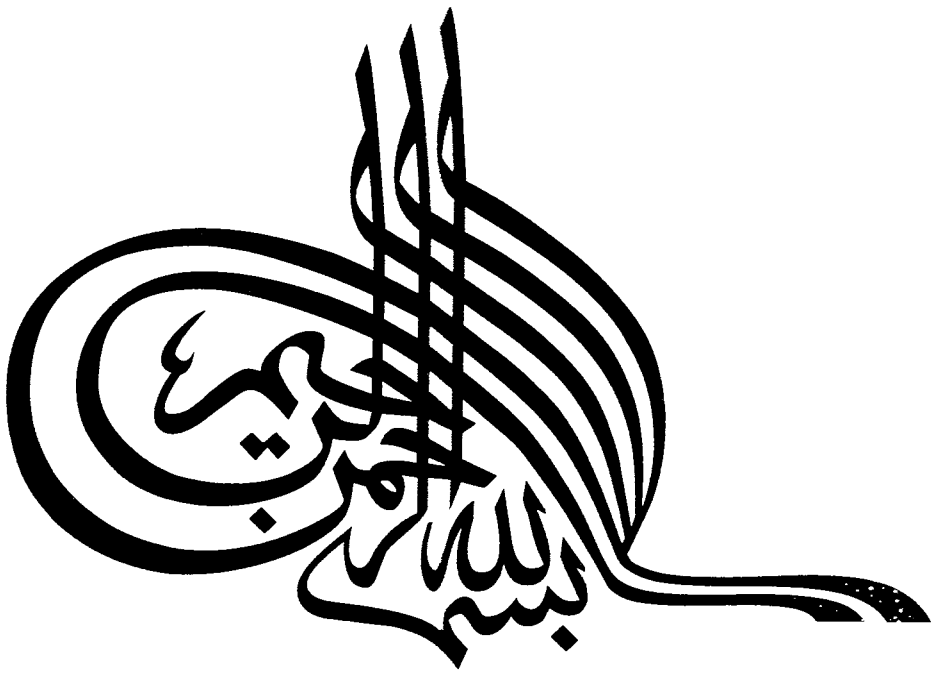

بسم الله الرحمن الرحيم

گزارش نهایی پروژه
طرح تحقیقاتی

موضوع:

طراحی و ساخت برد شبیه ساز خانواده *MCS51*

تهیه کننده:
علی سلیمانی ایوری
دانشگاه صنعتی شاهرود

اردیبهشت ۱۳۸۳

پیشگفتار

قابلیت اطمینان بالا، اپتیمم شدن برای کاربردهای کنترلی، هزینه پایین سخت افزار، کم بودن تعداد دستورالعملها، وجود قسمتهای مختلف ورودی / خروجی در میکروکنترلرها باعث شده است این عناصر در صنعت مورد توجه خاص قرار گیرند و از آنها در ساخت سیستمهای کنترلی مختلف استفاده کنند. از جمله میکروکنترلرهای معروف و پرکاربرد، میکروکنترلرهای خانواده *MCS51* است که انواع مختلفی از آنها با قابلیتهای مختلف در دسترس است. آشنایی با عملکرد آنها، طراحی سخت افزار لازم برای پروژه های کارشناسی و نوشتن نرم افزارهای مناسب، برای دانشجویان مهندسی برق لازم و ضروری است. بخاطر عدم وجود یک مرجع مناسب و یک سیستم آموزشی ساده، پروژه فوق پیشنهاد گردید. در پیشنهاد و اجرای این پروژه اهدافی چون فعال نمودن پروژه های تحقیقاتی، پروژه های کارشناسی مرتبط با طراحی و ساخت، ساخت یک سیستم آموزشی با مشخصات لازم و با استفاده از روش پیشنهادی مد نظر قرار گرفته است. اکنون این سیستم طراحی و ساخته شده و قابل استفاده برای علاقه مندان می باشد. در این راستا گزارشی با موضوعاتی چون سخت افزار میکروکنترلر ۸۰۵۱، چگونگی طراحی و ساخت سیستم مورد نظر، مبانی نرم افزار با اسمبلی و C به همراه مثالهای متعدد و راه حل عیب یابی ارائه شده است. با توجه به هدف اصلی پروژه در طراحی، ساخت و برنامه های نوشته شده یا پیشنهاد شده به ابعاد آموزشی و کاربردی توجه خاص شده بنحوی که دانشجویان بتوانند از آن بعنوان یک مرجع جهت کار با میکروکنترلرهای این خانواده استفاده کنند. علاوه بر گزارش فوق، گزارش کاربردی سیستم آموزشی نیز جهت سهولت استفاده ارائه شده است.

ایـن گزارش نتیجه طرح پژوهشی با عنوان طراحی و ساخت برد شبیه ساز خانواده *MCS51* است که در تاریخ ۸۳/۱/۳۰ به تصویب شورای پژوهشی دانشگاه رسیده است.

علی سلیمانی ایوری
اردیبهشت ۱۳۸۳

چکیده

میکروکنترلرها بخاطر سادگی و قابلیتهایی که در داخل آنها پیش بینی شده است بشدت مورد علاقه صنایع قرار گرفته اند. این امر طراحان سیستمهای میکروپروسسوری را برآن داشته است که هرچه بیشتر با این عناصر آشنا شده و از آنها در اجرای پروژه های صنعتی و آزمایشگاهی خود استفاده کنند. از طرف دیگر در حال حاضر دانشجویان در مطالب درسی خود با میکروکنترلرها به اندازه لازم آشنا نشده و لزوم فراگیری آن احساس می شود. چگونگی بکارگیری آنها و آموزش کار با آنها با هزینه کم و در مدت کوتاه از مسائلی است که بایستی مورد توجه قرار گیرد. علاوه بر این برای اجرای پروژه هایی که با این خانواده انجام می شود احتیاج به میکرو کنترلر، حافظه *EPROM* ، پاک کننده *EPROM* ، شبیه ساز *EPROM* ، برنامه ریز *EPROM* و اسیلوسکوپ می باشد که در صورت در دسترس بودن آنها، استفاده از آنها وقت گیر و مستلزم صرف هزینه های زیادی است. از طرف دیگر تعداد پاک کردنها و نوشتنها در حافظه *EPROM* و *EEPROM* میکرو کنترلر محدود است و گاهی باعث سوختن آنها می گردد.

این طرح برای حل این موضوع یعنی فراگیری سریعتر و صرفه جویی در هزینه، راه ساده و مؤثری پیشنهاد می نماید. این طرح به طراحی سخت افزاری می پردازد که قادر است با کامپیوتر ارتباط داشته، فایل اجرایی روی میکروکنترلر را از کامپیوتر دریافت و پس از ذخیره در حافظه *RAM* یا *EEPROM* اجرا نماید. این سخت افزار بنحوی طراحی می گردد که همه قابلیتهای سری *MCS51* را در اختیار برنامه نویس قرار دهد و برنامه نوشته شده بدون هیچ تغییری قابل انتقال و قابل اجرا در سخت افزار اصلی باشد. با وجود این سیستم، در کارهای آموزشی و تحقیقاتی و تست برنامه ها احتیاجی به برنامه ریزی آی سی های *EPROM* و میکرو کنترلرهای *89Cxx* نبوده و در نتیجه در هزینه و وقت بسیار صرفه جویی می شود. این طرح به سه بخش سخت افزار، نرم افزار روی سخت افزار و نرم افزار روی کامپیوتر و برنامه های آموزشی با زبان اسمبلی و C تقسیم می گردد.

این طرح با قابلیتهایی که در آن پیش بینی می شود و با تجاربی که در قالب گزارش نهایی ارائه می گردد به دانشجویان کمک می کند هرچه سریعتر با خانواده میکرو کنترلرهای *MCS51* آشنا شده و قادر باشند با آن، برنامه های کاربردی متنوع بنویسند و پروژه خود را بسرعت پیش ببرند و در صورت نیاز سخت افزار خاص پروژه خود را به مجموعه اضافه نمایند لذا به جنبه آموزشی این طرح توجه زیادی شده و با سخت افزار و نرم افزارهای موجود در بازار اختلافاتی دارد.

ایـن گزارش، که مرحله نهایی این پروژه می باشد، ابتدا پروژه و اهداف مورد نظر تعریف و مزایا و معایب آن و توجیه اقتصادی آن مورد ارزیابی قرار می گیرد و در ادامه، مبانی تئوری، بلوک دیاگرام کلی سیستم و قسمتهای مختلف آن تشریح می گردند. سخت افزار لازم طراحی و اشکالات موجود رفع می گردند، نرم افزارهای روی کامپیوتر، نرم افزار روی سخت افزار و نرم افزارهای آموزشی توضیح داده شده و پیاده سازی مـی گـردند و در انتها با جمع بندی کارهای انجام شده و نتایج بدست آمده و ارائه گزارش کاربردی سعی می گردد تاجایی که ممکن است ابعاد مختلف طرح تحت پوشش قرار گیرد.

فصل اول

مقدمه

فهرست صفحه

فصل دوم

مبانی سخت افزار میکروکنترلر ۸۰۵۱

فهرست صفحه

فصل سوم
طراحی سخت افزار

فهرست صفحه

فصل چهارم
مبانی نرم افزار

فهرست صفحه

فصل پنجم

نرم افزارهای پروژه

فهرست صفحه

فصل ششم

برنامه نویسی با اسمبلی و دستورالعملهای ۸۰۵۱

فهرست صفحه

فصل هفتم

برنامه نویسی به زبان C

فهرست صفحه

فصل هشتم

مثالهای برنامه نویسی به زبان اسمبلی و C

فهرست صفحه

فصل نهم

عیب یابی برنامه ها به زبان اسمبلی و C

فهرست صفحه

فصل دهم

مراحل انجام پروژه و نتیجه گیری

فهرست صفحه

فصل اول

مقدمه

تاکنون مدارهای بسیار مجتمعی طراحی و در صنعت، تجارت، اقتصاد، علوم مهندسی، پزشکی و غیره مورد استفاده قرار گرفته اند. به جهت صرفه جویی در هزینه و بالا بردن کارآیی مدارات مجتمع، در طراحی آنها سعی می شود نوع استفاده و قابلیتهای لازم و ضروری دقیقاً مورد توجه قرار گیرد و طرح برای آن کاربرد بهترین باشد. قسمتی از این مدارات مجتمع کارهای پردازش را انجام می دهند که به آنها پروسسور گفته می شود و تاکنون پروسسورهای متعددی طراحی و روانه بازار شده اند که هریک برای کاری خاص اپتیمم شده اند. میکروپروسسورها که بیشتر کاربرد آنها در کامپیوترهاست برای پردازش اطلاعات و ذخیره اطلاعات در مقیاس وسیع مورد استفاده قرار می گیرند. پروسسورهای *DSP*[1] در پردازش سیگنالهای دیجیتالی که در سیستمهای مخابراتی و پزشکی زیاد وجود دارند استفاده می شوند. میکرو کنترلرها برای کارهای کنترلی بهینه شده اند و در طراحی آنها سعی شده است همه یا قسمت زیادی از تجهیزات لازم برای یک کار کنترلی پیش بینی شده باشد. از آنجایی که اغلب کارهای کنترلی از برنامه های پیچیده ای برخوردار نیستند، حجم برنامه زیادی لازم ندارند و اغلب فضای آدرس دهی آنها به ۶۴ یا ۱۲۸ کیلوبایت محدود می شود. در عوض یک کار کنترلی بایستی از قابلیت اطمینان بالایی برخوردار باشد لذا در این میکرو کنترلرها این موضوع مورد توجه خاص قرار گرفته بطوری که در بعضی از آنها زمان سنج نگهبان که وظیفه نظارت برکار نرم افزار رابعهده دارد پیش بینی شده است و به جهت سادگی ساختار داخلی آنها و کم بودن فضای آدرس دهی امکانات اضافی دیگری همچون پورت سریال، پورت های ورودی خروجی، تایمر، شمارنده، مبدلهای *A/D* و *D/A* و ... در داخل آنها پیش بینی شده است. و این باعث شده است میکروکنترلرهای مختلف با قابلیتهای متفاوت همچون *80196*، *68HC11*، *PIC* ها، *AVR* ها با در مدلهای متنوع توسط سازندگان مختلف ارائه گردد. لذا برای هر پروژه ای میکروکنترلر ویژه ای بهترین انتخاب است که بایستی آنرا بدرستی انتخاب، و مورد استفاده قرار داد.

۱ـ۱ـ اهداف اجرای پروژه

در اجرای این پروژه اهداف زیادی مد نظر بوده که عمده آنها را بصورت زیر می توان برشمرد:

الف ـ فعال نمودن پروژه های تحقیقاتی و کارشناسی: به جهت تخصیص ندادن اعتبارات لازم برای پروژه های کارشناسی معمولاً دانشجویان در اجرای آنها با مشکلات عدیده ای برخورد کرده و نتیجه مناسبی عاید نمی شود مشکلاتی چون نداختیار نبودن تجهیزات لازم، در دسترس نبودن قطعات، در دسترس

---

[1] *Digital Signal Processing*

نبودن مرجع مناسب و ... همه همه باعث کند شدن روند اجرای پروژه می گردد. هدف این پروژه تهیه ابزاری مناسب با هزینه کم برای به حداقل رساندن مشکلات فوق می باشد.

ب ـ ساخت سخت افزاری با مشخصات زیر:

۱- حجم حافظه برنامه ۳۲ کیلو بایت و حجم حافظه دیتا ۶۴ کیلو بایت خواهد بود. به عبارت دیگر حجم برنامه در سیستمهایی که با ۸۹۵۱ نوشته می شوند به ۴ کیلوبایت محدود می گردد اما در این برد به ۳۲ کیلوبایت افزایش یافته در نتیجه بدون هیچ دغدغه ای می توان از کامپایلرها در برنامه نویسی استفاده کرد.

۲- احتیاجی به برنامه ریزی *EPROM* و برنامه ریزی میکروکنترلر ندارد لذا پاک کردن و برنامه ریزی لازم نبوده و دروقت وهزینه صرفه جویی خواهد شد.

۳- استفاده کننده به هیچ عنوان با آن، بعنوان یک جعبه سیاه برخورد نخواهد کرد. معمولاً شبیه سازهای معمولی ، برنامه أی مشابه سیستم عامل دارند که قسمتی از امکانات سیستم را در اختیار گرفته و برنامه تحت نظارت آن اجرا می گردد لذا استفاده کننده مبتدی چون فردی در کار او دخالت کرده است بدرستی متوجه روند کار نمی گردد.

۴- نرم افزاری که روی کامپیوتر نوشته شده و قابل اجراست به استفاده کننده جهت ارسال فایل و عیب یابی برنامه ها کمک می کند.

۶- برنامه مورد نظر جهت آموزش را می توان بصورت اسمبلی یا *C* نوشته و بصورت بلادرنگ مورد تست و ارزیابی قرار داده و عیب یابی کرد.

ج ـ تهیه گزارشی مناسب برای میکروکنترلرهای خانواده *MCS51* :

گزارش نهایی که قسمت اصلی پروژه می باشد شامل سخت افزار، نرم افزار با اسمبلی و *C* و راه حلهای عیب یابی است که می تواند کمک خوبی برای استفاده کنندگان باشد. در طراحی، ساخت و برنامه های آموزشی به ابعاد آموزشی و کاربردی آن توجه خاصی می شود بنحوی که بتوان از آن بعنوان یک مرجع جهت کار با میکروکنترلرهای این خانواده استفاده کرد.

۲ـ۱ـ انواع شبیه سازها

شبیه سازها کلاً به سه دسته نرم افزاری، سخت افزاری و سخت افزاری درون مداری تقسیم می گردند.

۱ـ۲ـ۱ـ شبیه سازهای نرم افزاری

این شبیه سازها بصورت نرم افزاری به شبیه سازی عملکرد میکروپروسسور یا میکروکنترلر هنگام اجرای برنامه می پردازند و نتایج اجرای برنامه را بفرمها و اشکال مختلف نمایش می دهند در این نوع شبیه سازها می توان برنامه لازم را خوانده و آنرا به یکباره و یا قدم بقدم اجرا نمود و وضعیت رجسترها و پورتهای مختلف را مشاهده کرد. عملکرد این شبیه سازها ساده بوده و هزینه کمی دارند.

اکنون برای خانواده ۸۰۵۱ شبیه سازهای نرم افزاری متنوعی ارائه شده و در دسترس می باشد بکار گیری این شبیه سازها برای تست بعضی از برنامه ها که با پورتهای ورودی خروجی و یا انترابتها سرو کار

ندارند مناسب است اما اگر عملکرد پورتها و سخت افزاری خاص مد نظر باشد بسادگی ممکن نیست لذا این نوع شبیه سازها کارآیی زیادی ندارند و تعقیب اجرای برنامه در آنها مشکل و وقت گیر است و علاوه بر اینها از بار آموزشی مناسبی برخوردار نیستند. مخصوصاً هنگامی که برنامه طولانی بوده و عملیات بخصوصی را که با سخت افزار نیز درگیر است انجام دهد.

۲ـ۲ـ۱ـ شبیه ساز سخت افزاری درون مداری

در این نوع شبیه سازها که قیمت آنها بسیار بالاست توسط سخت افزارها و نرم افزارهای خاص، عملکرد میکروکنترلر یا میکروپروسسور دقیقاً شبیه سازی می گردد. این نوع شبیه سازها از یک طرف به کامپیوتر و از طرف دیگر توسط یک کانکتور خاص به برد سخت افزاری متصل می گردد. با اتصال کانکتور که به جای میکروپروسسور یا میکروکنترلر قرار می گیرد مثل این است که خود میکروپروسسور یا میکروکنترلر در مدار قرار گرفته است. در این نوع سیستمها برنامه از طریق کامپیوتر دریافت و در حافظه سیستم ذخیره می گردد پس از اجرای برنامه سخت افزار به وضعیت پایه ها توجه نموده و وضعیت آنها را از طریق کامپیوتر به اطلاع می رساند با تحلیل اطلاعات دریافتی اشکالات سخت افزاری برد تا اندازه ای مشخص می گردد. این نوع شبیه سازها می توانند از حافظه سیستم و یا حافظه موجود در شبیه ساز که از طریق کامپیوتر مقدار دهی شده اند استفاده نموده و به اجرای برنامه بپردازند.

۳ـ۲ـ۱ـ شبیه سازهای سخت افزاری

حد فاصل بین دو شبیه ساز نرم افزاری و شبیه ساز درون مداری شبیه ساز سخت افزاری است. در این نوع شبیه ساز که گاهی به آن برد آموزشی نیز گفته می شود برنامه مورد نظر به حافظه سخت افزار منتقل و سپس توسط میکروپروسسور یا میکروکنترلر مربوطه بصورت زمان واقعی اجرا و نتایج بدست آمده، آن طور که هست نمایش داده می شود. در این سخت افزارها امکان ارتباط با کامپیوتر فراهم شده و در نتیجه می توان یک فایل را از کامپیوتر دریافت و سپس اجرا نمود. در این سخت افزارها نرم افزاری مشابه سیستم عامل روی سیستم قرار گرفته و عملیاتی مانند زیر را می توان انجام داد.

Fill a block with data
Copy a block of data to anather area
Change a single address data Content
Compare any two block of address data
Display data may be written to a file
Register /SFRS/Bit memory Examine or modify
Program variable Examine or modify

و علاوه بر اینها می توان برنامه را بطور یکجا و قدم به قدم اجرا و نتایج را مشاهده کرد. آنچه که در این پروژه مد نظر است سیستمی مشابه سیستم فوق بوده که با روشی خاص و باحذف سیسنم عامل به شبیه سازی زمان واقعی میکروکنترلر می پردازد و سعی می گردد در حین سادگی از بار آموزشی مناسبی برخوردار باشد.

فصل دوم

سخت افزار میکروکنترلر ۸۰۵۱

۱ـ۲ـ مقدمه

در این فصل سخت افزار میکرو کنترلر ۸۰۵۱ معرفی می گردد. در این راستا ساختار داخلی، وظایف پینها، ساختار حافظه، نحوه اتصال حافظه های *EPROM*، حافظه های *RAM*، رجسترهای داخلی، چگونگی استفاده از تایمرها و کانترها، پورت سریال و انتراپتها مورد بحث قرار می گیرند و در انتها مشخصات بعضی از میکروکنترلرهای این خانواده به جهت ضرورت بصورت مختصر بیان شده و با یکدیگر مقایسه می گردند.

۲ـ۲ـ میکروکنترلرهای خانواده *MCS51*

میکروکنترلرهایی که تاکنون براساس میکروکنترلر ۸۰۵۱ طراحی و روانه بازار شده اند و آنها را با خانواده *MCS51* می شناسند بسیار زیاد است. این میکروکنترلرها از لحاظ نرم افزاری از میکروکنترلر ۸۰۵۱ تبعیت می کنند اما از لحاظ سخت افزاری قابلیتهای متفاوتی دارند. در این میان میکروکنترلرهایی چون *8031*، *8032*، *AT89C51*، *AT89C2051* و ... نام برد. اکنون میکروکنترلر ۸۰۵۱ بعنوان پایه معرفی و قسمتهای مختلف آن تشریح می گردد.

۳ـ۲ـ میکرو کنترلر *8051*

ساختار داخلی میکروکنترلر۸۰۵۱ در شکل۱ـ۲ نشان داده شده است. در این شکل قسمتهای داخلی این میکروکنترلر و ارتباط آنها با یکدیگر نشان داده شده است. با توجه به این شکل و در نظر نگرفتن بلوک داخلی که پورت سریال، انتراپت و تایمر را شامل می گردد یک *CPU* هشت بیتی با رجسترهای لازم حاصل خواهد شد که می توان از آن استفاده کرد. بعبارت دیگر میکروکنترلر ۸۰۵۱ یک میکروپروسسور ۸ بیتی است که قابلیتهایی چون پورت سریال، تایمر/کانتر و انتراپت در آن پیش بینی شده است.

با توجه به ساختار داخلی و اطلاعات ارائه شده مشخصات این میکروکنترلر بصورت زیر خلاصه می گردد:

۱- دارای دو عدد تایمر یا کانتر ۱۶ بیتی است (۸۰۵۲ سه عدد تایمر یا کانتر ۱۶ بیتی دارد).

۲- دارای دو پایه انتراپت خارجی است،

۳- دارای پورتی سریال با چهار مد مختلف است،

۴- ۱۲۸ بایت حافظه *RAM* داخلی دارد. این مقدار در *8052*، ۲۵۶ بایت است،

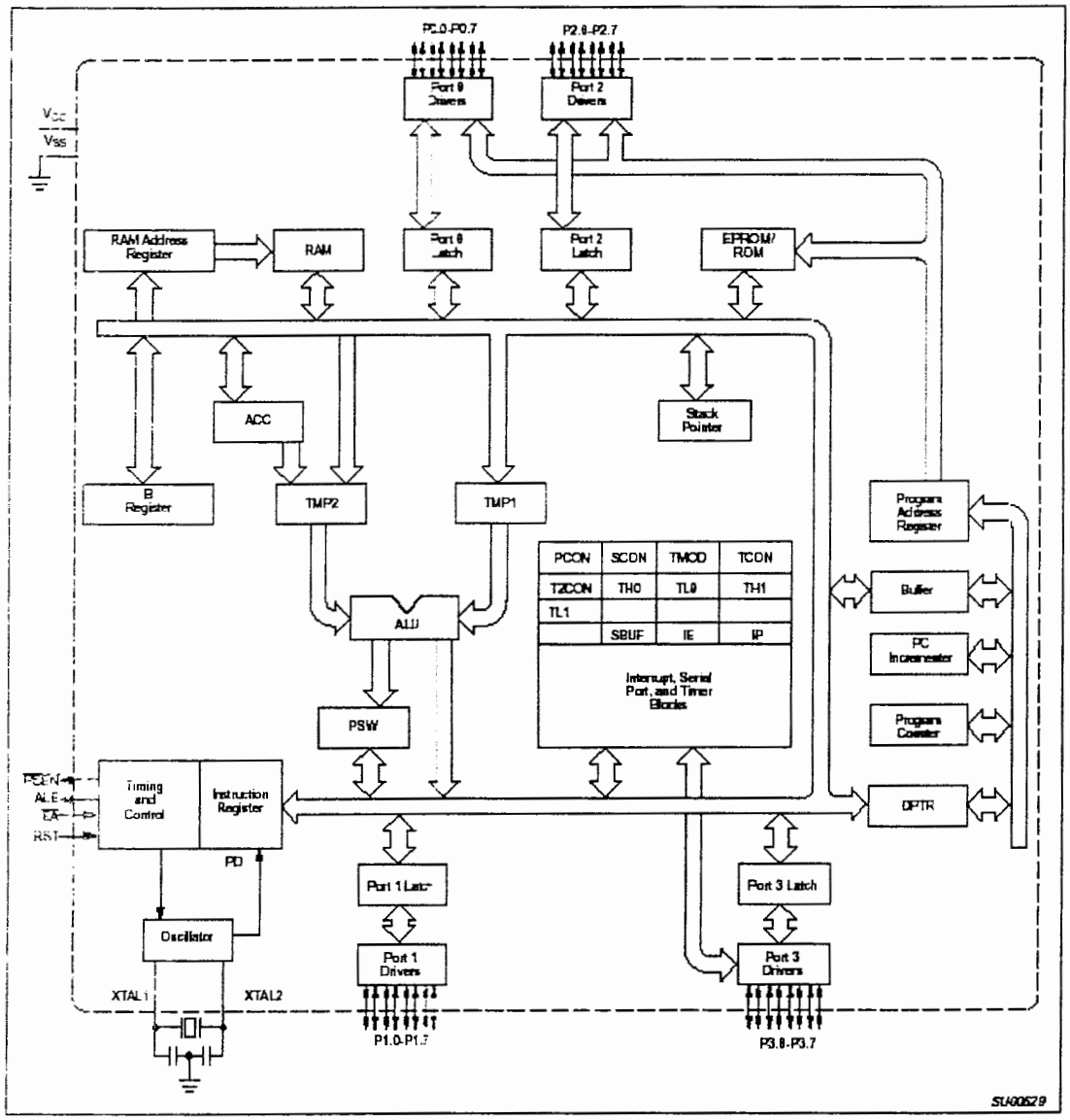
۵- ۴ کیلو بایت حافظه *ROM* داخلی دارد. در ۸۰۵۲ این مقدار به ۸ کیلو بایت می رسد. (البته قابل استفاده در حالت معمولی نیست)،

۶- ۴ پورت و به عبارتی ۳۲ پین ورودی خروجی دارد البته از آنجایی که از حافظه خارجی استفاده می گردد بیش از دو پورت آن قابل استفاده نیست.

۷- ۱۲۸ بیت قابل استفاده بصورت بیتی دارد که این بیتها ۱۶ بایت از حافظه *RAM* داخلی هستند.

۸- توانایی آدرس دهی ۶۴ کیلوبایت حافظه برنامه و ۶۴ کیلوبایت حافظه دیتا را دارد. یعنی حجم حافظه *EPROM* به ۶۴ کیلوبایت و حجم حافظه *RAM* نیز به ۶۴ کیلوبایت محدود می شود.

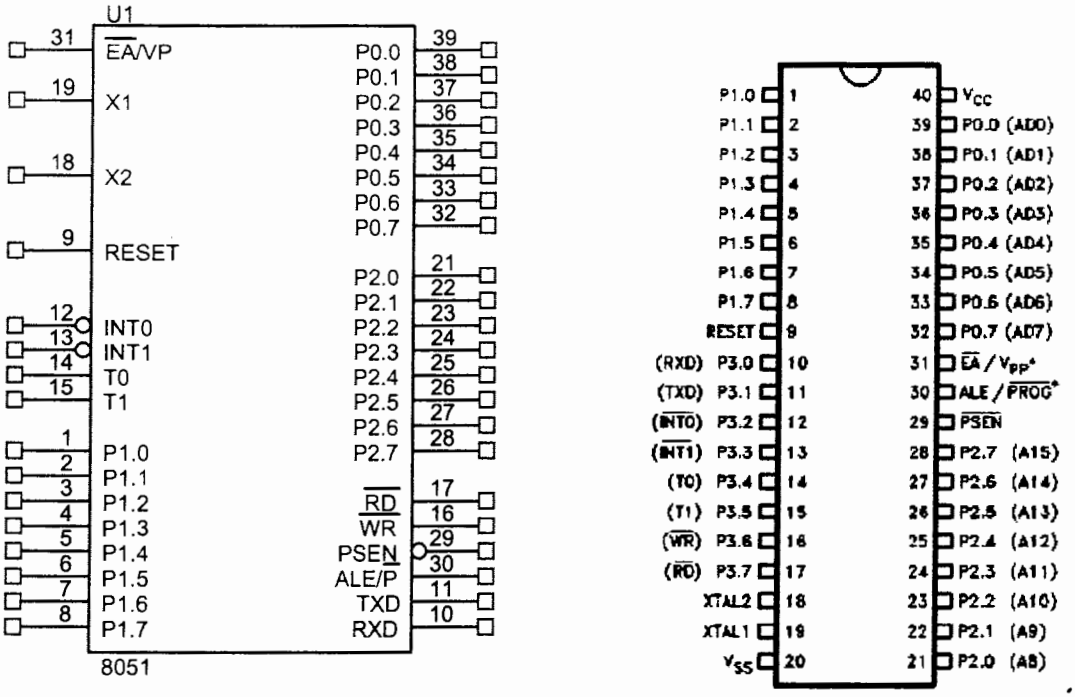
۹- *CPU* آن ۸ بیتی بوده و قادر است عملیات ضرب و تقسیم را انجام دهد.



شکل ۱ـ۲ ـ ساختار داخلی میکروکنترلر ۸۰۵۱

۴ـ۲ـ سخت افزار میکروکنترلر ۸۰۵۱

شـکل ۲ـ۲ پایـه هـای میکروکنترلر ۸۰۵۱ را به دو صورت ترتیبی و دسته بندی شده نمایش می دهد. وظایف این پایه ها و عملکرد آنها مورد بررسی و بحث قرار می گیرد.



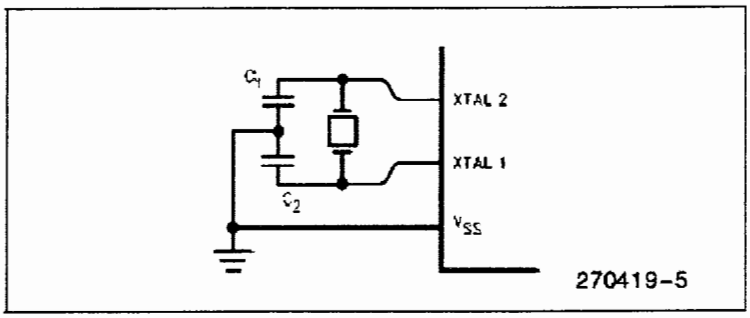
شکل ۲ـ۲ ـ ترتیب پایه های میکروکنترلر ۸۰۵۱

۱ـ تغذیه

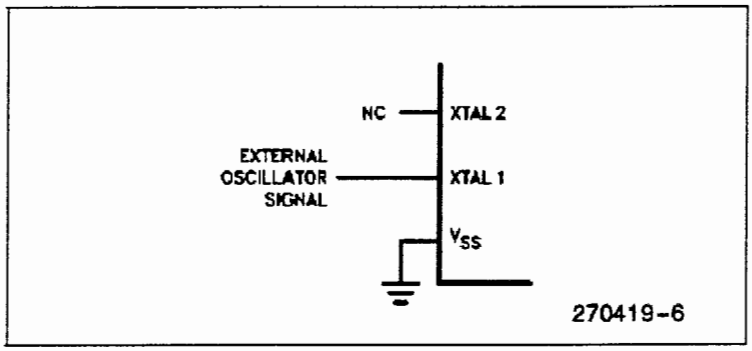
در ایـن میکروکنـترلر تغذیه بین پایه های ۲۰ و ۴۰ قرار می گیرد پایه ۲۰ به *GND* و پایه ۴۰ به *VCC* وصـل می گردد. *VCC*، ۵ ولت بوده و بایستی از کیفیت خوبی برخوردار باشد و بیش از *5%* رایپل نداشته باشد و جریان لازم را در تمام حالات تأمین نماید.

۲- پالس ساعت (پایه های ۱۸و۱۹)

این پایه ها جهت اتصال به کریستال نوسان ساز بکار می روند که با مدارات داخلی، **پالس** ساعت سیستم را تولید می کند. به دو صورت کلاک سیستم را می توان تأمین کرد یکی با استفاده از گذاشتن کریستال خارجی مانند شکل ۳ـ۲ و دیگر آنکه پالس ساعت خارجی را که به روشی دیگر و یا توسط میکروکنترلری دیگر تأمین شده است از طریق پایه ۱۹ به میکروکنترلر مانند شکل ۴ـ۲ اعمال کرد. در شکل ۳ـ۲ خازنهای *C1* و *C2* حدود ۲۲ پیکوفاراد انتخاب می گردند و حد ماکزیمم و مینیمم کریستال می تواند با توجه به مشخصات ارائه شده توسط سازنده میکروکنترلر انتخاب گردد.



شکل ۳ـ۲ ـ نحوه اتصال کریستال به ۸۰۵۱

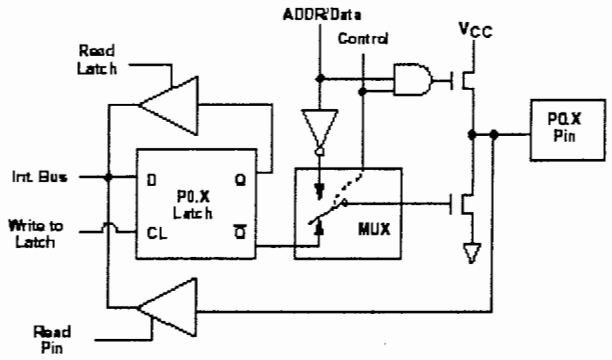


شکل ۴ـ۲ ـ اعمال کلاک خارجی به میکروکنترلر

۳- درگاه های موازی (*P3,P2,P1,P0*)

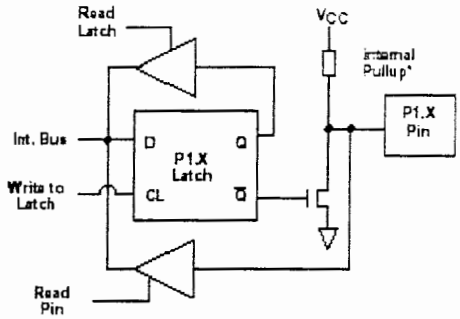
در میکروکنترلر ۸۰۵۱ درگاهها یا پورتهای ورودی خروجی ۸ بیتی با نامهای *P0,P1,P2,P3* منظور شده است که با هم ۳۲ پایه *I/O* را می سازند. که می توانند علاوه بر پایه های ورودی و خروجی عمومی برای منظورهای خاص نیز بکار روند.

P0 : پایه های ۳۲ تا ۳۹ که وضعیت آنها در شکل ۵ـ۲ نشان داده شده است پورت *P0* را می سازند. این پورت علاوه بر درگاه عمومی *I/O* در سیستم های با حافظه خارجی (چه حافظه برنامه و چه حافظه دیتا) می توان بعنوان باس داده (*D0* تا *D7*) و یا باس کم ارزش آدرس (*A0* تا *A7*) استفاده کرد. خطوط آدرس و دیتا روی این خط بصورت مالتی پلکس بوده که توسط پایه *ALE* و با استفاده از یک *Latch* ، ۸ بیتی قابل تفکیک از یکدیگر می باشند.



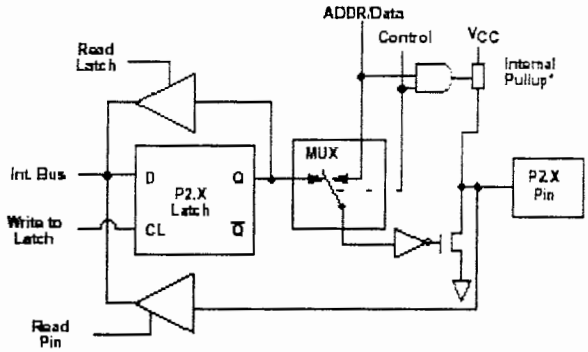
شکل ۵ـ۲ ـ وضعیت پینهای خروجی پورت *P0*

P1 : پایه های ۱ تا ۸ که وضعیت پینهای آن در شکل ۶ـ۲ نشان داده است درگاه *I/O* عمومی را تشکیل می دهند. این شکل نشان می دهد پینهای مربوطه توسط یک مقاومت داخلی *Pull up* شده اند و خواندن از پین و خواندن از لچ متفاوت بوده و عملکرد پورت را بهبود می دهد. همچنین با توجه به شکل ۶ـ۲ مشخص است که اگر لازم باشد از پورت بعنوان ورودی استفاده گردد بایستی در *Latch* یک نوشته شده و تراتزیستور متصل به *Latch* خاموش باشد زیرا اگر این ترانزیستور روشن باشد همیشه از پین صفر خوانده خواهد شد.



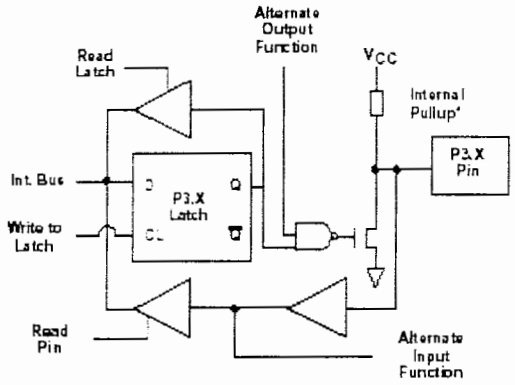
شکل ۶ـ۲ ـ وضعیت پینهای خروجی پورت *P1*

P2 : پایه های ۲۱ تا ۲۸ مانند شکل ۸ت۲ یک درگاه دو منظوره را تشکیل می دهند. این پایه ها می تواند به عنوان یک درگاه ۸ بیتی عمومی *I/O* بکار رود و یا در سیستم هائی که دارای بیش از ۲۵۶ بایت حافظه خارجی (دیتا یا برنامه) باشند به عنوان بایت بالای آدرس (*A8* تا *A15*) مورد استفاده قرار گیرند.



شکل ۷ـ۲ ـ وضعیت پینهای خروجی پورت *P2*

P3 : پایه های ۱۰ تا ۱۷ که ساختار ارتباطی آنها با *Latch* داخلی *P3* در شکل ۸ـ۲ نشان داده شده است علاوه بر یک درگاه عمومی *I/O* هر یک از آنها عملکرد دیگری مانند جدول ۱ـ۲ می توانند داشته باشد. وظایف پایه های *P3.6* و *P3.7* هنگامی که حافظه خارجی وصل شده است بصورت *WR* و *RD* تغییر کرده و از آنها بعنوان ورودی و خروجی نمی توان استفاده کرد، پایه های *P3.0* و *P3.1* هنگام استفاده از پورت سریال، *P3.2* و *P3.3* هنگام استفاده از پورت خارجی و پایه های *P3.4* و *P3.5* هنگام استفاده از کانتر کاربرد دارند.



شکل ۸ـ۲ ـ وضعیت پینهای خروجی پورت *P3*

جدول ۱ـ۲ ـ عملکرد ثانویه پورت ۳

Pin	Name	Alternate Function
P3.0	RXD	Serial Input Line
P3.1	TXD	Serial Output Line
P3.2	$\overline{INT0}$	External Interrupt 0
P3.3	$\overline{INT1}$	External Interrupt 1
P3.4	T0	Timer 0 External Input
P3.5	T1	Timer 1 External Input
P3.6	$\overline{WR}$	External Data Memory Write Strobe
P3.7	$\overline{RD}$	External Data Memory Read Strobe

۴- $\overline{PSEN}$ (پایه ۲۹ ، *Program Stor Enable*)

وقتی برنامه از حافظه خارجی اجرا می شود ، میکروکنترلر در زمان هایی که لازم است عمل خواندن کد دستورالعمل از حافظه برنامه را انجام دهد این سیگنال را فعال (صفر) می کند، این پایه خروجی به پایه *OE* حافظه متصل و در زمانهای خواندن کد دستورالعمل، حافظه را فعال می کند.

