که ثبات PC شماره دستوری که قرار است اجرا شود را در خود نگه می دارد. با توجه به ۱۶ بیتی بودن PC در ۸۰۵۱، نشان دهید شماره بزرگترین دستور در ۸۰۵۱ برابر ۶۵۵۳۵ (FFFF H) است.

ب) با توجه به پاسخ قسمت الف، نشان دهید که حداکثر حجم برنامه یک میکروکنترلر ۸۰۵۱ می تواند ۶۴ کیلوبایت (۶۵۵۳۶ بایت) باشد.

ج) با توجه به مقدار PC Wake-up Value در ۸۰۵۱ و پاسخ قسمت ب، چرا ۴ کیلوبایت ROM داخلی ۸۰۵۱، در ابتدای فضای ۶۴ کیلوبایتی (یعنی از خانه صفر به بعد) قرار داده می شود ؟

چون 0 ORG کاری انجام نمی دهد و تنها آغاز محل ذخیره برنامه را به اسمبلر و دستگاه برنامه ریز اعلام می کند، به جای دستور به آن "رهنمود[1]" گفته می شود.

دستورات برنامه

تعریف پورتهای ورودی

دو دستور اول برنامه، عدد شانزده‌تایی FF و عدد دهدهی255 (که هر دو برابر 11111111 دودویی هستند) را به پورتهای P0 و P2 ارسال می کنند.
در ۸۰۵۱ به دلایل فنی که در انتهای این فصل خواهد آمد، قبل از خواندن هر پورت ورودی، باید به بیتهای آن پورت "یک" ارسال کنیم؛ در اینجا چون می خواهیم ۸ بیت پورت P0 را بخوانیم، قبل از آن به هر ۸ بیت، "یک" ارسال می کنیم که معادل نوشتن عدد ۸ بیتی 11111111 دودویی (FF ۱۶تایی) در پورت P0 است و دستور اول برنامه این کار را انجام می دهد. دستور دوم نیز به همین نحو پورت P2 را به عنوان ورودی تعریف می‌کند.
توجه کنید که اعداد FF ۱۶تایی و 255 دهدهی معادلند و از هر دو برای تعریف یک پورت به عنوان ورودی استفاده می‌شود.

[1] ***Directive***

بدنه اصلی برنامه

چهار دستور بعدی بدنه اصلی برنامه را تشکیل می دهند :

- دستور اول (MOV A,P0) مقدار پورت P0 را در ثبات A کپی می کند.
- دستور دوم (ADD A,P2) مقدار فعلی ثبات A (که برابر مقدار P0 است) را با مقدار پورت P2 جمع کرده و نتیجه را در ثبات A ذخیره می کند.
- دستور سوم (ADD A,#10) مقدار ثبات A (که برابر P0 + P2 است) را با عدد ۱۰ جمع کرده و نتیجه را مجدداً در ثبات A ذخیره می کند.
- دستور چهارم (MOV P1,A) نتیجه اعمال فوق (که در ثبات A قرار دارد) را در پورت P2 می نویسد.

توجه کنید به دلیل ساختار ویژه پینهای پورتهای ورودی/خروجی، هنگامی که میکروکنترلر عددی را در یک پورت می نویسد تا زمانی که عدد جدیدی در پورت نوشته نشده است، عدد قبلی در پورت باقی می ماند و به اصطلاح **قفل**[1] می شود.

پرسش) با توجه به ساختار دستور ADD، توضیح دهید که چرا به جای دو دستور

```
MOV   A,P0
ADD   A,P2
```

به طور ساده از دستور ADD P0,P2 استفاده نمی کنیم ؟

دستور قطع برنامه

نامگذاری یک نقطه از برنامه: خط بعدی (HERE :) باعث می شود که نام HERE به این نقطه از برنامه نسبت داده شود.

نامگذاری یک نقطه از برنامه معمولاً برای استفاده در دستورات پرش انجام می‌شود. برای نامگذاری یک نقطه از برنامه، یک نام انتخاب کرده و با علامت " : " در آن نقطه برنامه می‌نویسیم. انتخاب این نام که اصطلاحاً برچسب یا Label نامیده می شود، کاملاً دلخواه است و می تواند شامل تمام حروف و ارقام و حتی نشانه هایی مانند؟ ، . ، @ ، _ ، $ و * باشد.

البته در انتخاب نام چند محدودیت وجود دارد که باید آنها را رعایت کنید :

۱) اولین کاراکتر نام انتخاب شده نباید رقم باشد.

۲) از نامهایی مانند A، B، R0، MOV و ... که توسط اسمبلر ۸۰۵۱ رزرو شده نباید استفاده کنید.

---

[1] *Latch*

۳) اگر خواستید چند نقطه از برنامه را نامگذاری کنید، نباید از اسمهای تکراری استفاده کنید.

۴) اسامی می‌توانند تا ۲۵۵ کاراکتر طول داشته باشند؛ اما اسمبلر تنها ۳۱ کاراکتر اول را در نظر می‌گیرد.

دستور SJMP HERE به میکروکنترلر می گوید که به نقطه ای از برنامه به نام HERE (که همان خط بالایی است !) "بپرد" و ادامه برنامه را از آنجا دنبال کند[1]. بنابراین ۸۰۵۱ به خط بالایی می پرد و برنامه را از آنجا ادامه می دهد. ادامه برنامه دستور SJMP HERE است که باعث می شود مجدداً به خط بالایی پرش انجام شود و دستور SJMP HERE دوباره اجرا شود و ....

می بینید که با نوشتن برنامه به این صورت، میکروکنترلر در این دو خط به دام می افتد. چرا این کار انجام شده است؟

میکروکنترلر۸۰۵۱ پس از روشن (یا Reset) شدن شروع به اجرای خط به خط برنامه می کند و با اجرای هر دستور، دستور بعدی را به طور خودکار از حافظه واکشی و اجرا می کند. آخرین دستور برنامه که می خواهیم اجرا شود، دستور MOV P1,A است؛ اما میکروکنترلر۸۰۵۱ نمی‌داند که این دستور، آخرین دستور برنامه است؛ بنابراین پس از اجرای دستور فوق باز هم به سراغ حافظه می رود تا دستورات بعدی (که در این برنامه وجود ندارند) را دریافت و اجرا کند ! اگر فکری برای این موضوع نکنیم، میکروکنترلر۸۰۵۱ همینطور ادامه حافظه ROM را (که برنامه ما نیست و خالی یا محتوی داده های ناشناخته است) را می‌خواند و بعنوان دستور اجرا می کند !

برای دوری از این مشکل، با استفاده از دو خط

```
HERE :
SJMP   HERE
```

(که البته در یک خط به صورت HERE : SJMP HERE نیز می تواند نوشته شود) میکروکنترلر را در همین نقطه برنامه مشغول می کنیم (به دام می اندازیم) که تا وقتی سیستم خاموش نشده در همین نقطه باقی بماند و پیش نرود.

به جای این دو خط می توان از دستور SJMP $ نیز استفاده کرد که معنای مشابه دارد؛ یعنی این دستور سبب می شود که میکروکنترلر در مکان اجرای دستور توقف کند.

تنها با Reset کردن سیستم، میکروکنترلر از این توقف دایمی رهایی می‌یابد.

---

[1] دستور **SJMP** عمل پرش (**JUMP**) به نقطه ای از برنامه که بعنوان مقصد برای آن مشخص می شود را انجام می دهد.

خط END

خط انتهای برنامه، **رهنمود** END است که به اسمبلر اعلام می کند دستورات برنامه ما پایان یافته است. همیشه آخرین خط برنامه های اسمبلی ۸۰۵۱، END است.
دقت داشته باشید که END به معنای قطع اجرای برنامه نیست و برای این کار حتماً باید در انتهای برنامه میکروکنترلر را به دام بیندازیم.

پرسش) اگر P0 = 00110011 و P2 = 00001110 ، چه عددی به پورت P1 ارسال می شود ؟

پرسش) اگر P0 = 11001101 و P2 = 10100001 نشان دهید عدد 01111000 به پورت P1 ارسال می شود. آیا این عدد برابر P0 + P2 + 10 است ؟ مشکل از کجاست ؟ نشان دهید حداکثر مقدار P0 + P2 می تواند ۲۴۵ باشد.

پرسش) اگر از مجموعه کلیدهای متصل به پورت P0، ۴ کلید متصل به بیتهای پرارزش P0 بسته و بقیه باز باشند و از مجموعه کلیدهای متصل به پورت P2، همه به جز کلید متصل به P2.1 بسته باشند، کدام LED های متصل به P1 روشن و کدام خاموش می شوند ؟

پرسش) برنامه نوشته شده فقط یک بار اجرا می شود؛ یعنی هنگام روشن شدن، میکروکنترلر عمل P1 ⟵ P0 + P2 + 10 را انجام داده و سپس در نقطه HERE به دام می افتد. بنابراین اگر بعد از روشن شدن میکروکنترلر مقادیر کلیدهای متصل به پورتهای ورودی P0 و P2 تغییر کنند، آنچه در LED های متصل به پورت خروجی P1 نمایش داده می شود، تغییری نمی کند؛ چون میکروکنترلر در نقطه پایانی برنامه به دام افتاده و دیگر پورتهای ورودی P0 و P2 را نمی خواند تا متوجه تغییر آنها شود؛ بنابراین همانطور که قبلاً گفتیم، آخرین عددی که میکروکنترلر در پورت P1 نوشته است، در پورت قفل می شود و باقی می ماند. تنها راه توجه میکروکنترلر به تغییرات پورتهای ورودی، Reset کردن آن است.