۵- *ALE* (پایه ۳۰ ، *Address Latch Enable*)

همانطور که قبلاً گفته شد روی درگاه *P0* باس داده و باس آدرس بصورت مالتی پلکس قرار گرفته اند. اینکه در هر لحظه باس به کدامیک اختصاص دارد توسط پایه خروجی *ALE* مشخص می شود . وقتی *ALE* فعال (*High*) شود به مفهوم این است که میکروکنترلر آدرس معتبری روی باس قرار داده است. می توان در لبه پایین رونده *ALE* بایت کم ارزش آدرس را در یک ثبات خارجی مانند *74HC373* ذخیره کرد، تا میکروکنترلر بتواند از همین باس به عنوان داده یا کد در لحظه بعد استفاده نماید .

۶- $\overline{EA}$ (پایه ۳۱ ، *External Access*)

اگر لازم باشد از حافظه برنامه داخلی استفاده گردد بایستی این پایه را غیر فعال (*High*) کرد در این صورت شروع حافظه برنامه از داخل بوده و ادامه آن از حافظه خارجی خواهد بود بعنوان مثال اگر

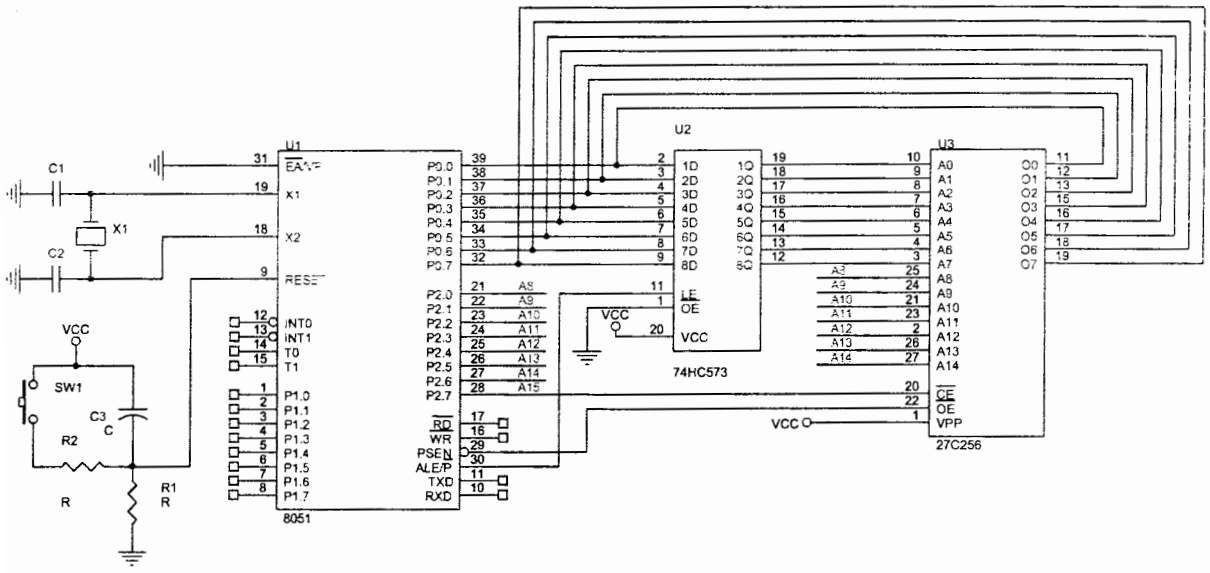
میکروکنترلر از نوع ۸۰۵۱ باشد که ۴ کیلوبایت حافظه داخلی دارد ۴ کیلوبایت اول از حافظه داخلی و مابقی از حافظه خارجی خوانده می شود . با فعال کردن این پایه (*Low* کردن آن) شروع حافظه برنامه که آدرس صفر می باشد از حافظه برنامه خارجی بوده و حافظه برنامه داخلی بلا استفاده خواهد ماند .

۷- ***RST*** (پایه ۹ ***RESET***)

با فعال کردن این پایه به عبارت دیگر *High* کردن آن حداقل به مدت دو سیکل کاری ماشین، رجیسترهای داخلی میکروکنترلر با مقادیر مناسبی پر شده و میکروکنترلر از آدرس *0000H* شروع به اجرای برنامه از حافظه برنامه می کند. لذا اگر از حافظه خارجی استفاده گردد بایستی در آدرس *0000H* حافظه *EPROM* یا *EEPROM* قرار گرفته و در آن دستورالعملهایی قرار گیرد که عملکرد میکروکنترلر را تحت نظارت در آورد.

۵ـ۲ـ اتصال حافظه برنامه به میکروکنترلر

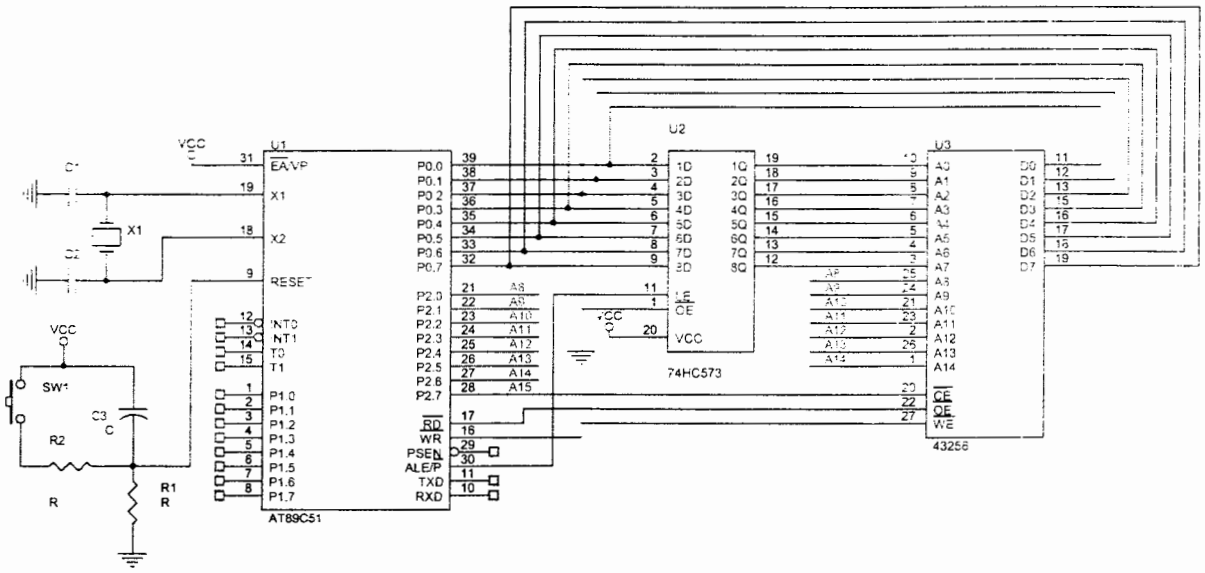
شکل ۹ـ۲ نحوه اتصال کریستال، مدار ریست و اتصال حافظه برنامه به ظرفیت ۳۲ کیلوبایت را به میکروکنترلر ۸۰۵۱ که از حافظه برنامه داخلی آن استفاده نمی شود نشان می دهد. لذا با توجه به این شکل اگر در میکروکنترلر ۸۰۵۱ یا ۸۹۵۱ از حافظه خارجی استفاده گردد تنها ۱۶ پایه ورودی خروجی در اختیار خواهد بود و مابقی استفاده خواهند شد.



شکل ۹ـ۲ ـ اتصال حافظه *EPROM* به میکروکنترلر ۸۰۵۱

۶ـ۲ـ اتصال حافظه دیتا به میکروکنترلر

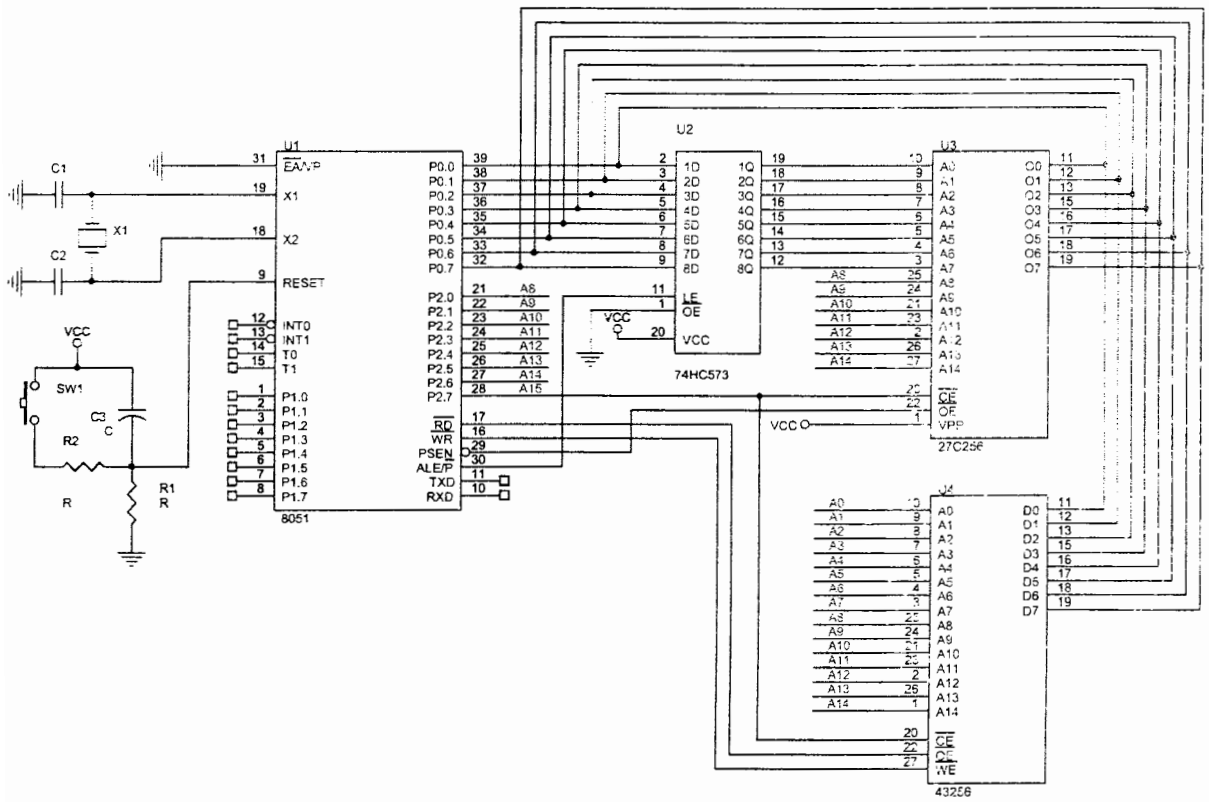
در شکل ۱۰ـ۲ فرض شده است از میکروکنترلر ۸۹۵۱ که حافظه برنامه داخلی دارد استفاده شده باشد و به جهت نوع کاربرد لازم است حافظه دیتا یا *RAM* خارجی به سیستم اضافه گردد. در این حالت تنها ۱۴ پایه میکروکنترلر آزاد بوده و از آنها بعنوان ورودی یا خروجی می توان استفاده کرد.



شکل ۱۰ـ۲ ـ اتصال حافظه دیتا به میکروکنترلر ۸۹۵۱

۷ـ۲ـ اتصال حافظه برنامه و حافظه دیتا به میکروکنترلر

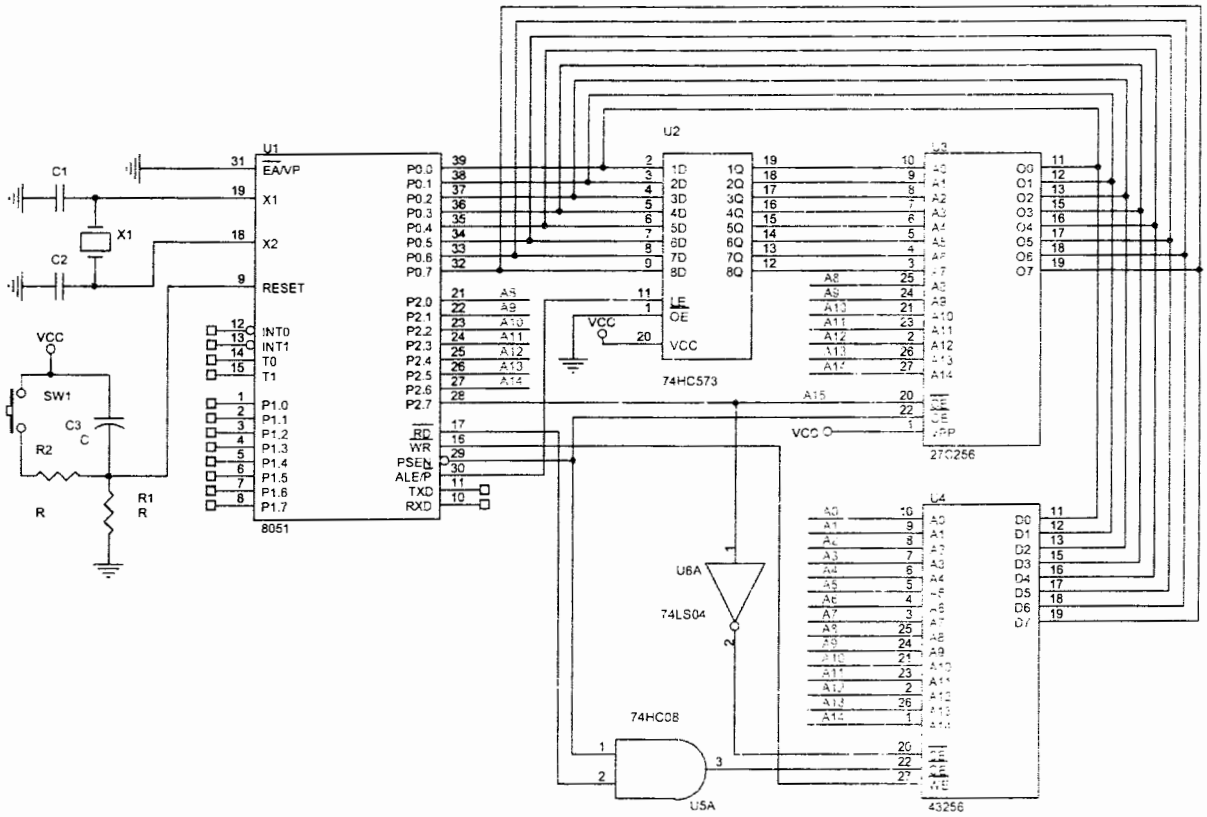
شکل ۱۱ـ۲ نحوه اتصال حافظه برنامه و حافظه دیتا هریک به ظرفیت ۳۲ کیلوبایت را به میکروکنترلر ۸۰۵۱ نشان می دهد در این حالت نیز ۱۴ پایه ورودی خروجی میکروکنترلر آزاد بوده و از آنها بعنوان پینهای ورودی و خروجی می توان استفاده کرد.



شکل ۱۱ـ۲ ـ اتصال حافظه برنامه و حافظه دیتا

۸ـ۲ـ اتصال حافظه برنامه و دیتا با قابلیت اجرای برنامه از روی حافظه *RAM*

با تغییری جزئی در شکل ۱۱ـ۲ می توان ترتیبی اتخاذ نمود که بتوان برنامه ای را روی حافظه *RAM* ذخیره کرد و سپس اجرا نمود. نوشتن برنامه خواندن از کامپیوتر و ذخیره روی *RAM* مشکل نبوده و براحتی این کار ممکن است. مدار طراحی شده در شکل ۱۲ـ۲ نشان داده شده است. این شکل نشان می دهد توسط *RD* و *PSEN* می توان از حافظه *RAM* خواند بعبارت دیگر هم بعنوان حافظه دیتا و هم بعنوان حافظه برنامه استفاده کرد.



شکل ۱۲ـ۲ ـ اتصال حافظه برنامه و دیتا به میکروکنترلر با قابلیت اجرای برنامه از روی *RAM*

۹ـ۲ـ سازمان دهی حافظه[1]

منظور از سازماندهی حافظه چگونگی دسترسی *CPU* به حافظه های *ROM/EPROM* و *RAM* و آدرس دهی آنها می باشد. در رابطه با دسترسی میکروپروسسور به حافظه می توان از دو روش استفاده کرد. در روش اول کل فضای آدرس دهی حافظه مشخص شده و حافظه های *EPROM* و *RAM* در مکانهای مختلف که بعضی از آنها را ساختار میکروپروسسور مشخص می کند قرار می گیرند. بعنوان مثال میکروپروسسور *Z80* حدود ۶۴ کیلوبایت فضای آدرس دهی داشته که می توان در فضاهای بین *0000h* الی *FFFFh* حافظه *EPROM* و *RAM* قرار داد اما از آنجایی که با ریست نمودن میکروپروسسور اولین آدرس خواندن حافظه *0000h* می باشد در این محدوده بایستی حافظه *EPROM* قرار گیرد. لذا می توان در این میکروپروسسور بعنوان مثال ۸ کیلوبایت حافظه *EPROM* و بقیه یا قسمت کمتری از فضای

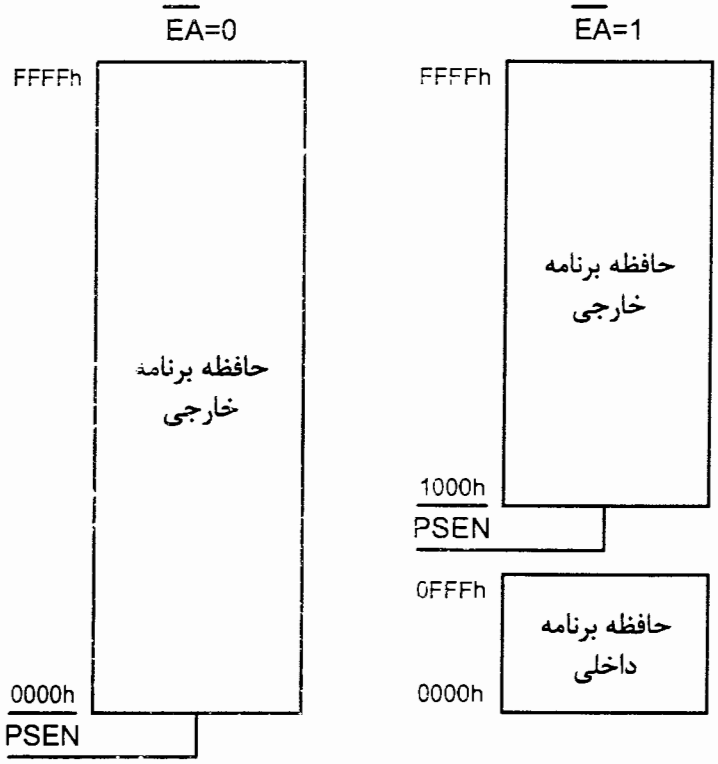
[1] *Memory Organization*

باقیمانده را *RAM* استفاده کرد اینکه در هرزمان چه حافظه ای فعال باشد دیکدر بکار رفته تعیین خواهد کرد.

در روش دوم که بعضی از میکروپروسسورها مانند *8051* از آن استفاده می کنند طوری طراحی شده اند که دستور رجوع به حافظه برنامه یعنی *ROM* یا *EPROM* را از رجوع به حافظه داده یعنی *RAM* تشخیص داده که همین امر سبب استفاده از فضای متفاوت حافظه برنامه و داده شده در نتیجه میتوان از فضای آدرس دهی بیشتری استفاده کرد. با این تکنیک فضای حافظه برنامه ۶۴ کیلوبایت و فضای حافظه دیتا ۶۴ کیلوبایت می گردد. بعبارت دیگر در این میکرو کنترلر برای استفاده بهتر از حافظه، فضای حافظه برنامه (*EPROM*) و فضای حافظه داده (*RAM*) متفاوت بوده اما خطوط آدرس و دیتا در حافظه داده و برنامه مشترک می باشند و تنها در سه خط کنترلی با هم اختلاف دارند.

۱ـ۹ـ۲ـ حافظه برنامه[1]

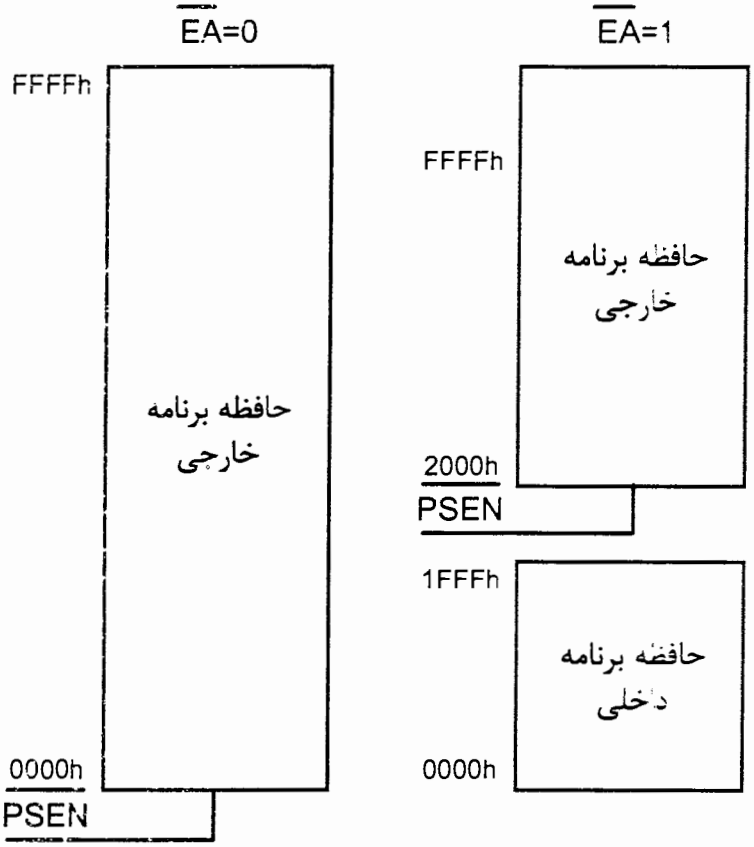
میکرو کنترلر *8051* طوری طراحی شده است که در رابطه با *ROM* و *EPROM* دو حالت مختلف دارد. حالت اول این است که از حافظه داخلی میکروکنترلر استفاده شود و در حالت دوم این است که از این حافظه استفاده نشود. در حالت اول ۴ کیلوبایت اول از حافظه داخلی و مابقی از حافظه خارجی استفاده می گردد و در حالت دوم تمام حافظه برنامه از حافظه خارجی استفاده گردد و از حافظه داخلی برفرض وجود استفاده نگردد. میکروکنترلر *8051* مانند شکل ۱۳ـ۲ حالت اول و دوم را به وسیله پایه *EA* از هم تشخیص می دهد.



شکل ۱۳ـ۲ ـ تقسیم بندی حافظه برنامه با توجه به پین *EA* در ۸۰۵۱

[1] *Program memory*

اگر این پایه به زمین وصل شده باشد حافظه برنامه داخلی غیر فعال و *CPU* یکسره حافظه برنامه خارجی را آدرس دهی می کند و اگر پایه *EA* یک باشد ابتدای حافظه برنامه داخلی و مابقی از حافظه خارجی استفاده می گردد. بعبارت دیگر در میکروکنترلر ۸۰۵۱ برای آدرس های کمتر از *0FFFH* حافظه برنامه داخلی و برای آدرس های بزرگتر از *1000H* به حافظه برنامه خارجی با فعال کردن پایه *PSEN* رجوع می کند. این وظیفه طراح است که با توجه به مداری که طراحی کرده است این پایه را به زمین یا *VCC* وصل کند. اگر میکروکنترلر ۸۰۵۲ باشد میزان حافظه داخلی متفاوت بوده و تقسیم بندی مطابق شکل ۱۴ـ۲ خواهد بود. لازم بذکر است هنگام استفاده از حافظه برنامه داخلی و سپس حافظه برنامه خارجی بایستی دیکدر لازم بنحوی طراحی گردد که حافظه خارجی با حافظه داخلی روی هم افتادگی نداشته باشد در غیر اینصورت حافظه خارجی روی هم افتاده کاربردی نخواهد داشت.



شکل ۱۴ـ۲ ـ تقسیم بندی حافظه برنامه با توجه به پایه *EA* در ۸۰۵۲

۲ـ۹ـ۲ـ حافظه داده[1]

این حافظه که در میکروکنترلر *8051* که به چند صورت تفکیک می شود در شکل ۱۵ـ۲ نشان داده شده است :

*64*کیلو بایت حافظه داده ی خارجی از آدرس *0000H* الی *FFFFH*

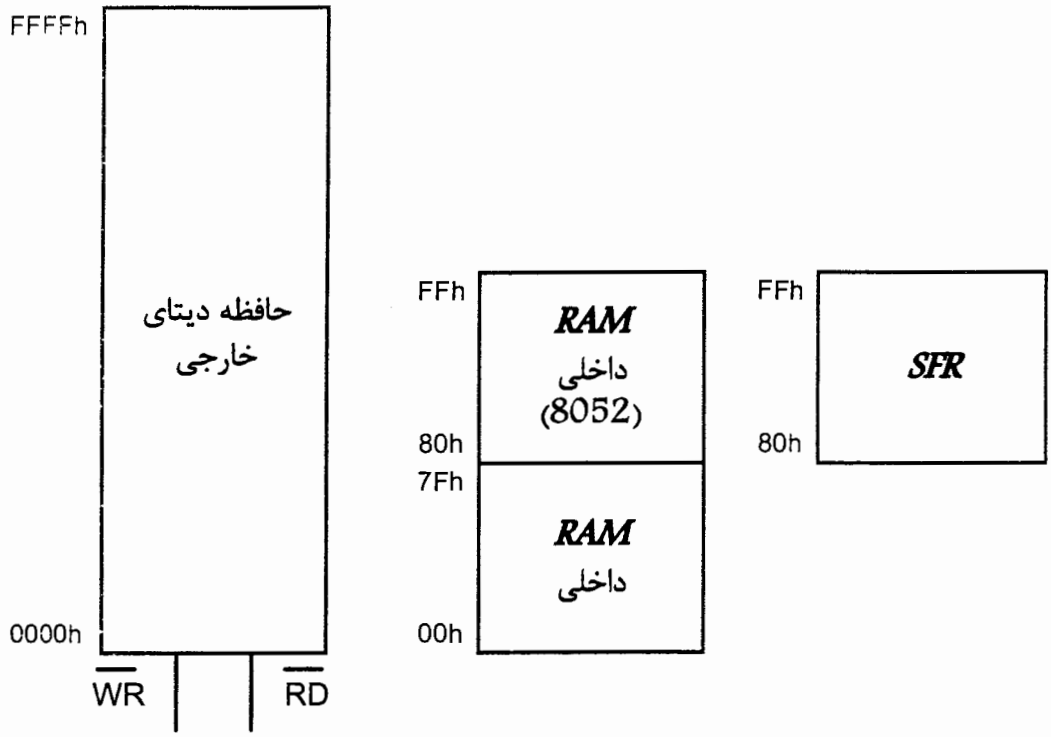
دارای *256* بایت حافظه داده ی داخلی که به دو بخش تقسیم می شود :

[1] *Data memory*

الف) دارای *128* بایت (یا *256* بایت برای *8052*) حافظه داده ی داخلی از آدرس صفر تا *7FH* (با از آدرس صفر تا *FFH* در *8052*)

ب) دارای *128*بایت حافظه داده داخلی تحت نام *SFR*[1] از آدرس *80H* تا *FFH*

چنانچه دیده می شود برای حافظه دیتا، دیگر دو وضعیت متمایز وجود ندارد بلکه قسمتی از آدرس حافظه دیتای داخلی با حافظه دیتای خارجی یکی است .(*256* بایت اول) و همچنین این آدرس برای حافظه برنامه هم استفاده می شود .



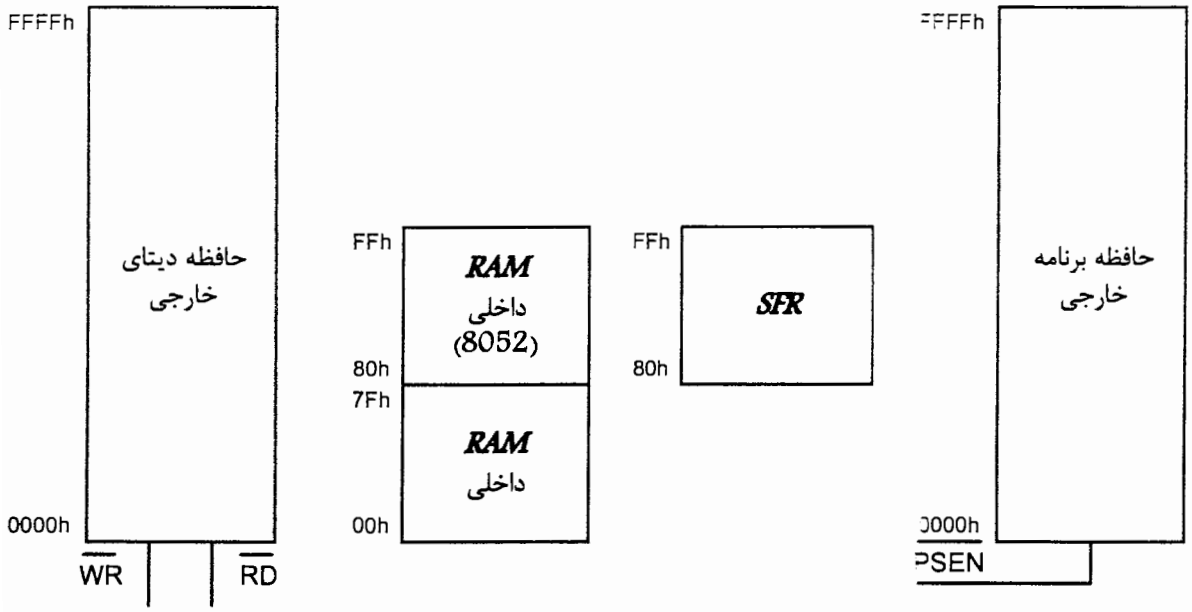
شکل ۱۵ـ۲ـ حافظه دیتا در میکروکنترلر ۸۰۵۱

شکل ۱۶ـ۲ حافظه برنامه و حافظه دیتا را بصورت یکجا در میکروکنترلر ۸۰۵۱ نشان می دهد همانطور که مشاهده می شود یک آدرس بخصوص مثلاً *45h* برای چند منظور استفاده خواهد شد.

همانطور که اشاره شد در این میکروکنترلر حافظه برنامه خارجی، حافظه برنامه داخلی، حافظه داده خارجی، حافظه داده داخلی وجود دارند که بایستی به روشهای سخت افزاری و نرم افزاری آنها را از یکدیگر تفکیک نمود.

روش سخت افزاری بیان می کند پین *EA* جهت استفاده از حافظه برنامه داخلی یا خارجی چه وضعیتی داشته باشد و روش نرم افزاری بیان می کند از چه دستورالعملهای جهت مشخص کردن نوع حافظه استفاده گردد. بعنوان مثال دستورالعمل *MOVX @DPTR,A* نوشتن دیتا در حافظه داده خارجی، دستورالعمل *MOV @R0,A* نوشتن دیتا در حافظه داده داخلی و دستورالعمل *MOVX @R0,A* نوشتن داده در حافظه داده خارجی را بصورت آدرس ۸ بیتس مشخص می سازد.

[1] *Special Function Rgister*



شکل ۱۶ـ۲ ـ حافظه برنامه و حافظه دیتا بصورت یکجا در میکروکنترلر ۸۰۵۱

۱۰ـ۲ـ رجیسترهای داخلی میکروکنترلر

رجیسترهای داخلی یا بعبارت دیگر حافظه داخلی میکروکنترلر ۸۰۵۱ را می توان به دو گروه اصلی زیر تقسیم کرد :

الف) رجیسترهای عمومی : رجیسترهای عمومی در واقع همان *RAM* داخلی هستند که به علت تعداد زیاد آنها ، به جای اسم مشخص آدرسی بین *00h* الی *7Fh* برای آنها منظور شده است.

ب) رجیسترهای خاص (رجیسترهای *SFR*)

این رجیسترها علاوه بر این که رجیسترهای معمولی هستند هر کدام کاربردی خاص نیز دارند. این رجیسترها، رجیسترهای مهم *CPU* را تشکیل می دهند و همان طوری که در بخش حافظه مشاهده گردید بخشی از حافظه داخلی را تشکیل می دهند (از آدرس *80H* الی *FFH*) که به صورت مستقیم قابل دسترسی هستند و کافی است در دستورات اسم آنها یا آدرس آنها ذکر گردد. همانطور که اشاره شد تنها بعضی از این ۱۲۸ بایت قابل استفاده اند و مابقی قابل استفاده نیستند.

۱۱ـ۲ـ رجستر های عمومی

از قسمتهای داخلی هر میکروکنترلر که استفاده کننده باید بداند و در برنامه نویسی آنها را مد نظر داشته باشد فضای حافظه داخلی، رجستر های داخلی و نحوه دسترسی به آنهاست. رجسترها و حافظه *RAM* داخلی ۸۰۵۱ در فضایی بین *00H* الی *7FH* قرار گرفته اند. از این حافظه ها جهت ذخیره موقت اطلاعات مانند یک رجستر یک بایتی می توان استفاده کرد. ساختار این حافظه با جزئیات در شکل ۱۷ـ۲ نشان داده شده است.

شکل ۱۷ـ۲ ـ ساختار حافظه *RAM* داخلی در ۸۰۵۱

همـانطور که این شکل نشان می دهد این فضای حافظه که بصورت مستقیم و غیر مستقیم قابل آدرس دهی است به سه گروه زیر تقسیم می گردد.

گروه اول آنهایی هستند که مانند یک رجستر یک بایتی عمل کرده و می توان در آنها نوشت و یا از آنها خواند و قابلیت دیگری ندارند. یعنی نامی خاص به آنها اختصاص داده نشده و بصورت بیتی قابل آدرس دهی نیستند. این گروه از فضای *30h* شروع و به *7Fh* خاتمه پیدا می کنند.

گروه دوم آنهایی هستند که ۱۶ بایت بوده بصورت بیتی قابل آدرس دهی هستند و از خانه *20H* شروع شده تا *2FH* ادامه دارند. ویژگی خاص این گروه این است که به صورت بیتی قابل آدرس دهی هستند. نحوه آدرس دهی آنها به این صورت است که به بیت های هر بایت شماره ای اختصاص می دهند. بدین ترتیب که بیت کم ارزش خانه *20H* را *00H* وبیت بعدی را *01H* و به همین ترتیب بیت آخر خانه *20H* را *07H* می نامند و شماره بیت های خانه *21H* به دنبال شماره های خانه *20H* قرار می گیرد. یعنی بیت کم ارزش خانه ۲۱ دارای شماره *08H* می باشد و آدرس بیت دوم آن *09H* و به همین ترتیب تک تک بیت های این *16* بایت هر کدام شماره ای از *00H* الی *7FH* خواهند داشت .

برای صفر و یک کردن و یا تست کردن و یا هر عمل دیگری روی آنها کافی است آدرس آن بیت که عددی بین *00h* الی *7Fh* است در دستورالعمل قید گردد. به عبارت دیگر هر بیت به طور جدا مستقل از بقیه بیت ها قابل تغییر و تست کردن است که به این خاصیت به صورت بیتی قابل آدرس دهی کردن[1] می گویند و با مجموعه دستورات بیتی یا عملیات بیتی[2] کار می کنند. علاوه بر استفاده بصورت بیتی از این ۱۶ بایت بصورت بایتی هم می توان استفاده کرد به جهت سادگی آدرس دهی می توان برای

---

[1] *Bit Addressable*

[2] *Boolean Manipulation*

هربیت مورد استفاده یک نام خاص انتخاب و در نرم افزار از آن استفاده کرد. در قسمت نرم افزار این موضوع بیشتر مورد بررسی قرار می گیرد.

گروه سوم ۳۲ بایت اول حافظه *RAM* داخلی از آدرس *00H* الی *1FH* می باشد که شامل بانک های ثبات بوده و به چهار گروه ۸ بایتی تقسیم می گردند و در هر لحظه فقط ۸ بایت از این ۳۲ بایت قابل دسترسی می باشند که به *R0,R1,R2, ... ,R7* نشان داده می شوند و در برنامه نویسی با این اسامی به آنها استناد شده و استفاده می شوند. علاوه براین می توان به صورت مستقیم در هر لحظه هر کدام از این ۳۲ بایت را مانند بقیه فضای *RAM* داخلی آدرس دهی کرد. با مقدار دهی ۲ بیت خاص از رجستر *PSW* (بعداً تشریح می گردد) می توان *R0* تا *R7* را به فضای *00H* تا *07H* یا *08H* تا *0FH* یا *10H* تا *17H* یا *18H* تا *1FH* حافظه داخلی نسبت داد.

در این میکروکنترلر بین رجسترهای *R0* و *R1* با رجسترهای *R2* تا *R7* تفاوتهایی وجود دارد یعنی رجسترهای *R0* و *R1* قابلیتهای بیشتری دارند و در دستورالعملهای بیشتری شرکت می کنند. رجسترهای *R0* و *R1* را می توان مقدار داد و در عملیات محاسباتی و منطقی دستورالعمل بنحوی وارد گردد که محتویات جایی که *R0* و *R1* اشاره می کند در عملیات منطقی و ریاضی شرکت کند بعنوان مثال دستورالعمل *MOV @R0,#55H* درست است اما دستورالعمل *MOV @R3,#55H* غلط است.

چرا چهار بانک رجستر تعریف شده است؟

اگر رجسترها بصورت *R0* الی *R31* تعریف می شدند و از عبارتی به نام بانک استفاده نمی شد فهم آنها خیلی ساده تر بود اما چرا اینکار را نکرده اند؟ این سؤالی است که اغلب برای استفاده کننده پیش می آید و دانستن آن خالی از لطف نیست. به چند دلیل این کار انجام شده است:

۱- رجسترها بصورت *R0* تا *R7* مشخص شده اند لذا با ۳ بیت می توان آنها را از یکدیگر تفکیک کرد. حال اگر این رجسترها در دستورالعملها استفاده شوند اختلاف کد دستورالعمل آنها فقط در سه بیت خواهد بود بعنوان مثال کد دستورالعمل *MOV A,R0* برابر *68H* و کد دستورالعمل *MOV A,R7* برابر *6FH* است که تنها در سه بیت کم ارزش با هم اختلاف دارند.

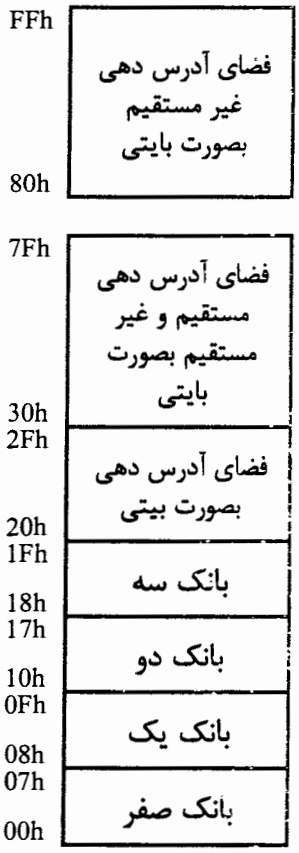
اگر رجسترها بصورت *R0* الی *R31* تعریف می شدند با یک بایت یعنی ۲۵۶ حالت مختلف نمی توان کل دستورالعملهای ۸۰۵۱ را پوشش داد در نتیجه لازم می شد کد دستورالعمل بجای یک بایت دو بایت تعریف گردد و این روند اجرای برنامه را کند می کرد. یعنی بعنوان مثال انتقال محتویات این رجسترها به آکومولاتور به جای ۸ کد مختلف ۳۲ کد مختلف می شد و همچنین برای دستورات دیگری چون *INC Rn* ، *DEC Rn* ، *ADD A,Rn* ، *ORL A,Rn* و غیره افزایش کد پیش می آمد و در نتیجه ۲۵۶ کد مختلف جوابگو تمامی دستورالعملها نبود پس استفاده از بانک به این صورت طول کد دستورالعملها را کاهش داده است. در واقع اختلاف سه بیت و پنج بیت در دو بیت رجستر *PSW* منظور شده است.