اگر بخواهیم میکروکنترلر مرتباً مقدار پورتهای ورودی را بخواند و نتیجه محاسبه P0 + P2 + 10 را روی پورت P1 نشان دهد، باید کاری کنیم که میکروکنترلر پس از اجرای آخرین دستور برنامه اصلی (MOV P1,A) به جای به دام افتادن، به ابتدای برنامه برگردد و دوباره آن را اجرا کند.

پرسش) نشان دهید که برنامه زیر این کار را انجام می دهد.

```
ORG  0

MOV  P0,#255
MOV  P2,#255

HERE :

MOV  A,P0
ADD   A,P2
ADD   A,#10
MOV  P1,A

SJMP  HERE

END
```

چند نکته راجع به زبان اسمبلی ۸۰۵۱

۱) زبان اسمبلی به بزرگی و کوچکی حروف به کار رفته در برنامه حساس نیست و می توانید برنامه را به دلخواه خود با حروف بزرگ یا کوچک بنویسید؛ مثلاً دستورات MOV P0,#255 و mov p0,#255 تفاوتی ندارند.

۲) چون برنامه های زبان اسمبلی، معمولاً به صورت کلی چندان قابل فهم نیستند، ممکن است در مراجعات بعدی به برنامه هایی که قبلاً نوشته ایم، عملکرد بعضی دستورات یا نقش قسمتی از برنامه نامفهوم شده باشد.
برای دوری از این مشکل، خوب است که در نقاط مناسب برنامه توضیحاتی بنویسیم تا در مراجعات بعدی راهنمای ما باشد.

در برنامه های اسمبلی برای نوشتن توضیحات از علامت ";" استفاده می کنیم؛ در هر نقطه‌ای از برنامه که این علامت گذاشته شود، تا انتهای خط حاوی این علامت، هرچه نوشته شده باشد توضیحات برنامه در نظر گرفته می شود و اسمبلر به آن توجهی نمی کند.
مثلاً دستور MOV P0,#255 را به صورت

```
MOV  P0,#255 ; Define P0 as input port.
```

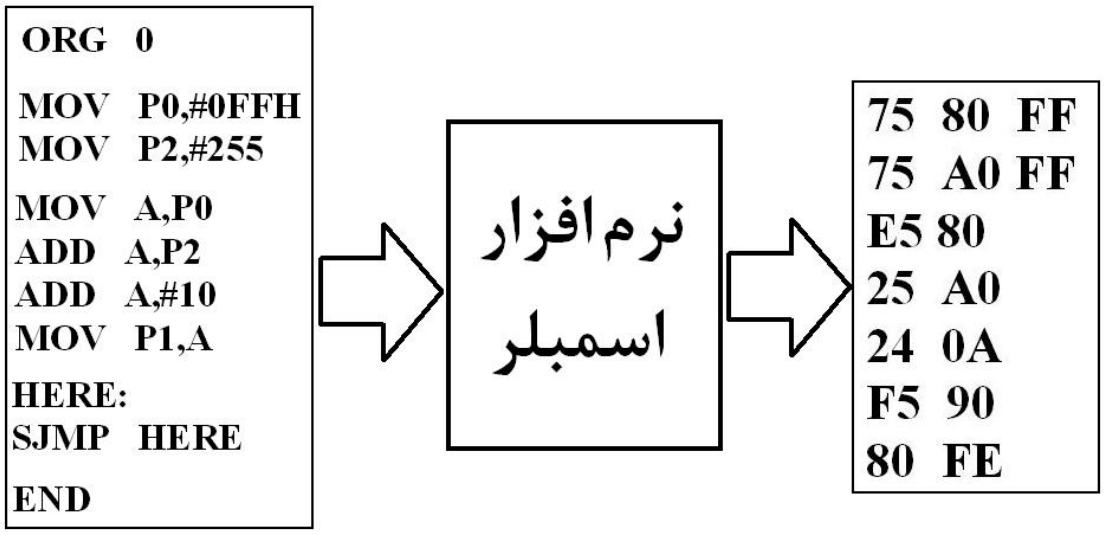
اصلاح می کنیم تا در مراجعات بعدی به برنامه مشخص باشد که منظور از نوشتن عدد 255 در پورت P0، تعریف آن به صورت ورودی بوده است.

یک عادت خوب این است که در ابتدای برنامه نام نویسنده، تاریخ نوشتن برنامه و عملکرد کلی برنامه نوشته شود. مثلاً در ابتدای برنامه قبلی می نویسیم :

```
; ///////////////////////////////////
; Author : J-Rasti
; Date : 1383 / 11 /29
; P1 <- P0 + P2 + 10
; ///////////////////////////////////
```

روند تبدیل برنامه به کد ماشین

برنامه مثال ۱ به زبان اسمبلی نوشته شده است. میکروکنترلر ۸۰۵۱ تنها کدهای ماشین ۸۰۵۱ را می فهمد؛ بنابراین برنامه اسمبلی فوق باید توسط نرم افزار اسمبلر به کدهای ماشین ۸۰۵۱ تبدیل شود :



نرم افزار اسمبلر ۸۰۵۱ شامل دو فایل اجرایی به نامهای asm51.exe و oh.exe است. فرض کنید این دو فایل در پوشه ای به آدرس C:\ASM ذخیره شده اند (این آدرس دلخواه است).

برنامه‌نویسی میکروکنترلر ۸۰۵۱

مراحل نوشتن یک برنامه اسمبلی ۸۰۵۱، ترجمه آن به کد ماشین و ذخیره آن در حافظه ROM داخلی میکروکنترلر به شرح زیر است :

۱) نوشتن و ذخیره فایل :

ابتدا برنامه خود را در با استفاده از یک ویرایشگر معمولی مانند ویرایشگر DOS (که با اجرای دستور edit در محیط DOS یا محیط command prompt ویندوز وارد آن

می‌شویم) یا Notepad ویندوز در یک فایل متنی بنویسید. بعضی ویرایشگرها مانند Microsoft Word یا Wordpad کاراکترهای خاصی برای قالب بندی وارد متن می کنند که برای اسمبلر ناشناخته هستند. از این نرم افزارها نمی توان برای نوشتن برنامه های اسمبلی استفاده کرد.

پس از نوشتن برنامه اسمبلی، فایل متنی مزبور را با یک نام دلخواه و پسوند asm در همان پوشه‌ای که فایلهای اسمبلر را در آن قرار داده‌اید (مثلاً پوشه C:\ASM) ذخیره کنید. مثلاً برنامه قبلی را به نام p1.asm در پوشه فوق ذخیره می کنیم.

۲) فراخوانی اسمبلر :

مرحله اول فراخوانی اسمبلر، استفاده از فایل اجرایی asm51.exe است.

یک روش، استفاده از محیط DOS یا command prompt ویندوز است. در محیط های ذکر شده، ابتدا وارد پوشه مورد نظر (مثلاً C:\ASM) می‌شویم و دستور زیر را وارد و اجرا می‌کنیم :

```
C:\ASM\asm51.exe  p1.asm
```

به جای این کار می توانید در محیط ویندوز و در پوشه C:\ASM، با استفاده از خاصیت Drag & Drop فایل p1.asm را با ماوس "کشیده" و روی فایل asm51.exe قرار دهید.

با فراخوانی asm51.exe، اسمبلر ۸۰۵۱ شروع به ترجمه ابتدایی برنامه می کند. نتیجه این مرحله تولید دو فایل به نامهای p1.lst و p1.obj است که در همان پوشه C:\ASM قرار می گیرند.

بخشی از محتویات فایل p1.lst که به آن فایل لیست[1] گفته می شود، شکل روبرو دیده می‌شود.

آنچه در ستون OBJ می بینید، معادل زبان ماشین دستورات است که به آنها کد عمل یا OP-Code گفته می شود. همانطور که

```
DOS 5.0 (038-N) MCS-51 MACRO ASSEMBLER,
OBJECT MODULE PLACED IN P1.OBJ

LOC    OBJ     LINE   SOURCE

0000            1      ORG  0
                2
0000   7580FF    3      MOV  P0,#0FFH
0003   75A0FF    4      MOV  P2,#255
                5
0006   E580      6      MOV  A,P0
0008   25A0      7      ADD  A,P2
000A   240A      8      ADD  A,#10
000C   F590      9      MOV  P1,A
                10
                11     HERE :
000E   80FE      12     SJMP HERE
                13
                14     END
  . . .
ASSEMBLY COMPLETE, NO ERRORS FOUND
```

[1] *List*

مـی‌بینیـد، رهنمودهـای ORG 0 و END دارای کـد عمـل نیـستند (چـون دسـتور نیستند).

ستون LOC نشانگر شماره خانه حافظه ROM است که کد عمل مقابلش در آن خانه ذخیره می شود. توجه کنید که استفاده از ORG 0 باعث شده که دستورات با شروع از صفر شماره گذاری شوند. دستگاه برنامه ریز با توجه به این شماره گذاری، کدهای عمل را با شروع از خانه صفر حافظه ROM ذخیره می کند.

یافتن خطاهای ترجمه برنامه به کمک فایل لیست

به آخرین خط فایل لیست توجه کنید :

ASSEMBLY COMPLETE, NO ERRORS FOUND

این پیغام که پس از فراخوانی asm51.exe روی صفحه نمایش نیز ظاهر می شود، نشان می دهد که برنامه ما *از نظر املایی و قواعد زبان اسمبلی* درست است و اسمبلر در ترجمه ابتـدایی برنامه موفق شده است. در این حالت، فایل p1.obj (که فایل مقصد[1] نامیده می شود) حـاوی کدهای عمل صحیح است که می توان آن را در حافظه ROM میکروکنترلر ذخیره کرد.

اگر اشتباهی از نظر املایی (مثلاً نوشتن دستور MOV به صورت MOVE) یـا قواعـد زبـان اسمبلی (مثلاً استفاده از دستور ADD P0,P1) در برنامه وجود داشته باشد، اسـمبلر خطـا می گیرد و محل وقوع و نوع خطاها را در فایل لیست مشخص می کند.
برای مثال فایل p1.asm را با چند خطای عمدی بازنویسی می کنیم :

```
ORG 0

MOV P0,# FFH ; Error – Number should start with a digit

MOV P2,#255

MOVE A,P0 ; Error – Syntax Error
ADD P2,A ; Error – P2 can not be destination of ADD
ADD A,#10
MOV P1,A

HERE :
SJMP HERE
```

[1] ***Object***

END

با فراخوانی asm51.exe پیغام زیر نمایش داده می شود :

ASSEMBLY COMPLETE, 2 ERRORS FOUND

در فایل لیست، خطاها زیر خطوط اشتباه تذکر داده شده‌اند.

پس از اصلاح خطاها در **فایل اسمبلی**[1]، مجدداً asm51.exe را فراخوانی می کنیم تا در نهایت پیغام عدم وجود خطا را مشاهده کنید.

توجه کنید که اسمبلر فقط متوجه خطاهای املایی و خطاهای مربوط به قواعد زبان اسمبلی می شود. اگر در برنامه خطای منطقی داشته باشید (مثلاً فراموش کردن علامت # قبل از عدد ثابت یا وقوع سرریز در عمل جمع) اسمبلر به شما پیغامی نمی دهد و تنها نتایج نادرست برنامه شما را متوجه وقوع خطا خواهد کرد. این خطاها بدترین خطاهای برنامه نویسی هستند که باید مراقب آنها باشید.

۳) تبدیل به کد ماشین ۸۰۵۱ :

فایل اجرایی oh.exe کار تبدیل فایل مقصد که در مرحله پیش تولید شده به کد ماشین ۸۰۵۱ را بر عهده دارد.

برای این کار دستور زیر را در محیط DOS یا command prompt ویندوز اجرا می کنیم :

```
C:\ ASM\oh.exe   p1.obj
```

یا می توان در محیط ویندوز با "کشیدن" فایل p1.obj با ماوس و قرار دادن آن روی فایل oh.exe همین عمل را انجام داد.

نتیجه این کار، ایجاد فایلی به نام p1.hex است که حاوی کدهای ماشین برنامه است و می‌توان آن را در حافظه میکروکنترلر ذخیره و اجرا نمود. شکل زیر محتویات فایل P1.hex را نشان می‌دهد.

```
:0E0000007580FF75A0FFE580240AF59080FE54
:00000001FF
```

حجم کدهای ماشین این برنامه کمتر از ۳۰ بایت است؛ در AT89S51 برنامه‌هایی با اندازه حداکثر ۴ کیلوبایت می‌توانند در حافظه ROM داخلی میکروکنترلر ذخیره شوند. اگر حجم برنامه شما از این مقدار بیشتر باشد، باید از میکروکنترلرهایی با ROM داخلی بیشتر (مانند

[1] نکته مهم این است که خطاهای رخ داده باید در **فایل اسمبلی** اصلاح شوند نه در فایل لیست.

AT89S52 که هشت کیلوبایت ROM داخلی دارد) استفاده کنید یا به میکروکنترلر خود حافظه ROM خارجی متصل کنید.

شاید کنجکاو باشید بدانید اطلاعات داخل فایل HEX چیست؟ با کمی دقت می‌توانید کدهای زبان ماشین برنامه خود را در این فایل بیابید. اطلاعات دیگر داخل فایل، مربوط به ساختار استاندارد فایلهای HEX شرکت اینتل (که INHX8M نام دارد) می‌باشد.

۴) برنامه ریزی میکروکنترلر :

مرحله آخر ذخیره کدهای فایل p1.hex در حافظه ROM داخلی تراشه است[1]. دستگاههای برنامه ریز این کار را برای ما انجام می دهند.

دستگاه برنامه ریز باید با توجه به نوع میکروکنترلر مورد استفاده انتخاب شود؛ مثلاً برای برنامه ریزی تراشه AT89S51 باید از دستگاه برنامه ریزی که از این تراشه حمایت می کند، استفاده شود. این موضوع بسیار مهم است؛ چون دستگاهی که مثلاً تراشه AT89**C**51 را برنامه ریزی می کند ممکن است قادر به برنامه ریزی تراشه AT89**S**51 نباشد و بالعکس.

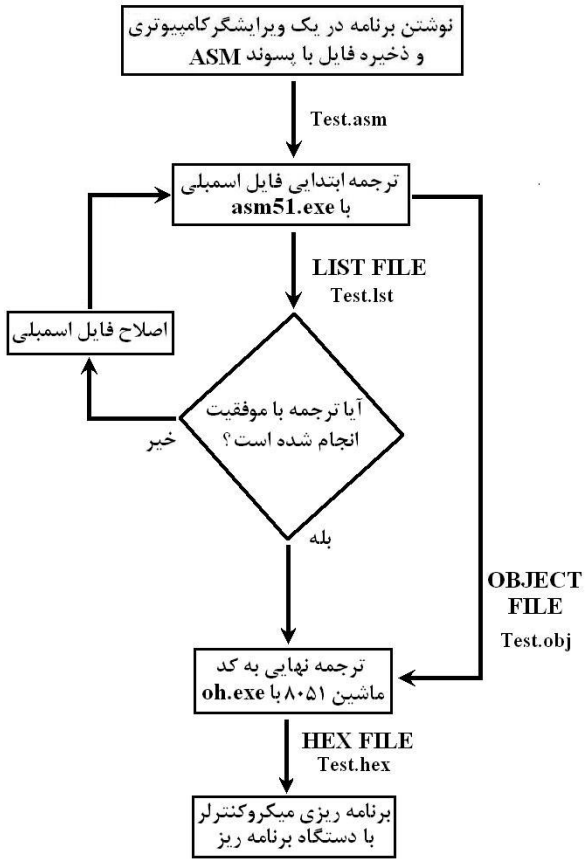
پس از قرار دادن تراشه در دستگاه برنامه ریز، ارتباط دستگاه با کامپیوتر را از طریق کابل مخصوص برقرار می کنیم. نرم افزار دستگاه برنامه ریز که در کامپیوتر نصب می شود، مسیر فایل حاوی کدهای ۸۰۵۱ (در اینجا فایل p1.hex) را از کاربر گرفته و از طریق کابل برای دستگاه برنامه ریز ارسال می کند. دستگاه نیز کدهای فوق را در حافظه ROM داخلی تراشه ذخیره می کند و پس از اتمام کار، یک بار از صحت ذخیره اطلاعات مطمئن می شود[2].

اکنون تراشه آماده کار است. شکل صفحه بعد به طور خلاصه مراحل فوق را در مورد فایلی به نام Test نشان می دهد. فایلهای asm51.exe و oh.exe معمولاً در نرم افزار دستگاههای برنامه ریز گنجانده شده اند.

---

[1] البته بعضی از انواع ۸۰۵۱ (مانند ۸۰۳۱) حافظه ROM داخلی ندارند و برنامه آنها باید در یک تراشه ROM خارجی ذخیره و به میکروکنترلر متصل شود.