۲- دلیل دیگر اینست که می توان رجستر های هر بانک را به یک انتراپت خاص و یا یک زیر برنامه خاص نسبت داد و هنگام شروع روتین انتراپت و یا شروع زیر برنامه بجای اینکه در بقیه میکروپروسسورها رجسترها با دستورات متعددی در پشته قرار گیرد تنها کافی است رجستر *PSW* متناسب با آن انتراپت یا

زیر برنامه تغییر کند و بدون هیچ مشکلی از *R0* الی *R7* استفاده گردد. با توجه به دو دلیل فوق استفاده از بانک قابل توجیه است.

در میکرو کنترلر ۸۰۵۲ مانند شکل ۱۸ـ۲ بجای ۱۲۸ بایت *RAM* داخلی ۲۵۶ بایت *RAM* داخلی با فضای آدرس دهی *00H* الی *FFH* پیش بینی شده است که در ۱۲۸ بایت اول با ۸۰۵۱ مشترک می باشد. از آنجایی که از آدرس *80H* الی *FFH* هم فضای آدرس دهی رجسترهای خاص و هم فضای بالای حافظه است بایستی بنحوی این دو را از یکدیگر تفکیک نمود. به این جهت دسترسی به حافظه های داخلی به دو صورت مستقیم و غیر مستقیم پیش بینی شده است. آدرس دهی رجسترهای خاص بصورت مستقیم و آدرس دهی حافظه بالا بصورت غیر مستقیم منظور شده است و از طرفی آدرس دهی حافظه پایین هم بصورت مستقیم و هم غیر مستقیم ممکن است. در آدرس دهی غیر مستقیم ابتدا بایستی در رجستر *R0* یا *R1* آدرس حافظه مورد نظر را قرار داده و سپس از آن رجستر برای آدرس دهی استفاده کرد در صورتیکه در آدرس دهی مستقیم استفاده از رجستر دیگری لازم نیست.

بعنوان مثال دستور *MOV 45H,#55H* آدرس دهی مستقیم می باشد و مفهوم آن اینست که عدد *55H* را در مکان *45H* حافظه قرار دهد و دستور *MOV @R0,#55H* آدرس دهی غیر مستقیم می باشد و مفهوم آن اینست که عدد *55H* را در جایی که رجستر *R0* اشاره می کند قرار دهد.



شکل ۱۸ـ۲ ـ ساختار حافظه *RAM* داخلی در ۸۰۵۲

۱۲ـ۲ـ رجسترهای با کاربرد خاص

در میکروکنترلر ۸۰۵۱ مکانهایی که در آدرسهایی بین *80h* الی *FFh* قرار گرفته اند برای رجسترهای با کاربرد خاص منظور نموده اند از این ۱۲۸ مکان مختلف تعداد کمی از آنها که در شکل ۱۹ـ۲ نشان داده شده اند استفاده شده است و مابقی رزرو بوده و کاربر نمی تواند از آنها برای مقاصد خاص استفاده کند بعبارت دیگر نوشتن در آنها و خواندن از آنها مقدار معتبری را در اختیار نمی گذارد. بعضی از این مکانها که در شکل ۱۹ـ۲ در ستون سمت چپ قرار گرفته اند و آدرس آنها به *X0h* و *X8h* ختم می شود بصورت بیتی قابل آدرس دهی هستند و آدرس این بیتها از *80h* شروع و به *FFh* ختم می شود در ادامه بعضی از این رجسترها توضیح داده می شود.

	F8h	F9h	FAh	FBh	FCh	FDh	FEh	FFh	
F8h									FFh
F0h	B								F7h
E8h									EFh
E0h	ACC								E7h
D8h									DFh
D0h	PSW								D7h
C8h	T2CON		RCAP2L	RCAP2H	TL2	TH2			CFh
C0h									C7h
B8h	IP								BFh
B0h	P3								B7h
A8h	IE								AFh
A0h	P2								A7h
98h	SCON	SBUF							9Fh
90h	P1								97h
88h	TCON	TMOD	TL0	TL1	TH0	TH1			8Fh
80h	P0	SP	DPL	DPH				PCON	87h

شکل ۱۹ـ۲ـ رجسترهای با کاربرد خاص در میکروکنترلر ۸۰۵۱

۱ـ ثبات اکومولاتور (*ACCUMULATOR*)

اکومولاتور یا *ACC* که به اختصار در دستورات بصورت *A* هم نوشته می شود و به آن انباره نیز گفته می شود یک رجیستر ۸ بیتی بوده که تقریباً بیشتر عملیات انتقال و منطق و شیفت روی آن انجام می گیرد و نسبت به سایر دستورات سریعتر اجرا شده و تعداد بایتهای کمتری دارد. آدرس این رجیستر *E0H* بوده و بصورت بیتی قابل آدرس دهی می باشد.

۲ـ رجستر B

در فضای *SFR* رجستر دیگری بنام *B* با آدرس *F0h* وجود دارد که از آن می توان برای ذخیره موقت اطلاعات استفاده کرد و علاوه براین در دستورالعملهای ضرب و تقسیم نقش مهمی ایفا می کند.

۳ـ رجستر وضعیت برنامه (*PSW*)[1]

بیت های این رجستر با آدرس *D0H* تحت تأثیر بعضی عملیات های محاسباتی یا منطقی فعال شده و بصورت بیتی قابل آدرس دهی هستند و شامل بیت های زیر می باشد:

PSW:	CY	AC	F0	RS1	RS0	OV	--	P

پرچم نمایش پریتی (*P*): هنگام اجرای بعضی از دستورالعملها این بیت تغییر وضعیت می دهد. اگر این بیت یک باشد نشان می دهد یکهای موجود در رجستر *A* تعدادی فرد می باشد و اگر این بیت صفر باشد نشان می دهد یکهای موجود در رجستر *A* تعدادی زوج می باشد بعبارت دیگر این بیت توازن زوج را نشان می دهد.

بیت دوم یا بیت پرچم بدون نام : از این پرچم به منظور خاصی استفاده نشده و کاربر می تواند بعنوان یک بیت قابل برنامه ریزی استفاده کند.

پرچم سرریز (OV) : هرگاه نتیجه محاسبات بیشتر از بیتهای در نظر گرفته جا لازم داشته باشد این بیت یک می گردد بعنوان مثال جمع دو عدد علامت دار ۷۰+ و ۸۰+ برابر ۱۵۰+ می گردد در حالی که با ۷ بیت ماکزیمم عدد مثبتی که می توان نشان داد ۱۲۷ است لذا سرریز اتفاق افتاده است

پرچم بیت نقلی (CY) : اگر نتیجه محاسبات دستورالعمل قبلی بیش از ۸ بیت باشد این بیت یک می گردد. تا نشان دهد رقم نقلی بوجود آمده و در صورت لزوم بایستی آنرا در محاسبات بعدی دخیل نمود.

پرچم نقلی کمکی (*AC*) : اگر در نتیجه محاسبات نیبل اول بیت نقلی بوجود آمده باشد این بیت یک میگردد و بیشتر برای جمع و تفریق اعداد *BCD* استفاده می شود. بعنوان مثال هنگام جمع دو عدد *48h* و *69h* چون بیت چهارم هر دو یک است سرریز بوجود می آید لذا *AC* یک می گردد.

بیت های کنترل رجستر های بانک *RS0* و *RS1* : این دو بیت که در شکل ۲۰ـ۲ وضعیت مختلف آنها نشان داده شده است توسط برنامه نویس مقدار دهی می شوند و توسط آن مشخص می گردد که چه مکانهایی از حافظه به رجسترهای *R0* الی *R7* اختصاص پیدا کند.

بیت پرچم *F0* : این بیت نیز قابل دسترسی توسط کاربر است و از آن می تواند استفاده کند. بعنوان مثال می توان با ترکیب آن با بیت های *RS0* و *RS1* هنگام استفاده از بانکهای رجستر شرط و شروطی گذاشت و به قابلیتهای برنامه افزود بعنوان مثال اگر این بیت یک بود یک زیر برنامه یک و اگر صفر بود زیر برنامه ۲ اجرا گردد.

همانطور که مشاهده می شود *PSW* دارای پرچم صفر نیست و این مشکلی ایجاد نمی کند چون ۸۰۵۱ دارای دستورالعملی است که براحتی می تواند صفر بودن انباره را چک کند.

---

[1] *Program Status Word*

شماره بانک	RS1	RS0	R0	R1	R2	R3	R4	R5	R6	R7
بانک صفر	0	0	00h	01h	02h	03h	04h	05h	06h	07h
بانک یک	0	1	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
بانک دو	1	0	10h	11h	12h	13h	14h	15h	16h	17h
بانک سه	1	1	18h	19h	1Ah	1Bh	1Ch	1Dh	1Eh	1Fh

شکل ۲۰ـ۲ـ انتخاب فضای بانک با توجه به *RS0* و *RS1*

۴ـ رجستر اشاره گر پشته (*SP*)

در هر میکرو پروسسوری قسمتی از حافظه *RAM* داخلی یا خارجی را بعنوان پشته در نظر می گیرند. پشته برای ذخیره موقت اطلاعات استفاده می شود و وجود آن در دستورالعملهایی چون *PUSH*، *POP*، *CALL* و اجرای برنامه های انترایت ضروری است. در ۸۰۵۱ این رجستر محل قرارگیری اطلاعات را در پشته مشخص می کند و پس از ریست سیستم این رجستر مقدار *07H* را گرفته و در صورت تغییر ندادن آن اطلاعات *PUSH* شده را از مکان هشتم حافظه به بالا می نویسد لذا اگر از مکانهای بعدی به منظور دیگری استفاده شده باشد در بعضی از حالات مشکل ساز خواهد بود لذا بایستی ابتدا *SP* را مقدار مشخصی داد که روی قسمتهای استفاده شده حافظه اطلاعاتی را ننویسد. این مقدار می تواند بین صفر تا ۱۲۸ باشد. بعنوان مثال اگر *SP* مقدار *40H* قرار داده شود ۶۴ بایت یعنی از آدرس *40H* الی *7FH* برای پشته در نظر گرفته شده است و بایستی دقت نمود که زیر برنامه های تودرتو و انتراپتهای اعمالی و دستورات *PUSH* استفاده شده حافظه آی بیش از ۶۴ بایت نیاز نداشته باشند اگر این مقدار حافظه کافی نبود بایستی *SP* را مقداری کوچکتر داد البته بایستی به حافظه مورد نیاز بقیه برنامه نیز توجه نمود.

۵ـ رجستر اشاره گر داده (*DPTR*)

۸۰۵۱ دو رجستر ۱۶ بیتی قابل استفاده برای برنامه نویس دارد این دو رجستر اشاره گر داده (*DPTR*) و شمارشگر برنامه (*PC*) می باشند. از کنار هم قرار گرفتن رجسترهای *DPL* و *DPH* رجستر *DPTR* حاصل می شود. این رجستر جهت آدرس دهی فضای حافظه دیتا و خواندن اعدادی از حافظه برنامه استفاده می شود. دستورات زیر مثالهایی از کاربرد این رجستر را نشان می دهد.

محتویات *A* را در جایی که *DPTR* در حافظه دیتا اشاره می کند قرار می دهد.

MOVX @DPTR, A

از جایی که *DPTR* در حافظه دیتا اشاره می کند یک بایت را خوانده در *A* قرار می دهد

MOVX A,@DPTR

از حافظه برنامه جایی که مجموع *DPTR* و رجستر *A* اشاره می کند یک بایت خوانده شده در *A* قرار می گیرد.

MOVC A,@A+DPTR

قابل ذکر است از شمارشگر برنامه برنامه نویس تنها می تواند در خواندن اعدادی از حافظه برنامه استفاده کند.

۶ـ ثبات های درگاه

چهار پورت با اسامی *P0*، *P1*، *P2* و *P3* برای میکروکنترلر ۸۰۵۱ و چهار رجستر با اسامی مشابه در فضای *SFR* برای آن در نظر گرفته شده است که ارتباط تنگاتنگی با یکدیگر دارند. ارتباط بین پورتها و رجستر موجود در فضای *SFR* در شکلهای ۵ـ۲، ۶ـ۲، ۷ـ۲ و ۸ـ۲ نشان داده شد. در حالت عادی آنچه که در رجستر ذخیره می گردد روی پورت نوشته می شود اما در حالت خواندن با توجه به نوع دستورالعمل ممکن است از پورت یا رجستر خوانده شود. لذا عملکرد این رجسترها مطابق پایه های میکروکنترلر است که قبلاً توضیح داده شد.

۷ـ بافر داده سریال[1] (*SBUF*)

پورت سریال دو بافر یک بایتی دارد که یکی برای ارسال و دیگری برای دریافت استفاده می شود ولی آدرس آندو یکی است. اگر عددی از این بایت (*SBUF*) خوانده شود از بافر دریافت و اگر بایتی فرستاده شود در بافر ارسال قرار می گیرد.

۸ـ رجسترهای تایمر/کانتر

۸۰۵۱ دو تایمر/ کانتر ۱۶ بیتی به نامهای تایمر/کانتر صفر و تایمر کانتر یک دارد. تایمر /کانتر صفر از رجسترهای *TL0* و *TH0* و تایمر / کانتر یک از رجسترهای *TL1* و *TH1* استفاده می کنند.

در ۸۰۵۲ که به جای دو تایمر سه تایمر دارد رجسترهای *TL2* و *TH2* نیز پیش بینی شده است و علاوه بر آن دو رجستر ۸ بیتی به نام رجسترهای تسخیر کننده (*RCAP2L* و *RCAP2H*) وجود دارد که به منظور نگهداشتن یک نسخه اضافی از محتویات رجسترهای *TL2* و *TH2* بکار می روند. نحوه استفاده از این تایمرها بعداً توضیح داده خواهد شد.

۹ـ رجستر های کنترل

علاوه بر رجسترهای فوق رجسترهای *SFR* دیگری وجود دارند که برای کنترل و تعیین وضعیت وقفه، تایمر/ کانتر و درگاه پورت سریال میکروکنترلر بکار می روند. اگر در سخت افزاری از این قابلیتهای میکروکنترلر استفاده نشده باشد لزومی به مقدار دهی و برنامه ریزی آنها نیست اما اگر از این قابلیتها استفاده شده باشد تعیین مقدار آنها ضروری است. این رجسترها که نحوه برنامه ریزی و مقدار دهی آنها بعداً تشریح خواهد شد عبارتند از *IP* (حق تقدم وقفه)، *IE* (فعال ساز وقفه)، *TMOD* (تعیین کننده حالت تایمر)، *TCON* (کنترل تایمر)، *T2CON* (کنترل تایمر ۲ در ۸۰۵۲)، *SCON* (کنترل پورت سریال) و *PCON* (کنترل توان مصرفی در میکروکنترلرهایی که با تکنولوژی *CMOS* ساخته شده اند).

---

[1] *Serial Data Buffer*

۱۳ـ۲ـ تایمر یا کانتر در میکروکنترلر ۸۰۵۱

همانطور که اشاره شد میکروکنترلر ۸۰۵۱ دو تایمر/کانتر ۱۶ بیتی دارد. با برنامه ریزی میتوان مشخص کرد که تایمر یا کانتر(شمارنده) مد نظر است. تایمر به مفهوم اینست که فاصله زمانی بین دو رویداد (مثلاً تغییر وضعیت یکی از پایه ها) اندازه گیری گردد در این حالت جهت اندازه گیری زمان از فرکانس ناشی از کریستال سیستم که تقسیم بر ۱۲ شده است استفاده می شود لذا می توان با دقت خوبی زمان را اندازه گیری کرد.

کانتر به مفهوم اینست که در یک فاصله زمانی (مثلاً یک ثانیه) تعداد پالسهای اعمال شده به یکی از پایه های *T0* یا *T1* شمرده شود.

اینکه رجسترهای ۱۶ بیتی که از ترکیب *TL0* و *TH0* یا از ترکیب *TL1* و *TH1* بدست می آیند چگونه عمل کنند آیا تایمر باشند یا کانتر، در چه مدی عمل کنند، چگونه فعال گردند با برنامه ریزی رجستر *TMOD* مشخص می گردد و اینکه کی شروع به شمارش کنند و کی متوقف گردند و سرریز را نمایش دهند از رجستر *TCON* استفاده می گردد. رجسترها و بیتهای مرتبط با این تایمرها و موقعیت آنها بدون در نظر گرفتن بیتهای مربوط به وقفه تایمر در شکل ۲۱ـ۲ نشان داده شده است.

TL0:	D7	D6	D5	D4	D3	D2	D1	D0
TH0:	D7	D6	D5	D4	D3	D2	D1	D0
TL1:	D7	D6	D5	D4	D3	D2	D1	D0
TH1:	D7	D6	D5	D4	D3	D2	D1	D0
TMOD:	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TCON:	TF1	TR1	TF0	TR0				

شکل ۲۱ـ۲ـ رجسترها و بیتهای مرتبط با تایمر ها در ۸۰۵۱

۱ـ۱۳ـ۲ـ کاربرد رجستر *TCON* در تایمر/کانتر صفر و یک

بیتهای مورد استفاده این رجستر توسط تایمر صفر و یک در زیر نشان داده شده که وظیفه هریک توضیح داده شده است.

TCON:	TF1	TR1	TF0	TR0

TR0 **(بیت کنترل راه انداز تایمر صفر):** با یک شدن این بیت توسط نرم افزار تایمر/کانتر صفر شروع به شمارش می کند و با صفر شدن آن توسط نرم افزار تایمر/کانتر صفر متوقف می گردد.

TF0 **(پرچم سرریز تایمر/کانتر صفر):** اگر هنگام شمارش سرریز اتفاق بیفتد یعنی شمارش به ماکزیمم مقدار ممکن مثلاً در شمارش ۱۶ بیتی به *FFFFh* برسد این بیت توسط سخت افزار یک می شود و وقتی پردازنده اقدام به اجرای روتین وقفه می کند این بیت توسط سخت افزار صفر می گردد. یا می توان بصورت سرکشی وضعیت این بیت را خوانده و اقدامات لازم را انجام داد.

TR1 **(بیت کنترل راه انداز تایمر یک):** با یک شدن این بیت توسط نرم افزار تایمر/کانتر یک شروع به شمارش می کند و با صفر شدن آن توسط نرم افزار تایمر/کانتر یک متوقف می گردد.

TF1 **(پرچم سرریز تایمر/کانتر یک):** اگر هنگام شمارش سرریز اتفاق بیفتد این بیت توسط سخت افزار یک می شود و وقتی پردازنده اقدام به اجرای روتین وقفه می کند این بیت توسط سخت افزار صفر می گردد.

۲ـ۱۳ـ۲ـ برنامه ریزی رجستر *TMOD*

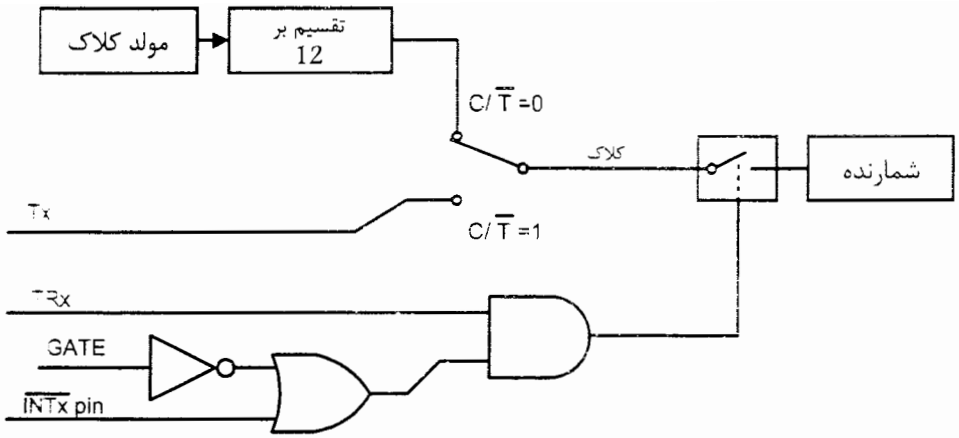
بیتهای رجستر *TMOD* و موقعیت آنها در زیر نشان داده شده است وظیفه هر بیت و عملکرد آنها توضیح داده می شود.

TMOD:	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
	تایمر یک				تایمر صفر			

GATE : وقتی پین کنترل *GATE* یک باشد تایمر/کانتر هنگامی فعال می شود که پایه های *INTx* و *TRx* یک باشند و وقتی بیت کنترل *GATE* صفر باشد تایمر/کانتر فقط وقتی فعال می شود که بیت کنترل *TRx* یک باشد. بعبارت دیگر این پین مانند دروازه أی عمل می کند که به طریق سخت افزاری بتوان شمارشی را انجام داد یا متوقف کرد. کاربرد این بیت و برنامه ریزی آن در ادامه آورده خواهد شد.

$C/\overline{T}$: با صفر شدن این بیت توسط نرم افزار رجسترهای مرتبط (*TLx* و *THx*) بعنوان تایمر عمل می کنند (ورودی از ساعت سیستم دریافت می گردد). با یک شدن این بیت توسط نرم افزار رجسترهای مرتبط بصورت شمارنده عمل می کنند (ورودی پالسی که باید شمرده شود به *Tx* اعمال می گردد). بعبارت دیگر این بیت تعیین می کند که حالت اندازه گیری زمان مد نظر است یا شمارش تعداد پالسهایی که به پایه های *T1* یا *T0* اعمال می شود.

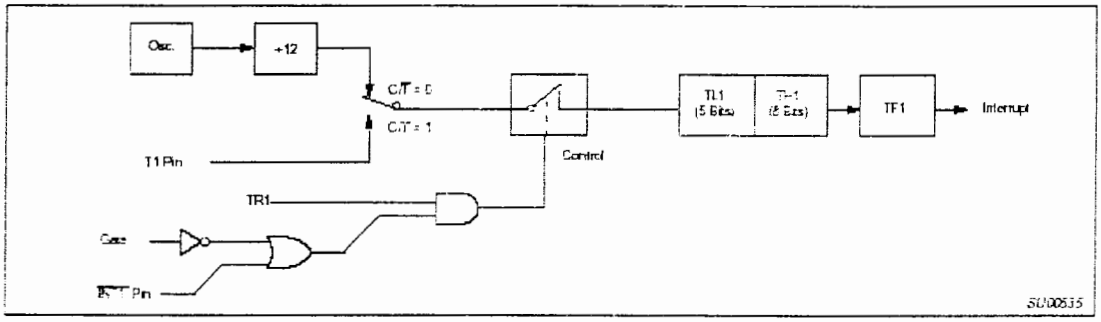
شکل ۲۲ـ۲ ارتباط بین بیتهای *GATE*، *TRx*، *INTx* و $C/\overline{T}$ را نشان می دهد. این شکل نشان می دهد اگر $C/\overline{T}$ برابر صفر باشد کلاک از نوسانساز داخلی و اگر یک باشد از پایه *Tx* تأمین خواهد شد.



شکل ۲۲ـ۲ـ چگونگی فعال سازی تایمر یا کانتر

بیتهای تعیین مد عملیاتی *M0*، *M1*: بخاطر بالا بردن قابلیتهای تایمرها و کانترها امکاناتی پیش بینی شده که بتوانند این دو در حالتهای متفاوتی عمل کنند اینکه چگونه عمل کنند توسط این دو بیت بصورت نرم افزاری مشخص می گردد. دو بیت چهار حالت مختلف می گیرند لذا این تایمر یا کانترها نیز به چهار صورت مختلف می توانند عمل کنند که بصورت زیر است:

M0=0*، *M1=0 : تایمر ۱۳ بیتی که هشت بیت کم ارزش آن در *THx* و پنج بیت بعدی در *TLx* قرار گرفته است. عملکرد این مد در شکل ۲۳ـ۲ نشان داده شده است.

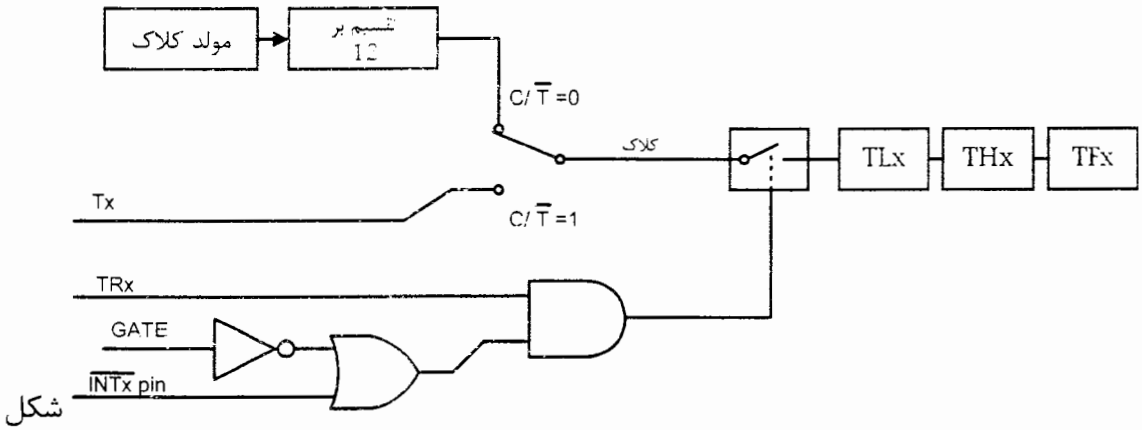


شکل ۲۳ـ۲ـ عملکرد تایمر یا کانتر در حالت صفر

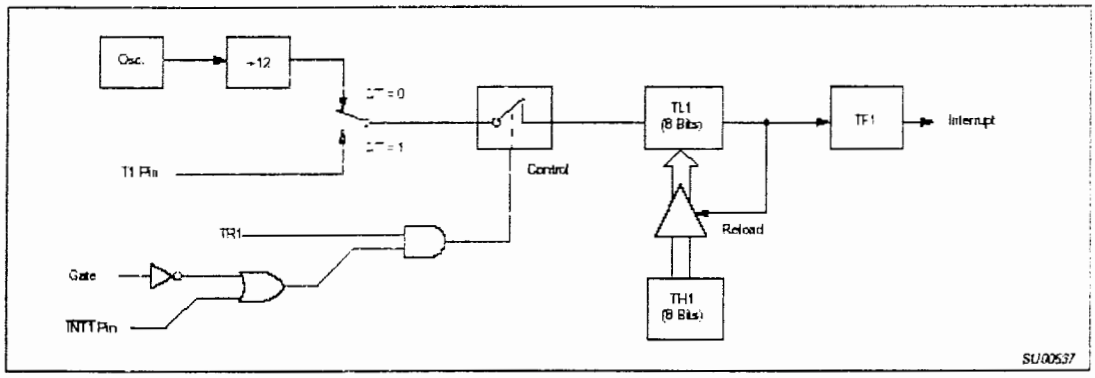
M0=1*، *M1=0: در این حالت با سری شدن *TLx* و *THx* تایمر/کانتر ۱۶ بیتی حاصل می شود که در شکل ۲۴ـ۴ نحوه فعال سازی آن نمایش داده شده است.

M0=0*، *M1=1: **تایمر/کانتر ۸ بیتی با بار شدن خود کار**. در این حالت رجستر *THx* مقداری را در خود نگهمیدارد و هرگاه در *TLx* سرریز ره دهد مقدار *THx* در *TLx* بصورت خود کار قرار می گیرد و شمارش مجدداً شروع می گردد. نحوه فعال سازی تایمر کانتر در این حالت در شکل ۲۵ـ۲ نشان داده شده است. یکی از کاربردهای این حالت تعیین بادریت پورت سریال توسط تایمر/کانتر یک می باشد.

M0=1*، *M1=1: در این حالت *TL0* تایمر/کانتر صفر ۸ بیتی است که توسط بیتهای کنترلی تایمر/کانتر صفر کنترل می شود و *TH0* نیز یک تایمر/کانتر ۸ بیتی است که تنها توسط بیتهای کنترلی تایمر/کانتر یک کنترل می شود. در این حالت تایمر ۱ متوقف است.

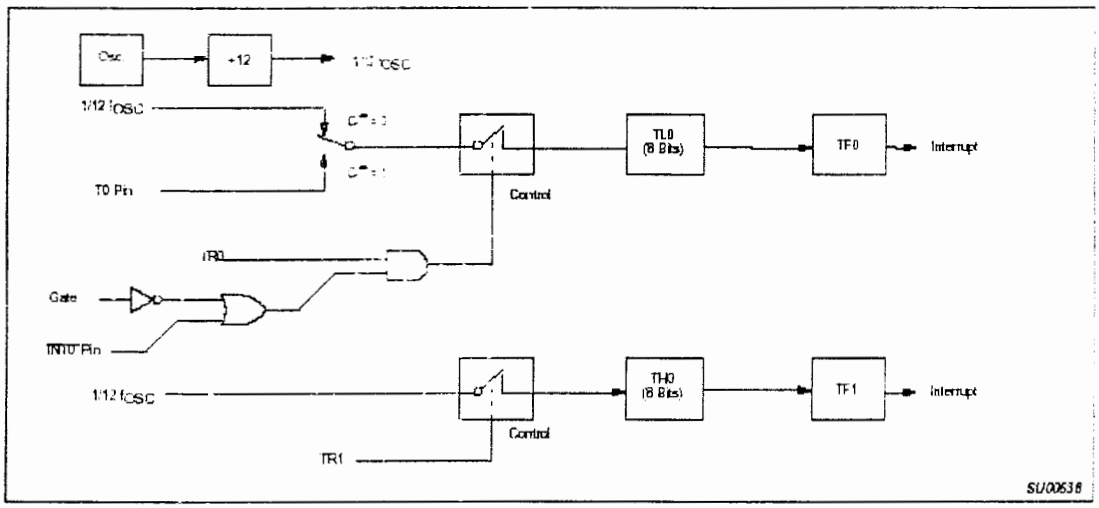


شکل ۲۴ـ۲ـ عملکرد تایمر یا کانتر در حالت یک



شکل ۲۵ـ۲ـ عملکرد تایمر یا کانتر در حالت دو

شکل ۲۶ـ۲ عملکرد این تایمر و کانترها را بصورت ساده نشان می دهد که با تحلیل آن بهتر می توان عملکرد تایمر و کانترها فراگرفت. همانطور که مشاهده می شود در گاه سریز اتفاق یبفتد *TFx* یک می گردد که اگر انتراپت مربوطه فعال شده باشد انتراپت نیز اتفاق می افتد



شکل ۲۶ـ۲ـ عملکرد تایمر یا کانتر در حالت سه

تایمر/ شمارنده یک بعنوان تایمر

حالت	عملکرد تایمر یک	TMOD	
		کنترل داخلی	کنترل خارجی
0	تایمر ۱۳ بیتی	00h	80h
1	تایمر ۱۶ بیتی	10h	90h
2	تایمر ۸ بیتی با بار شدن مجدد	20h	A0h
3	دو تایمر ۸ بیتی	30h	B0h

تایمر/ شمارنده یک بعنوان شمارنده

حالت	عملکرد شمارنده یک	TMOD	
		کنترل داخلی	کنترل خارجی
0	کانتر ۱۳ بیتی	40h	C0h
1	کانتر ۱۶ بیتی	50h	D0h
2	کانتر ۸ بیتی خودکار	60h	E0h
3	در دسترس نمی باشد	--	--

لازم بذکر است با یک کردن بیت *TR1* توسط نرم افزار، تایمر/شمارنده روشن و با صفر کردن آن خاموش می گردد و اگر *TR1* یک باشد، تایمر/شمارنده با تغییر سطح $\overline{INT1}$ (*P3.3*) از صفر به شروع به شمارش می کند.

۱۴ـ۲ـ ارتباط بصورت سریال

انتقال اطلاعات از نقطه ای به نقطه ای دیگر به دو صورت موازی و سریال ممکن است. انتقال اطلاعات بصورت موازی سریع بوده اما در مقابل فاصله ارتباطی کم و هزینه انتقال زیاد بوده و به تجهیزات بیشتری نیاز دارد. انتقال اطلاعات به صورت سریال کند بوده، فاصله ارتباطی بسته به نرخ تبادل اطلاعات تا ۴۰۰۰ فوت می تواند طول داشته باشد و همچنین هزینه سخت افزار کمی نیاز دارد. انتقال اطلاعات بصورت سریال به دو صورت همزمان (سنکرون) و ناهمزمان (آسنکرون) ممکن است که بعداً مورد بحث قرار می گیرند.

۱ـ۱۴ـ۲ـ نرخ انتقال اطلاعات

در انتقال سریال نرخ انتقال اطلاعات بر حسب *bps* (بیت بر ثانیه) نمایش داده می شود. اصطلاح دیگری بنام *Baud rate* وجود دارد که بیان می کند تعداد تغییرات سیگنال در ثانیه چقدر است. این دو لزوماً با هم برابر نیستند. زیرا ممکن است با هر تغییر سیگنال مانند مدمها چند بیت اطلاعات منتقل شود. در کامپیوتر نرخ انتقال اطلاعات با بادریت با هم برابرند. بادریتهای استاندارد که معمولاً استفاده می شوند ۱۵۰، ۳۰۰، ۱۲۰۰، ۲۴۰۰، ۴۸۰۰، ۹۶۰۰، ۱۹۲۰۰ و ... هستند که بصورت بیت بر ثانیه بیان می گردد.

۲ـ۱۴ـ۲ـ استاندارد *RS232*

این استاندارد وضعیت ولتاژی سیگنالهای ارتباطی و کانکتورهای رابط را مشخص می کند. سیگنالهای ارتباطی در این حالت بین *3-* تا *25-* برای سطح منطقی یک و *3+* تا *25+* برای سطح منطقی صفر تعریف شده اند. استفاده از این ولتاژها طول کابل ارتباطی را بیشتر می کند.

آی سی ها و چیپ های خاصی وجود دارند که ولتاژهای *TTL* را به *RS232* تبدیل و یا عمل عکس را انجام می دهند. چیپ هایی چون *1488* جهت تبدیل *TTL* به *RS232* و *1489* جهت تبدیل *RS232* به *TTL* مورد استفاده قرار می گیرند. آی سی *1488* برای انجام این تبدیل به ولتاژهای *12+* و *12-* ولت نیاز دارد.

آی سی های دیگری مانند *MAX232* یا *ICL232* ارائه شده اند که این مهم را تنها با تغذیه *5* ولت انجام می دهند. از آنجایی که سیگنالهای مورد استفاده از نوع ولتاژی است لذا نسبت به نویز حساس بوده و به فاصله کمی می توان اطلاعات را رد وبدل کرد. برای حل این مشکل استاندارد دیگری از نوع سیگنال جریانی بنام *RS422*، *RS423* ارائه شده است که مطابق شکل ۲۷ـ۲ فاصله ارتباطی آنها بیشتر است.

RS423	RS422	RS232	
4000ft	4000ft	50ft	حداکثر طول کابل
100k/30ft	10M/40ft	20k	حداکثر سرعت
10k/300ft	1M/400ft		
1k/4000ft	100k/4000ft		

شکل ۲۷ـ۲ـ مقایسه طول کابل ارتباطی در استانداردهای مختلف سریال

۳ـ۱۴ـ۲ـ انتقال اطلاعات سریال بصورت سنکرون

در این حالت سه سیم جهت تبادل اطلاعات مورد استفاده قرار می گیرد. یک سیم *GND*، یک سیم جهت ارسال دریافت اطلاعات و سیم دیگر کلاک ارسال دریافت اطلاعات را نشان می دهد.

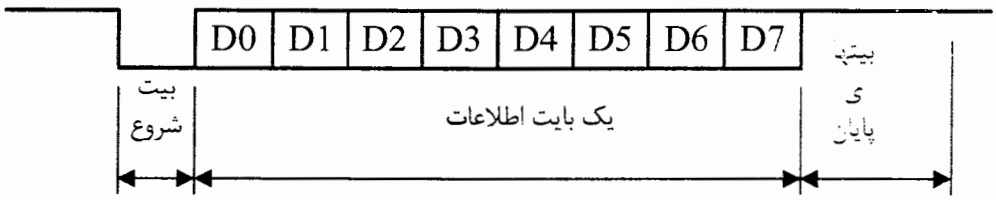
در این حالت پروتکلهای خاصی طراحی و بکار گرفته شده اند. در این حالت اطلاعات بصورت قاب در آمده و ارسال می گردند و در گیرنده شروع و انتهای قابها مشخص و اطلاعات استخراج می گردند. شکل ۲ـ۲۸ بعنوان مثال یک قاب پیام سنکرون را در پروتکل *SDLC*[1] نشان می دهد.

01111110	چک سام	اطلاعات	میدان کنترل	میدان آدرس	01111110

شکل ۲ـ۲۸ـ قاب یک پیام سنکرون در پروتکل *SDLC*

۴ـ۱۴ـ۲ـ انتقال اطلاعات بصورت آسنکرون

نوع دیگری از ارتباط سریال از نوع آسنکرون بوده که نرخ تبادل اطلاعات و فاصله ارتباطی در آن پایین بوده و کاربرد زیادی در ارتباط بین کامپیوتر و تجهیزات جانبی بخاطر سادگی می تواند داشته باشد. در ارتباط به صورت آسنکرون از دو منبع کلاک جداگانه استفاده شده و صفر و یکها ساخته شده و ارسال می گردند لذا تا اندازه ای این منابع کلاک می تواند متفاوت بوده و یا تغییرات داشته باشند. از آنجایی که منبع کلاک فرستنده و منبع کلاک گیرنده با هم متفاوت هستند لازم است در فاصله زمانی نه چندان زیاد مبدأ اندازه گیری آنها تنظیم گردد به این خاطر مطابق شکل ۲ـ۲۹ فرمت تبادل اطلاعاتی مشخص شده است.



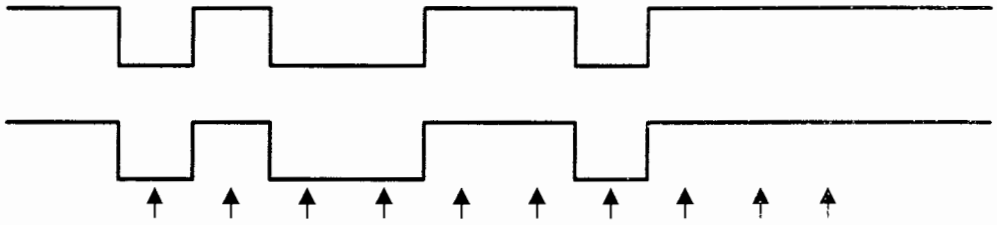
شکل ۲ـ۲۹ـ فرمت تبادل اطلاعات در انتقال اطلاعات بصورت آسنکرون

این شکل نشان می دهد جهت همزمانی مبدأ برای تبادل هر بایت اطلاعات یک بیت بعنوان بیت شروع منظور شده است. فرستنده این بیت را فرستاده و گیرنده نیز این بیت را دریافت می کند و شروع آنرا مبدأ زمان برای بایت دریافتی تعیین می کند. بیتهای بعد از بیت شروع دیتا بوده که اولین بیت دریافتی کم ارزشترین بیت بایت ارسالی است. تعداد بیتهای اطلاعاتی می تواند ۵، ۶، ۷، ۸ بیت باشد و علاوه بر اینها ممکن است بیت دیگری بعنوان پریتی وجود داشته باشد. بیتهای بعدی که ۱، ۱/۵ یا ۲ بیت می باشند بیتهای پایان بوده که شرایط را برای ارسال بیت شروع مشخص می کنند.

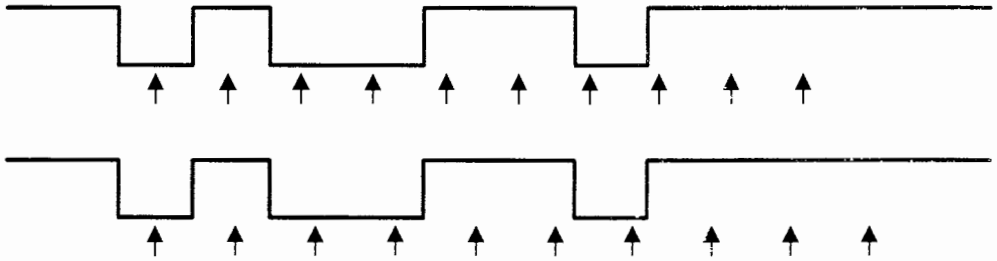
فرستنده بیتهای بعد از بیت شروع را باتوجه به منبع کلاک خود ارسال می کند. در گیرنده نرخ تبادل اطلاعات با فرستنده تنظیم شده است. لذا اگر منبع کلاک فرستنده و گیرنده مطابق شکل ۲ـ۳۰ دقیقاً

[1] *Serial Data Link Control*

یکسان باشد گیرنده درست در وسط هربیت دریافتی از سیگنال نمونه خواهد گرفت اما اگر منبع کلاک فرستنده و گیرنده یکسان نباشد مانند شکل ۳۱ـ۲ در وسط نمونه نگرفته و بسته به زیاد یا کم بودن فرکانس کلاک گیرنده در طرف چپ یا طرف راست وسط بیت نمونه برداری خواهد کرد و هرچقدر تعداد بیتها بیشتر باشد در بیتهای آخری نمونه گیری به طرف چپ یا راست بیشتر متمایل خواهد شد. لذا اختلاف منبع کلاک فرستنده و گیرنده تنها می تواند آنقدر باشد که در بیت آخری نمونه گیری از لبه بیت کمتر یا بیشتر نگردد.