[2] ***Verification***

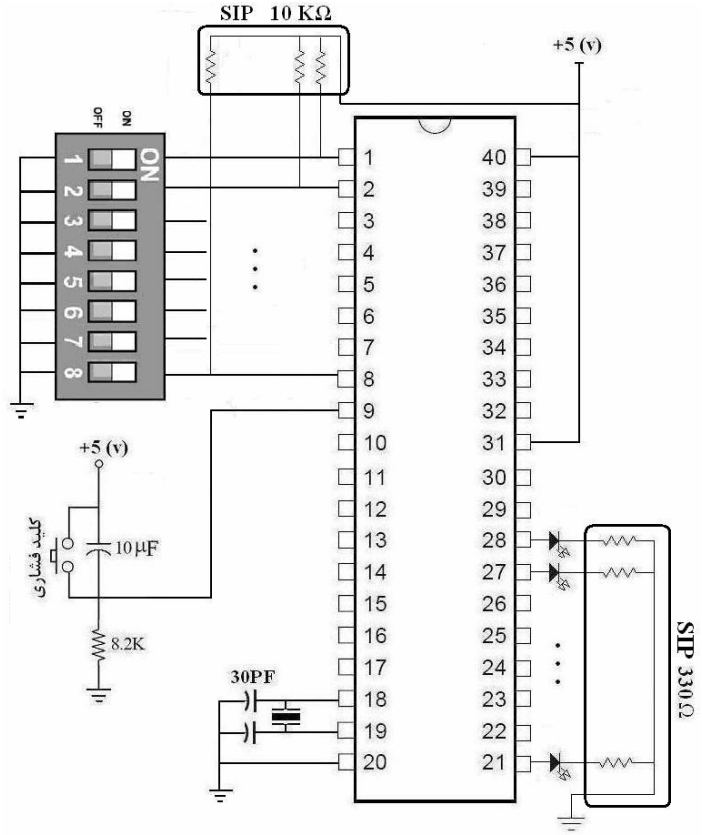


این مثال طراحی سیستم را برای شروع کار به طور کامل انجام دهید. سخت افزار آن را روی بورد ببندید و میکروکنترلر را پس از برنامه ریزی به مدار اضافه کنید و نتیجه اجرای برنامه به ازای مقادیر مختلف کلیدها را روی LED ها مشاهده کنید.

نرم‌افزارهای شبیه‌ساز مانند Prog-Studio، µVision، Franklin و Proteus تمام امکانات سخت‌افزار طراحی سیستم را به صورت مجازی داخل خود دارند. در این نرم‌افزارهای می‌توانید برنامه خود را بنویسید، به راحتی اسمبل کنید، فایل hex آن را برای ذخیره در حافظه ROM میکروکنترلر ایجاد کنید و از همه مهمتر اجرای برنامه‌ها را شبیه سازی کنید (مثلاً مقادیر پورتها را در حین اجرای خطبه‌خط برنامه ببینید).

مثال طراحی سیستم ۲) بعنوان مثالی دیگر می خواهیم سیستمی دیگر طراحی کنیم که **مرتباً** یک عدد ۸ بیتی را از کاربر گرفته و معکوس (NOT) آن را نمایش دهد.

در این مثال پورت P1 را بعنوان ورودی و پورت P2 را بعنوان خروجی انتخاب می کنیم. سخت افزار سیستم فوق را در شکل روبرو مشاهده می کنید.
می بینید که استفاده از مقاومتهای آرایه ای (SIP) و DIP Switch مدار را چقدر ساده تر کرده است.



برنامه سیستم فوق را در زیر مشاهده می کنید :

```
;////////////////////////////////
; Author : J-Rasti
; Date : 83/11/29
; P2 <- NOT(P1)
;////////////////////////////////

ORG   0

MOV   P1,#255 ; Define P1 as input port.

AGAIN :

MOV   A,P1     ; A ← P1
CPL   A        ; A ← NOT(A)
MOV   P2,A     ; Send result (A = NOT (P1)) to output.

SJMP   AGAIN ; Jump back to the start point.

END
```

دستور CPL A ، مقدار ثبات A را معکوس[1] می کند.

به یاد داشته باشید که هرگاه صورت مسأله ای ایجاب کند که کاری **مرتباً** انجام شود (مانند این مسأله)، *دستور انتهایی برنامه باید پرش به ابتدای برنامه باشد*. به اصطلاح می گوییم برنامه باید به صورت **حلقه بینهایت** نوشته شود (حلقه ای که بینهایت بار (قبل از خاموش شدن سیستم) اجرا می شود).
چون عمل ارسال عدد 255 به پورت ورودی کافی است یکبار اجرا شود، نقطه ابتدایی حلقه اصلی (AGAIN) بعد از این دستور تعریف شده تا در هر مرتبه اجرای برنامه، این دستور اجرا نشود.
اگر دستور انتهایی برنامه $ SJMP باشد، برنامه در همان نقطه متوقف می شود.

```
ORG   0
MOV   P1,#255
LOOP :
MOV   A,P1
CPL   A
ADD   A,#1
MOV   P0,A
SJMP  LOOP
END
```

پرسش) برنامه روبرو مرتباً P1 را خوانده و منفی آن را در پورت P0 می نویسد (منفی یک عدد دودویی، معکوس آن عدد به اضافه ۱ یا همان مکمل دو آن است).
دستور INC A[2] می تواند به جای ADD A,#1 به کار رود.
الف) سخت افزار سیستم فوق را طراحی کنید.
ب) برنامه را به نحوی تغییر دهید که تنها یکبار اجرا شود.
ج) اگر P1 = 10110001، ولتاژ هرکدام از پینهای P0 چقدر است؟

پرسش) مانند آنچه در مثالهای بالا انجام شد، سخت افزار و برنامه سیستم های زیر را طراحی کنید. پورتهای ورودی/خروجی ذکر شده پیشنهادی است و می توانید آنها را به دلخواه خود تغییر دهید :

الف) سیستمی که مقدار پورتهای P0 و P1 را خوانده و حاصل P1 – P0 را در پورت P2 بنویسد. توجه کنید که P1 – P0 = P1 + (-P0) .

ب) سیستمی که مقدار پورت P3 را بخواند و دو برابر آن را در P0 بنویسد.

---

[1] ***ComPLement***

[2] مخفف INCrease به معنای افزایش یک واحدی است.

مثال طراحی سیستم ۳) می خواهیم سیستمی طراحی کنیم که یک LED را به صورت چشمک زن در آورد. این سیستم، مثالی از سیستمهایی است که ورودی ندارند و تنها خروجی تولید می کنند. یک تابلو تبلیغاتی ساده که یک پیغام را به صورت متحرک روی مجموعه‌ای از LEDها نمایش می دهد، مثالی عملی از این سیستمها می باشد.

سخت افزار سیستم مورد نظر ما بسیار ساده است و فقط شامل یک LED است که باید به یکی از پینهای ورودی/خروجی اضافه شود که در این مثال آن را به پین P1.0 متصل می کنیم. برای تمرین سخت افزار این سیستم را رسم کنید.

برای نوشتن برنامه این سیستم باید به نکته ای توجه کنیم. در این مثال ما می خواهیم مرتباً بیت P1.0 را صفر و یک کنیم تا LED خاموش و روشن شود. برنامه زیر این کار را انجام می‌دهد:

```
ORG   0

AGAIN :

MOV   P1,#00000001 B
MOV   P1,#00000000 B

SJMP   AGAIN

END
```

همانطور که می بینید با این کار بیت P1.0 مرتباً صفر و یک می شود و LED متصل به آن روشن و خاموش خواهد شد.
اگر تنها کار سیستم ما همین باشد، برنامه فوق درست است؛ اما اگر بیتهای دیگر پورت P1 به ورودیها یا خروجیها دیگر متصل باشند و از آنها استفاده دیگری شود، این برنامه درست نیست؛ چون علاوه بر صفر و یک کردن بیت P1.0، بیتهای دیگر را نیز دستکاری می کند.
بنابراین به دستوراتی نیاز داریم که بتوانند روی بیت کار کنند، نه روی بایت (در اینجا به دستوراتی نیاز داریم که بتوانند **یک بیت** را مستقل از بیتهای دیگر صفر و یک کنند). دستورات فوق، **دستورات بیتی** نامیده می شوند که موضوع فصل ۷ کتاب هستند.

آشنایی با چند دستور بیتی

چند دستور بیتی مهم را که کاربردهای زیادی دارند را در اینجا معرفی می کنیم :

دستور "بیت" SETB

این دستور "بیت" ذکر شده را "یک" می کند.
مثلاً دستور SETB P1.0 ، بیت شماره صفر پورت P1 (P1.0) را بدون تغییر بیتهای دیگر این پورت، "یک" می کند (ولتاژ حدود (v) 5+ روی آن ظاهر می‌شود).

دستور "بیت" CLR

این دستور "بیت" ذکر شده را "صفر" می کند. مثلاً دستور CLR P3.4 ، بیت شماره ۴ پورت P3 (P3.4) را بدون تغییر بیتهای دیگر این پورت، "صفر" می کند (ولتاژ آن تقریباً صفر می‌شود).

دستور "بیت" CPL

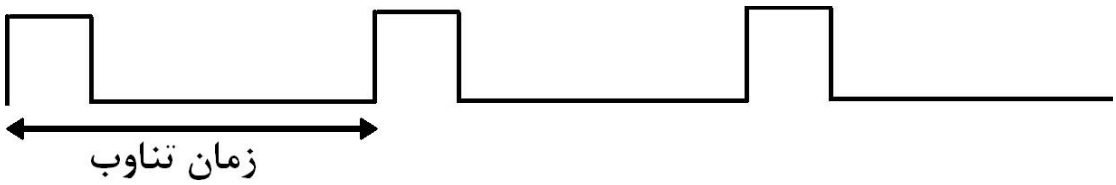
این دستور "بیت" ذکر شده را معکوس[1] (NOT) می کند (اگر "بیت" ذکر شده "یک" باشد آن را "صفر" و اگر "صفر" باشد آن را "یک" می کند). مثلاً دستور CPL P2.1 ، بیت شماره یک پورت P2 (P2.1) را بدون تغییر بیتهای دیگر این پورت، معکوس می کند.