شکل ۳۰ـ۲ـ نمونه برداری با یکسان بودن بادریت ارسال و دریافت



شکل ۳۱ـ۲ـ نمونه برداری با یکسان نبودن بادریت ارسال و دریافت، بالایی فرکانس گیرنده بیشتر و پایینی کمتر

۵ـ۱۴ـ۲ـ کانکتور پورت سریال

در کامپیوترها معمولاً کانکتور پورت سریال به دو صورت *D* کانکتور ۹ پین و *D* کانکتور ۲۵ پین بصورت نری مورد استفاده قرار می گیرد. شکل ۳۲ـ۲ این دو کانکتور را نشان می دهد. این کانکتورها بصورت مادگی یا نری ارائه شده و شماره هر پین روی آن مشخص شده است. از آنجایی که کاربرد اصلی پورت سریال در اتصال مدمهای خارجی بوده است وظایف این پایه ها برای اتصال مدم تعریف شده و با توجه به آن تعاریف و وظایف پایه ها می توان از آنها در اتصال دیگر سیستمها به یکدیگر استفاده نمود.

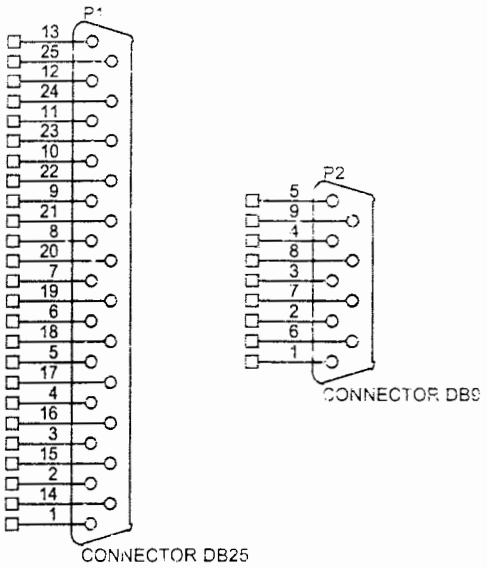
۱ـ ***DCD*** (**تشخیص حامل معتبر**)[1] مدم این سیگنال را فعال می کند تا به کامپیوتر اطلاع دهد که یک حامل معتبر توسط مدم شناخته شده است و اتصال بین این مدم و مدم دیگر برقرار شده است از این رو *DCD* خروجی از مدم و ورودی به کامپیوتر است.

۲ـ ***RXD*** : داده از طریق این پایه به طریق سریال توسط کامپیوتر دریافت می شود.

۳ـ ***TXD*** : داده از طریق این پایه به طریق سریال توسط کامپیوتر ارسال می شود.

[1] *Data Carrier Detect*

۴ـ ***DTR*** **(آماده بودن پایانه)**[1]: زمانیکه کامپیوتر روشن می شود بعد از تست داخلی این پایه را فعال می کند تا نشان دهد برای ارتباط آماده است. اگر اشکالی در پورت سریال وجود داشته باشد این سیگنال فعال نمی شود. این پایه خروجی از کامپیوتر است.



شکل ۳۲ـ۲ـ *D* کانکتور ۹ پین و ۲۵ پین مورد استفاده در پورت سریال کامپیوتر

۵ـ ***GND*** **(زمین)**: توسط این پین می توان زمین کامپیوتر و سخت افزار جانبی همچون مدم را به یکدیگر وصل نمود. اطلاعات در صورتی منتقل می گردند که دو سخت افزار زمین مشترک داشته باشند و یا با استفاده از اپتوکوپلر به طریقی از یکدیگر جدا شده باشند.

۶ـ ***DSR*** **(آماده بودن برای دریافت داده)**[1]: زمانیکه دستگاه جانبی همچون مدم روشن می شود بعد از یک تست داخلی *DSR* را فعال می کند تا نشان دهد که برای ارتباط آماده است. بنابر این یک سیگنال ورودی به کامپیوتر است.

۷ـ ***RTS*** **(تقاضا برای ارسال)**[2]: هنگامی که کامپیوتر یک بایت برای انتقال دارد جهت اینکه به دستگاه جانبی اطلاع دهد این پایه را فعال می کند. لذا یک پایه خروجی از کامپیوتر است.

۸ـ ***CTS*** **(خالی برای ارسال)**[3]: اگر دستگاه جانبی آمادگی دریافت داده را داشته باشد این سیگنال را فعال می کند تا آمادگی خود را به اطلاع کامپیوتر برساند. این یک سیگنال ورودی به کامپیوتر است.

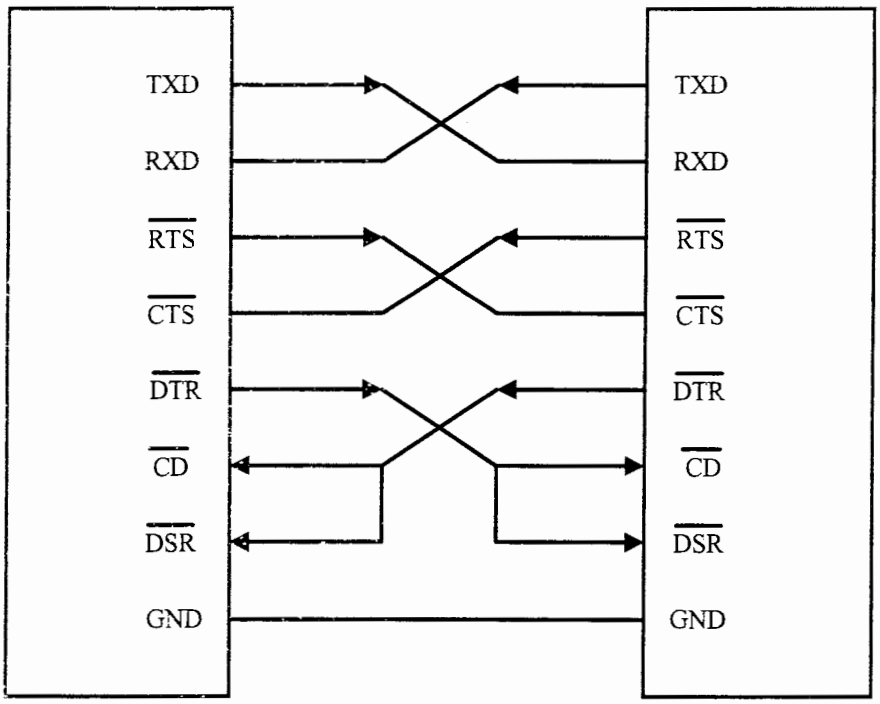
۹ـ ***RI*** **(مشخص کننده زنگ)**[4]: این سیگنال نشان می دهد تلفن زنگ زده مدم آنرا تشخیص داده و می خواهد آنرا به اطلاع کامپیوتر برساند. لذا این پایه خروجی از مدم یا دستگاه جانبی و ورودی به کامپیوتر است.

---

[1] *Data Terminal Ready*
[1] *Data Set Ready*
[2] *Request To Send*
[3] *Clear To Send*
[4] *Ring Indicator*

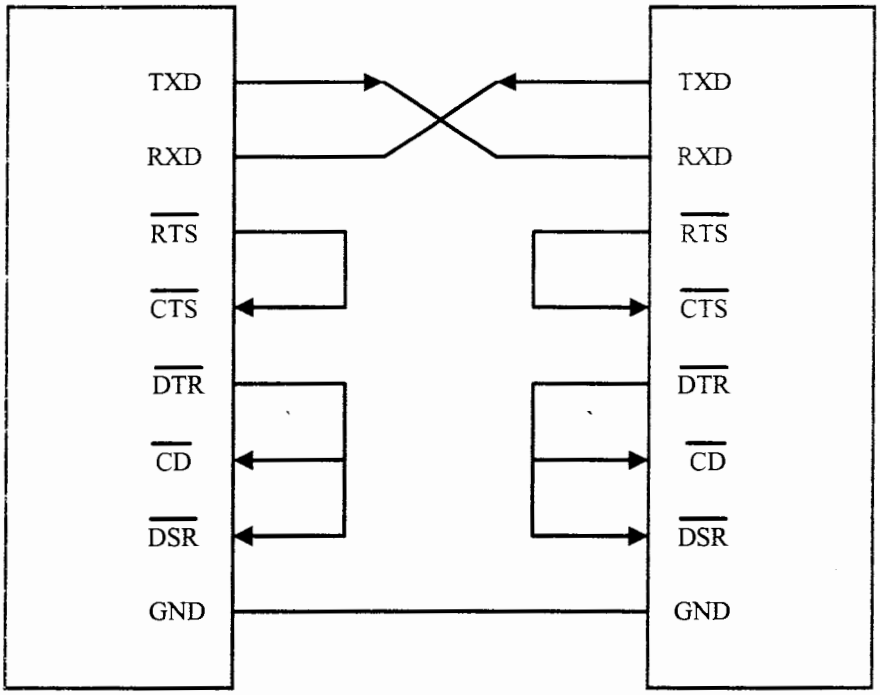
همانطور که اشاره شد کاربرد اصلی پورت سریال کامپیوتر که این سیگنالها برای آن تعریف شده است مدم است این نوع مدمها را مدمهای خارجی نامیده که می توان از آنها استفاده کرد. از پورت سریال مربوطه می توان در ارتباط سخت افزارهای مختلف با کامپیوتر استفاده کرد در این حالت کافی است سیگنالهای فوق به نحوی ساخته شوند که شرایط برای ارسال و دریافت اطلاعات مهیا باشد.

نحوه اتصال دو کامپیوتر به یکدیگر توسط پورت سریال در شکل ۳۳ـ۲ نشان داده شده است. در این شکل که حالت کلی است برای ارتباط به ۷ سیم ارتباطی نیاز است.



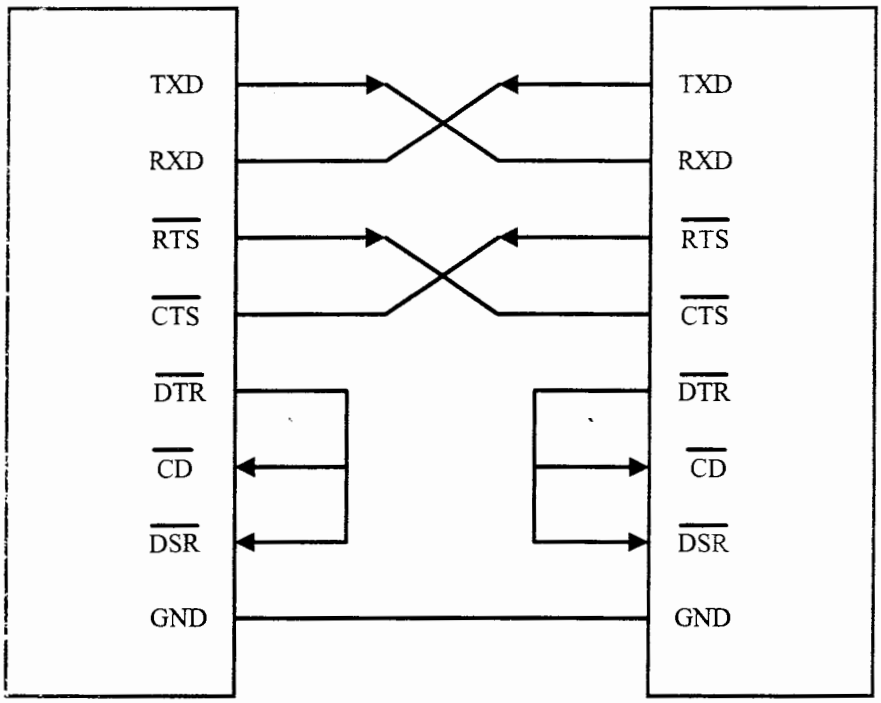
شکل ۳۳ـ۲ـ ارتباط دو کامپیوتر از طریق پورت سریال در حالت کامل

شکل ۳۴ـ۲ که کانکتور ارتباطی طرف کامپیوتر را نشان می دهد از این قاعده استفاده شده است که اگر کامپیوتر آمادگی داشت یعنی پایه *DTR* را فعال کرده بود فرض می گردد دستگاه جانبی نیز آمادگی دارد و حامل معتبر را تشخیص داده است. لذا پایه ۴ که خروجی است بایستی به پایه های ۱ و ۶ وصل گردد. از طرف دیگر اگر کامپیوتر می خواهد دیتایی را بفرستد پایه *RTS* را فعال می کند و در صورتی دیتا را می فرستد که دستگاه جانبی آمادگی داشته باشد به عبارت دیگر *CTS* فعال شده باشد لذا اگر پایه *RTS* به *CTS* وصل شده باشد کامپیوتر با فعال سازی *RTS* دستگاه جانبی را آماده می بیند و بایت مربوطه را ارسال می کند. مزیت استفاده از روش فوق سادگی و استفاده از سه سیم است و اما عیب عمده آن این است که ارسال کننده اطلاعات و دریافت کننده اطلاعات از یکدیگر بی اطلاعند لذا اگر با سرعت زیادی لازم باشد اطلاعات مبادله گردد ممکن اطلاعاتی در این بین حذف گردد.



شکل ۳۴ـ۲ـ نحوه اتصال پینها جهت تبادل اطلاعات بصورت ساده

روشی دیگر که اشکال روش بالا را نداشته باشد در شکل ۳۵ـ۲ نشان داده شده است. در این شکل فرض شده است کامپیوتر و دستگاه جانبی آمادگی دارند اما هرگاه قرار است اطلاعاتی بین آندو ارسال گردد فرستنده بررسی می کند که آیا وسیله جانبی آمادگی دریافت داده را دارد یا نه. اگر گیرنده آمادگی داشته باشد بایستی سیگنال *CTS* را فعال نماید. اتصال به این صورت به ۵ سیم نیاز دارد.



شکل ۳۵ـ۲ـ نحوه اتصال پینها جهت تبادل اطلاعات بصورت دست دادن

۱۵ـ۲ـ پورت سریال ۸۰۵۱

یکی از قابلیتهای خوب میکروکنترلر ۸۰۵۱ وجود پورت سریال آن است. توسط این پورت بسادگی می توان سخت افزار استفاده کننده از ۸۰۵۱ را به کامپیوتر متصل و اطلاعات لازم را بین آندو رد و بدل کرد. اگر این قابلیت پیش بینی نمی شد لازم بود سخت افزارهای مبدل موازی به سریال و بالعکس همچون آی سی ۸۲۵۱ به سیستم اضافه گردد. با وجود این پورت که تنها دو پایه *RXD* و *TXD* آن در دسترس است می توان توسط نرم افزار آنرا برنامه ریزی و استفاده کرد. رجستر های مرتبط با آن *SBUF*، *SCON*، *TMOD* و *TCON* می باشند که مورد بررسی قرار می گیرند.

۱ـ۱۵ـ۲ـ رجستر *SBUF*

این رجستر بافر فرستنده و بافر گیرنده پورت سریال می باشد. گرچه آدرس بافر فرستنده و گیرنده هر دو *99h* بوده و با *SBUF* شناخته می شوند ولی در واقع به دو مکان مختلف اشاره می کنند. هنگام ارسال یک بایت، آن بایت در بافر فرستنده قرار گرفته و هنگامی که یک بایت دریافت می گردد در بافر گیرنده قرار می گیرد. جهت نوشتن در بافر فرستنده و خواندن از بافر گیرنده از دستورالعملهای زیر استفاده می شود.

دستورالعمل خواندن از بافر سریال	*MOV A,SBUF*
دستورالعمل نوشتن در بافر سریال	*MOV SBUF,A*

۲ـ۱۵ـ۲ـ رجستر *SCON*

این رجستر که آدرس آن *98h* بوده و بصورت بیتی می توان بیتهای آنرا تغییر داد برای کنترل عملکرد پورت سریال مورد استفاده قرار می گیرد. همانطور که یک پورت جانبی همچون *8255* جهت عملکرد مناسب به برنامه ریزی نیاز دارد پورت سریال میکروکنترلر نیز به برنامه ریزی نیاز دارد. بیتهای مختلف این رجستر در زیر داده شده است.

SCON:	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

بیت *RI* : اگر یک بایت از طریق پایه *RXD* پورت سریال درست دریافت شود این بیت بصورت سخت افزاری یک می گردد تا نشان دهد یک بایت دریافت شده است در نرم افزار می توان در فاصله زمانی مشخص این بیت را خواند و تشخیص داد که آیا یک بایت دریافت شده است یاخیر. هنگامی که بایت دریافتی خوانده شد این بیت را بایستی بصورت نرم افزاری صفر نمود.

بیت *TI* : اگر در بافر پورت سریال یک بایت نوشته شود این بیت صفر شده و ارسال آن بایت بصورت سریال شروع می گردد پس از اینکه آن بیت کامل ارسال شد این بیت یک می گردد تا نشان دهد بایت مربوطه بطور کامل ارسال شده و می توان بایت بعدی را ارسال کرد.

بیت *RB8* : اگر ارسال و دریافت اطلاعات بصورت ۹ بیتی برنامه ریزی شده باشد این مکان وضعیت بیت نهم دریافتی را نشان می دهد. لازم بذکر است بین نهم مبین بیت توازن و یا بیت نهم هنگام کار در محیط چند پردازنده ای مورد استفاده قرار می گیرد.

بیت *TB8*: بیت نهمی که قرار است ارسال گردد در این مکان توسط برنامه نویس قرار می گیرد.

بیت *REN* : برای اینکه گیرنده پورت سریال فعال گردد بایستی این بیت را یک نمود.

بیتهای *SM0* و *SM1*

بیتهای *SM0* و *SM1* مطابق شکل ۳۶ـ۲ چهار حالت کاری پورت سریال را نشان می دهند. از این ۴ حالت کاری یک حالت سنکرون و سه حالت بصورت آسنکرون مورد استفاده قرار می گیرد در حالت کاری صفر و ۲ نرخ ارسال یا دریافت اطلاعات تنها به کریستال مورد استفاده بستگی دارد اما در حالت کاری یک و ۳ می توان با استفاده از تایمر ۱ یا ۲ (تنها در ۸۰۵۲) نرخ ارسال و دریافت قابل برنامه ریزی داشت.

SM0	SM1	MODE	FUNCTION	LENGTH	PERIOD
0	0	0	Sync	8 bits	4/12 t_{CLK}
0	1	1	Async	10 bits	Timer 1 or 2*
1	0	2	Async	11 bits	64/32 t_{CLK}
1	1	3	Async	11 bits	Timer 1 or 2*

شکل ۳۶ـ۲ـ حالات کاری مختلف پورت سریال ۸۰۵۱

بیت *SM2* : هنگامی چند پردازنده از طریق پورت سریال به یکدیگر وصل شده اند و لازم است بین آنها اطلاعاتی رد و بدل گردد حالت کاری را ۲ یا ۳ انتخاب کرده و این بیت را یک می نمایند. اگر در این حالت کاری نهمین بیت دریافتی صفر باشد *RI* که مبین دریافت یک بایت می باشد فعال نمی گردد. به عبارت دیگر در صورتی یک پردازنده آن بایت را دریافت می کند که *SM2* آن صفر باشد. نحوه استفاده به این صورت است که در حالت عادی تمام پردازنده هایی که قرار است اطلاعاتی دریافت کنند حالت کاری آنها ۲ یا ۳ انتخاب شده و بیت *SM2* آنها یک می گردد. حال فرستنده هنگام شروع ارسال اطلاعات ابتدا نه بیت را که حتماً بین نهم آن یک بوده و مابقی مشخص کننده پردازنده انتخابی می باشد را ارسال می کند با توجه به شرایط موجود همه آنرا دریافت می کنند پس از پردازش بایت دریافتی تشخیص می دهند که آیا اطلاعات بعدی مربوط به آنهاست یا خیر در صورتی که مربوط به آنها نباشد کار خاصی نمی کنند اما اگر مربوط به پردازنده خاصی باشد بیت *SM2* خود را صفر می کند و فرستنده از آن به بعد اطلاعاتی را ارسال می کند که بیت نهم آنها صفر است در نتیجه تنها پردازنده ای که بیت *SM2* خود را صفر نموده است اطلاعات را دریافت می کند و این روند ادامه دارد تاهنگامی که اطلاعات لازم مبادله گردد. پس از مبادله اطلاعات پردازنده گیرنده بیت *SM2* خود را یک نموده و مشابه دیگر پردازنده ها خواهد شد و روند تکرار خواهد شد.

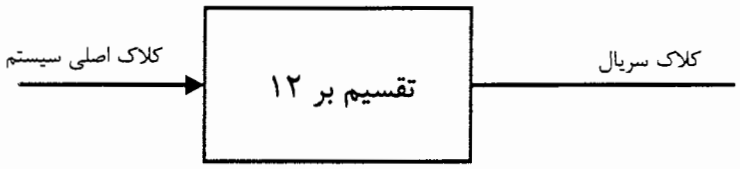
در حالت کاری یک اگر *SM2* یک باشد در صورتی *RI* یک خواهد شد که بیت توقف درستی دریافت شده باشد. به عبارت دیگر بیت توقف را می توان نهم بیت دریافتی در نظر گرفت و مشابه حالت بالا در صورتی پیام دریافت خواهد شد که این بیت یک باشد و بیت توقف در صورتی درست است که یک باشد. در حالت کاری صفر این بیت بایستی صفر باشد.

۳ـ۱۵ـ۲ـ حالتهای کاری پورت سریال

همانطور که در شکل ۳۶ـ۲ نشان داده شد پورت سریال ۸۰۵۱ می تواند چهار حالت کاری مختلف داشته باشد. رجسترهای خاص مرتبط با حالات کاری مختلف *SCON*، *PCON*، *TMOD* و *TCON* می باشند که در بعضی حالات کاری بایستی مقدار آنها را تعیین کرد. با مقدار دهی این رجسترها بادریت پورت سریال و تعداد بیتها انتقال داده شده تعیین می گردد. در ادامه این حالات توضیح داده شده و مقدار هر رجستر در صورت نیاز مقدار دهی می گردد.

۱ـ ارسال و دریافت بصورت سنکرون (حالت صفر)

در این حالت کاری که اطلاعات بصورت سنکرون ارسال و دریافت می گردند پایه *RXD* جهت ارسال و دریافت اطلاعات و پایه *TXD* جهت ارسال و دریافت کلاک مورد استفاده قرار می گیرد. نرخ ارسال اطلاعات در این حالت ثابت بوده و مطابق شکل ۳۷ـ۲، *1/12* کلاک اصلی سیستم می باشد. در این حالت کاری تنها لازم است *SCON=10h* مقدار دهی گردد و *SCON* خوانده شود تا وضعیت *RI* و *TI* بررسی گردد درنتیجه به مقدار دهی *PCON*، *TMOD* و *TCON* هیچ نیازی نیست.

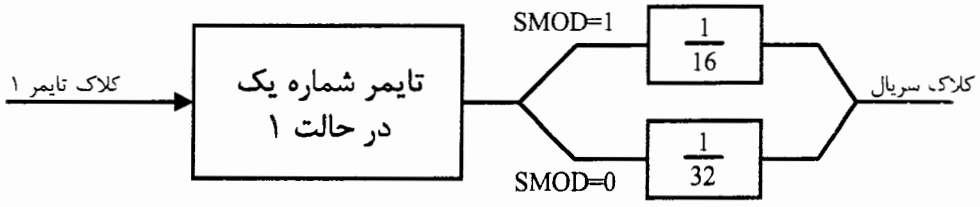


شکل ۳۷ـ۲ـ نحوه تولید کلاک سریال در حالت کاری صفر

۲ـ *UART* ۸ بیتی با نرخ ارسال قابل تنظیم (حالت یک)

در این حالت کلاک ارسال و دریافت اطلاعات بااستفاده از تایمر شماره یک ساخته می شود. در این حالت تایمر شماره یک در حالت بار نمودن خودکار بصورت ۸ بیتی (حالت ۱) برنامه ریزی می گردد. دراین حالت بایستی *TMOD=50h* و *TH1* از رابطه زیر مقدار آن تعیین گردد تا بادریت مشخص تولید گردد.

$$\text{Reload} = 256 - \frac{2^{\text{SMOD}} * \text{Oscillator Freq.}}{384 * \text{Baud rate}}$$



شکل ۳۸ـ۲ـ نحوه تولید کالا سریال در حالت کاری یک

اگر از تایمر شماره ۲ در ۸۰۵۲ برای تعیین بادریت استفاده شود بایستی *RCAP2L* و *RCAP2H* از رابطه زیر تعیین گردند.

$$RCAP2H, RCAP2L = 65536 - \frac{\text{Oscillator Frequency}}{32 * \text{Baud rate}}$$

مقادیری که بایستی در *TH1* قرار گیرند برای چند بادریت مختلف و چند کریستال مختلف در شکل ۳۹ ـ۲ و برای *RCAP2L* و *RCAP2H* در شکل ۴۰ـ۲ نشان داده شده است.

Baud Rate	f_{osc}	SMOD	Timer 1		
			C/T	Mode	Reload Value
Mode 0 Max: 1.67MHz	20MHz	X	X	X	X
Mode 2 Max: 625k	20MHz	1	X	X	X
Mode 1, 3 Max: 104.2k	20MHz	1	0	2	FFH
19.2k	11.059MHz	1	0	2	FDH
9.6k	11.059MHz	0	0	2	FDH
4.8k	11.059MHz	0	0	2	FAH
2.4k	11.059MHz	0	0	2	F4H
1.2k	11.059MHz	0	0	2	E8H
137.5	11.986MHz	0	0	2	1DH
110	6MHz	0	0	2	72H
110	12MHz	0	0	1	FEEBH

۳۹ـ۲ـ مقدار بار گیری *TH1* با کریستال *11.0592 MHZ*

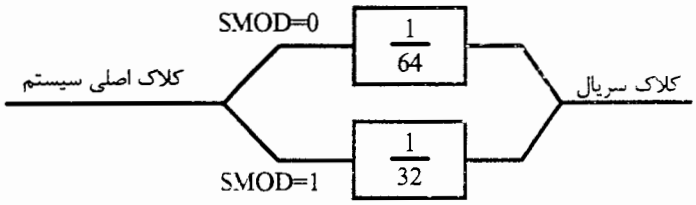
BAUD RATE	RCAP2H	RCAP2L
57600	0FFh	0FAh
9600	0FFh	0DCh
2400	0FFh	090h
1200	0FEh	0E0h

۴۰ـ۲ـ مقدار بارگیری برای تایمر ۲ با بادریتهای مختلف با کریستال *11.0592 MHZ*

۳ـ *UART* ۹ بیتی با نرخ ارسال ثابت (حالت دو)

در این حالت بادریت ارسال دریافت اطلاعات مطابق رابطه زیر و شکل ۴۱ـ۲ از تقسیم کلاک اصلی سیستم بر ۳۲ یا ۶۴ بدست می آید. در این حالت به هیچ یک ازتایمرها نیازی نیست و تنها می توان با یک کریستال مشخص تنها دو بادریت مختلف داشت.

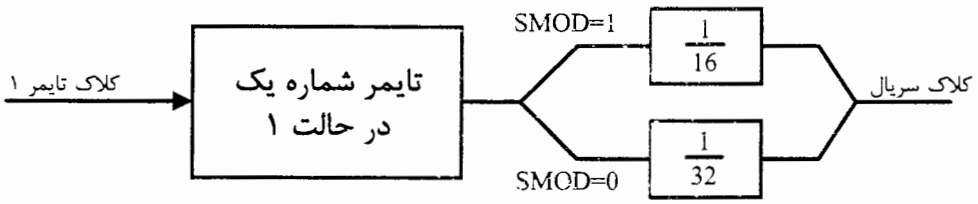
$$\text{baud rate} = \frac{2^{\text{SMOD}} * \text{Oscillator Freq.}}{64}$$



شکل ۴۱ـ۲ـ نحوه تولید کالا سریال در حالت کاری دو

۴ـ *UART* ۹ بیتی با نرخ ارسال قابل تنظیم (حالت سه)

این حالت مشابه حالت سه است با این تفاوت که در *TMOD* مقدار متفاوتی قرار گرفته و اطلاعات بصورت ۹ بیتی ارسال و دریافت می گردند.



شکل ۴۲ـ۲ـ نحوه تولید کلاک سریال در حالت کاری سه

شکل ۴۳ـ۲ ماکزیمم بادریت بدست آمده در حالتهای کاری مختلف را با کریستال های مختلف نشان می دهد. همانطور که مشاهده می شود و انتظار می رود ارسال، دریافت اطلاعات بصورت سنکرون بیشترین سرعت را دارد.

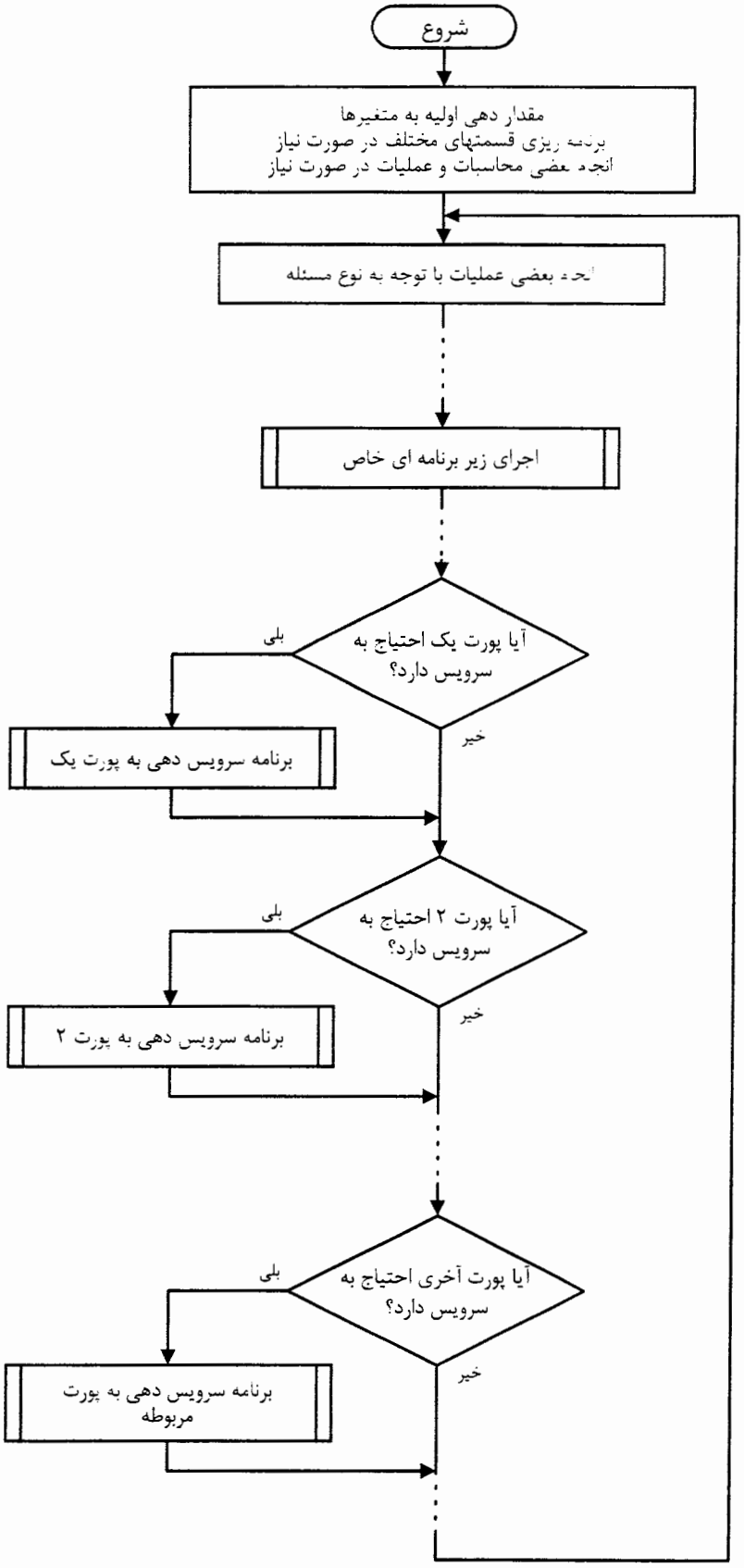
MODE	MAXIMUM BAUD, 25 MHz	MAXIMUM BAUD, 33 MHz	MAXIMUM BAUD, 40 MHz
0	6,250,000	8,250,000	10,000,000
1	390,625	515,625	625,000
2	781,250	1,031,250	1,250,000
3	390,625	515,625	625,000

۴۳ـ۲ـ ماکزیمم بادریت

۱۶ـ۲ـ وقفه ها در میکروکنترلر ۸۰۵۱

برای سرویس دهی به پورتها در میکروپروسسورها و میکروکنترلرها دو روش سرکشی و انتراپتی وجود دارد. در روش سرکشی برنامه لازم ساده بوده و از قابلیت اطمینان بالایی برخوردار است میکروپروسسور یا میکروکنترلر در حلقه اصلی برنامه مطابق شکل ۴۴ـ۲ به سراغ پورتهای مختلف رفته و وضعیت آنها را چک نموده و در صورت نیاز زیر برنامه سرویس دهی به آن پورت را اجرا می کند و اگر پورت مربوطه به سرویس دهی نیاز نداشت آنرا رها کرده و به سراغ پورت دیگر می رود و این روند را آنقدر تکرار می کند که همه پورتها مورد بررسی قرار گیرند و سپس روند فوق از ابتدا شروع می گردد.

نوشتن برنامه بصورت سرکشی و اجرای آن اگر ممکن باشد ساده ترین و در عین بهترین و مطمئن ترین روش بوده و در کاربردهای صنعتی این روش ترجیح داده می شود. اما همیشه این روش بدلایلی همچون نوع مسئله و کند بودن این روش ممکن نبوده و بایستی از روش دیگری جهت سرویس دهی به پورتها استفاده کرد. لذا برای بالا بردن کیفیت و سرعت اجرای برنامه های مختلف روی میکروپروسسورها و میکروکنترلرها قابلیت وقفه یا اینتراپت پیش بینی شده است. با وجود این قابلیت اگر میکروپروسسور یا میکروکنترلر درحال اجرای برنامه اصلی باشد و شرایطی خاص بوجود آید و پورتی تقاضای سرویس دهی کند میکروپروسسور یا میکروکنترلر قادر است برنامه اصلی در حال اجرا را رها کرده و به سراغ برنامه مورد نظر رفته و آنرا اجرا کند و پس از اتمام سرویس دهی به پورت مربوطه به سراغ برنامه اصلی برگشته و اجرای آنرا دنبال کند. از آنجایی که لحظه تقاضای سرویس از دید برنامه اصلی در حال اجرا مشخص نیست و ممکن است در وضعیتها و شرایط مختلف اتفاق افتد در نوشتن برنامه بایستی دقت کافی بعمل آورده و برنامه نوشته شده بدقت و تحت شرایط مختلف تست گردد.



شکل ۴۴ـ۲ـ اجرای برنامه با روش سرکشی

در ۸۰۵۱ پنج منبع وقفه وجود دارد. این منابع وقفه، دو وقفه سخت افزاری، دو وقفه تایمر یا کانتر و یک وقفه پورت سریال می باشد در میکروکنترلر ۸۰۵۲ منبع وقفه تایمر ۲ نیز پیش بینی شده است. می توان میکروکنترلر را بنحوی برنامه ریزی کرد که وقفه ها غیر فعال باشند و یا یک، دو و یا چند تای آنها فعال و بقیه غیر فعال باشند و همچنین می توان اولویت آنها را تغییر داد. برای نیل به این هدف رجسترها و بیتهایی منظور شده است که بایستی آنها را بفرم مناسب مقدار دهی کرد.

۱ـ۱۶ـ۲ـ فعال کردن وقفه ها

هنگامی که میکروکنترلر ریست می گردد بعضی از رجسترها را بنحوی مقدار دهی می کند که میکروکنترلر تحت شرایط پیش فرض که کمترین خطا را بوجود می آورد قرار گیرد از جمله این رجسترها رجستر *IE* به آدرس *A8h* از رجسترهای *SFR* می باشد. بعبارت دیگر پس از ریست میکروکنترلر رجستر *IE* که وظیفه فعال سازی وقفه ها را بعهده دارد صفر شده و همه انتراپتها غیر فعال می گردند. لذا در صورت استفاده از وقفه ای خاص در برنامه مورد نظر بایستی وقفه مربوطه را فعال نمود برای این منظور بایستی رجستر *IE* را که در زیر آورده شده است بفرم مناسب مقدار دهی کرد.

IE:	EA		ET2	ES	ET1	EX1	ET0	EX0

بیت ***EA***: جهت غیر فعال سازی کل وقفه ها بکار می رود. اگر این بیت صفر باشد هیچ وقفه ای پذیرفته نمی شود. اما اگر این بیت یک باشد وقفه مربوطه در صورتی پذیرفته می شود که بیت اختصاصی به آن وقفه یک شده باشد.

بیت ***ET2*** : برای فعال کردن وقفه تایمر ۲ در میکروکنترلر ۸۰۵۲ این بیت بایستی یک گردد.

بیت ***ES*** : برای فعال کردن وقفه پورت سریال بایستی این بیت یک گردد. اگر این بیت یک باشد ولی *EA* صفر باشد وقفه پذیرفته نمی شود.

بیت ***ET1*** : برای فعال سازی وقفه تایمر یک بایستی این بیت یک گردد. اگر این بیت یک باشد ولی *EA* صفر باشد وقفه پذیرفته نمی شود.

بیت ***EX1*** : برای فعال سازی وقفه خارجی یک (*INT1*) این بیت بایستی یک گردد. اگر این بیت یک باشد ولی *EA* صفر باشد وقفه پذیرفته نمی شود.

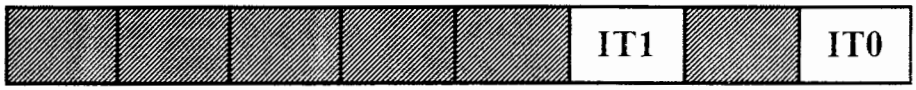
بیت ***ET0*** : برای فعال سازی وقفه تایمر صفر بایستی این بیت یک گردد. اگر این بیت یک باشد ولی *EA* صفر باشد وقفه پذیرفته نمی شود.

بیت ***EX0*** : برای فعال سازی وقفه خارجی یک (*INT0*) این بیت بایستی یک گردد. اگر این بیت یک باشد ولی *EA* صفر باشد وقفه پذیرفته نمی شود.

۲ـ۱۶ـ۲ـ پذیرش وقفه ها

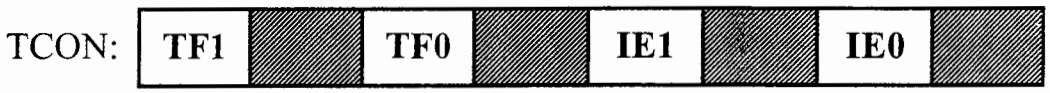
با فرض اینکه بیت های مربوط به وقفه ها یک شده باشند در چه صورتی وقفه اتفاق می افتد؟ قبل از پاسخ گویی به این سؤال دو بیت بنامهای *IT0* و *IT1* مطابق شکل ۴۵ـ۲ در رجستر *TCON* وجود دارد که

بایستی مقدار آندو را در صورت استفاده از انتراپتهای سخت افزاری بدرستی تعیین نمود. می توان وقفه سخت افزاری صفر را به نحوی برنامه ریزی کرد که به لبه یا سطح حساس باشد اگر *IT0* یک باشد حساس به سطح و اگر صفر باشد حساس به لبه خواهد بود. همچنین برای انتراپت سخت افزاری یک اگر بیت *IT1* یک باشد حساس به سطح و اگر صفر باشد حساس به لبه خواهد بود.