حال برنامه قبلی را با دستورات SETB و CLR می نویسیم :

```
ORG   0

AGAIN :

SETB  P1.0
CLR   P1.0

SJMP  AGAIN

END
```

این برنامه بدون تغییر بیتهای دیگر پورت P1، بیت P1.0 را صفر و یک می کند.
اگر موج تولید شده روی پین P1.0 را با اسیلوسکوپ ببینید، یک موج مربعی به صورت زیر مشاهده خواهید کرد :

[1] ***Complement***



پرسش) اگر زمان تناوب موج مربعی فوق ۴ میکروثانیه باشد، نشان دهید زمان یک بار اجرای کامل حلقه اصلی برنامه نیز ۴ میکروثانیه است.

انتظار داریم اگر یک LED به P1.0 متصل شود، به صورت چشمک زن درآید؛ اما در عمل این اتفاق نمی افتد و LED را همیشه روشن می بینیم !

دلیل این موضوع آن است که اثر یک نقطه روشن تا ۴۰ میلی ثانیه در چشم باقی می ماند، حتی اگر خاموش شده باشد. مدت زمانی که در برنامه LED را روشن، خاموش و مجدداً روشن می کنیم، تنها چند میکروثانیه طول می کشد و در این فرصت کم، چشم اصلاً متوجه خاموش شدن LED نمی شود؛ بهمین خاطر LED همیشه روشن دیده می شود.

چاره کار چیست ؟ طبیعتاً باید به چشم فرصت داد تا خاموش و روشن شدن LED را ببیند؛ یعنی به نحوی بین دستورات SETB و CLR متوالی، تأخیر ایجاد کرد.

برنامه زیر را ببینید :

```
ORG   0

AGAIN :

SETB  P1.0
```

دستورات ایجاد تأخیر

```
CLR   P1.0
```

دستورات ایجاد تأخیر

```
SJMP AGAIN

END
```

دستورات ایجاد تأخیر، دستوراتی هستند که بدون اینکه تأثیری روی عملکرد برنامه داشته باشند، تنها میکروکنترلر را در نقطه ای از برنامه معطل می کنند. دستور NOP[1] معروفترین این دستورات است که هیچ کاری غیر از معطل کردن میکروکنترلر انجام نمی دهد ! با روش ایجاد و محاسبه زمان تأخیر در فصل بعد آشنا خواهید شد.

[1] ***No Operation***

پرسش) نشان دهید که برنامه زیر معادل برنامه قبل است.

```
ORG   0
AGAIN :
CPL    P1.0
```

دستورات ایجاد تأخیر

```
SJMP   AGAIN
END
```

توجه کنید که میکروکنترلر در هنگام روشن شدن به همه پورتها "یک" ارسال می کند؛ بنابراین اجرای متوالی دستور CPL P1.0 باعث خاموش و روشن شدن متوالی P1.0 و LED متصل به آن می‌شود.

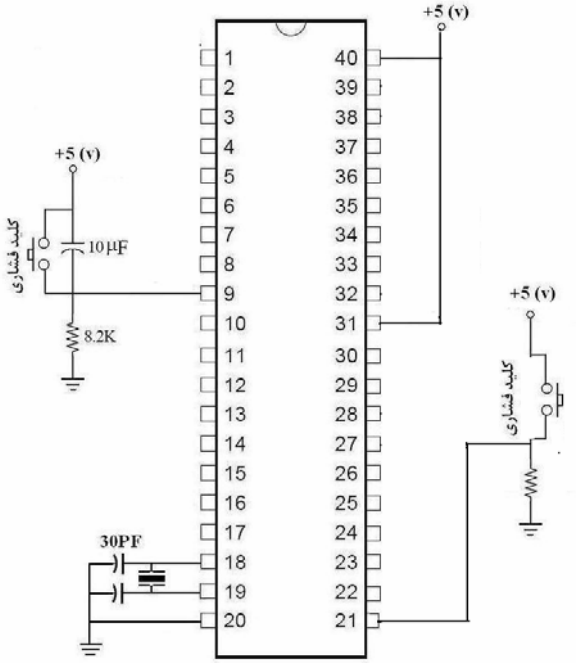
به یک نکته توجه کنید. در سیستم فوق به خاطر اینکه روشن و خاموش شدن یک نقطه **دیده شود**، عمداً بین روشن و خاموش شدن آن تأخیر ایجاد کردیم تا به چشم فرصت درک این تغییر را بدهیم.
در بعضی از سیستمهای نوری مانند تلویزیون، نمایشگر کامپیوتر، تابلوهای تبلیغاتی و ... عکس این موضوع اتفاق می افتد؛ یعنی در این سیستمها در هر لحظه فقط یک نقطه روشن است ! اما از لحظه ای که یک نقطه روشن و بعد خاموش می شود تا لحظه ای که دوباره روشن می شود، این قدر کم به طول می انجامد که چشم متوجه خاموش شدن آن نمی شود و تصور می کند که دائماً روشن است؛ مثلاً وقتی تصویر یک گل را در تلویزیون می بینید، تصور نکنید تمام نقاط این تصویر (که پیکسل نامیده می شوند) همیشه روشن هستند؛ بلکه در هر لحظه فقط یک نقطه روشن است و کنترل کننده نمایشگر با شروع از یک گوشه تصویر، هر نقطه را متناسب با تصویر مورد نظر روشن می کند و به سراغ نقطه بعدی می رود. این کار بسیار سریع انجام می شود[1]، آن قدر سریع که وقتی کنترل کننده به نقطه شروع باز می گردد و دوباره آن را روشن می کند، چشم هنوز متوجه خاموش شدن آن نشده است؛ به این ترتیب همه نقاط

[1] در تلویزیون ۵۰ بار در ثانیه و در نمایشگر کامپیوتر حدود دو برابر آن. اما برای اینکه یک LED همیشه روشن دیده شود (مثلاً در تابلوهای تبلیغاتی که متشکل از تعداد زیادی LED است) ۲۵ بار روشن شدن در ثانیه نیز کافی است.

تصویر همیشه روشن دیده می شوند. اگر این سرعت کم باشد، پرشهایی در تصویر دیده می‌شود.

یک مثال دیگر از این سیستمها، Refreshing 7-Segments است که در فصل ۶ با آن آشنا خواهیم شد.

مثال ۴ طراحی سیستم) در مثالهای ۱ و ۲ طراحی سیستم، اعداد ۸ بیتی به صورت ۸ کلید دوحالته توسط میکروکنترلر خوانده شده و پس از پردازش، نتیجه ۸ بیتی روی ۸ عدد LED نمایش داده می‌شود. در مثال ۳، سیستم تنها دارای یک بیت خروجی بود.
در این مثال سیستمی را بررسی می‌کنیم که تنها یک بیت اطلاعات را به عنوان ورودی بخواند. شکل زیر را ببینید:



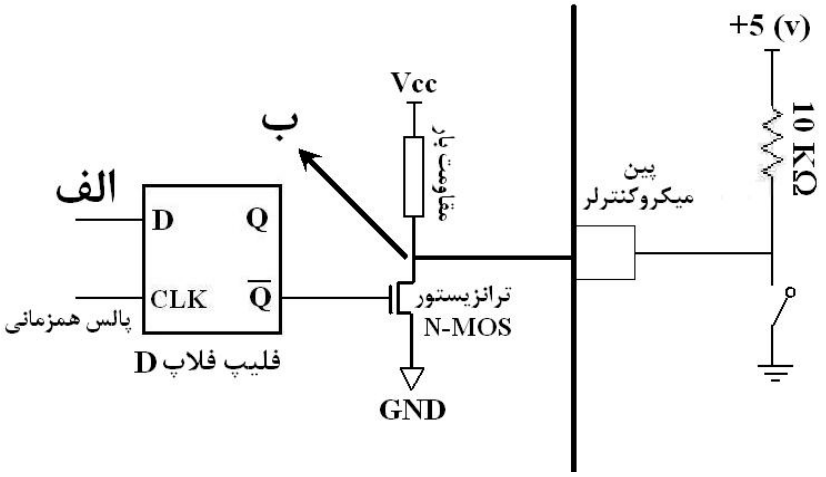
اگر کلید فشرده شده باشد، مقدار P2.0 برابر «یک» و اگر فشرده نشده باشد مقدار P2.0 برابر «صفر» خواهد بود.
طراحی برنامه‌ای که مقدار P2.0 را بخواند و بر اساس مقدار آن (که نشانه فشرده شدن یا نشدن آن است) تصمیم‌گیری کند، منوط به یادگیری دستورات پرش شرطی است که در فصل بعد مورد بررسی قرار خواهد گرفت.

نکاتی در مورد پورتهای ورودی/خروجی ۸۰۵۱

نکته ۱) دیدیم وقتی که میکروکنترلر می خواهد مقدار یک پورت را بعنوان ورودی بخواند، باید حتماً پیش از عمل خواندن به بیتهای آن پورت "یک" ارسال کند. ارسال عدد 255 یا

FF شانزده‌تایی (معادل 11111111 دودویی) بـه یـک پـورت ورودی بـه همـین دلیـل صورت می گیرد.

دلیل این موضوع این است که هر پین پورتهای ورودی/خروجی ۸۰۵۱ ساختاری به شکل زیر دارد :



در این شکل پین میکروکنترلر به یک کلید دو حالته متصل شده و قرار است مقدار آن توسط میکروکنترلر خوانده شود.

هنگامی که یک بیت ("**صفر**" یا "**یک**") توسط میکروکنترلر به یک پین ارسال می شود، در نقطه **الف** نوشته می شود. وقتی دستور خواندن از پورت توسط میکروکنترلر اجرا می شود، ولتاژ نقطه **ب** خوانده می شود[1].

گفتیم میکروکنترلر پیش از خواندن یک پین، بایـد در آن پـین (و در واقـع در نقطـه (**الـف**) ساختار آن پین) "**یـک** " بنویسد. حال فرض کنید پیش از عمل خواندن، در نقطه (**الف**) صفر نوشته شده است. خروجی $\overline{\mathbf{Q}}$ فلیپ فلاپ D، "**یک**" یا همان (v) 5+ شده و باعـث روشـن شدن ترانزیستور و اتصال نقطه (**ب**) به زمین داخلی میکروکنترلر می شود. بنابراین چه کلیـد دوحالته باز باشد چه بسته (چه "**صفر**" وارد میکروکنترلر شـود چـه "**یـک**") نقطـه (**ب**) بـه زمین متصل است و "**صفر**" در نظر گرفته می شود؛ یعنی در هر دو صورت با اجـرای دسـتور خواندن، "**صفر**" خوانده می شود!

اما اگر میکروکنترلر پیش از خواندن پین، "**یـک**" به آن ارسال کند، خروجی $\overline{\mathbf{Q}}$ فلیپ فـلاپ D برابر صفر و ترانزیستور خاموش می شود و نقطه (**ب**) از طریق مقاومت بار به Vcc داخلـی

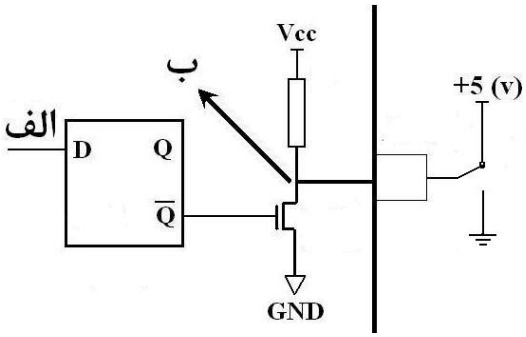
[1] در دستورات خواندن- اصلاح - نوشتن (Read-Modify-Write) این موضوع مستثناست که آن را بعدها بررسی خواهیم کرد.

میکروکنترلر متصل می شود. حال اگر کلید دو حالته باز باشد، ولتاژ نقطه (**ب**) برابر 5+ (v) ("یک") و اگر بسته باشد، این نقطه به زمین خارجی ("صفر") متصل است؛ پس در هر دو صورت مقدار کلید درست خوانده می شود.

بنابراین هرگاه در برنامه نیاز به خواندن یک پورت ورودی بود، پیش از دستور خواندن پورت، حتماً دستور ارسال "یک" به بیتهای پورت مورد نظر را می نویسیم.

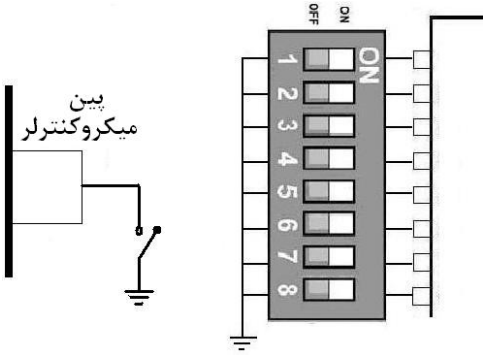
در میکروکنترلر ۸۰۵۱ بعد از Reset پورتها به صورت ورودی پیکربندی می شوند؛ یعنی ۸۰۵۱ در ابتدای روشن شدن به همه پورتها "یک" ارسال می کند؛ اما بعنوان یک عادت خوب حتی اگر در ابتدای برنامه می خواهید مقدار یک پورت را بخوانید، قبل ازدستورخواندن به بیتهای آن "یک" ارسال کنید.

ارسال "یک" به بیتهای یک پورت ورودی پیش از خواندن آن دلیل فنی دیگری نیز دارد؛ فرض کنید برای ایجاد یک بیت از ساختاری متفاوت استفاده کرده ایم و یک پین را به شکل روبرو مستقیماً (بدون استفاده از مقاومت 10 KΩ) به (v) 5+ متصل کرده ایم.



اگر پیش از خواندن پورت در بیتهای آن "صفر" نوشته شود، ترانزیستور روشن و نقطه (**ب**) به زمین متصل می شود. در این حالت چون مقاومتی بین راه (v) 5+ خارجی تا زمین داخلی میکروکنترلروجود ندارد، جریان زیادی از ترانزیستور می گذرد که باعث سوختن آن و غیرقابل استفاده شدن پورت می شود.

پرسش) الف) با توجه به ساختار پینهای پورتهای ۸۰۵۱ نشان دهید هرگاه پین میکروکنترلر از خارج به جایی متصل نباشد و مقدار آن را بخوانیم، "یک" خوانده می شود.



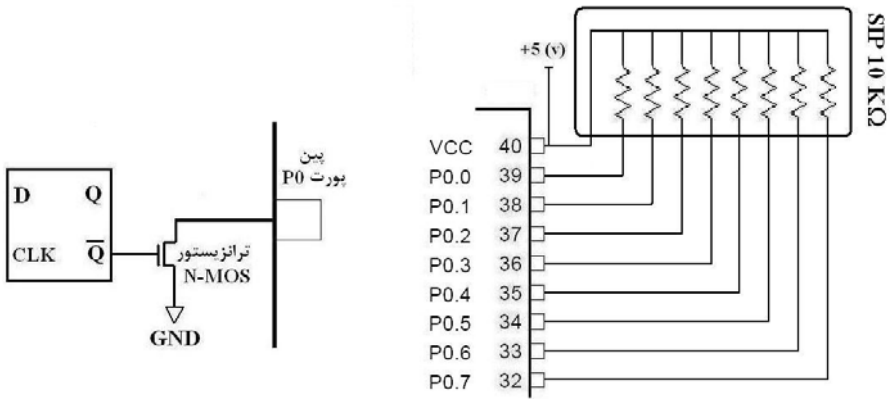
ب) با توجه به قسمت *الف* توضیح دهید چگونه ساختار روبرو می تواند برای ایجاد یک عدد ۸ بیتی به کار رود؟

البته این ساختار به دلیل عدم مقاومت در مقابل نویز

محیطی توصیه نمی‌شود. بهتر است پین فوق را با مقاومت به (v) 5+ متصل کنید. این کار به اصطلاح بالاکشی یا Pull-up نام دارد.

پرسش) با توجه به ساختار پینهای پورتهای ورودی/خروجی، توضیح دهید چرا همیشه آخرین مقداری که میکروکنترلر در پورت نوشته است، تا زمانی که مقدار جدیدی نوشته نشده حفظ می شود ؟

نکته ۲) پینهای پورت P0 به دلیل ایفای نقش آدرس و داده در هنگام اتصال حافظه خارجی[1]، مقاومت بار داخلی ندارند (اصطلاحاً ترانزیستور پینهای آن **درین باز**[2] است).
به همین دلیل *اگر می خواهید از پورت P0 بعنوان ورودی/خروجی استفاده کنید*، باید پینهای آن را با مقاومت بالاکش[3] به (v) 5+ متصل کنید.



همانطور که می بینید در این حالت استفاده از مقاومتهای آرایه ای (SIP) مناسبتر است. معمولاً از مقاومتهای 10 KΩ برای این کار استفاده می شود.

با تغییر اندازه مقاومتهای بالاکش، می توان جریانهای متفاوتی از پورت P0 کشید که از این لحاظ از پورتهای دیگر قابل انعطاف تر است.

پرسش) نشان دهید در مدار مثال طراحی سیستم ۱، با توجه به ساختار کلیدهای ورودی که به پورت P0 متصل شده است، نیازی به مقاومتهای بالاکش جداگانه نیست.

نکته ۳) پورت P3 نیز مانند پورتهای دیگر می تواند بعنوان ورودی/خروجی مورد استفاده قرار گیرد؛ اما از آنجایی که پینهای این پورت وظایف دوگانه دارند، بهتر است در صورت امکان

---

[1] برای آگاهی در مورد نقش پورت P0 در اتصال حافظه خارجی به میکروکنترلر، به ضمیمه (ج.۲) مراجعه کنید.