شکل ۴۵ـ۲ـ رجستر *TCON* و بیتهای تعیین کننده حساس به سطح یا لبه وقفه های سخت افزاری

در رجستر *TCON* و *SCON* که در شکل ۴۶ـ۲ نشان داده شده اند بیتهایی وجود دارد که وضعیت پایه های سخت افزاری *INT0* و *INT1* ، تایمر صفر و یک و پورت سریال را نشان می دهد. بعبارت دیگر این بیتها نشان می دهند که آیا تغییر خاصی که منجر به تصمیم گیری شود روی آنها اتفاق افتاده است یا خیر. این بیتها می توانند شرایط لازم جهت بوجود آمدن انتراپت را فراهم کنند و یا می توان آنها را خواند و به روش سرکشی اعمال لازم را انجام داد.



شکل ۴۶ـ۲ـ رجستر *TCON* و *SCON* و بیتهای تعیین کننده وضعیت انتراپتها

بیت ***IE0***: اگر روی پین ورودی *INT0* تغییر سطح از یک به صفر اتفاق افتاده باشد و یا *IT0* یک باشد و پایه *INT0* در سطح صفر قرار داشته باشد این بیت یک می گردد.

بیت ***IE1***: اگر روی پین ورودی *INT1* تغییر سطح از یک به صفر اتفاق افتاده باشد و یا *IT1* یک باشد و پایه *INT1* در سطح صفر قرار داشته باشد این بیت یک می گردد.

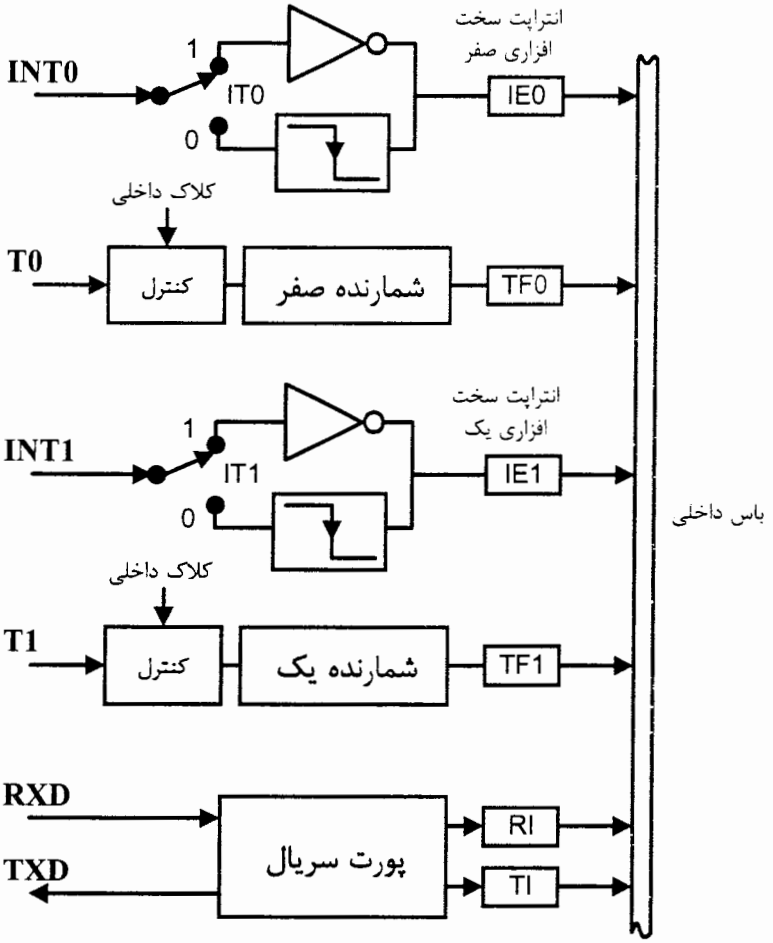
بیت ***TF0***: اگر در شمارنده صفر سرریز اتفاق افتد این بیت یک می گردد. با پذیرش انتراپت این بیت بصورت سخت افزاری صفر می گردد.

بیت ***TF1***: اگر در شمارنده یک سرریز اتفاق افتد این بیت یک می گردد. با پذیرش انتراپت این بیت بصورت سخت افزاری صفر می گردد.

بیت ***RI*** : اگر یک بایت از طریق پورت سریال دریافت گردد این بیت یک می شود.

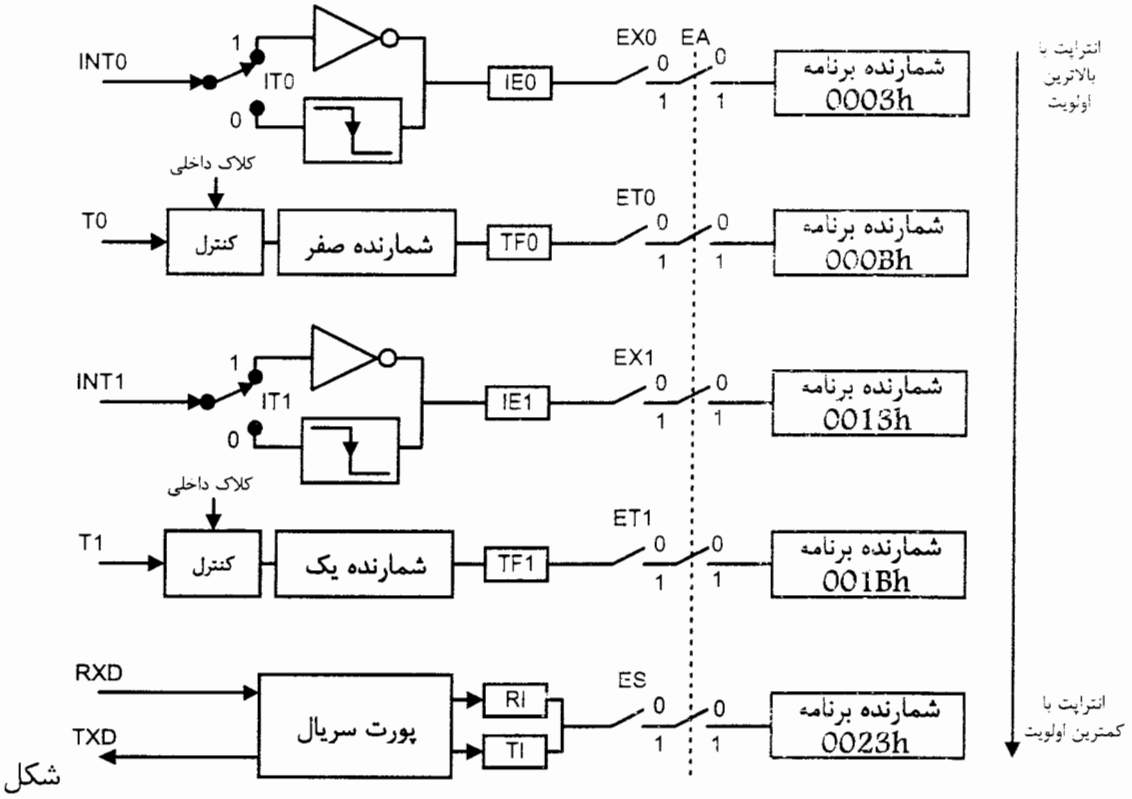
بیت ***TI*** : اگر بایتی که در پورت سریال قرار گرفته است بطور کامل ارسال گردد این بیت یک می گردد.

از قابلیتهای میکروکنترلر ۸۰۵۱ همچون تایمرها و پورت سریال می توان به روش سرکشی مطابق شکل ۴۷ـ۲ بهره جست. در این حالت کافی است بیتهای مربوط به قسمتهای مختلف و موجود در *TCON* و *SCON* را به روش سرکشی بررسی کرده و در صورت وجود شرایط خاص برنامه مورد نظر را اجرا کرد.



شکل ۴۷ـ۲ـ استفاده از قسمتهای مختلف میکروکنترلر ۸۰۵۱ با روش سرکشی

شکل ۴۸ـ۲ ارتباط بین بیتهای رجسترهای *IE* ، *TCON* و *SCON* را بهمراه بردار وقفه مربوطه نشان می دهد. در این شکل *TF0* و *TF1* مطابق آنچه که قبلاً در مورد تایمرها گفته شد از آنها فرمان گرفته و تحت تأثیر قرار گرفته اند. همچنین *RI* و *TI* از پورت سریال فرمان گرفته و تغییر وضعیت می دهند.



شکل ۲-۴۸- ساختار وقفه در میکرو کنترلر ۸۰۵۱

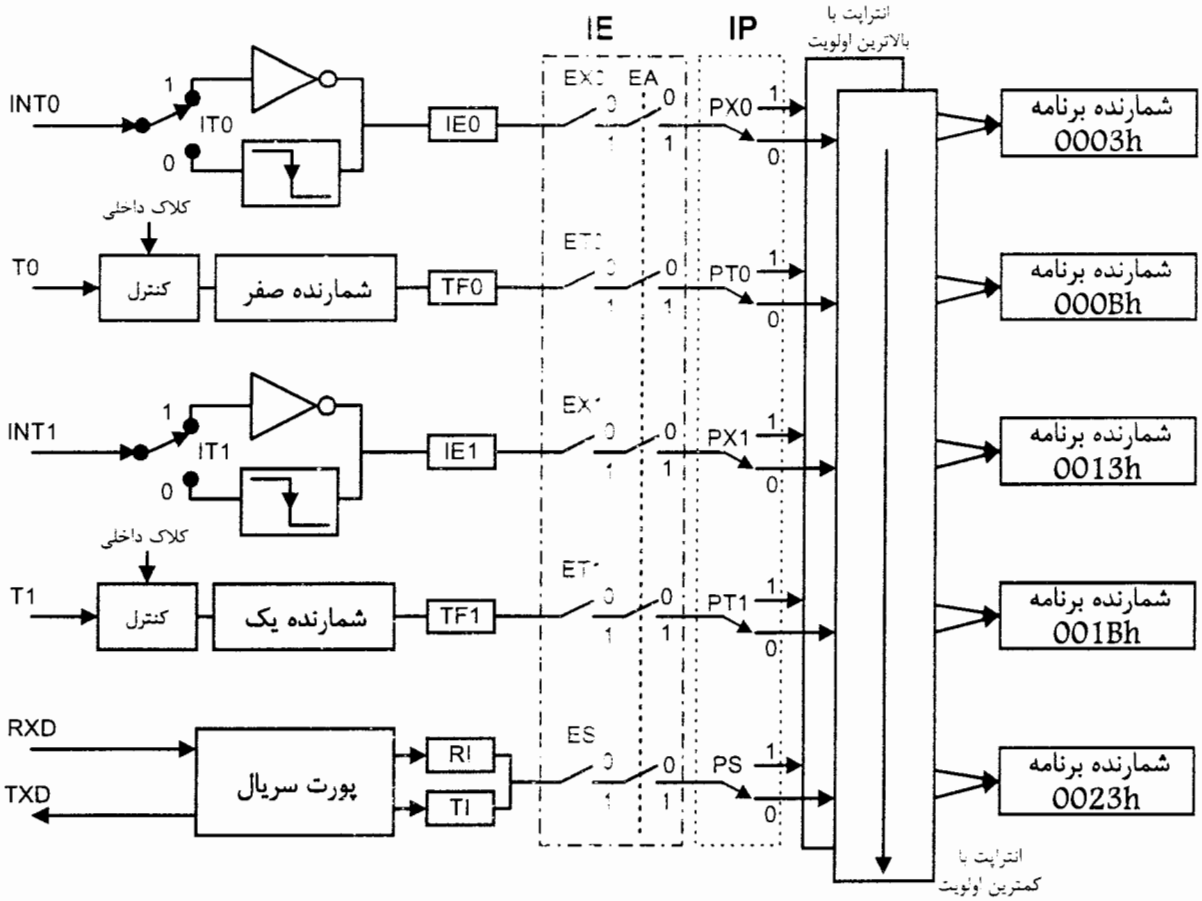
۳-۱۶-۲- تقدم وقفه ها

در حالت عادی اگر رجستر *IP* محتوی عدد صفر باشد اولویت انتراپتها بصورت پیش فرض بوده که در آن انتراپت سخت افزاری صفر بالاترین اولویت و در ۸۰۵۱ پورت سریال و در ۸۰۵۲ تایمر ۲ کمترین اولویت را دارند. اولویت انتراپتها نیز در رجستر *IP* نشان داده شده است.

IP:

		PT2	PS	PT1	PX1	PT0	PX0

در این رجستر بیت *PX0* مربوط به انتراپت سخت افزاری صفر، بیت *PT0* مربوط به تایمر صفر، بیت *PX1* مربوط به انتراپت سخت افزاری یک، بیت *PT1* مربوط به تایمر یک، بیت *PS* مربوط به پورت سریال و بیت *PT2* مربوط به تایمر دو می باشد. حال اگر بعضی از این بیتها به یک تغییر وضعیت دهند اولویت انتراپتها تغییر خواهد کرد اما اگر تمامی آنها به یک تغییر وضعیت دهند در اولویت انتراپتها تغییری ایجاد نخواهد شد. بعنوان مثال اگر *PS=1* و بقیه برابر صفر باشند انتراپت ناشی از پورت سریال بالاترین اولویت سپس انتراپت سخت افزاری صفر، تایمر صفر، انتراپت سخت افزاری یک و تایمر یک بترتیب اولویت های بعدی را خواهند داشت. شکل ۲-۴۹ علاوه بر بیتهای فعال سازی وقفه ها، بیتهای اولویت دهنده وقفه ها و بردار وقفه را بصورت همزمان نشان می دهد.



شکل ۴۹ـ۲ـ نمایش چگونگی پذیرش و اولویت دهی به وقفه ها و بردار وقفه هریک

۱۷ـ۲ـ معرفی دیگر میکروکنترلرهای خانواده *MCS51*

تاکنون میکروکنترلرهای مختلفی از این خانواده توسط شرکتهای مختلف ساخته و روانه بازار شده اند در این بین میکروکنترلرهای زیر بیشتر مورد توجه قرار گرفته اند که مشخصات آنها مختصراً توضیح داده می شود.

۱ـ۱۷ـ۲ـ میکروکنترلر *8031* و *8032*

مشابه میکروکنترلرهای ۸۰۵۱ و ۸۰۵۲ هستند با این تفاوت که در آنها حافظه برنامه داخلی پیش بینی نشده است البته از آنجایی که حافظه داخلی ۸۰۵۱ و ۸۰۵۲ نیز قبلاً برنامه ریزی شده است در حالت معمولی کارآیی ۸۰۵۱ دقیقاً مشابه ۸۰۳۱ و ۸۰۵۲ دقیقاً مشابه ۸۰۳۲ می باشد.

۲ـ۱۷ـ۲ـ میکرو کنترلر *8751* و *8752*

مشابه میکروکنترلرهای ۸۰۵۱ و ۸۰۵۲ هستند با این تفاوت که در آنها حافظه برنامه داخلی از نوع *EPROM* پیش بینی شده و توسط برنامه ریزهای خاص قابل برنامه ریزی و توسط نور ماورای بنفش قابل پاک شدن و برنامه ریزی مجدد می باشند.

۳ـ۱۷ـ۲ـ میکرو کنترلر *89C51* و *89C52*

این میکرو کنترلرها دقیقاً مشابه *8051* و *8052* می باشند با این تفاوت که به جای حافظه *ROM* داخلی حافظه *EEPROM* داخلی در آن پیش بینی شده است. لذا کاربر خود می تواند آنرا برنامه ریزی و مورد استفاده قرار دهد و در صورت نیاز حافظه مورد نظر راپاک نموده و برنامه ریزی مجدد نماید. استفاده از این نوع میکروکنترلر در کاربردهای کنترلی ساده حجم سخت افزار را کاهش داده و به قابلیت اطمینان سیستم می افزاید.

۴ـ۱۷ـ۲ـ میکرو کنترلر*89S51*

این میکروکنترلر علاوه بر داشتن حافظه *EEPROM* داخلی به ظرفیت ۴ کیلوبایت، قابلیت برنامه ریزی به طریق سریال در آن پیش بینی شده است. به عبارت دیگر می توان ترتیبی اتخاذ نمود که روی برد سخت افزاری به طریق سریال قابل برنامه ریزی باشد.

۵ـ۱۷ـ۲ـ میکرو کنترلر *89C2051،89C1051* و *89C4051*

این میکرو کنترلرها از خانواده *8051* بوده با این تفاوت که تعداد پایه های آنها ۲۰ تا بوده و به جای حافظه *ROM* داخلی حافظه *EEPROM* داخلی به ظرفیت ۱، ۲ و ۴ کیلوبایت در آنها پیش بینی شده است. لذا کاربر خود می تواند این حافظه را برنامه ریزی و مورد استفاده قرار دهد. دو عدد تایمر یا کانتر ۱۶ بیتی، دو پایه انتراپت خارجی، پورت سریال با چهار مد مختلف، ۱۲۸ بایت حافظه *RAM* داخلی، ۱، ۲ و ۴ کیلوبایت حافظه *EEPROM* داخلی، ۱۵پین ورودی خروجی، ۱۲۸ بیت قابل استفاده بصورت بیتی از مشخصات این میکروکنترلرها می باشد. استفاده از این نوع میکروکنترلر در کاربردهای کنترلی ساده، حجم سخت افزار را کاهش داده و به قابلیت اطمینان سیستم می افزاید.

۶ـ۱۷ـ۲ـ میکرو کنترلر *89C55WD*

این میکروکنترلر دقیقاً مشابه *89C51* است با این تفاوت که حجم حافظه *EEPROM* داخلی آن بجای ۴ کیلوبایت ۲۰ کیلوبایت و حافظه *RAM* داخلی بجای ۱۲۸ بایت ۲۵۶ بایت استو تایمر ۲ نیز در آن پیش بینی شده است و علاوه بر این در داخل آن زمان سنج نگهبان که در کارهای کنترلی مهم است پیش بینی شده است لذا هنگامی که طول برنامه زیاد می گردد و یا نیاز به حافظه *RAM* داخلی بیشتر است و یا زمان سنج نگهبان برای سخت افزار ضروری است از آن استفاده می گردد.

۱۸ـ۲ـ زمان سنج نگهبان

اگر در سیستمهای میکروپروسسوری یا میکروکنترلری در حین اجرای برنامه بدلیل وجود نویز میکروپروسسور یا میکروکنترلر در خواندن یا نوشتن با مشکل برخورد کرده و حتی یک بیت را به غلط بخواند یا بنویسد به احتمال زیاد سیستم از روند جاری خود منحرف شده و دیگر برنامه عادی خود را اجرا نکرده و ممکن است تا سیستم بصورت دستی ریست نگردد در این وضعیت بماند. برای سیستمهای کنترلی این مسئله جدی بوده و بایستی ترتیبی اتخاذ نمود که اولاً این مشکل بوجود نیاید و اگر این مشکل بوجود آمد سخت افزار خود آنرا در کوتاهترین زمان ممکن تشخیص و به رفع عیب بپردازد. در

سیستمهای میکروپروسسوری به سخت افزاری که این مهم را انجام می دهد زمان سنج نگهبان گفته می شود. این زمان سنج را می توان با شمارنده ها و یا منواستابل ها ساخت و با نرم افزاری که در سیستم پیش بینی می گردد هدف مورد نظر را برآورده ساخت. با وجود سخت افزار زمان سنج نگهبان در سیستم، عمکرد بصورت زیر خواهد بود. نرم افزار اگر روند عادی خود را طی نماید بایستی در یک دوره زمانی مثلاً ۱۰۰ میلی ثانیه سخت افزار زمان سنج نگهبان را ریست نماید اگر در این مدت سخت افزار زمان سنج ریست نشد (مثلاً شمارنده ریست نشد) سخت افزار زمان سنج نگهبان کل سخت افزار را ریست خواهد کرد. لذا اگر در حین اجرای برنامه مشکلی بوجود بیاید و نرم افزار از روند عادی خود خارج گردد سخت افزار زمان سنج با ریست کردن سیستم، سیستم را به روند عادی خود هدایت خواهد کرد.

در میکروکنترلر ۸۰۵۱ این قابلیت پیش بینی نشده و در صورت نیاز بایستی سخت افزاری خاص به سیستم اضافه کرد.

فصل سوم

طراحی سخت افزار پروژه

۱ـ۳ـ مقدمه

در این فصل ابتدا مشخصات سیستم مورد نظر بیان شده و با توجه به آن بلوک دیاگرام کلی سیستم ارائه شده و توضیحات لازم داده می شود سپس قسمتهای مختلف سخت افزار با جزئیات لازم طراحی و توضیحات لازم بیان می گردد.

۲ـ۳ـ مشخصات مورد نیاز

همانطور که قبلاً بیان شد سیستم مورد نظر لازم است براحتی در کاربردهای آموزشی مورد استفاده قرار گیرد به نحوی که با هزینه و صرف زمان کم بتوان آزمایش مورد نظر را انجام داد. برای نیل به این هدف سیستمی با مشخصات زیر طراحی و ساخته می شود.

۱ـ براحتی با کامپیوتر ارتباط برقرار نموده و اطلاعات لازم بین سخت افزار و کامپیوتر منتقل گردد و صحت انتقال اطلاعات نمایش داده شود.

۲ـ یک فایل با فرمت *HEX* از کامپیوتر توسط برد سخت افزاری دریافت و روی حافظه *RAM* یا *EEPROM* قرار گیرد.

۳ـ پس از اینکه فایل دریافت و ذخیره شد ترتیبی اتخاذ گردد که بتوان برنامه را بدون دخالت هیچ برنامه ای دیگر و با دراختیار گرفتن کل قابلیتهای مختلف میکرو کنترلر اجرا نمود.

۴ـ اگر لازم است برنامه اصلاح و یا برنامه جدیدی اجرا گردد براحتی این کار ممکن باشد.

۵ـ برنامه های مختلف را بتوان با آن اجرا و ترتیبی اتخاذ نمود که بتوان بعضی سخت افزارها را که در سیستم وجود ندارد شبیه سازی کرد.

۶ـ علاوه بر آموزش نرم افزار و تست برنامه های مختلف بتوان آزمایشهای پرکاربرد همچون کار با *7Segment* ، *LCD* ، کی برد، *Step Motor* ، مبدل آنالوگ به دیجیتال و بالعکس را انجام داد.

۷ـ تغذیه های لازم روی سخت افزار پیش بینی شده باشد و به تغذیه جداگانه نیاز نداشته باشد.

۸ـ حجم حافظه برنامه و دیتا به اندازه لازم باشد و برای همه کاربردهای آموزشی کفایت کند.

۹ـ سیستم در دو محیط *DOS* و *WINDOWS* به جهت آموزش بهتر قابل استفاده باشد. محیط *DOS* برای افراد کم تجربه و محیط *WINDOWS* برای افراد با تجربه بیشتر استفاده شود.

۱۰ـ برای اغلب میکروکنترلرهای این خانواده قابل استفاده باشد. بعبارت دیگر نوع میکروکنترلر به نحوی انتخاب گردد که میکروکنترلرهای بیشتری را پوشش دهد.

۱۱ـ لزومی به برنامه ریزی میکروکنترلر یا حافظه بصورت جداگانه نباشد.

علاوه بر اینها گزارش تکمیلی و نحوه عیب یابی و موارد لازم در گزارش کار منعکس گردد.

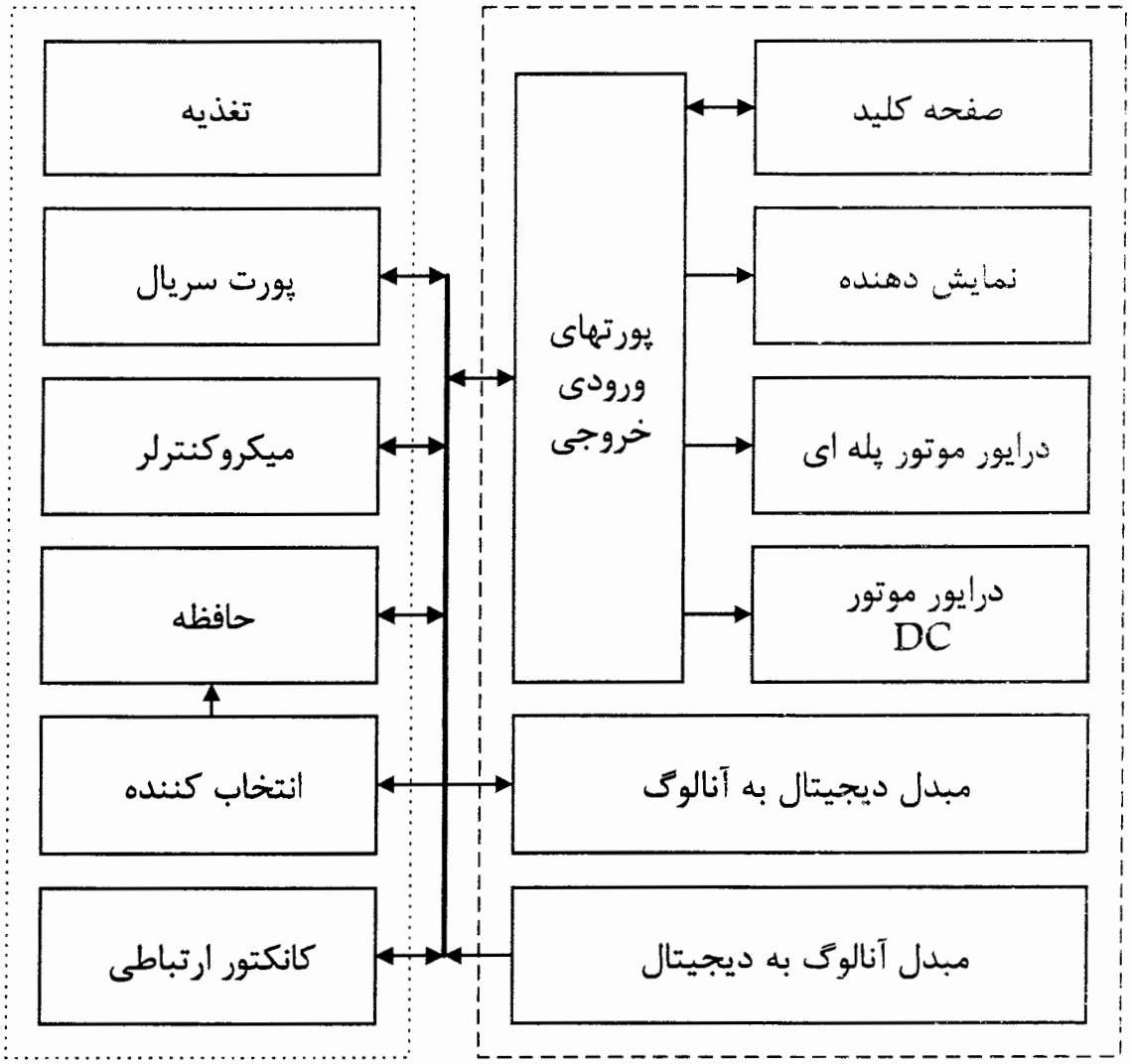
۳ـ۳ـ بلوک دیاگرام کلی سیستم

به جهت آشنایی بیشتر با کلیات و قسمتهای مختلف سیستم در شکل ۱ـ۳ بلوک دیاگرام کلی سیستم نشان داده شده است. نمایش بصورتی ارائه شده است که سیستم را به دو بخش سیستم مینیمم و پورتهای ورودی خروجی تفکیک می نماید. آنچه که در این بین اهمیت بیشتری دارد سیستم مینیمم و نرم افزار روی آن است و پورتهای ورودی خروجی به این خاطر اضافه شده اند که بسادگی بتوان آزمایشهای مختلف را انجام داده و تبحر افراد را در برنامه نویسی افزایش داد بنحوی که دید کاربران مبتدی را افزایش داده و نتیجه فعالیت خود را بصورت ساده مشاهده نمایند. در روندی که در قالب نرم افزار علاوه بر سخت افزارهای موجود پیگیری شده است سعی شده است پورتهای ورودی خروجی بصورتی شبیه سازی گردند که بتوان روی کامپیوتر نتایج لازم را مشاهده نمود این قسمت کار برای افراد خبره برای انجام بعضی پروژه ها کارساز بوده و در وقت و هزینه صرفه جویی می نمایند بعنوان مثال پورتهای *P0* و *P2* در سخت افزار مورد نظر اشغال شده اند ولی سخت افزار واقعی که با استفاده از *89C51* ساخته خواهد شد این پورتها در دسترس هستند برای اینکه در نرم افزار از این پورتها بتوان بعنوان ورودی یا خروجی استفاده کرد شبیه سازی لازم انجام شده است تعداد این پورتها را براحتی می توان به تعداد مورد نیاز بدون هیچ محدودیتی افزایش داده و پس از انجام آزمایشها و تست های مختلف با اندک تغییراتی آنرا به سیستم واقعی منتقل نمود. برای این شبیه سازی پروتکل ارتباطی خاصی که شامل دو بایت شروع، آدرس ۱۶ بیتی، یک بایت دیتا و بایت اصلاح خطا (چک سام)[1] می باشد پیش بینی شده است که توسط آن می توان ۶۵۰۰۰ پورت مجازی مختلف تعریف و نتایج لازم را روی کامپیوتر مشاهده کرد و یا اطلاعات خاصی را بدین وسیله برای سخت افزار ارسال نمود.

در ادامه سیستم به دو بخش سیستم مینیمم و پورتهای ورودی و خروجی تقسیم و قسمتهای مختلف آنها و وظایف آنها مورد بحث قرار می گیرد. سپس با کنار هم قرار دادن سخت افزار طراحی شده، سخت افزار اولیه که پس از تست و آزمایشهای متعدد احتمالاً تغییراتی در آن اعمال خواهد شد ساخته می شود.

---

[1] *Checksum*



شکل ۱ـ۳ـ بلوک دیاگرام کلی برد سخت افزاری

۴ـ۳ـ سیستم مینیمم

وظیفه این قسمت سیستم، دریافت فایل مورد نظر از کامپیوتر و ذخیره آن روی حافظه *RAM* یا *EEPROM* می باشد. در مرحله بعد می توان توسط انتخاب کننده برنامه مورد نظر را اجرا و نتیجه مورد نظر را مشاهده نمود . با بکاربردن تکنیکی ساده اما با اهمیت که در اینجا بکار رفته است می توان کل مشخصات میکرو را دراختیار گرفته و بدون دخالت هیچ برنامه خارجی برنامه مورد نظر را اجرا نمود. ساخت این قسمت سیستم با وجود منبع تغذیه، پورت سریال، میکروکنترلر *89C55WD* یا *89C52*، حافظه، انتخاب کننده و کانکتور ارتباطی مقدور است که به تفصیل مورد بحث قرار می گیرند.

۱ـ۴ـ۳ـ منبع تغذیه

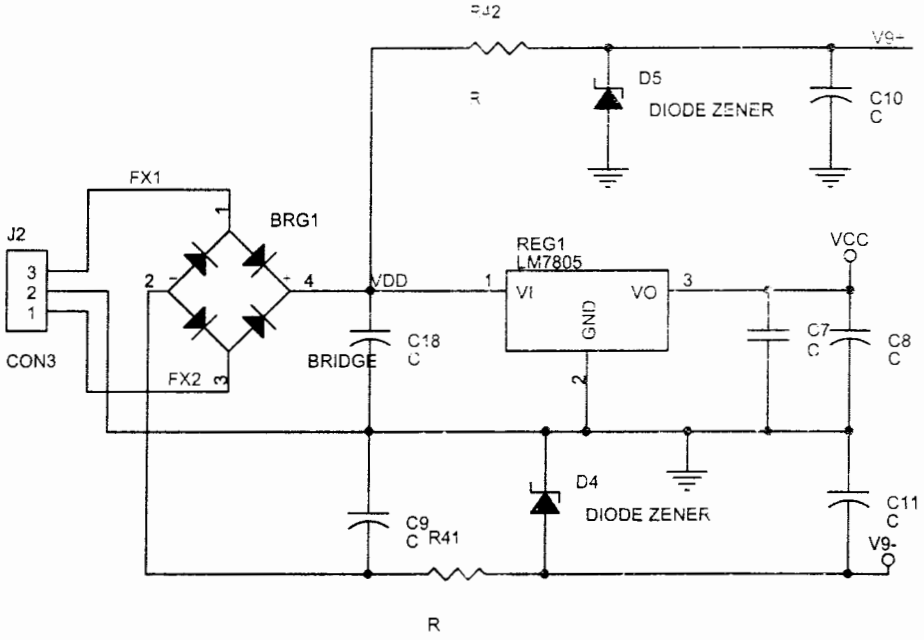
منابع تغذیه به دو نوع خطی و سویچینگ تقسیم می گردند. منابعی که از ترانس و پل دیودی و رگولاتور استفاده می کنند منابع خطی و منابعی که از قطعات الکترونیکی برای قطع و وصل ولتاژ با فرکانس بالا (چند کیلو هرتز) و سپس عبور آن از یک ترانسفورماتور کوچک و سپس یکسوسازی آن

استفاده می کنند تغذیه سویچینگ می گویند، دراین بحث منابع تغذیه سویچینگ که برای جریانهای بالا استفاده می گردند مورد نظر نبوده و حالت اول مختصری توضیح داده می شود.

یکی از قسمتهای اساسی و در عین حال ساده سیستم، منبع تغذیه آن است. وجود هرگونه اشکالی در این قسمت روی عملکرد بقیه قسمتها اثر سوء داشته و گاهی عملکرد کل سیستم را زیر سؤال می برد زیرا اشکال بوجود آمده توسط آن اغلب تصادفی بوده و در زمان و شرایط مشخصی اتفاق نمی افتد. بدین جهت بایستی در طراحی آن جریانهای مصرفی، حرارت بوجود آمده و حرارت گیر لازم، ولتاژ ترانسها، ولتاژ خازنها و دیگر موارد بخوبی مد نظر قرار گیرد.

منابع تغذیه خطی با استفاده از یک ترانس دو سر یا سه سر، یک پل دیودی یا دو عدد دیود و خازن به ظرفیت نسبتاً بالا و یک رگولاتور ساخته می شوند. این مدارات در عین سادگی از اهمیت خاصی برخودارند یعنی بایستی با توجه به جریان و ولتاژ مورد نیاز و میزان تغییرات آن، ولتاژ و جریان ترانس، نوع پل دیودی و اندازه خازن را تعیین کرد. اگر جریان ترانس به اندازه کافی نباشد در لحظاتی که مدار مورد نظر جریان زیاد می کشد ولتاژ افت کرده و باعث می گردد مدار درست کار نکند یا موقعیکه ترانس سرد است کار کند و موقعی که ترانس گرم شد کار نکند یا اگر ولتاژ برق شهر مناسب است کار کند و اگر ولتاژ کم شد کار نکند از آنجایی که این کار کردنها و نکردنها از روند درستی پیروی نمی کند عیب یابی آن بسیار مشکل است. لذا بایستی ولتاژ و جریان ترانس با توجه به مصرف مدارات و تلرانسهای ساخت ترانس و نوسان ولتاژ برق شهر و میزان حرارت بوجود آمده روی رگولاتور، مناسب انتخاب گردد و علاوه بر آن جریان دیودها، ظرفیت خازن مورد نیاز و رگولاتور لازم مناسب انتخاب گردد.

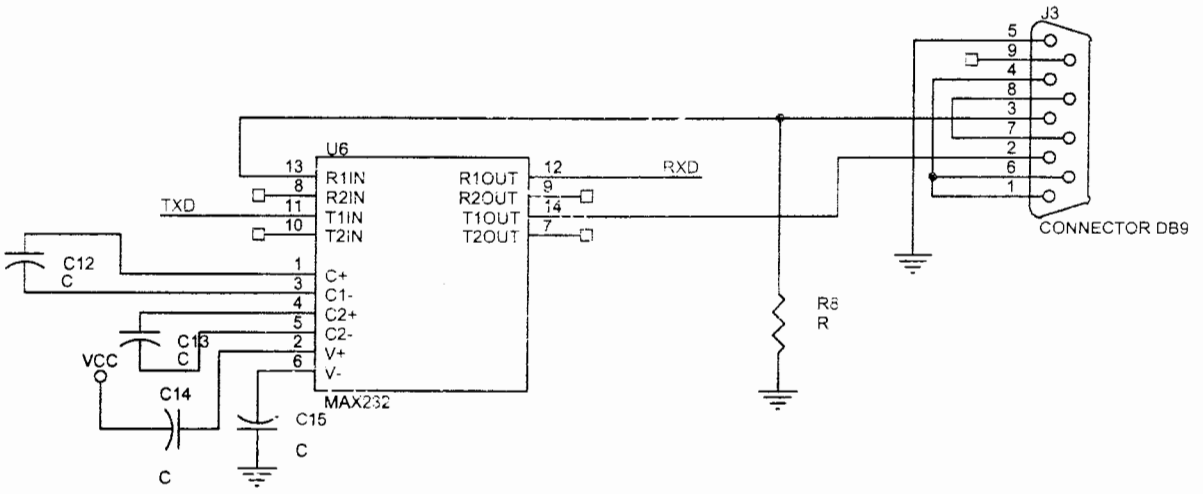
در سیستم مورد نظر که به ولتاژهای 5+، 9+ و 9- ولت نیاز است از مدار شکل ۲ـ۳ استفاده شده است. در این مدار از ترانس ۹ ولت سه سر، زنرهای ۹ ولت، رگولاتور ۵ ولت و خازنهای الکترولیتی و عدسی استفاده شده است.



شکل ۲ـ۳ـ منبع تغذیه مورد نیاز

۲ـ۴ـ۳ـ پورت سریال

با استفاده از پورت سریال و پورت موازی می توان اطلاعاتی را بین کامپیوتر و یک سخت افزاری جانبی انتقال داد. در تمامی کامپیوترهایی که پورت سریال دارند با استفاده از آن می توان اطلاعاتی را ارسال و دریافت نمود اما در حالت موازی همیشه می توان اطلاعاتی را از کامپیوتر دریافت ولی در بعضی از آنها نمی توان اطلاعاتی را برای کامپیوتر ارسال نمود بعبارت دیگر در پورت موازی اطلاعات ممکن است تنها در یک جهت منتقل گردند. از طرف دیگر جهت تبادل اطلاعات بین کامپیوتر و یک سخت افزار به روش سریال به سه سیم و به روش موازی حداقل ۱۲ سیم مورد نیاز است. با این توضیح تبادل اطلاعات بین کامپیوتر و پورت سریال مزایایی چون هزینه کم و سادگی به همراه خواهد داشت. در این پروژه نیز جهت تبادل اطلاعات بین کامپیوتر و برد سخت افزاری مورد نظر از پورت سریال استفاده شده است. پورت کامپیوتر از استاندارد ۲۳۲ استفاده می کند که در آن سطوح ولتاژ صفر و ۵ ولت به ترتیب به 12+ و 12- ولت تبدیل می گردند سخت افزار مورد نظر بایستی عمل عکس را انجام داده و سطوح ولتاژ دریافتی را به صفر و ۵ ولت تبدیل نماید و علاوه بر آن سطوح ولتاژ صفر و ۵ ولت را به 12+ و 12- ولت تبدیل نماید مدار مورد نظر که این وظیفه را انجام می دهد در شکل ۳ـ۳ نشان داده شده است. مدار مجتمع *ICL232* ، *MAX232* یا مشابه آنها وظیفه تبدیل سطوح ولتاژ را تنها با استفاده از تغذیه ۵ ولت انجام می دهد و قادر است دو سیگنال *TTL* را دریافت و آنرا به *RS232* و دو سیگنال *RS232* را دریافت و آنرا به *TTL* تبدیل نماید. این شکل نیز نحوه اتصال درست پایه های کانکتور طرف کامپیوتر را نیز نشان می دهد.



شکل ۳ـ۳ـ مدار واسط بین پورت سریال میکروکنترلر و پورت سریال کامپیوتر

۳ـ۴ـ۳ـ میکروکنترلر *89C55WD*

انتخاب این میکروکنترلر که از خانواده *MCS51* و ساخت شرکت *ATMEL* می باشد مزایا و قابلیتهای زیر را بهمراه خواهد داشت.