[2] ***Open-Drain***

[3] ***Pull-up***

از این پینها بعنوان ورودی/خروجی استفاده نشود؛ چون ممکن است در ابتدا بعنوان ورودی/خروجی در نظر گرفته شوند و در ادامه طراحی سیستم، وظایف دوم آن مد نظر قرار گیرد که در این صورت ناچار به تغییر کلی برنامه و سیستم خواهیم شد.

اشکال زدایی سیستم

فرض کنید تمام کارهای لازم برای طراحی سیستم را انجام داده اید؛ سخت افزار طراحی و برنامه پس از ترجمه به کد ماشین ۸۰۵۱ در حافظه ROM میکروکنترلر ذخیره شده و میکروکنترلر در مدار قرار گرفته است؛ اما سیستم کار نمی کند.

برای یافتن مشکل سیستم، روند زیر را پیشنهاد می کنیم :

۱) ابتدا پالس ALE را چک کنید. اگر این پالس وجود دارد، یعنی میکروکنترلر کار می کند و اشکال از سخت افزارهای جانبی یا برنامه سیستم است.
اگر ALE وجود دارد، سخت افزارها را کنترل کنید و با قرار دادن مجدد میکروکنترلر در دستگاه برنامه ریز، از ذخیره صحیح برنامه در حافظه مطمئن شوید (یکی از گزینه های نرم افزار کامپیوتری دستگاه برنامه ریز، گزینه خواندن (Read) است که با انتخاب آن، می توانید برنامه روی حافظه تراشه را مشاهده کنید).
اگر با وجود صحیح بودن اتصال سخت افزارها و وجود برنامه در حافظه میکروکنترلر، سیستم کار نمی کند یا *نتایج آن مورد انتظار شما نیست*، باید منطق برنامه خود را بازبینی و اصلاح کنید.

۲) اگر پالس ALE وجود ندارد، میکروکنترلر به درستی کار نمی کند. چند مشکل ممکن است وجود داشته باشد :

- اتصال پایه EA/VPP به $\overline{\text{Vcc}}$ را فراموش کرده اید.
- مدار Reset به درستی بسته نشده و پایه Reset فعال ("یک") باقی مانده باشد.
- کریستال به درستی کار نمی کند. اگر روی پایه XTAL2 فرکانسی دیده نشود، این مشکل وجود دارد. اگر با تعویض کریستال و خازنها مشکل برطرف نشد، نوسان ساز داخلی ۸۰۵۱ دچار مشکل شده و باید میکروکنترلر را تعویض یا از نوسان ساز خارجی برای تولید پالس ساعت استفاده کرد.

RAM داخلی تراشه ۸۰۵۱

در میکروکنترلر ۸۰۵۱، ۱۲۸ بایت حافظه RAM وجود دارد که برای ذخیره موقت و دستکاری داده ها به کار می روند. بعنوان مثال می خواهیم مرتباً مقدار یک پورت را بخوانیم و جایی ذخیره کنیم تا بعداً روی آن پردازش انجام شود. در یک میکروکنترلر اطلاعات باید یا در حافظه RAM ذخیره شود یا در حافظه ROM. طبیعی است که داده های مثال ما باید در RAM ذخیره شوند؛ چون اولاً ROM محل ذخیره برنامه کاربر است، ثانیاً نوشتن در ROM نیاز به دستگاه برنامه ریز دارد، در صورتی که نوشتن در RAM بسیار ساده و سریع است[1]. تنها داده‌هایی در ROM ذخیره می‌شوند که اولاً اندازه آنها زیاد نباشد که فضای برنامه را اشغال کنند، ثانیاً در طول برنامه ثابت بمانند. در مبحث مدهای آدرس‌دهی این موضوع را بیشتر بررسی می‌کنیم.

نقشه RAM ۱۲۸ بایتی ۸۰۵۱ در مقابل آمده است.

با بانکهای ثبات که در خانه صفر تا ۳۱ حافظه RAM قرار دارند، قبلاً آشنا شده اید.

از خانه شماره ۳۲ تا خانه شماره ۴۷، قسمت **قابل آدرس دهی بیتی** حافظه RAM است؛ یعنی به خانه های این قسمت حافظه RAM هم به صورت بیتی می توان مراجعه کرد و هم به صورت بایتی. با این مبحث در فصل ۷ بیشتر آشنا می شویم.

از خانه شماره ۴۸ تا خانه شماره ۱۲۷، قسمت یادداشت حافظه RAM است. خانه های این قسمت برای مقاصد مختلف برنامه نویسی مورد استفاده قرار می گیرند؛ مثلاً برای ذخیره ۱۰ مقدار اخیر که از پورت P1 خوانده شده است، می توان از خانه های ۵۰ تا ۶۰ حافظه RAM استفاده کرد.

رهنمود EQU

در زبان اسمبلی ۸۰۵۱، اعداد به وفور استفاده می شوند؛ اعداد ثابتی که برای منظوری در برنامه نویسی به کار می روند، اعدادی که شماره خانه های RAM هستند و ... .

---

[1] البته در بعضی انواع میکروکنترلرها علاوه بر حافظه های فوق، یک حافظه از نوع E^2PROM هم وجود دارد که در حین کارکرد سیستم می توان اطلاعاتی در آن ذخیره کرد که بعد از قطع برق نیز از بین نمی روند. با بعضی از این میکروکنترلرها در ضمیمه (ج.۱) آشنا خواهید شد.

گاهی وجود این اعداد برنامه را به صورتی پیچیده و گیج کننده درمی آورد؛ بهمین دلیل وجود راهی برای گذاشتن نام روی اعداد و استفاده از اعداد به جای نامشان در برنامه ضروری به نظر می رسد.

رهنمود EQU[1] برای نامگذاری به صورت زیر به کار می رود :

نام اصلی EQU نام انتخابی

مثلاً وقتی قرار است فشار یک مخزن را به صورت یک عدد ۸ بیتی از پورت P1 بخوانیم و در خانه شماره ۷۸ حافظه RAM ذخیره کنیم، به جای دستور

```
MOV   78,P1
```

که از نظر فهم برنامه کمی نامفهوم و پیچیده است، می توان در ابتدای برنامه با توجه به اینکه از خانه شماره ۷۸ حافظه RAM برای ذخیره **فشار** یک مخزن استفاده می کنیم، با رهنمود

```
Pressure   EQU   78
```

نام Pressure را برای خانه شماره ۷۸ برگزید و در ادامه برنامه با دستور

```
MOV   Pressure,P1
```

که مفهوم دستور را بهتر می رساند، عمل ذخیره سازی را انجام داد. اسمبلر در هنگام ترجمه این برنامه، تمام کلمات Pressure را با مقدار عددی معادل آن (۷۸) جایگزین می‌کند.
از رهنمود EQU می‌توان برای تعریف نام برای ثباتهای A و R0 تا R7 نیز استفاده کرد.
رهنمود EQU تأثیری در نتیجه برنامه ندارد.
پرسش) اگر در ابتدای برنامه از رهنمود Variable EQU 20 استفاده کرده باشیم، دو دستور زیر چه تفاوتی دارند ؟

```
MOV   A,Variable          MOV   A,#Variable
```

استفاده از رهنمودها علاوه بر خواناتر شدن برنامه مزیت دیگری نیز دارد؛ در پرسش بالا تصور کنید عدد ۲۰ را در چند جای مختلف برنامه به کار برده ایم. اگر به دلیلی بخواهیم این عدد ۲۰ را به عدد ۳۰ تبدیل کنیم کافی است رهنمود اول برنامه را به صورت

[1] ***EQUate***

Variable EQU 30 اصلاح کنیم. اگر از این رهنمود استفاده نکرده باشیم، ناچاریم در تمام نقاطی از برنامه که عدد ۲۰ به کار رفته این اصلاح را انجام دهیم.

پرسش) توضیح دهید برنامه زیر چه عملی انجام می دهد ؟

```
ORG  0

ON     EQU  #10101010B
OFF    EQU  #01010101B

AGAIN :

MOV  P1,ON
```

دستورات ایجاد تأخیر

```
MOV  P1,OFF
```

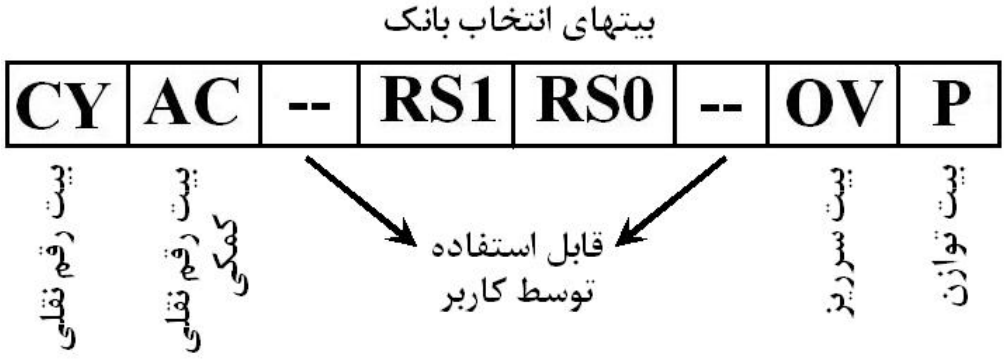
دستورات ایجاد تأخیر

```
SJMP AGAIN

END
```

رهنمود BIT مانند رهنمود EQU است؛ با این تفاوت که روی یک بیت نامگذاری انجام می‌دهد. مثلاً رهنمود PIN BIT P1.0، پایه P1.0 را به نام PIN نامگذاری انجام می‌دهد. بعد از این رهنمود، دستور SETB PIN باعث «یک» شدن P1.0 می‌شود.