۱ـ تمامی قابلیتهای *89C51* و *89C52* در آن پیش بینی شده است.

۲ـ حجم حافظه *EEPROM* داخلی آن بجای ۴ یا ۸ کیلوبایت ۲۰ کیلوبایت بوده و می توان برنامه های حجیم تری را در آن ذخیره نمود.

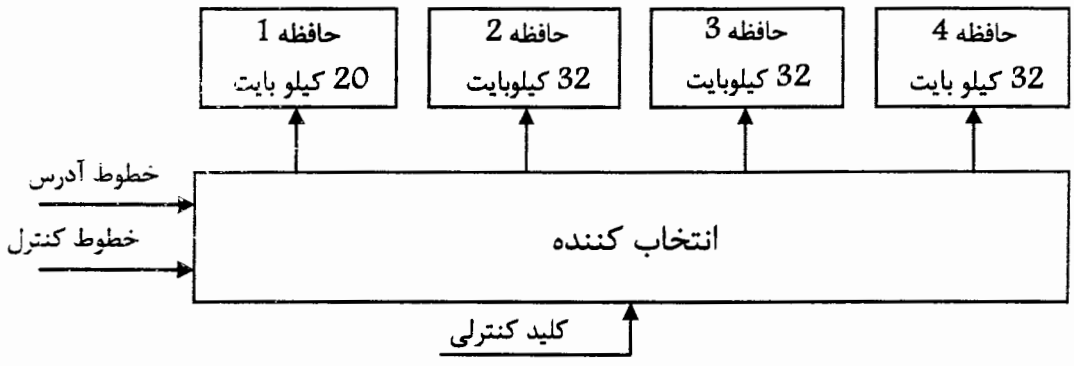
۳ـ در داخل آن زمان سنج نگهبان پیش بینی شده است که کاربرد مؤثری در سیستمهای کنترلی حساس می تواند داشته باشد.

۴ـ مانند ۸۰۵۲ سه تایمر داخلی داشته که از تایمر ۲ آن برای ارتباط با کامپیوتر از طریق پورت سریال می توان استفاده کرد در نتیجه تمامی قابلیتهای ۸۰۵۱ را می توان دراختیار داشت.

از طرفی چون از لحاظ وضعیت پینها دقیقاً مشابه *89C51* و *89C52* می باشد براحتی می توان بجای آن از *89C51* یا *89C52* در صورت دراختیار نبودن آن استفاده کرد.

۴ـ۴ـ۳ـ انتخاب کننده و حافظه ها

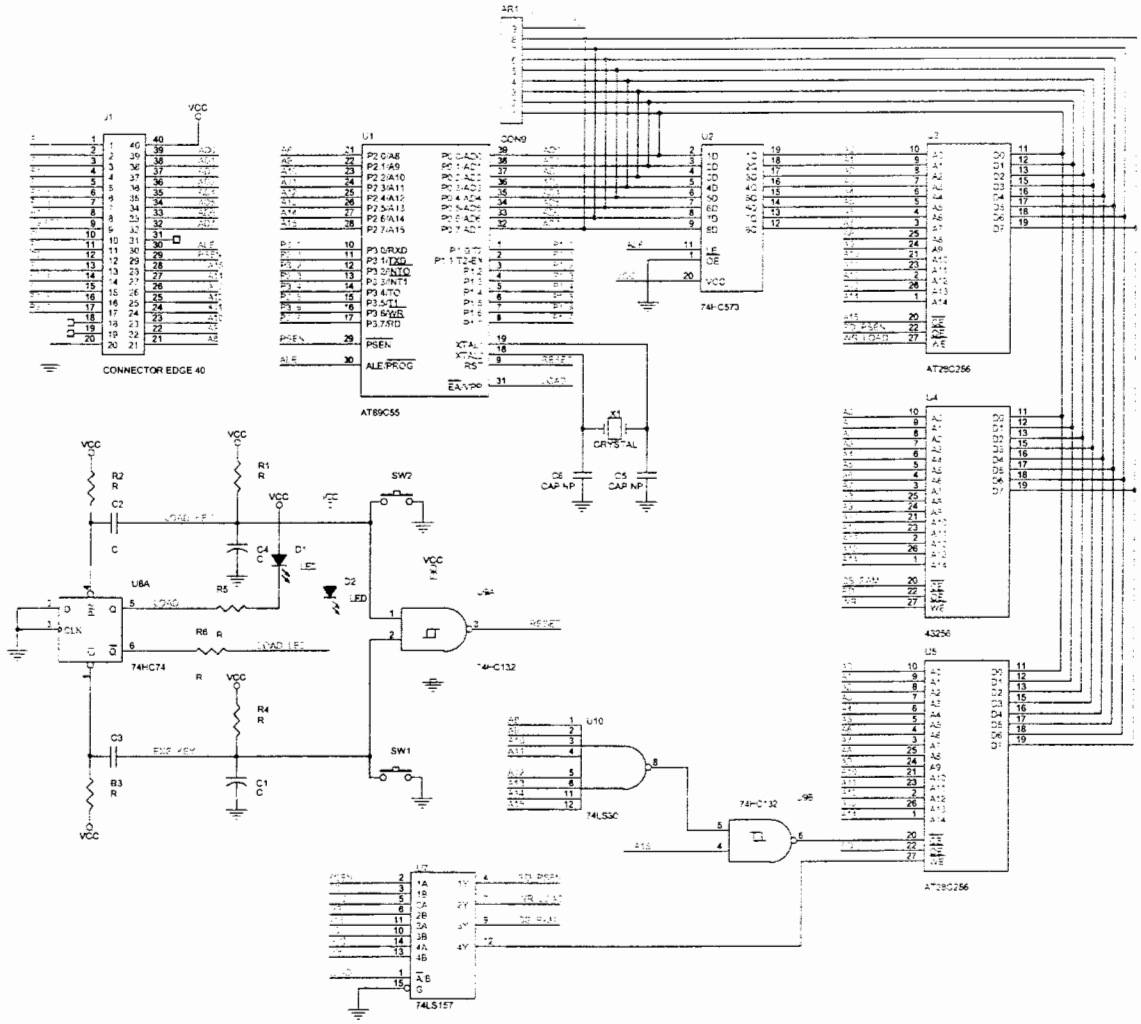
از قسمتهای اصلی سیستم می نیمم این قسمت بوده که وظیفه مهمی را بعهده دارد ساخت این قسمت به روشهای مختلفی ممکن و عملی است. تاکنون نگارنده در سخت افزارهای مختلف با این موضوع برخورد نکرده و ایده بکارگیری آن و چگونگی ساخت آن توسط طراح پیشنهاد می گردد و تاکنون به چند روش آنرا روی میکروکنترلرهای *8051*، *68HC11* و *80196* پیاده نموده است. جهت ساخت آن بایستی توابع مورد نظر را که برای هر میکروکنترلر متفاوت است تشکیل داده سپس با استفاده از مدارات منطقی مانند گیتها، دیکدر و مالتی پلکسر آنرا پیاده سازی کرد. در روش دوم می توان بجای مدارات منطقی از مدارت قابل برنامه ریزی مانند *PAL* استفاده کرد و در روش سوم که جدیدتر بوده تکمیل شده روشهای قبلی می باشد ترتیبی داده شده است که پس از عملکرد مدار انتخاب کننده سیستم بلافاصله ریست شده و تغییرات جدید بدون هیچ مشکلی در عملکرد سیستم تأثیر گذاشته و روند مورد نظر طی شود. بطور خلاصه در این سخت افزار خاص مانند شکل ۴ـ۳ چهار حافظه پیش بینی شده است که حافظه ۱ و ۲ ضروری بوده اما حافظه سوم و چهارم برای افزایش قابلیتهای سیستم منظور شده است. در این سخت افزار حافظه ۱ از نوع *EEPROM* حافظه ۲، ۳ و ۴ از نوع *RAM* یا *EEPROM* می باشند. حافظه ۱ نقش حافظه برنامه، حافظه ۲ نقش حافظه دیتا یا برنامه و حافظه های سوم وچهارم نقش حافظه دیتا را در سخت افزار بازی می کنند البته توابع استخراج شده که این حافظه ها را کنترل می کنند با توجه کار خواسته شده می توانند آنها را غیر فعال یا فعال نموده و نقش آنها را در سخت افزار تعیین نمایند.



شکل ۴ـ۳ـ بلوک دیاگرام انتخاب کننده و حافظه

۵ـ۴ـ۳ـ کانکتور ارتباطی

جهت اینکه بتوان کل پایه های میکروکنترلر را در اختیار یک برد تکمیلی خارجی قرار داد و از آنها در صورت لزوم استفاده کرد یک کانکتور ۴۰ پین به سخت افزار اضافه شده است. شکل ۵ـ۳ این کانکتور *(J1)* را بهمراه کل سخت افزار اولیه برای سیستم مینیمم فوق نشان می دهد.

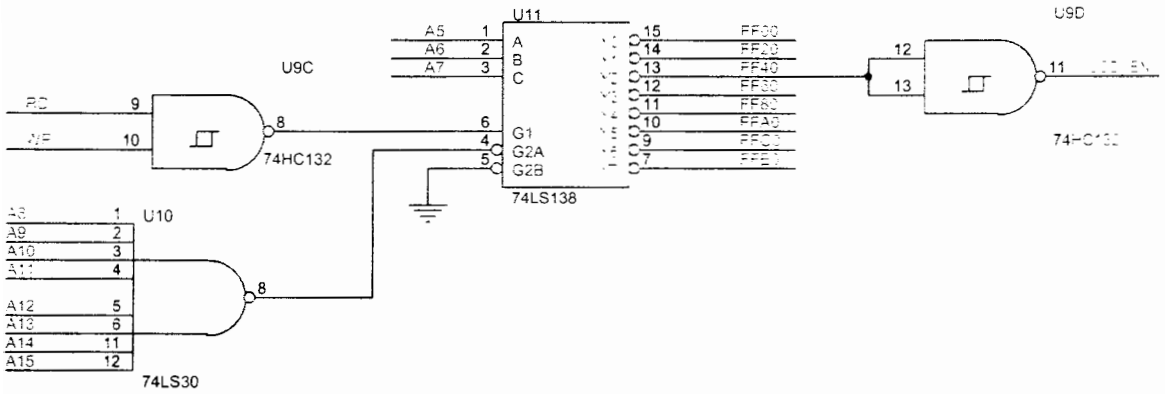


شکل ۵ـ۳ـ سخت افزار سیستم مینیمم

۵ـ۳ـ پورتهای ورودی خروجی

همانطور که اشاره شده وجود این قسمت در سخت افزار برای افزایش کارآیی سیستم برای مقاصد آموزشی و گاهی صنعتی در نظر گرفته شده است. در این قسمت سخت افزارهایی که معمولاً دانشجویان با آن سرو کار دارند و از لحاظ آموزشی می تواند مؤثر باشد انتخاب شده است علاوه بر این سخت افزارهایی که می تواند در کاربردهای تحقیقاتی و پژوهشی مورد استفاده قرار گیرد همچون داریور موتور های پله ای و *DC* منظور شده اند با وجود سخت افزارهای موجود تعداد آزمایشهایی که می توان برای آن تعریف نمود بسیار زیاد بوده و می تواند بسیار ساده یا پیچیده باشد. دیگر مورد استفاده جهت انتخاب پورتهای مختلف در شکل ۶ـ۳ نشان داده شده است همانطور که مشاهده می شود آدرس پورتها در فضای

FF00h الی *FFFFh* قرار گرفته اند که در این فضا مطابق شکل ۵ـ۳ حافظه مورد استفاده نیز غیر فعال شده ست.

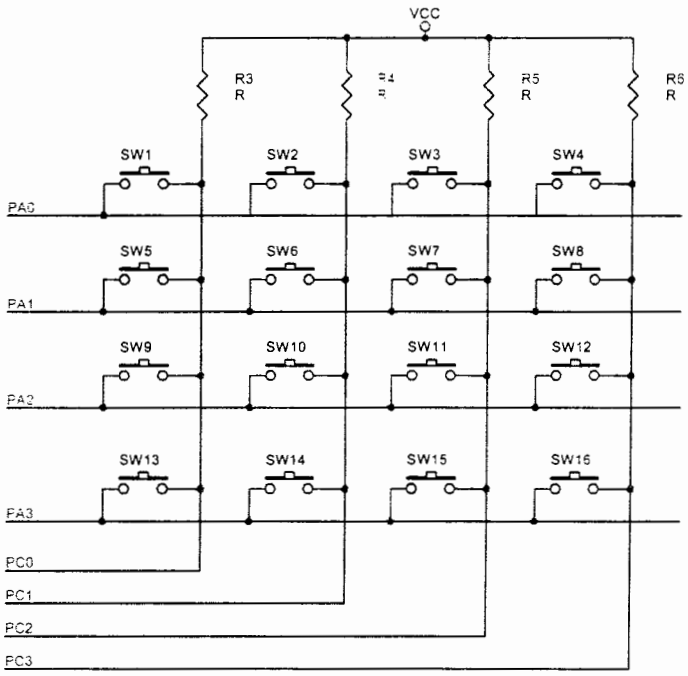


شکل ۶ـ۳ـ دیکدر مورد استفاده جهت انتخاب پورتها

در ادامه سخت افزارهای اضافه شده بصورت مختصر بهمراه شکلهای مربوطه نشان داده شده اند.

۱ـ۵ـ۳ـ صفحه کلید

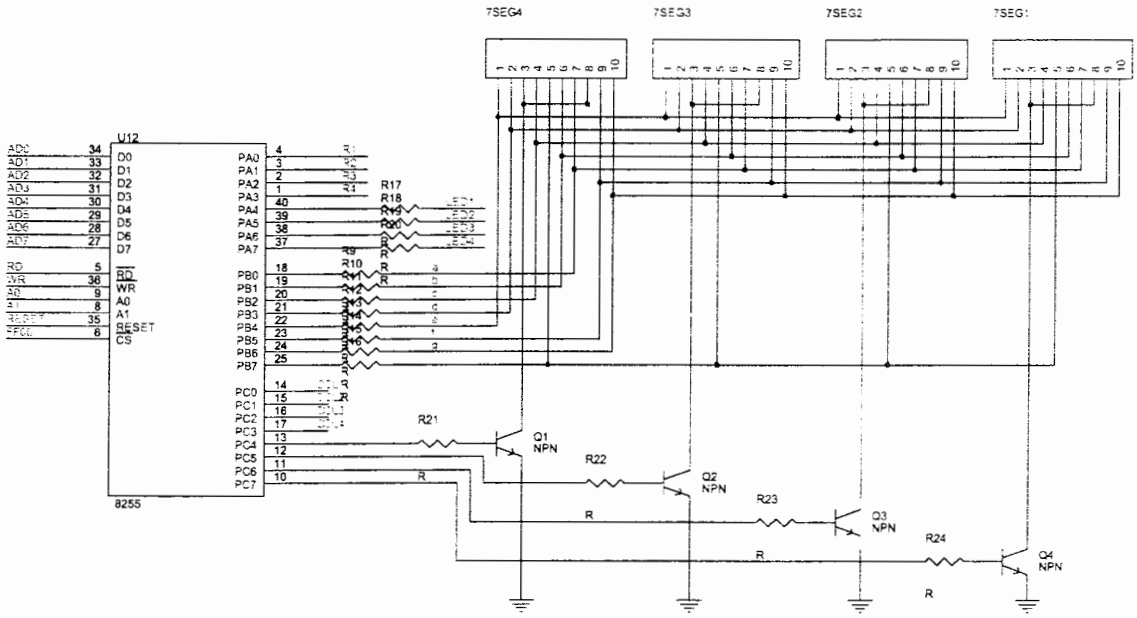
برای سیستم مورد نظر صفحه کلیدی ۱۶ تایی که بصورت ماتریسی مانند شکل ۷ـ۳ قرار گرفته اند پیش بینی شده است در حالت معمولی این کلیدها برای سخت افزار هیچ مفهوم خاصی نداشته و تنها روی آن اعدادی نوشته شده است این برنامه نویس است که بایستی الگوریتم مربوطه را نوشته برای هرکلید وظیفه ای در صورت نیاز تعیین نماید. برنامه ریزی پورت ورودی خروجی *8255* و خواندن صفحه کلید از طریق آن از موارد آموزشی مناسبی است که در بخش نرم افزار در مرور مثالها به آن پرداخته می شود.



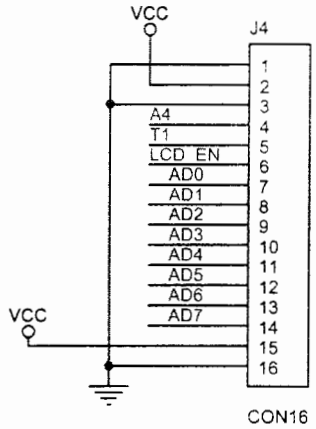
شکل ۷ـ۳ـ صفحه کلید ماتریسی ۱۶ تایی

۲ـ۵ـ۳ـ نمایش دهنده

۴ عدد *7_Segment* کاتد مشترک و یک عدد *LCD 2x16* نمایش دهنده های پیش بینی شده روی سخت افزار مطابق شکل ۸ـ۳ و ۹ـ۳ می باشند. کد گذاری *7_Segment* ها، نحوه روشن کردن آنها، جاروب کردن آنها و نمایش ۴ عدد در مبنای ۱۶ روی آنها با کیفیت مناسب از موارد آموزشی مد نظر است. علاوه بر این برنامه ریزی *LCD* و نحوه کار با آن خالی از لطف نبوده و می تواند در کیفیت آموزش مؤثر باشد.



شکل ۸ـ۳ـ نحوه اتصال ۴ عدد *7_Segment* به پورت های خروجی *8255*

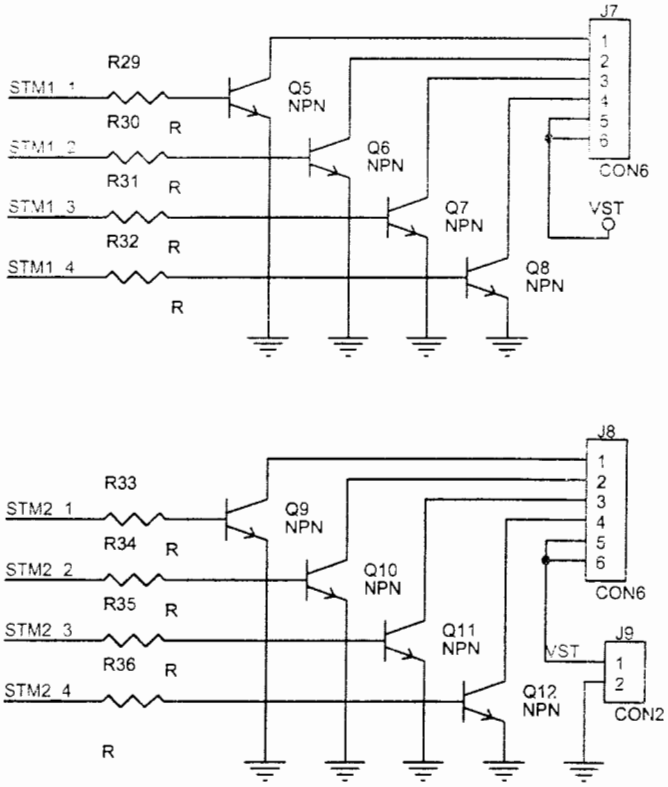


شکل ۹ـ۳ـ اتصال *LCD* به سخت افزار مورد نظر

۳ـ۵ـ۳ـ درایور موتور پله ای

اکنون کار با موتورهای پله ای مخصوصاً جهت کنترل حرکت روباتها لازم و ضروری بنظر می رسد. برای این منظور سخت افزار لازم برای کنترل دو موتور پله ای مانند شکل ۱۰ـ۳ پیش بینی شده است. در این

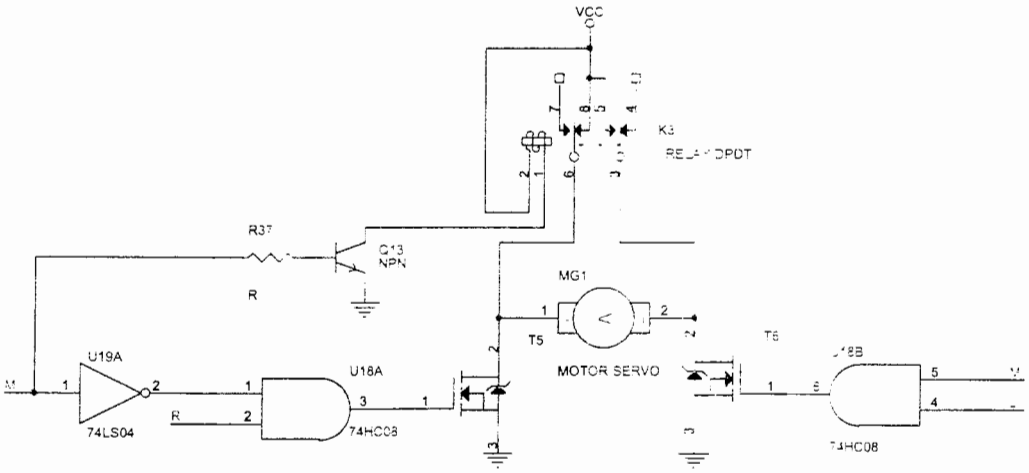
شکل ۸ خروجی دیجیتال که توسط پورت ورودی خروجی *8255* تولید می شوند جریان آنها بوسیله ترانزیستور تقویت شده و مورد استفاده قرار گرفته است.



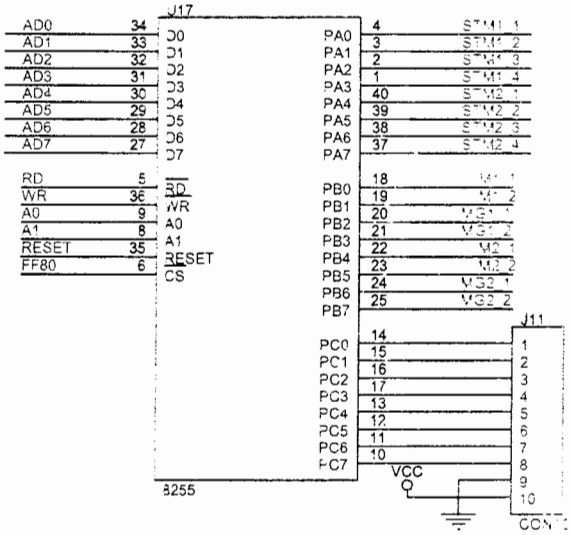
شکل ۱۰ـ۳ـ داریو موتورهای پله ای توسط سخت افزار

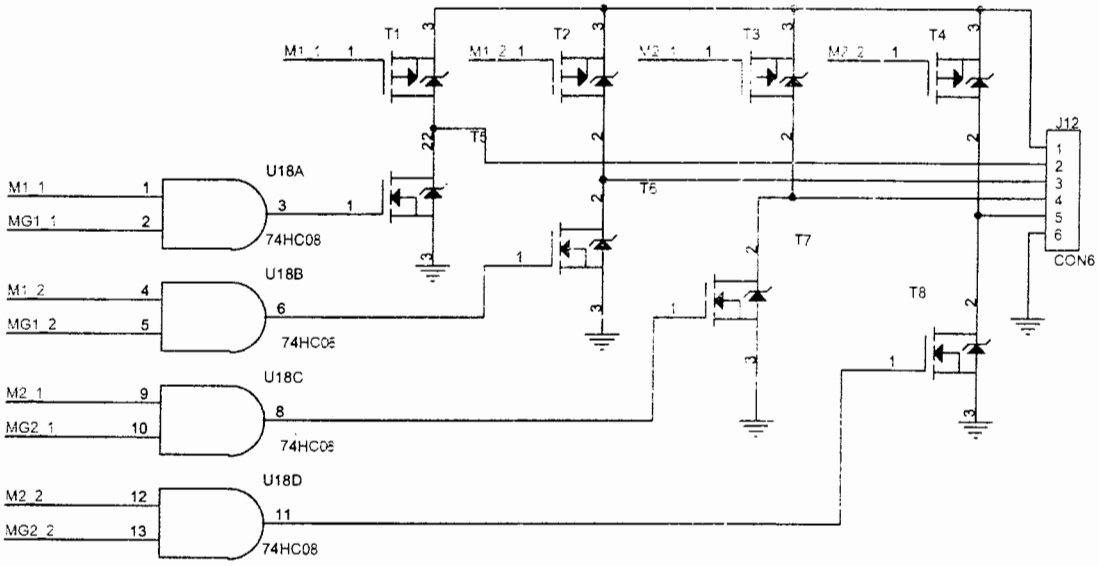
۴ـ۵ـ۳ـ درایور موتور *DC*

فرقی که این درایور با درایور موتور پله ای دارد به ساختار داخلی موتور *DC* و موتور پله ای برمی گردد در موتور پله ای اگر پالسها بترتیبی خاص که بستگی به نوع موتور پله ای دارد از بالا به پایین اعمال گردد موتور در یک جهت و اگر پالسها از پایین به بالا اعمال گردند در جهت عکس خواهد چرخید اما در موتور *DC* وضعیت فرق خواهد کرد. برای چرخش موتور در یک جهت مشخص بایستی جهت ولتاژ اعمالی به موتور بفرمی و اگر چرخش در جهت عکس مد نظر باشد بایستی جهت ولتاژ تغییر نماید. مداری که می تواند جهت ولتاژ اعمالی به موتور را با یک رله تغییر دهد با استفاده از ترانزیستور *MOSFET* در شکل ۱۱ـ۳ نشان داده شده است عیب این مدار در این است که با سرعت زیاد نمی توان جهت را تغییر داد. مداری بهتر اما با هزینه بیشتر که مورد آزمایش قرار گرفته است در شکل ۱۲ـ۳ نشان داده شده است مدار بنحوی طراحی شده است که همزمان ترانزیستورهای موجود در یک ستون تحت هیچ شرایطی روشن نگردند. در این شکل ترانزیستورهای تعیین کننده جهت حرکت *MOSFET* کانال *P* و ترانزیستورهای روشن یا خاموش کردن موتور از نوع *MOSFET* کانال *N* می باشد جهت روشن نمودن ترانزیستور *MOSFET* کانال *P* بایستی یک ولتاژ صفر به گیت آن اعمال کرد و برای روشن نمودن ترانزیستور *MOSFET* کانال *N* بایستی ولتاژی برابر ولتاژ سورس به گیت آن اعمال کرد. در مدار مورد نظر می توان دو موتور *DC* را با جریانی حدود ۵ آمپر و ولتاژی حدود ۱۰۰ ولت کنترل نمود.



شکل ۱۱ـ۳ـ کنترل جهت حرکت موتور با استفاده از رله و ترانزیستور



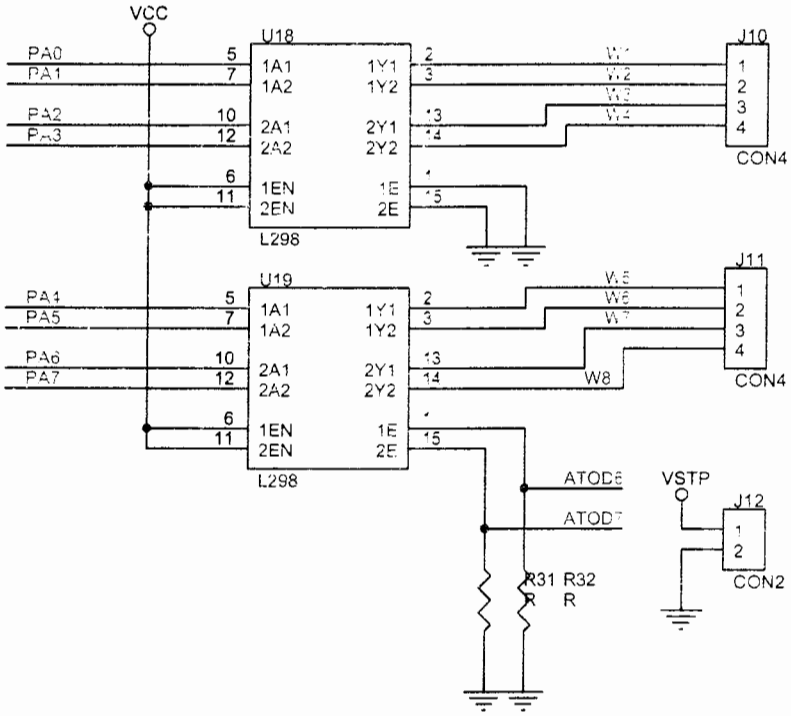


شکل ۱۲ـ۳ـ دایو موتورهای *DC* توسط سخت افزار

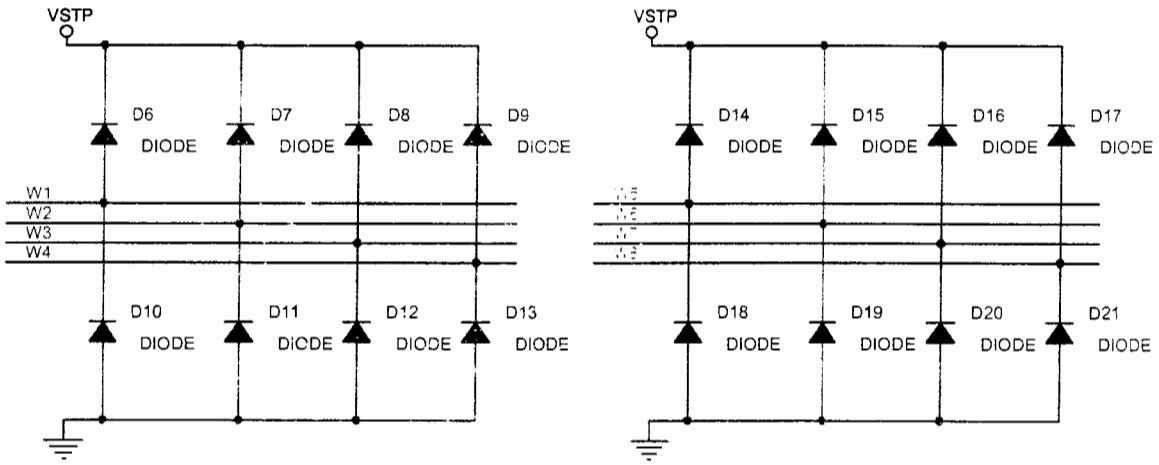
۵ـ۵ـ۳ـ درایور موتور پله ای و *DC* با استفاده از *L298*

L298 مدار مجتمع مناسبی برای راه اندازی موتورهای *DC* می باشد. این مدار مجتمع قادر است همزمان برای راه اندازی دو موتور *DC* استفاده شود. با استفاده از آن می توان جهت ولتاژ را تغییر و در نتیجه جهت چرخش را نیز مستقل از هم عوض نمود. علاوه بر این می توان از آن برای راه اندازی موتورهای پله ای خاص که چهار سر سیم دارند کمک گرفت البته با ندیده گرفتن سیمهای مشترک بین دو سیم پیچ در موتورهای پله ای می توان برای راه اندازی موتورهای پله ای ۶ سیم نیز استفاده کرد.

مدار شکل ۱۳ـ۳ نحوه اتصال آنرا به پورت *A*، ۸۲۵۵ نشان می دهد لذا با استفاده از *L298* بجای مدارت قبلی که در شکلهای ۱۰ـ۳، ۱۱ـ۳ و ۱۲ـ۳ نشان داده شدند دو پورت *B* و *C* آزادند لذا می توان از آنها برای منظورهای دیگری کمک گرفت. جهت حفاظت مدارهای مجتمع فوق، لازم است دیودهایی مطابق شکل ۱۴ـ۳ به خروجیهای آنها متصل گردد.



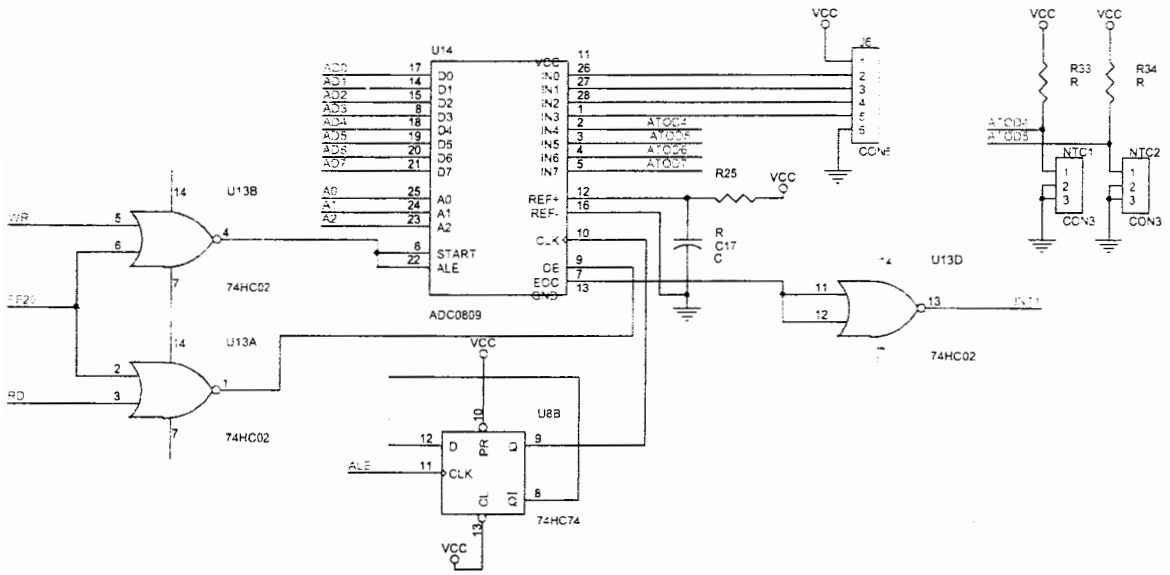
شکل ۱۳ـ۳ـ اتصال دو عدد *L298* به پورت *A* ۸۲۵۵



شکل ۱۴ـ۳ـ حفاظت *L298* با دیودهای هرزگرد

۶ـ۵ـ۳ـ مبدل آنالوگ به دیجیتال

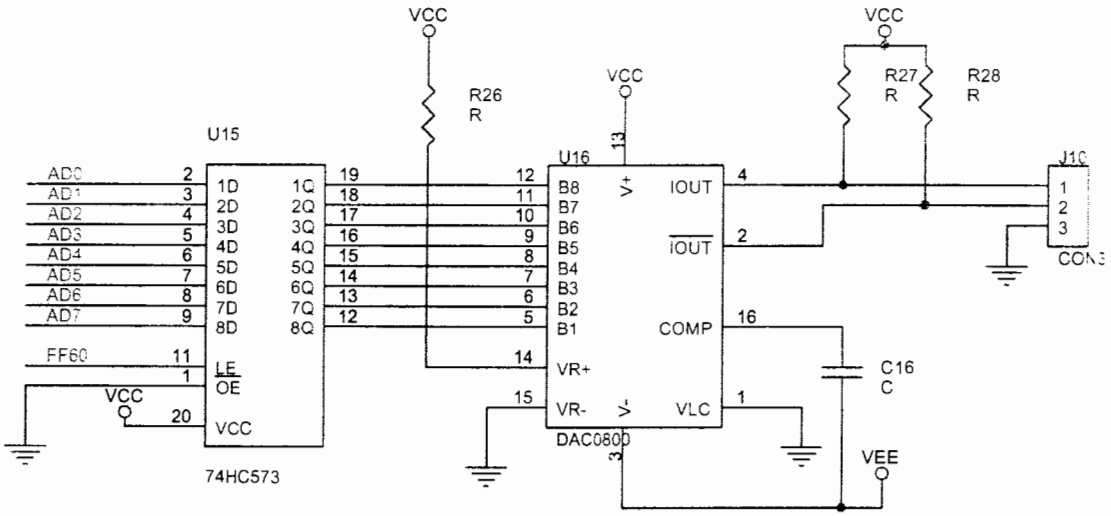
با استفاده از مبدلهای آنالوگ به دیجیتال می توان ارتباط سیگنالهای آنالوگ را که به وفور یافت می شوند و اغلب سنسورها آنرا در اختیار قرار می دهند با برد سخت افزاری دیجیتال برقرار نمود. مبدل مورد استفاده مانند شکل ۱۵ـ۳ . *ADC0809* می باشد که قادر است ۸ سیگنال آنالوگ را بصورت مالتی پلکس توسط یک کانکتور دریافت دارد. این مبدل ۸ بیتی بوده و سرعت تبدیل آن ۱۰۰ میکروثانیه است که برای خیلی از کاربردها مناسب است. این شکل نشان می دهد کلاک لازم برای *A/D* از سیگنال *ALE* میکروکنترلر که تقسیم بر ۲ می گردد استفاده شده است.



شکل ۱۵ـ۳ـ اتصال مبدل آنالوگ به دیجیتال

۷ـ۵ـ۳ـ مبدل دیجیتال به آنالوگ

کنترل سیستمهای واقعی معمولاً با یک سیگنال پیوسته کنترل می گردد بدین جهت این مبدل در سخت افزار مطابق شکل ۱۶ـ۲ پیش بینی شده است. این مبدل ۸ بیت اطلاعات را دریافت و متناسب با آن یک ولتاژ پیوسته بین ۵- و ۵+ و یا یک ولتاژ پیوسته بین صفر و ۵+ را در اختیار قرار می دهد.



شکل ۱۶ـ۳ـ اتصال مبدل دیجیتال به آنالوگ

۶ـ۳ـ نتیجه گیری

در این فصل قسمتهای مختلف سخت افزار طراحی شده توضیح داده شد. ابتدا این سخت افزار روی برد سوراخ دار بصورت وایرپ بسته شده و آزمایشهای مختلف روی آن صورت گرفت. پس از دریافت جواب مطلوب نقشه ارکد تهیه و مقدمات طراحی *Layout* فراهم گردید. نتایج حاکی از آن است که سخت افزار در حال حاضر مشکل خاصی نداشته و می توان اقدامات بعدی جهت تکمیل طرح را انجام داد.

فصل چهارم

مبانی نرم افزار

۱ـ۴ـ مقدمه

در این فصل مبانی نرم افزار و آنچه که به بحث نرم افزار پروژه مربوط است مورد نقد و بررسی قرار می گیرد. مطالبی چون ملاحظات لازم در برنامه نویسی، زبان برنامه نویسی، ساخت فایل قابل اجرا و ... بررسی شده و در ضمن بیان می دارد که جهت انجام پروژه چه اطلاعاتی بایستی جمع آوری و مورد نظر قرار گیرد.

۲ـ۴ـ برنامه نویسی

بحث برنامه نویسی و چگونگی اجرای آن بسیار وسیع و گسترده بوده و در اینجا مورد بحث قرار نمی گیرد. هدف از بیان این موضوع بررسی روند و قواعد برنامه نویسی هنگام کار با بردهای سخت افزاری و به نتیجه رساندن یک پروژه در حد متوسط است، لذا آنچه که به این بحث مربوط می شود بطور خلاصه مورد بررسی قرار می گیرد تا راهنمایی برای افراد مبتدی باشد تا سریعتر به هدف مورد نظر برسند. در برنامه نویسی معمولاً بایستی مراحل زیر را بررسی و کامل نمود تا نتیجه مطلوب حاصل آید.

الف- صورت مسئله یا پروژه مورد نظر بفرم واضح بیان گردد.

ب- با توجه به صورت مسئله الگوریتم یا فلوچارت برنامه ترسیم گردد.

ج- برنامه با رعایت قواعد برنامه نویسی با یک متن نگار نوشته شود.

د- برنامه نوشته شده عیب یابی دستوری گردد.

ه- فایل قابل اجرا ساخته شود.

و- برنامه نوشته شده اجرا شده و عیب یابی گردد.

زـ برنامه نوشته شده مستند سازی گردد.

با اتمام مرحله به مرحله موارد فوق انتظار می رود حل مسئله در زمان کوتاهی به نتیجه مطلوب برسد. اما بایستی توجه نمود هر مرحله قید شده خود از مراحل جزئی تر تشکیل شده که بایستی بدقت تکمیل گردند و اطلاعات مرتبط با آن جمع آوری گردد.