ثبات PSW

ثبات PSW، ثبات پرچم میکروکنترلر ۸۰۵۱ است. بیتهای این ثبات را در شکل زیر می بینید :



با استفاده از شماره بیتها و علامت نقطه، می توان به آنها مراجعه کرد؛ مثلاً به بیت شماره صفر می توان به صورت PSW.0 مراجعه کرد یا PSW.3 همان RS0 است. بعلاوه به هر کدام از بیتها با نامشان نیز می توان مراجعه کرد؛ مثلاً دستورات زیر معادلند :

```
SETB    AC          یا          SETB    PSW.6
```

بیتهای این ثبات را به طور مختصر در اینجا معرفی می کنیم و بحث مفصلتر را به فصول آینده واگذار می کنیم :

بیت توازن[1] (PSW.0)

این بیت نشان دهنده توازن زوج محتویات ثبات A است و در کشف خطا مورد استفاده قرار می گیرد که در فصل ۱۰ درباره آن سخن خواهیم گفت.

بیت سرریز[2] (PSW.2) :

از این بیت برای کشف خطای سرریز در اعمال ریاضی علامتدار و نیز اعلان خطای تقسیم بر صفر استفاده می شود که در فصل ۶ به آن خواهیم پرداخت.

بیتهای انتخاب بانک[3] (PSW.3 و PSW.4) :

با نقش این دو بیت که RS0 و RS1 نیز نام دارند، در تغییر بانک ثباتها آشنا شدیم. مثلاً برای انتخاب بانک شماره ۱ (RS1 = 0 , RS0 = 1) از دستورات زیر می توان استفاده کرد :

```
CLR     RS1             یا          CLR     PSW.4
SETB    RS0                         SETB    PSW.3
```

پس از اجرای این دستورات، با مراجعه به R0، به ثبات R0 بانک شماره ۱ (خانه شماره ۸ حافظه RAM) مراجعه می شود.

پرسش) دستورات لازم برای انتخاب بانک شماره ۳ را بنویسید.

بیت رقم نقلی کمکی[4] (PSW.6) :

این بیت وجود رقم نقلی از بیت شماره ۳ به بیت شماره ۴ در حاصل یک عمل ریاضی را نشان می دهد که در محاسبات BCD مورد استفاده قرار می گیرد.

---

[1] ***Parity***
[2] ***Overflow***
[3] ***Register Select***
[4] ***Auxiliary Carry***

بیت رقم نقلی[1] (PSW.7) :

این بیت وجود رقم نقلی (سرریز) در محاسبات ریاضی بدون علامت را نشان می دهد (فصل ۶). بعلاوه این بیت در دستورات بیتی و منطقی نیز به کار می رود (فصل ۷)

بیتهای بدون استفاده (PSW.1 و PSW.5) :

این دو بیت در ۸۰۵۱ استفاده خاصی ندارند و کاربر می تواند در برنامه های خود از آنها بعنوان یک پرچم منطقی استفاده کند. در فصل ۷ مثالی راجع به این موضوع خواهید دید.

خلاصه

این فصل آغاز آشنایی ما با میکروکنترلرهای خانواده ۸۰۵۱، ساختار داخلی و برنامه نویسی آنها بود. در این فصل با ویژگیهای سخت افزاری یکی از اعضای این خانواده به نام AT89S51 آشنا شدیم. طریقه بستن مدار پایه ۸۰۵۱ و نیز طرز استفاده از پورتها بعنوان ورودی/خروجی از مسایل دیگر مورد بحث بود. در بخش دیگر این فصل نیز با تعدادی از ثباتهای داخلی ۸۰۵۱ آشنا شدیم.

چند سیستم نیز از نظر سخت افزاری و نرم افزاری به طور کامل طراحی و بررسی شد. در انتهای فصل نیز با نقشه RAM داخلی ۸۰۵۱ و نیز ثبات PSW آن آشنا شدیم.

در فصول آینده با جنبه های مختلف برنامه نویسی اسمبلی ۸۰۵۱ آشنا می شویم.

پرسشهای دوره ای

۱) مهمترین دلایل محبوبیت میکروکنترلر ۸۰۵۱ را ذکر کنید.

۲) الف) امکانات میکروکنترلر ۸۰۵۱ پایه (MCS-51) را نام ببرید.

ب) در این میکروکنترلر حداکثر حجم برنامه کاربر چقدر می تواند باشد ؟

ج) اگر حجم برنامه کاربر بیش از اندازه فوق باشد، چه باید کرد ؟

۳) تراشه AT89S51 در یک لحظه می تواند چند واحد ۸ بیتی اطلاعات را با جهان خارج مبادله نماید ؟ چرا ؟

۴) الف) مدار لازم برای استفاده از نوسان ساز داخلی ۸۰۵۱ را رسم کنید.

ب) چگونه می توان از صحت کار این نوسان ساز مطمئن شد ؟

---

[1] *Carry*

ج) حداکثر فرکانس کریستال که می تواند به میکروکنترلر AT89S51-16PC متصل شود، چقدر است ؟

۵) الف) مدار Auto Power-on Reset را رسم کنید و عملکرد آن را توضیح دهید.

ب) مقادیر مقاومت و خازن این مدار چگونه تعیین شده اند ؟

۶) اگر

الف) RS1 = RS0 = 1

ب) RS1 = 0 و RS0 = 1

ج) میکروکنترلر تازه روشن شده باشد

با مراجعه به R2، به کدام خانه حافظه RAM مراجعه می شود ؟

۷) تفاوت برنامه نویسی با زبان اسمبلی ۸۰۵۱ و زبان C-51 چیست ؟

۸) هرکدام از مجموعه دستورات زیر چه می کنند ؟

```
MOV   A,R0          MOV   A,#10         MOV   A,#10
MOV   R0,R3         ADD   A,10          ADD   A,#10D
MOV   R3,R7         MOV   10,A          ADD   A,#10B
MOV   R7,A                              ADD   A,#10H
```

۹) یک سیستم کامل (شامل سخت افزار و نرم افزار) طراحی کنید که دو عدد m و n را از کاربر گرفته و حاصل 2m-n را نمایش دهد.

۱۰) الف) چگونه می توان برنامه ای نوشت که فقط یک بار اجرا شود ؟

ب) چگونه می توان برنامه ای نوشت که مرتباً اجرا شود ؟

۱۱) الف) دستور SJMP $ به چه منظور در انتهای برنامه استفاده می شود ؟

ب) تفاوت این دستور با END چیست ؟

۱۲) روندنمای تبدیل برنامه اسمبلی به کد ماشین ۸۰۵۱ را رسم کنید و توضیح دهید.

۱۳) در صورت وجود خطا در برنامه، چگونه می توان به محل و نوع این خطاها پی برد؟

۱۴) اگر در ابتدای برنامه مثال ۱، از ORG 50 استفاده شده باشد، ستون LOC فایل لیست این مثال چگونه خواهد بود ؟

۱۵) سخت‌افزار مثال طراحی سیستم ۲ را با میکروکنترلر ۲۰ پایه AT89S2051 که تنها دو پورت ورودی/خروجی دارد طراحی کنید. برگه اطلاعاتی میکروکنترلر را از اینترنت (مثلاً سایت اینترنتی شرکت ATMEL) دانلود کنید.

۱۶) چرا در برنامه مثال ۳، باید بین خاموش و روشن شدنهای متوالی LED تأخیر ایجاد کرد؟

۱۷) با توجه به ساختار پینهای ورودی/خروجی،

الف) چرا باید پیش از خواندن یک پین، میکروکنترلر به آن پین "یک" ارسال کند؟

ب) چگونه با یک کلید فشاری می توان یک بیت را تولید کرد ؟

ج) چرا آخرین مقداری که میکروکنترلر در یک پورت می نویسد، تا زمان نوشتم عدد بعدی تغییر نمی کند ؟

۱۸) چرا باید در هنگام استفاده از پورت P0 بعنوان ورودی/خروجی، از مقاومتهای بالاکش استفاده کرد ؟

۱۹) اگر

الف) سیستم اصلاً کار نکند (پالس ALE نداشته باشیم)

ب) سیستم کارکند، اما نتایج آن مورد انتظار ما نباشد

مشکل ممکن است از کدامیک از موارد زیر باشد ؟

- مدار نوسان ساز
- اتصال EA/Vpp به Vcc
- مدار Reset
- منطق برنامه

۲۰) رهنمود EQU در چه مواردی استفاده می شود ؟

۲۱) منظور از RAM قابل آدرس دهی بیتی چیست ؟