۱ـ۲ـ۴ـ صورت مسئله

در پروژه های کاربردی و عملی یکی از موارد مشکل و وقت گیر تعریف صورت مسئله است. اگر پروژه از جانب فرد یا گروه یا شرکتی پیشنهاد شده باشد و تمامی جزئیات در آن منعکس شده باشد طراح بدون هیچ دغدغه ای به طراحی سخت افزار و و نوشتن نرم افزار پرداخته و تنها مشکل او بعد علمی بوده و کار

او مشخص و روتین وار می باشد. اما همیشه اینطور نیست گاهی صورت مسئله کامل نبوده و تنها کلیات در آن بیان شده است در این صورت کار طراح یا برنامه نویس مشکل بوده و بایستی با پرس و جوهای مختلف، با در نظر گرفتن محدودیتها، با در نظر گرفتن امکانات و وقت و هزینه پیش بینی شده کلیه جزئیات لازم را استخراج و در طراحی و برنامه نویسی لحاظ نماید.

گاهی اوقات صورت مسئله یا پروژه در ابتدای امر بظاهر مناسب تعریف شده است و مشخصات لازم تعیین شده اند اما با گذشت زمان و تجربه بیشتر، صورت مسئله دچار تغییراتی می گردد که گاهی نمی توان از آن اجتناب کرد بعنوان مثال طرح قبلی اعتبار خود را از دست داده و بایستی آنرا با قابلیتهای بیشتری جهت جلب مشتری ارائه کرد و یا پیش فرضهای نامناسبی در صورت مسئله منظور شده است که بایستی اصلاح گردند. در هر صورت با در نظر گرفتن موارد بالا طراح بایستی بداند جهت انجام یک پروژه نرم افزاری یا سخت افزاری دانستن کلیه جزئیات لازم و ضروری بوده و بایستی قبل از هر اقدامی به مشخص کردن آنها اقدام کند. بعنوان مثال مسئله زیر را در نظر بگیرید:

سیستمی طراحی کنید که درجه حرارت اتاق را کنترل نماید.

صورت مسئله از دید یک کارفرما کامل بوده و نقصی ندارد اما برای طراح سخت افزار و برنامه نویس جزئیات بسیاری است که بایستی روشن گردد بعنوان مثال:

۱ـ سیستم سرمایشی یا گرمایشی از چه نوعی است و چگونه کنترل می گردد.

۲ـ آیا بهترین کنترل از لحاظ مصرف انرژی مد نظر است.

۳- آیا از سخت افزارهای موجود می توان استفاده کرد و یا بایستی سخت افزار خاصی طراحی کرد.

۴- سنسور گرما چه باشد و طبق چه مکانیسمی عمل کند.

۵- زبان برنامه نویسی چه باشد.

۶- از چه الگوریتمی برای کنترل استفاده گردد بعبارت دیگر کنترل کننده از چه نوعی باشد.

۷- هزینه تمام شده چقدر باشد.

وجود این نوع سؤالها هنگام اجرای یک پروژه عملی کاملاً طبیعی بوده که تنها با مطالعه و تحقیق و انجام پرس و جوهای متعدد می توان به قسمتی از آنها پاسخ گفت. لذا قبل از هر اقدامی بایستی صورت مسئله دقیقاً تشریح و خواسته ها مشخص گردند.

یکی از مشکلات موجود در اجرای این پروژه نیز تعیین مشخصات لازم برای سخت افزار و تعیین قابلیتهای لازم برای نرم افزار بود که پس از بررسی های فراوان سخت افزار و نرم افزار مورد نظر که اشکالاتی نیز خواهد داشت ارائه گردید.

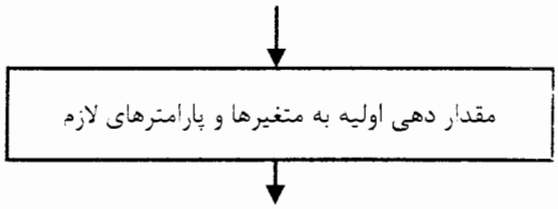
۲ـ۲ـ۴ـ الگوریتم یا فلوچارت برنامه نویسی

پس از اینکه صورت مسئله تشریح شد و خواسته ها مشخص شد بایستی تمامی راه حلهای ممکن برای حل مسئله مورد نظر قرار گیرد و با توجه به نوع مسئله، هزینه در نظر گرفته شده، تخصص لازم، دردسترس بودن تجهیزات لازم و تجربه طراح از میان راه حلهای ممکن مناسب ترین راه حل انتخاب و سپس اقدامات بعدی صورت گیرد. با انتخاب راه حل مناسب، بایستی نحوه پیاده سازی آن با جزئیات

کامل بیان گردد. اجرا ممکن است شامل قسمتهای مختلف باشد که یک قسمت آن نوشتن برنامه لازم است. منطق اجرای برنامه را می توان بصورت کلامی و یا بصورت تصویری نمایش داد. اگر منطق برنامه بصورت کلامی بیان گردد به آن الگوریتم و اگر بصورت تصویری نمایش داده شود به آن فلوچارت گفته می شود. لذا قبل از اینکه اقدام به نوشتن برنامه گردد لازم است ابتدا چارچوب کلی آنرا که روند کار را بسادگی نشان دهد ترسیم گردد. این ترسیم بایستی بنحوی باشد که مستقل از زبان برنامه نویسی باشد. با پیاده سازی منطق برنامه بصورت فلوچارت و سپس نوشتن برنامه حل مسئله از یک مشکل پیچیده به دو مشکل تقریباً مستقل از هم تفکیک می گردد. بعبارت دیگر برای حل مسئله هنگام نوشتن برنامه به منطق کاری نداشته و هنگام ترسیم منطق به زبان برنامه نویسی و مشکلات آن کاری نداشته لذا فرد در آن واحد با یک مشکل دست به گریبان بوده در نتیجه راحتتر می تواند بر مشکلات فائق آید. علاوه بر این می توان حل مسئله را به دو گروه کاری مستقل از هم که یکی فلوچارت و دیگری برنامه را می نوید تفکیک کرد. همچنین در پیاده سازی منطق برنامه که در نهایت بصورت فلوچارت ارائه می گردد سعی می شود مسئله به اجزاء کوچکتر تقسیم و برای هر جزء راه حلی ارائه گردد و فلوچارت مربوطه ترسیم گردد.

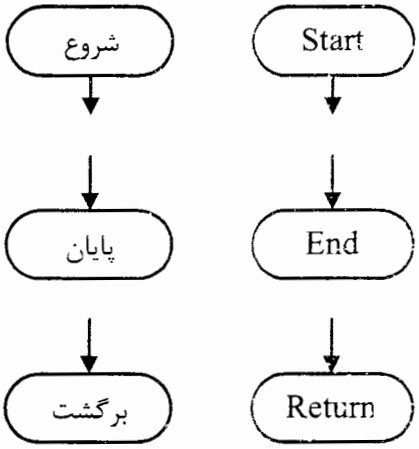
فلوچارت ترسیم شده سعی می شود برای افراد مختلف خوانایی لازم را داشته باشد در نتیجه استفاده از نمادهای استاندارد که بعضی از آنها بیان می گردند استفاده می شود.

نماد عملیات: این نماد که بصورت مستطیل بوده و در شکل ۴ـ۱ نشان داده شده است بیانگر شرح عملیات است. این عملیات می تواند محاسباتی یا منطقی باشد. شرح عملیات مورد نظر معمولاً به صورت فرمولهای ریاضی و یا بصورت گزاره ای بیان می گردد که در داخل آن نوشته می شود. ممکن است یک یا چند عملیات در یک نماد نمایش داده شود و ترتیب قرار گیری آنها، ترتیب اجرای آنها را در برنامه مشخص کند.



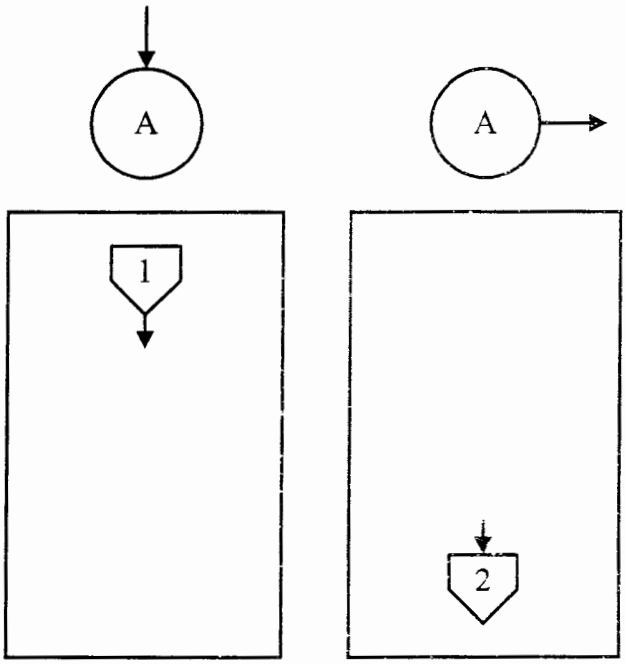
شکل ۴ـ۱ـ نماد عملیات و پردازش

نماد شروع و پایان: این نماد که در شکل ۴ـ۲ نشان داده شده است بصورت بیضی بوده و شروع یک فلوچارت و یا خاتمه آنرا نشان می دهد چنانچه فلوچارت رسم شده انتهای یک زیر برنامه را مشخص نماید از نماد برگشت استفاده می گردد.



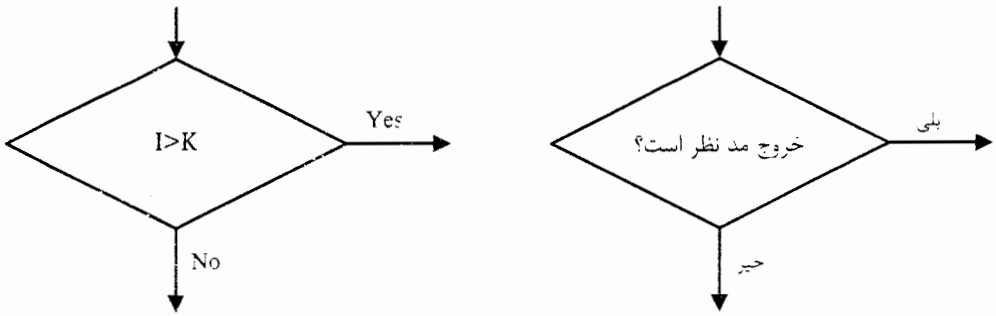
شکل ۴ـ۲ـ نماد شروع یا پایان یک فلوچارت

نماد های ارتباط: در فلوچارت برای جلوگیری از قطع شدن مسیری توسط مسیر دیگر به جای رسم خط از نمادهای رابط جهت نشان دادن پیوند استفاده می گردد. جهت هر پیوند دو رابط که هرکدام به شکل دایره می باشند به کار برده می شود و روی هر دو یک نوع مشخصه مثلاً حرف یا عدد نوشته می شود. حروف یا اعداد یکسان نشان دهنده یک پیوند و جهت پیکان نشان دهنده جریان عملیات می باشد. اگر فلوچارت در بیش از یک صفحه رسم گردد جهت پیوند صفحات می توان از نماد فوق و یا نمادهایی مطابق شکل ۴ـ۳ استفاده کرد.



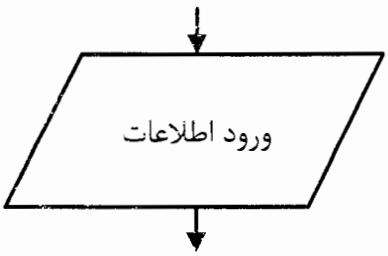
شکل ۴ـ۳ـ نمادهای ارتباط با قسمت دیگر فلوچارت در همان صفحه و یا صفحه بعد

نماد تصمیم و داوری: این نماد مطابق شکل ۴ـ۴ از یک لوزی تشکیل شده است که در داخل آن عباراتی چون آیا با علامت سؤال آورد شده و بطور ضمنی سؤالی مطرح می گردد و مطابق با شرایط موجود که حاصل عملیات قبلی است مسیر اجرا مشخص می گردد.



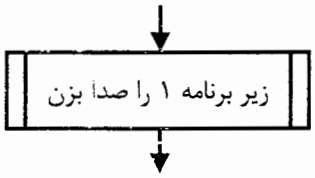
شکل ۴ـ۴ـ نماد تصمیم و داوری

نماد های ورودی و خروجی: جهت خواندن اطلاعات از دستگاههای ورودی و یا نوشتن وثبت اطلاعات در دستگاههای خروجی از نمادی مطابق شکل ۵ـ۴ استفاده می گردد. در داخل نماد مشخص می گردد چه اطلاعاتی خوانده و چه اطلاعاتی نوشته می شود. دستگاههای ورودی می توانند صفحه کلید، پورت سریال، یک درگاه ورودی و دستگاههای خروجی می توانند پورت سریال، درگاه خروجی، مونیتور و ... باشد.



شکل ۵ـ۴ـ نماد ورودی و خروجی

نماد فراخوانی زیر برنامه: در هریک از مراحل پردازش که نیاز به فراخوانی زیر برنامه ای باشد در فلوچارت با استفاده از نماد فراخوانی زیربرنامه لازم مشخص و صدا زده می شود. مطابق شکل ۶ـ۴ نماد فراخوانی زیر برنامه به شکل مستطیل می باشد که عرض آن با دو خط قائم مشخص می گردد و نام زیر برنامه که بایستی فراخوانی شود در داخل مستطیل نوشته می شود.



شکل ۶ـ۴ـ نماد فراخوانی زیر برنامه ها

۳ـ۲ـ۴ ـ نوشتن برنامه با رعایت قواعد برنامه نویسی

بعد از تهیه فلوچارت برنامه و قبل از شروع برنامه نویسی چند موضوع بایستی دقیقاً مورد بررسی قرار گرفته، نقاط ضعف و قدرت مشخص شده و انتخابهای لازم صورت گیرد که در این بین می توان به مسائل زیر اشاره کرد:

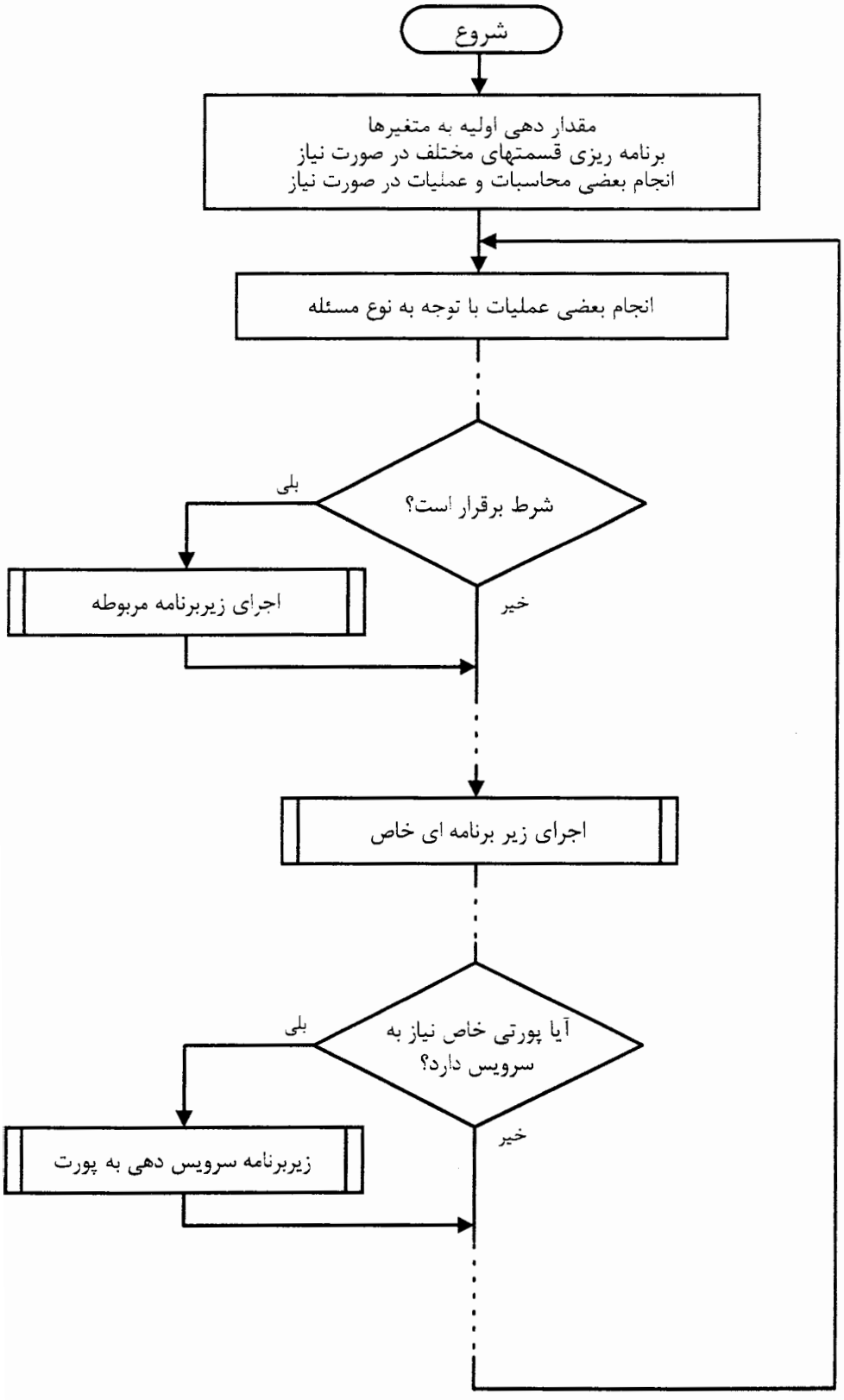
۱ـ **زبان برنامه نویسی:** یک برنامه را می توان به زبانهای مختلف برنامه نویسی همچون اسمبلی، *C*، پاسکال، ویژوال *C*، ویژوال بیسیک، دلفی و ... نوشت. انتخاب زبان برنامه نویسی بسیار اهمیت دارد. انتخاب آن بسته به هدف مورد نظر، دراختیار بودن امکانات لازم، داشتن تخصص لازم و قابلیتهای زبان برنامه نویسی بستگی دارد. که در این بین داشتن تخصص لازم از اهمیت زیادی برخوردار است. برنامه نویس بایستی سعی نماید در آن واحد تنها با یک مشکل دست به گریبان بوده و به حل مسئله بپردازد وجود چند مشکل همزمان در برنامه نویسی باعث کند شدن برنامه نویسی شده و جواب مناسبی عاید نخواهد شد.

هنگام کار با میکروکنترلرها اغلب نوشتن برنامه با اسمبلی یا *C* مد نظر است آگاهی کامل با موضوعات مربوطه، مشخص بودن قواعد برنامه نویسی، در اختیار بودن اسمبلر یا کامپایلر و داشتن اطلاعات مربوط به آنها از ضروریات بوده که بایستی مورد توجه قرار گیرد.

۲ـ **محیط برنامه نویسی:** یک برنامه را می توان با ویرایشگرهای مختلف و تحت سیستم عامل *DOS* یا *Windows* نوشت بایستی با توجه به هدف مورد نظر و آشنایی قبلی با آن محیطها به نوشتن برنامه اقدام کرد. انتخاب نابجا مشکلات برنامه نویسی را زیادتر نموده و باعث کند شدن برنامه نویسی می گردد.

۳ـ **چارچوب کلی یک برنامه نرم افزاری:** برنامه نویس بایستی چارچوب کلی برنامه نرم افزار را مشخص نماید. در این رابطه طول حلقه اصلی برنامه، میزان استفاده از انتراپتها، میزان استفاده از زیربرنامه ها، اسامی انتخابی برای متغیرها، سادگی عیب یابی برنامه، میزان استفاده از متغیرهای محلی و عمومی و ... را مد نظر قرار داده و شروع به برنامه نویسی کند. شکل ۷ـ۴ چارچوب کلی یک برنامه کامل را نشان می دهد. از ویژگیهای این ساختار کوتاه بودن حلقه اصلی و عیب یابی ساده برنامه می باشد.

۴ـ **نوشتن برنامه با توجه به فلوچارت تعیین شده:** با در نظر گرفتن موارد فوق برنامه نویس اقدام به نوشتن برنامه در یک محیط مناسب همچون ادیتور *PE2*، *TC* یا ... نموده و سپس آنرا با یک نام ذخیره کرده و اقدامات لازم جهت به نتیجه رساندن مسئله را به انجام می رساند.



شکل ۷ـ۴ـ چارچوب کلی پیشنهادی جهت نوشتن برنامه

۴ـ۲ـ۴ـ عیب یابی دستوری برنامه نوشته شده

پس از اینکه برنامه با در نظر گرفتن تمام ملاحظات برنامه نویسی نوشته شد بایستی توسط اسمبلر یا کامپایلر مورد تجزیه و تحلیل قرار گرفته و اشکالاتی چون غلطهای املائی، کاربرد دستورات غیر مجاز، عدم تعریف مناسب و بجای متغیرها، استفاده نامناسب از فرامین و ... مشخص و مرتفع گردد.

۵ـ۲ـ۴ـ ساخت فایل قابل اجرا

اگر برنامه نوشته غلط دستوری نداشته باشد و این موضوع توسط اسمبلر یا کامپایلر تأیید گردد فایلی با پسوند *obj* ایجاد می گردد. این فایل حاوی کد های برنامه بوده و می توان توسط نرم افزاری دیگر که به آن لینکر گفته می شود آنرا در صورت نیاز با دیگر فایلها لینک نموده و فایلی اجرایی که قابل انتقال به میکروپروسسور یا میکروکنترلر باشد ساخت و سپس اقدامات لازم جهت اجرای آنرا فراهم کرد.

۶ـ۲ـ۴ـ اجرای برنامه و عیب یابی منطقی آن

با توجه به منطق برنامه، حجم برنامه و تخصص برنامه نویس احتمال اینکه از برنامه نوشته شده در مراحل اولیه جواب منطقی و مناسب دریافت گردد کم است لذا بایستی برنامه مورد نظر را اجرا کرده و نتایج آنرا مشاهده کرد. در صورت عدم پاسخ مناسب به برنامه اصلی برگشته و اصلاحات لازم را انجام داد. اجرای قدم به قدم برنامه و استفاده از برنامه های شبیه ساز می تواند کمک مؤثری در عیب یابی برنامه بنماید. هرچقدر برنامه نوشته شده طولانی تر باشد احتمال وجود اشکال نیز بیشتر می باشد لذا مناسب ترین راه این است که ابتدا زیر برنامه های لازم را نوشته پاسخ آنها را مورد ارزیابی قرار داده و با کنار هم قرار دادن آنها به حل مشکل پرداخت، استفاده از برنامه نویسی مدولار و بکارگیری کمتر متغیرهای عمومی به حل مشکل کمک خواهند کرد.

۷ـ۲ـ۴ـ مستند سازی

پس از اینکه برنامه کامل و جواب مورد نظر حاصل شد ممکن است برنامه موجود برای مدت کوتاه یا طولانی جوابگوی نیازها باشد اما با گذشت زمان انتظارات و خواسته ها تغییر نموده و لازم باشد تغییراتی در برنامه ایجاد گردد. برای این منظور بایستی شرح برنامه، نمودار ساختار برنامه، منطق برنامه، لیست برنامه همراه با توضیحات لازم در دسترس باشد و این ممکن نیست مگر اینکه پس از تکمیل برنامه اقدامات لازم جهت مستند سازی صورت گیرد.

در پروژه فوق تمامی مراحل فوق بررسی و انتخابهای مناسب صورت گرفته است.

۳ـ۴ـ مراحل ساخت یک برنامه قابل اجرا روی میکروکنترلر با زبان اسمبلی

فرض می گردد با درنظر گرفتن موارد مطرح شده برنامه به زبان اسمبلی نوشته شده و هدف این است که آنرا با نرم افزار مشخصی غلط گیری و آنرا اجرا نماییم. برای عیب یابی برنامه های نوشته شده به زبان اسمبلی از اسمبلر *A51.exe* که توسط شرکت فرانکلین ارائه شده است استفاده می گردد. برای این منظور پس از اصلاح فایل دستوری *Autoexec.bat* که بتوان این برنامه را از هر دایرکتوری یا زیر دایرکتوری اجرا کرد دستور زیر را وارد می گردد. اسم فایل اسمبلی *Test.asm* و دایرکتوری مربوطه *C51* فرض می گردد :

C:>C51>a51 test.asm

اگر برنامه مربوطه غلط دستوری داشته باشد تعداد آنها را نمایش داده و جزئیات را می توان در *Test.lst* مشاهده کرد.

سپس غلطهای برنامه اصلاح و روند غلط یابی تکرار می گردد تا هنگامی که تعداد خطاها صفر گردد. با صفر شدن تعداد خطاها فایل *Test.obj* ایجاد خواهد شد.

حال با استفاده از نرم افزار *L51.exe* برنامه لینک می گردد. با استفاده از این برنامه می توان دو یا چند فایل با پسوند *obj* را با یکدیگر و با برنامه های موجود در کتابخانه نرم افزار لینک نمود. مانند زیر:

C:>C51>L51 test.obj

اگر تمامی متغیرها و زیر برنامه هایی که استفاده شده اند در برنامه های مشخص شده وجود داشته باشند فایلی با نام *Test* بدون پسوند ایجاد خواهد شد.

حال با استفاده از دستور زیر فایلی با فرمت *Hex* و پسوند *Hex* ایجاد خواهد شده که می توان آنرا به سخت افزار منتقل و اجرا نمود.

C:>C51>OHS51 Test

۴ـ۴ـ مراحل ساخت یک برنامه قابل اجرا روی میکروکنترلر با زبان *C*

مراحل ساخت یک فایل با استفاده از زبان *C* مشابه ساخت فایل با زبان اسمبلی است با این تفاوت که نرم افزار مورد استفاده بجای *A51.exe* نرم افزار *C51.exe* می باشد در نتیجه مراحل کار بصورت زیر خلاصه می گردد فرض می گردد نام فایل مورد نظر *Test.c* باشد :

C:>C51>C51 Test.c
C:>C51>L51 Test.obj
C:>C51>OHS51 Test

۵ـ۴ـ پروتکل ارتباطی

برای تبادل صحیح اطلاعات بین کامپیوتر و سخت افزار مورد نظر در مد کارکرد به صورت سریال در حالت آسنکرون بایستی شروع و خاتمه اطلاعات، تعداد بایتهای اطلاعاتی و درست منتقل شدن آنها مشخص گردد برای نیل به این منظور بایستی فرمت خاصی را که به آن پروتکل ارتباطی گفته می شود و توسط آن پیامها منتقل می گردند مشخص گردد. پروتکلهای ارتباطی متنوعی که اصول آنها مشخص است تاکنون برای این منظور ارائه شده است. می توان در این پروژه یکی از آنها را انتخاب و یا پروتکل ارتباطی مناسب تعریف کرد. به جهت سادگی و اینکه در این پروژه تعداد بایتهای ارسالی و دریافتی

همیشه یکسان در نظر گرفته می شوند فرمت پیامی بدون در نظر گرفتن تعداد بایتهای لازم برای هر قسمت بصورت شکل ۸ـ۴ پیشنهاد شده است. بطور کلی فرمت پیامها بدون در نظر گرفتن مبدأ و مقصد آنها، تعداد بایتهای ارسالی در این شکل نشان داده شده است. در ادامه قسمتهای مختلف این پیام توضیح داده می شود.

شکل ۸ـ۴- فرمت کلی پیام

الف- رمز شروع پیام: رمز شروع پیام که ممکن است یک یا چند بایت باشد در ابتدای پیام قرار گرفته و از آن به دو منظور یکی تشخیص شروع پیام و دیگری تشخیص سیستم ارسال کننده اطلاعات استفاده می شود. اگر رمز دریافت شده، مربوط به سیستم مورد نظر بود سیستم اطلاعات بعدی را دریافت و روی آنها پردازش انجام می دهد در غیر اینصورت منتظر پیام بعدی می ماند. در این فرمت سیستم دریافت کننده پیام بطور مرتب چک می کند که آیا بایت دریافتی با اولین بایت رمز یکسان است یا خیر اگر یکسان بود بایت دریافتی بعدی را با بایت بعدی رمز مقایسه می کند و اگر مجدداً یکسان بود این روند را ادامه می دهد تا کل رمز دریافت شود. لذا اگر کل رمز یکسان بود به تعداد بایتهای پیام، اطلاعات و چک سام و پریتی ها را دریافت و ذخیره می نماید. پس از دریافت کامل پیام، پریتی پیام دریافت شده محاسبه و با پریتی موجود در پیام مقایسه می گردد چنانچه پیام درست دریافت شده بود و مربوط به سیستم دریافت کننده اطلاعات بود پیام، مورد تحلیل قرار گرفته و خواسته مربوطه برآورده می شود. در این پروژه رمز پیام دوبایت در نظر گرفته شده است که در شکل نشان داده شده اند.

ب - آدرس : در سیستم مورد نظر می توان ۶۵۰۰۰ آدرس مختلف برای پورتها منظور نمود اگر کامپیوتر پیام را با رمز مناسب دریافت کند پیام از جانب سخت افزار ارسال شده است و اگر سخت افزار طراحی شده آنرا دریافت کند فرستنده کامپیوتر می باشد این آدرس ۲ بایت بوده و بعد از رمز پیام قرار می گیرد.

ج- بایتهای اطلاعاتی: در فرمت مشخص شده پس از رمز شروع پیام و آدرس دو بایت اطلاعاتی مربوط به پورت مورد نظر قرار می گیرد و می تواند یک دیتای خام یا فرمان باشد.

د - **چک سام** : برای اینکه بتوان به صحت پیام دریافت شده پی برد لازم است قبل از اینکه پیام ارسال گردد چک سام آن محاسبه و در پیام قرار گیرد. پس از دریافت پیام مجدداً چک سام پیام محاسبه شده اگر با چک سام دریافت شده برابر بود پیام درست مخابره شده است در غیر اینصورت بایستی پیام دوباره مخابره گردد. برای محاسبه چک سام عمودی اولین بایت پیام را از عدد *FF h* کم نموده و نتیجه محاسبه می گردد. از نتیجه بدست آمده بایت دوم کم شده و نتیجه محاسبه می گردد و عملیات مجدداً روی نتیجه تکرار می گردد و این عمل تا آخرین بایت پیام قبل از چک سام عمودی ادامه دارد. نتیجه بدست آمده در محل چک سام عمودی قرار می گیرد.

۶ـ۴ـ نتیجه گیری

در این فصل کلیاتی راجع به برنامه نویسی، نحوه ساخت یک فایل اجرایی با اسمبلی ۸۰۵۱، کامپایلر *C51* مورد بررسی قرار گرفت و در انتها فرمت ارسال دریافت اطلاعات بین سخت افزار و کامپیوتر در حالت شبیه سازی پورتها مورد استفاده قرار خواهد گرفت ارائه گردید. همانطورکه مشاهده می شود این فصل ناقص بوده لازم است در آینده تکمیل گردد.

فصل پنجم

نرم افزار پروژه

۱ـ۵ـ مقدمه

در این فصل نرم افزاری پروژه مورد نقد و بررسی قرار می گیرد. از آنجایی اکنون این نرم افزارها مراحل طراحی و برنامه ریزی خود را می گذرانند نمی توان آنها را با جزئیات کامل بیان نمود لذا سعی می گردد اجزاء لازم بصورت فلوچارت بیان شده و اصول چارچوب مشخص و سپس نسبت به تکمیل آنها اقدامات لازم صورت گیرد.

۲ـ۵ـ نرم افزار پروژه

بحث نرم افزار پروژه از سخت افزار آن وسیعتر بوده و موضوعات زیادی را به خود اختصاص می دهد. در این بحث بایستی نرم افزار روی کامپیوتر، نرم افزار روی سخت افزار و نرم افزارهای آموزشی به زبان اسمبلی و C مورد بحث و بررسی قرار گیرد.

۳ـ۵ـ نرم افزاری روی کامپیوتر

به جهت کارآیی بیشتر سیستم در آموزش افراد مبتدی و با تجربه، نرم افزار روی کامپیوتر به دو صورت استفاده از سیستم عامل *DOS* و استفاده از سیستم عامل *WINDOWS* تقسیم می گردد. در سیستم عامل *DOS* دقیقاً می توان مراحل انجام کار را نمایش داده و بدین وسیله آنچه را که اتفاق می افتد به افراد آموزش داد. برعکس هنگام استفاده از سیستم عامل ویندوز خیلی از اعمال از دید فرد مخفی بوده و بعد آموزشی کمتری دارد اما هنگامی که فرد تجربه لازم را کسب و از روند کار مطلع شد می توان از برنامه تحت سیستم عامل *WINDOWS* که با ویژوال بیسیک نوشته شده است استفاده کرد.

۴ـ۵ـ وظایف برنامه روی کامپیوتر در محیط *DOS*

در نوشتن و بکارگیری این برنامه سعی شده است برنامه به ساده ترین فرم ممکن نوشته شود بنحوی که کاربر دقیقاً از آنچه که اتفاق می افتد مطلع و احساس بهتری داشته باشد. بکارگیری این برنامه برای افراد مبتدی بسیار مفید بوده و موقعی که پیشرفت لازم حاصل شد حتی می توانند برنامه ای مشابه، مطابق با سلیقه خود با اطلاعاتی که در اختیار دارند بنویسند. همچنین می توانند از برنامه تکمیلی که تحت سیستم عامل *WINDOWS* نوشته شده است استفاده کنند. وظایف این برنامه بصورت زیر است:

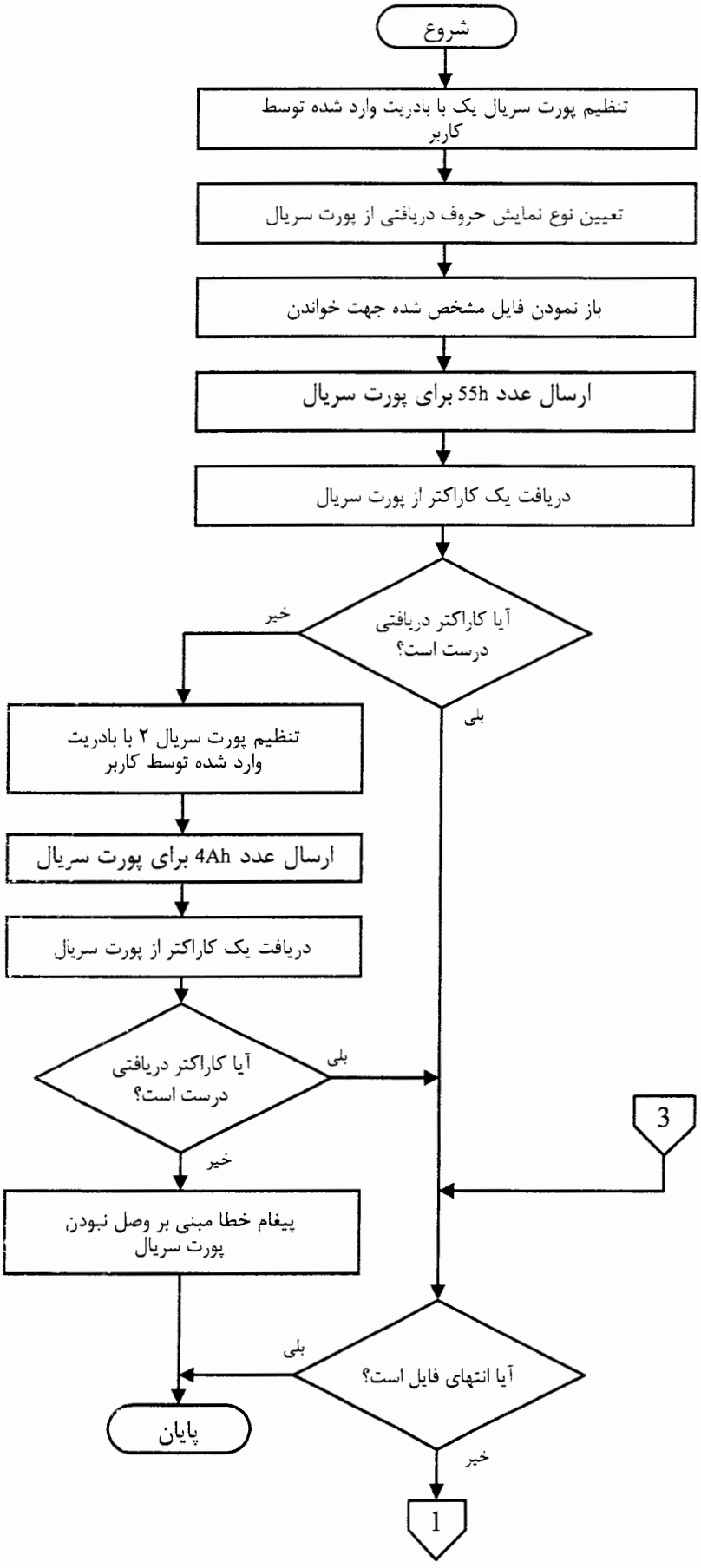
۱ـ تشخیص پورت سریال اتصالی بین کامپیوتر و سخت افزار بصورت اتوماتیک: در این حالت اگر کانکتور پورت سریال متصل نشده باشد و یا ارتباط بدلایلی قطع باشد پیغام خطا نمایش داده می شود.

۲ـ باز نمودن فایل با پسوند *HEX* و ارسال آن از طریق پورت سریال: برای انجام این امر شناخت فایل با پسوند *HEX* ، چگونگی باز نمودن فایل، خواندن اطلاعات از فایل، تبدیل اسکی به *HEX*، چگونگی ارسال و دریافت اطلاعات از پورت سریال لازم و ضروری است. در گزارش سعی شده است این موارد تا حدودی توضیح داده شوند.

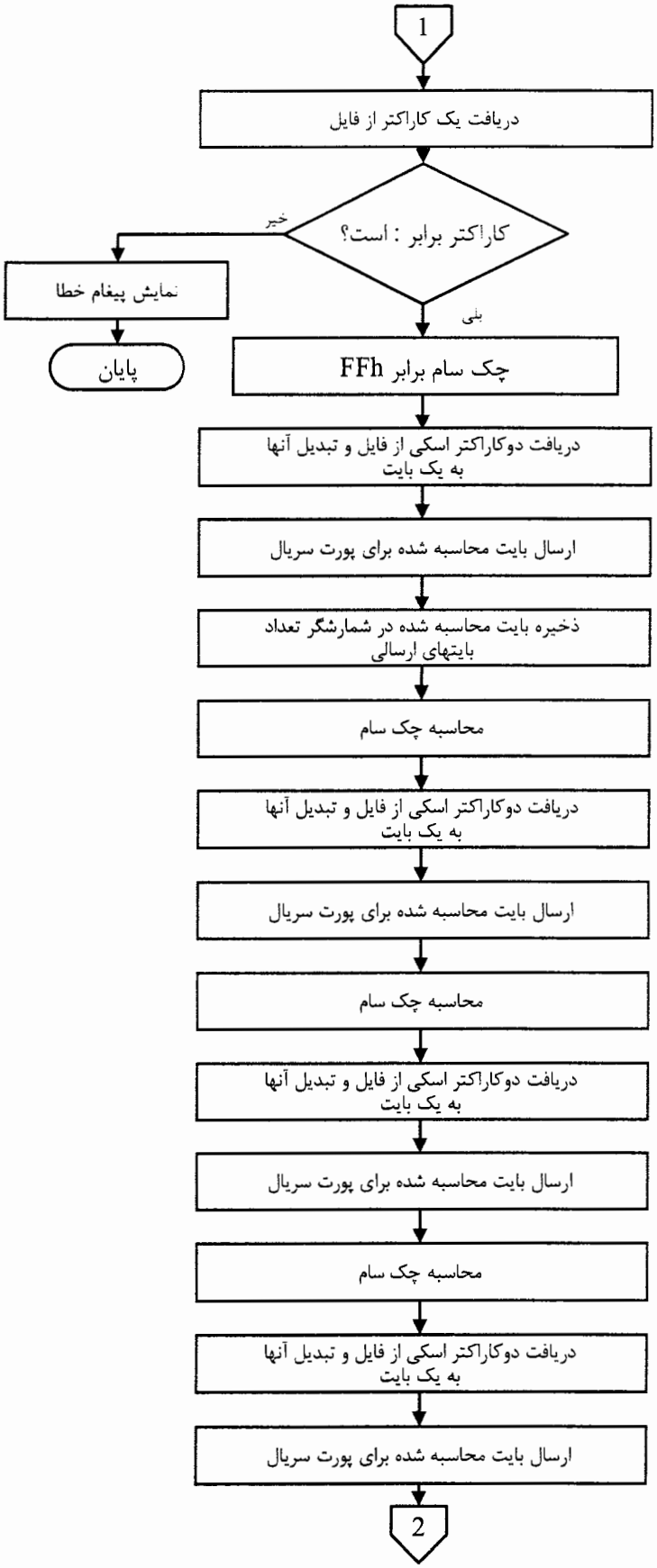
۳ـ بررسی صحت ارسال و دریافت فایل: جهت اطمینان از قرار گیری درست فایل روی حافظه این برنامه بر روند کار نظارت نموده و در صورت بروز هر مشکلی آنرا به اطلاع می رساند.

۴ـ دریافت اطلاعات رسیده از پورت سریال و نمایش آنها به دو صورت اسکی و *HEX* : همانطور بعداً اشاره خواهد شد یکی از روشهای بسیار مؤثر و کارا در عیب یابی منطقی برنامه های نوشته شده استفاده از پورت سریال و نمایش اطلاعات مورد نیاز در ساده ترین حالت ممکن روی صفحه مونیتور است. با بکارگیری این روش در اغلب موارد احتیاجی به اسیلوسکوپ، لاجیک آنالایزر و برنامه های شبیه ساز نیست و بدین وسیله با پیگیری مکان خطا می توان به عیب موجود پی برد و به رفع آن اقدام نمود.

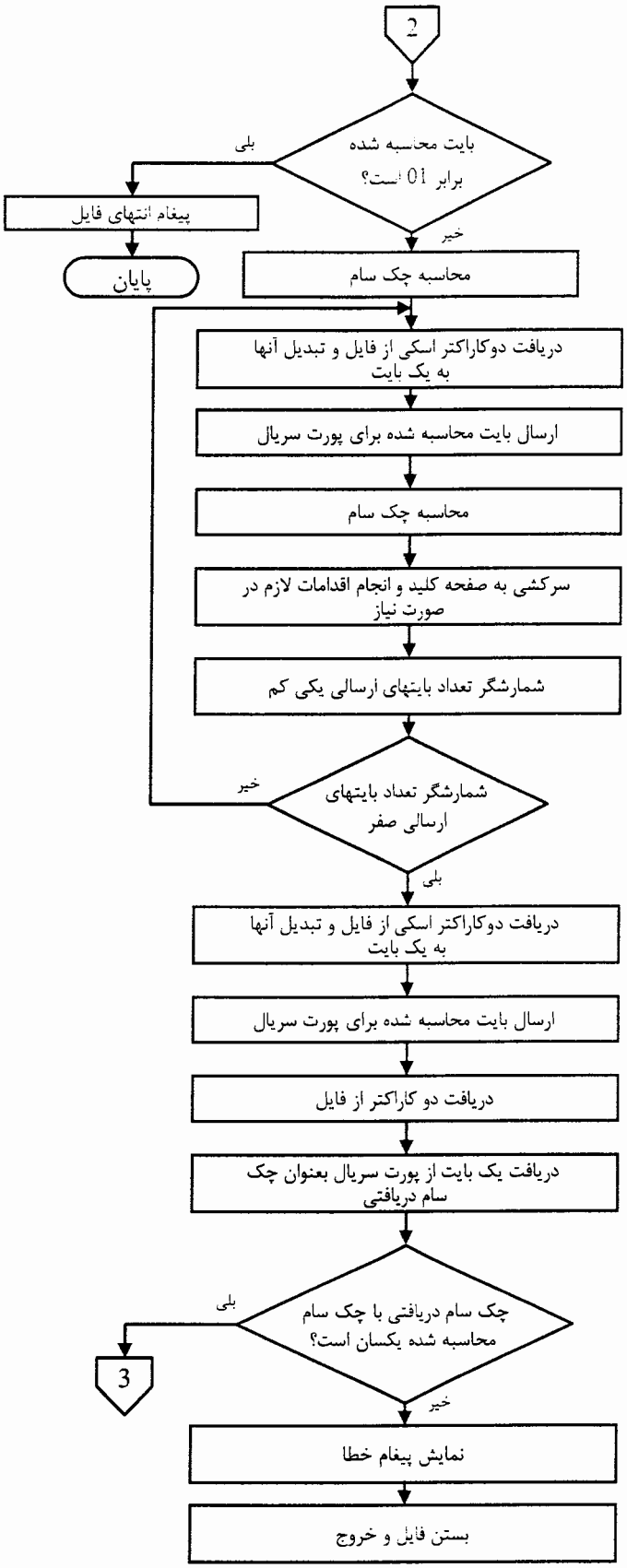
این برنامه قادر است اطلاعات دریافتی از پورت سریال را بصورت اسکی یا *HEX* که کاربر آنرا تعیین می کند روی صفحه مونیتور بنحوی که بسادگی قابل تشخیص باشد با نظم و قاعده مشخصی بنویسد و برنامه نویس با توجه به آنچه که انتظار دارد به بررسی موضوع بپردازد. بعنوان مثال برنامه ای نوشته شده و اجرا می گردد اما نتایج لازم حاصل نمی شود اشکال کجاست؟ اشکال سخت افزاری است یا نرم افزاری؟ اگر اشکال نرم افزاری است کجای برنامه ایراد دارد؟ برای پاسخ به این پرسشها به سراغ برنامه نوشته شده رفته و ترتیبی اتخاذ می گردد که در صورت اجرای هرقسمت گزارش لازم که ارسال یک حرف بخصوص می باشد برای پورت سریال داده شود. حال با اجرای این برنامه اطلاعات دریافتی از پورت سریال روی صفحه مونیتور نمایش داده شده و کاربر می تواند آنها را با آنچه که بایستی دریافت کند مقایسه کند و بدین وسیله به عیب یابی برنامه بپردازد. این روش تاکنون جواب بسیار خوبی داده و ابزار بسیار سود مندی می تواند باشد. فلوچارت این برنامه در شکل ۱ـ۵ نشان داده شده است.



شکل ۱ـ۵ ـ فلوچارت برنامه لازم روی کامپیوتر در محبط *DOS*



شکل ۱ـ۵ـ ادامه فلوچارت برنامه لازم روی کامپیوتر در محیط *DOS*



شکل ۱ـ۵ـ ادامه فلوچارت برنامه لازم روی کامپیوتر در محیط *DOS*

۵ـ۵ـ وظایف برنامه روی کامپیوتر در محیط *WINDOWS*

این برنامه که با نرم افزار ویژوال بیسیک نوشته و تکمیل می گردد وظایفی مشابه برنامه نوشته شده در محیط *DOS* دارد با این تفاوت که با ویژوال بیسیک نوشته شده و از محیط گرافیکی مناسبتر با قابلیت نمایش اطلاعات بیشتری برخوردار است وظایف این برنامه بصورت زیر است:

۱ـ باز نمودن فایل با پسوند *HEX* و ارسال آن از طریق پورت سریال و بررسی صحت ارسال و دریافت فایل

۲ـ دریافت اطلاعات رسیده از پورت سریال و نمایش آنها به دو صورت اسکی و *HEX*

۳ـ شبیه سازی پورتهای ورودی و خروجی

با توجه به ویژوال بودن محیط برنامه نویسی می توان قابلیتهای خوبی به نرم افزار اضافه کرد البته نباید قابلیتها را آنقدر زیاد نمود که کار با نرم افزار مشکل و از بار آموزشی آن بکاهد.

۶ـ۵ـ نرم افزار روی سخت افزار

همانطور که قبلاً اشاره شد مبنا بر سادگی و در عین حال آموزش بیشتر گذاشته شده است لذا سعی شده است تا جایی که ممکن است علاوه برسخت افزار نرم افزار نیز ساده باشد و از اضافه نمودن برنامه های کم کاربرد اجتناب شود و ترتیبی داده شود که کاربر خود زیر برنامه های اسمبلی یا C لازم را از مجموعه برنامه های ارائه شده انتخاب و برنامه مورد نظر را ساخته و نتیجه لازم را استخراج نماید. بعنوان مثال زیر برنامه ای که وظیفه آن ارسال و دریافت حروف و کاراکترها برای پورت سریال باشد در متن برنامه که در اختیار کاربر باشد پیش بینی نشده است در عوض این زیر برنامه و زیر برنامه های مشابه که کاربرد ببشتری دارند در گزارش و به همراه برنامه اصلی آورده می شود. لذا آنچه که روی حافظه سخت افزار بصورت ماندگار ذخیره خواهد شد برنامه ساده ای می باشد که وظایف آن به قرار زیر است:

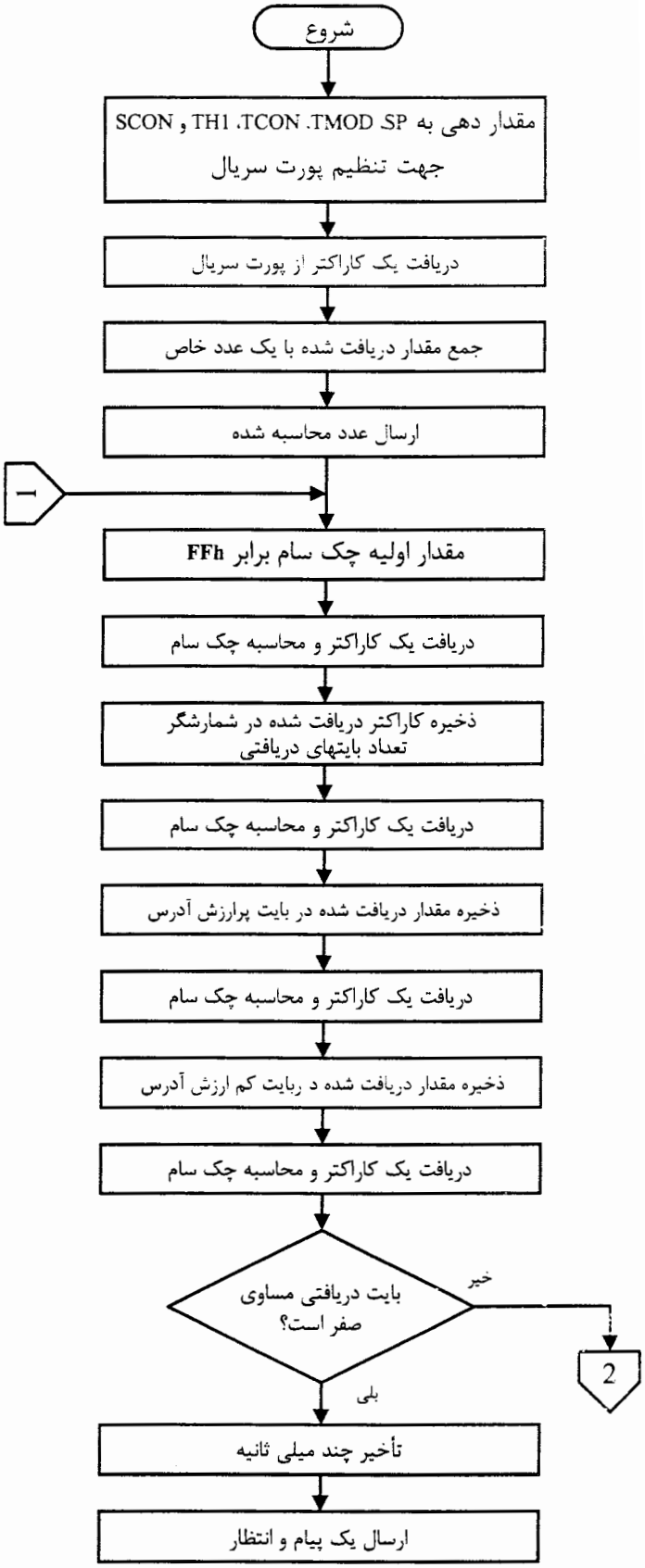
۱ـ برقراری ارتباط با پورت سریال کامپیوتر با دریافت و ارسال رمز مورد نظر: توسط این قسمت برنامه می توان اتصال پورت سریال و شماره پورت و تنظیم بادریت را بررسی نمود.

۲ـ دریافت اطلاعات دریافتی با فرمت و نظم مشخص و قرار دهی اطلاعات در آدرس تعیین شده با توجه به فرمت *HEX* فایل دریافت شده.

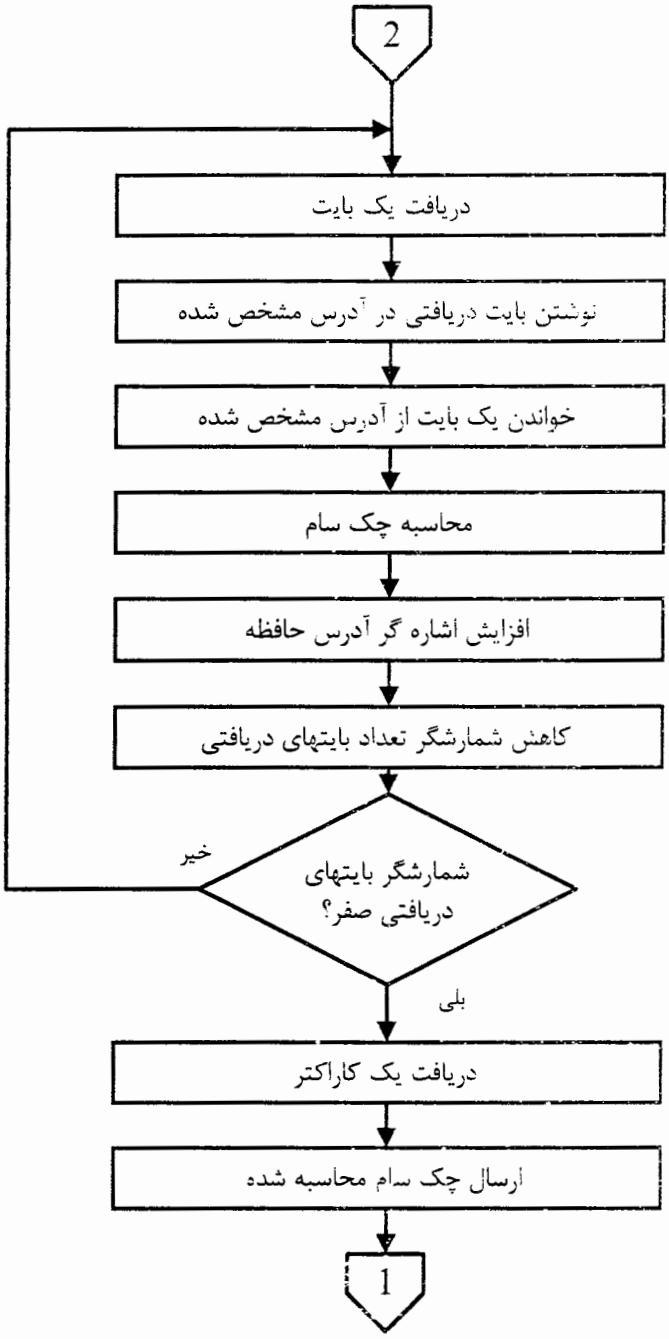
۳ـ محاسبه چک سام یک خط اطلاعات دریافتی و ارسال آن برای کامپیوتر

۴ـ دریافت فایل بطور کامل و ارسال پاسخ مناسب

فلوچارت این برنامه در شکل ۲ـ۵ نشان داده شده است.



شکل ۲ـ۵ ـ فلوچارت برنامه لازم روی سخت افزار



شکل ۲ـ۵ـ ادامه فلوچارت برنامه لازم روی سخت افزار

۷ـ۵ـ الگوریتم استفاده از برد سخت افزاری

۱ـ کابل رابط *RS232* به کامپیوتر و برد سخت افزاری وصل گردد.

۲ـ کامپیوتر را روشن نموده و تغذیه برد سخت افزاری وصل گردد.

۳ـ برنامه مورد نظر نوشته شده یا اصلاح و غلط گیری گردد و از آن فایلی با فرمت *HEX* ساخته شود.

۳ـ وضعیت برد سخت افزاری در حالت *Load* قرار گیرد.

۴ـ فایل مورد نظر در محیط *DOS* یا *Windows* مطابق دستورات لازم اجرا گردد.

۵ـ در صورت انتقال صحیح فایل به برد سخت افزاری وضعیت برد سخت افزاری در حالت *Execute* قرار گیرد.

۶ـ در صورت عدم دریافت جواب مناسب به مرحله ۳ برگشته و روند فوق تکرار گردد.

۸ـ۵ـ نتیجه گیری

در این فصل قسمتهای مختلف نرم افزار مورد بررسی قرار گرفت. این نرم افزارها بهمراه سخت افزار ساخته شده، شرایط لازم جهت آموزش برنامه نویسی با میکروکنترلر ۸۰۵۱ را به زبان اسمبلی و C فراهم می کند که در فصول آینده این موضوعات تشریح خواهد شد.

فصل ششم

برنامه نویسی با اسمبلی و دستورالعملهای ۸۰۵۱

۱ـ۶ـ مقدمه

مجموعه دستورالعملهای ۸۰۵۱ برای کاربردهای کنترلی هشت بیتی بهینه شده اند. این دستورالعملها که ۲۵۵ عدد می باشند روشهایی را جهت دسترسی به حافظه *RAM* داخلی و خارجی و انجام عملیات منطقی و ریاضی و عملیات بیتی و کنترلی ارائه می دهند.

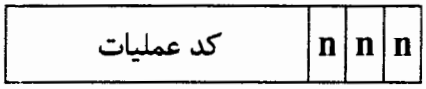
۲ـ۶ـ روشهای آدرس دهی

آنچه را که در یک میکروپروسسور مشخص می کند که چگونه می توان به حافظه ها، رجسترها و به قسمتهای مختلف برنامه مراجعه کرد، روشهای آدرس دهی گفته می شود. برای میکروکنترلر ۸۰۵۱ هشت روش آدرس دهی استفاده شده است.

۱ـ۲ـ۶ـ آدرس دهی ثبات[1]

رجسترهای کاری در ۸۰۵۱ از *R0* الی *R7* شماره گذاری شده اند. کد دستورالعمل این نوع آدرس دهی در شکل ۱ـ۶ نشان داده شده است. سه بیت کم ارزش شماره رجستر و ۵ بیت پر ارزش نوع عملیات را مشخص می کند. مانند دستورالعملهای زیر که انباره با رجستر *R7* جمع و یا انباره با رجستر *R6* عمل منطقی *AND* می گردد.

```
ADD  A,R7
ANL  A,R6
```



شکل ۱ـ۶ـ کد دستورالعمل آدرس دهی ثبات

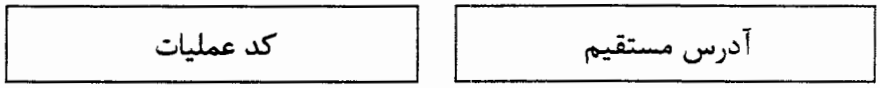
[1] - *Register Addressing*

۲ـ۲ـ۶ـ آدرس دهی مستقیم[1]

برای دسترسی به ۱۲۸ بایت پایین حافظه *RAM* داخلی و ۱۲۸ بایت فضای رجسترهای خاص از این نوع آدرس دهی استفاده می گردد. مطابق شکل ۲ـ۶ کد دستورالعملهای این نوع آدرس دهی دو بایتی بوده که بایت اول، نوع عملیات و بایت دوم مکان حافظه را مشخص می نماید.

```
ADD A,45h
MOV P1,A
```

در این دو دستورلعمل محتوی رجستر *A* با محتوی مکان *45h* جمع شده و نتیجه در *A* ذخیره می گردد و در دستورلعمل بعدی محتویات *A* را روی پورت *P1* قرار می گیرد.

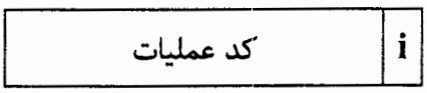


شکل ۲ـ۶ـ کد دستورالعمل در آدرس دهی مستقیم

۳ـ۲ـ۶ـ آدرس دهی غیر مستقیم[2]

برای دسترسی به محتویات مکانهای متوالی حافظه *RAM* و یا مکانهایی که آدرس آنها بایستی هنگام اجرای برنامه محاسبه گردد از آدرس دهی غیر مستقیم استفاده می گردد. در این روش تنها از رجسترهای *R0* و *R1* بعنوان اشاره گر می توان استفاده کرد. مطابق شکل ۳ـ۶ کد این دستورالعملها یک بایتی بوده که بیت اول آن *R0* یا *R1* را مشخص می کند. بعنوان مثال در دستورلعمل زیر اگر *R0* شامل *50h* و در مکان *50h* حافظه *RAM* عدد *54h* قرار گرفته باشد. پس از اجرای دستورالعمل فوق عدد *54h* به انباره منتقل می گردد.

```
MOV A,@R0
```



شکل ۳ـ۶ـ کد دستورالعمل در آدرس دهی غیر مستقیم

۴ـ۲ـ۶ـ آدرس دهی فوری[3]

اگر عملوند مبدأ به جای یک متغیر یک عدد ثابت باشد این مقدار ثابت مانند شکل ۴ـ۶ می تواند به عنوان یک بایت داده فوری به دستورالعمل بپیوندد. در زبان اسمبلی نشانه "#" جلو عملوندهای فوری می آید. عملوند ممکن است یک عدد ثابت، یک عبارت ریاضی مشتمل بر ثابت ها، نمادها و عملگرها باشد اسمبلر مقدار آنرا محاسبه کرده و داده فوری حاصل را در دستورالعمل جایگزین می نماید. برای مثال دستورالعملهای:

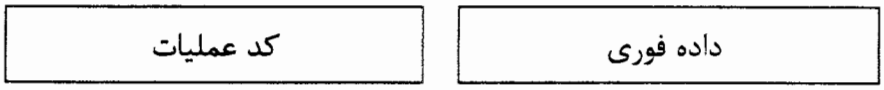
```
MOV A,#18
MOV A,#18H
MOV A,#NUM1+NUM2
```

---

[1] - *Direct Addressing*

[2] - *Indirect Addressing*

[3] - *Immediate Addressing*



شکل ۴ـ۶ـ کد دستورالعمل آدرس دهی فوری

بجـز در دسـتورالعمل زیـر بقیه دستورالعملها مانند شکل ۴ـ۶ دو بایتی بوده که بایت اول کد عملیات و بایت دوم عدد فوری را مشخص می کند.

MOV DPTR,#0FF00h

این دستورالعمل سه بایتی بوده که دو بایت آن عدد فوری را مشخص می سازد.

بـا توضـیحات بـالا دسـتورالعملهای زیـر با یکدیگر اختلاف دارند زیرا اولی آدرس دهی مستقیم و دومی آدرس دهی فوری می باشد.

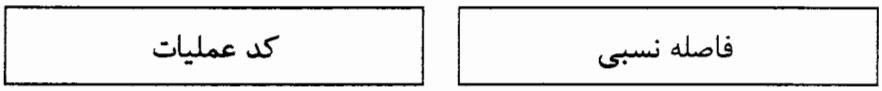
MOV A,25h
MOV A,#25h

۵ـ۲ـ۶ـ آدرس دهی نسبی[1]

ایـن نـوع آدرس دهـی در بعضی از دستورالعملهای پرش مورد استفاده قرار می گیرد. یک آدرس نسبی یک عدد هشت بیتی علامتدار است که با شمارنده برنامه جمع و آدرس دستورالعمل بعدی را که باید اجرا شـود مـی سـازد. محدوده این عدد علامتدار هشت بیتی بین ۱۲۸- و ۱۲۷ قرار می گیرد. این عدد هشت بیتی مانند شکل ۵ـ۶ به عنوان یک بایت اضافی به دستورالعمل می پیوندد.

SJMP THERE

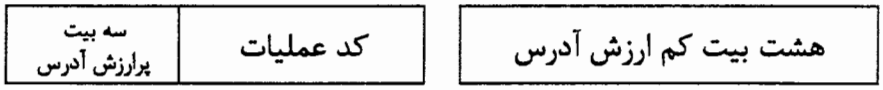
هـنگام استفاده از این دستورالعمل از آنجایی که از برچسبها استفاده می گردد اسمبلر فاصله محل پرش را محاسـبه مـی کند. اگر این فاصله در محدوده مورد نظر قرار نگرفته باشد ایراد خواهد گرفت که بایستی اصلاح گردد.



شکل ۵ـ۶ـ کد دستورالعمل در آدرس دهی نسبی

۶ـ۲ـ۶ـ آدرس دهی مطلق[2]

ایـن نـوع آدرس دهـی تـنها توسط دستورالعمل *ACALL* و *AJMP* استفاده می شود و فرق آن با آدرس دهـی نسبی در این است که بجای ۱۲۷ یا ۱۲۸- بایت پرش می توان ۲ کیلوبایت پرش داشت. مزیت آن نسـبت بـه آدرس دهـی طولانی در این است که مطابق شکل **۶ـ۶ کد** دستورالعمل آن بجای سه بایت دو بایت می باشد در نتیجه اجرای برنامه سریعتر می گردد.



شکل **۶ـ۶ـ** کد دستورالعمل در آدرس دهی مطلق

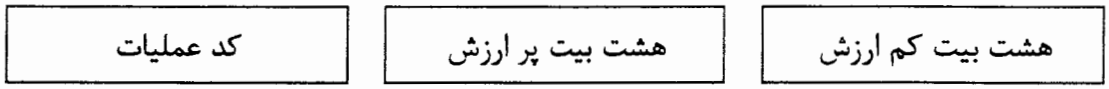
---

[1] - *Relative Addressing*

[2] - *Absolute Addressing*

۷ـ۲ـ۶ـ آدرس دهی طولانی[1]

دستورالعملهای *LJMP* و *LCALL* از این نوع آدرس دهی استفاده می کنند. این دستورالعملها مطابق شکل ۷ـ۶ از سه بایت که دو بایت آنرا آدرس و یک بایت آنرا کد عملیات مشخص می کند تشکیل شده اند. لذا می توان به کل ۶۴ کیلوبایت حافظه برنامه دسترسی داشت و زیر برنامه های مختلف را فراخوانی و اجرا کرد.

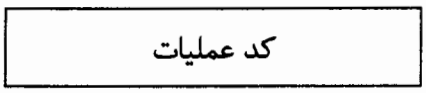


شکل ۷ـ۶ـ کد دستورالعمل در آدرس دهی طولانی

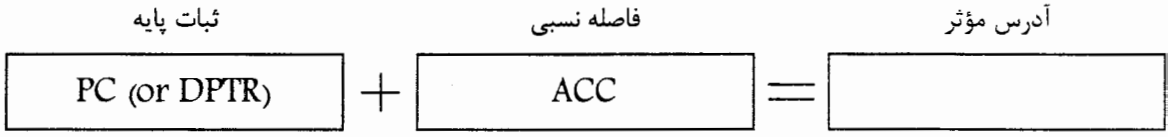
۸ـ۲ـ۶ـ آدرس دهی اندیس دار[2]

کد دستورالعمل در این نوع آدرس دهی مطابق شکل ۸ـ۶ یک بایتی می باشد. در این نوع آدرس دهی مانند شکل ۹ـ۶ برای ساخت آدرس مؤثر از یک ثبات پایه (شمارنده برنامه و یا اشاره گر داده) و یک فاصله نسبی (انباره) استفاده می شود. جدولهای پرش و یا جدولهای جستجو را براحتی با استفاده از آدرس دهی اندیس دار می توان ایجاد کرد.

JMP @A+DPTR
MOVC A,@A+DPTR
MOVC A,@A+PC



شکل ۸ـ۶ـ کد دستورلعمل در آدرس دهی اندیس دار



شکل ۹ـ۶ـ نحوه ساخت آدرس در آدرس دهی اندیس دار

۳ـ۶ـ انواع دستورالعملها

۸۰۵۱ دارای ۱۱۱ دستورالعمل می باشد که از ۴۹ دستورالعمل تک بایتی، ۴۵ دستورالعمل ۲ بایتی و ۱۷ دستورالعمل ۳ بایتی تشکیل یافته است. با شمردن متغیرهای موجود در هر دستورالعمل، مشاهده می شود که تعداد ۲۵۵ دستورالعمل مجزا وجود دارد، کد هر دستورالعمل در مبنای ۱۶ عددی بین *00h* الی *FFh* بوده و تنها کد *A5h* که برای نسخه های بعدی رزرو شده است استفاده نشده است. دستورالعملها به ۵ دسته حسابی، منطقی، انتقال داده، بولی، انشعاب دار تقسیم می شوند. که خلاصه ای از هریک به همراه جدول بیان کننده دستورالعملها آورده شده است.

در دستورالعملهای زیر نکات زیر بایستی مورد توجه قرار گیرد:

Rn ـ به رجسترهای *R0* الی *R7* بانک انتخاب شده اشاره می کند.

---

[1] - Long Addressing

[2] - Indexed Addressing

direct ـ آدرس ۸ بیتـی مکانهـای داده حافظـه داخلی است. این آدرس می تواند مکانی از حافظه *RAM* داخلی (۰ تا ۱۲۸) یا یکی از رجسترهای ناحیه *SFR* باشد.

Ri ـ به رجسترهای *R0* و *R1* بانک انتخاب شده اشاره می کند.

@Ri ـ مکانهـای ۸ بیتی از حافظه *RAM* داخلی (۰ تا ۲۵۵) که بصورت غیر مستقیم توسط رجسترهای *R0* و *R1* آدرس دهی می شوند.

#data ـ عدد ۸ بیتی ثابت که دستورالعملهای خاصی شامل آن می شود.

#data16 ـ عدد ۱۶ بیتی ثابت که دستورالعملهای خاصی شامل آن می شود.

addr16 ـ آدرس ۱۶ بیتـی مقصد. توسط دستورالعملهای *LCALL* و *LJMP* مورد استفاده قرار می گیرد. امکان پرش به هر مکانی از فضای آدرس پذیر حافظه برنامه ۶۴ کیلوبایتی را فراهم می کند.

addr11 ـ آدرس ۱۱ بیتی مقصد. توسط دستورالعملهای *ACALL* و *AJMP* مورد استفاده قرار می گیرد. امکـان پـرش بـه در محدوده ۲ کیلوبایتی بعد از اولین بایت دستورالعمل بعدی را در حافظه برنامه فراهم می کند.

rel ـ بایـت ۸ بیتـی علامـتدار نسبی. در دستورالعمل *SJMP* و تمام دستورالعملهای شرطی استفاده می شود. محدوده آن از ۱۲۷ تا ۱۲۸- بایت نسبت به اولین بایت دستورالعمل بعدی می باشد.

bit ـ بیتـی از حافظـه *RAM* داخلـی یا رجسترهای کاربرد خاص که بصورت مستقیم و بیتی قابل آدرس دهی می باشند.

۱ـ۳ـ۶ـ دستورالعملهای حسابی

در ایـن دسـتورالعملها کـه در جدول ۱ـ۶ آورده شده اند عملیات جمع، تفریق، افزایش، کاهش، ضرب و تقسـیم توسـط آنهـا انجـام می گیرد. تمامی دستورالعملهای حسابی در یک سیکل ماشین اجرا می شوند بجـز دسـتورالعملهای *INC DPTR* کـه در دو سـیکل ماشـین و *MUL AB* و *DIV AB* کـه در چهار سیکل ماشین انجام می گیرند.

دسـتورالعمل *MUL AB* ، محتویات رجستر *B* را با محتویات انباره بعنوان دو رقم بدون علامت ضرب می کـند و یـک نتـیجه ۱۶ بیتـی حاصـل می شود که ۸ بیت کم ارزش آن در انباره و ۸ بیت پرارزش آن در رجستر *B* قرار می گیرد.

دسـتورالعمل *DIV AB*، محتویات انباره را بر محتویات رجستر *B* بعنوان دو رقم بدون علامت تقسیم می کند و خارج قسمت حاصل از تقسیم در انباره و باقیمانده در رجستر *B* قرار می گیرد.

جدول ۱ـ۶ـ دستورالعملهای عملیات محاسباتی

••• •••••	توضیح	تعداد بایت	تعداد پالس
ADD A,Rn	محتویات رجستر *Rn* را با انباره جمع و نتیجه را در انباره می گذارد	1	12
ADD A,direct	بایتی با آدرس مستقیم را با انباره جمع و نتیجه را در انباره می گذارد	2	12
ADD A,@Ri	انباره را با جایی که *R0* یا *R1* اشاره می کند جمع و نتیجه را در انباره می گذارد	1	12
ADD A,#data	انباره را با عدد مشخصی (داده بی واسطه) جمع و نتیجه را در انباره می گذارد	2	12
ADDC A,Rn	رجستر *Rn* را با انباره و رقم نقلی جمع و نتیجه را در انباره می گذارد	1	12
ADDC A,direct	بایتی با آدرس مستقیم را با انباره و رقم نقلی جمع و نتیجه را در انباره می گذارد	2	12
ADDC A,@Ri	انباره را با رقم نقلی و جایی که *R0* یا *R1* اشاره می کند جمع و نتیجه را در انباره می گذارد	1	12
ADDC A,#data	انباره را با عدد مشخصی (داده بی واسطه) و رقم نقلی جمع و نتیجه را در انباره می گذارد	2	12
SUBB A,Rn	مجموع *Rn* و رقم قرضی را از انباره کم و نتیجه را در انباره می گذارد	1	12
SUBB A,direct	مجموع یک مکان مشخص از حافظه و رقم قرضی را از انباره کم و نتیجه را در انباره می گذارد	2	12
SUBB A,@Ri	مجموع رقم قرضی و جایی را که *Ri* اشاره می کند از انباره کم و نتیجه را در انباره می گذارد	1	12
SUBB A,#data	مجموع رقم قرضی و یک عدد مشخص را از انباره کم و نتیجه را در انباره می گذارد.	2	12
INC A	به انباره یک واحد اضافه می کند	1	12
INC Rn	به رجستر *Rn* یک واحد اضافه می کند	1	12
INC direct	به محتویات مکانی از حافظه که در دستور مشخص شده است یک واحد اضافه می کند	2	12
INC @Ri	محتویات مکانی از حافظه را که رجستر *Ri* به آن اشاره می کند یک واحد اضافه می کند	1	12
DEC A	از انباره یک واحد کم می کند	1	12
DEC Rn	از رجستر Rii یک واحد کم می کند	1	12
DEC direct	از محتویات مکانی از حافظه که در دستور مشخص شده است یک واحد کم می کند	2	12
DEC @Ri	محتویات مکانی از حافظه را که رجستر *Ri* به آن اشاره می کند یک واحد کم می کند	1	12
INC DPTR	به رجستر ۱۶ بیتی *DPTR* یک واحد اضافه می کند	1	24
MUL AB	محتویات رجستر *A* را در محتویات رجستر *B* ضرب و نتیجه پر ارزش را در *A* و کم ارزش را در *B* ذخیره می کند	1	48
DIV AB	محتویات رجستر *A* را بر محتویات رجستر *B* تقسیم و حاصل را در *A* و باقیمانده را در *B* ذخیره می کند	1	48
DA A	اگر دو عدد *BCD* با یکدیگر جمع گردند نتیجه غیر *BCD* است این دستورالعمل نتیجه را بصورت *BCD* بر می گرداند	1	12

۲ـ۳ـ۶ـ دستورالعملهای منطقی

در ایـن دستورالعملها که در جدول ۲ـ۶ آورده شده اند عملیات منطقی همچون *AND* ، *OR*، *XOR*، پاک کردن و متمم کردن، چرخش براست و بچپ روی بایتها توسط آنها انجام می گیرد.

دستورالعمل *SWAP A* محتویات چهار بیت پایین و چهار بیت بالای انباره را با یکدیگر عوض می کند.

جدول ۲ـ۶ـ دستورالعملهای عملیات منطقی

نماد	توضیح	تعداد بایت	تعداد پالس
ANL A,Rn	محتویات رجستر *A* را با رجستر *Rn* ، *AND* نموده و نتیجه را در *A* ذخیره می کند	1	12
ANL A,direct	محتویات رجستر *A* را با مکانی از حافظه *RAM* ، *AND* نموده و نتیجه را در *A* ذخیره می کند	2	12
ANL A,@Ri	محتویات رجستر *A* را با مکانی از حافظه *RAM* که *Ri* به آن اشاره می کند ، *AND* نموده و نتیجه را در *A* ذخیره می کند	1	12
ANL A,#data	محتویات رجستر *A* را با عددی مشخص ، *AND* نموده و نتیجه را در *A* ذخیره می کند	2	12
ANL direct,A	محتویات رجستر *A* را با مکانی از حافظه *RAM* ، *AND* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	2	12
ANL direct,#data	محتویات مکانی از حافظه *RAM* را با عددی مشخص ، *AND* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	3	24
ORL A,Rn	محتویات رجستر *A* را با رجستر *Rn* ، *OR* نموده و نتیجه را در *A* ذخیره می کند	1	12
ORL A,direct	محتویات رجستر *A* را با مکانی از حافظه *RAM* ، *OR* نموده و نتیجه را در *A* ذخیره می کند	2	12
ORL A,@Ri	محتویات رجستر *A* را با مکانی از حافظه *RAM* که *Ri* به آن اشاره می کند ، *OR* نموده و نتیجه را در *A* ذخیره می کند	1	12
ORL A,#data	محتویات رجستر *A* را با عددی مشخص ، *OR* نموده و نتیجه را در *A* ذخیره می کند	2	12
ORL direct,A	محتویات رجستر *A* را با مکانی از حافظه *RAM* ، *OR* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	2	12
ORL direct,#data	محتویات مکانی از حافظه *RAM* را با عددی مشخص ، *OR* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	3	24
XRL A,Rn	محتویات رجستر *A* را با رجستر *Rn* ، *XOR* نموده و نتیجه را در *A* ذخیره می کند	1	12

XRL A,direct	محتویات رجستر *A* را با مکانی از حافظه*RAM* ، *XOR* نموده و نتیجه را در *A* ذخیره می کند	2	12
XRL A,@Ri	محتویات رجستر *A* را با مکانی از حافظه *RAM* که *Ri* به آن اشاره می کند ، *XOR* نموده و نتیجه را در *A* ذخیره می کند	1	12
XRL A,#data	محتویات رجستر *A* را با عددی مشخص ، *XOR* نموده و نتیجه را در *A* ذخیره می کند	2	12
XRL direct,A	محتویات رجستر *A* را با مکانی از حافظه*RAM* ، *XOR* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	2	12
XRL direct,#data	محتویات مکانی از حافظه *RAM* را با عددی مشخص ، *XOR* نموده و نتیجه را در آن مکان حافظه ذخیره می کند	3	24
CLR A	محتویات رجستر *A* صفر می گردد	1	12
CPL A	محتویات رجستر *A* مکمل می گردد	1	12
RL A	محتویات رجستر *A* یک بیت به سمت چپ می چرخند	1	12
RLC A	محتویات رجستر *A* از میان رقم نقلی یک بیت به سمت چپ می چرخند	1	12
RR A	محتویات رجستر *A* یک بیت به سمت راست می چرخند	1	12
RRC A	محتویات رجستر *A* از میان رقم نقلی یک بیت به سمت راست می چرخند	1	12
SWAP A	نیبل کم ارزش و پرارزش انباره با یکدیگر جابجا می گرند	1	12

۳ـ۳ـ۶ـ دستورالعملهای انتقال داده

مهمترین دستورالعمل انتقال داده عبارت *move* است که سه شکل کلی *MOV*، *MOVC* و *MOVX* دارد و فرمهای مختلف آنها در جدول ۳ـ۶ نشان داده شده است. همچنین دستورالعملهایی چون *PUSH* جهت قرار دادن داده در پشته، *POP* جهت خواندن از پشته و *XCH* (معاوضه) نیز از نوع انتقال داده هستند. در این نوع دستورالعمل ها از همه روشهای آدرس دهی استفاده می شود.

در دستورالعملهای *MOV* از حافظه داخلی *RAM* و ناحیه *SFR* استفاده می شود. دستورالعمل *MOVC*، برای انتقال بایتهایی از حافظه برنامه *ROM*، که شبیه یک جدول داده است، به انباره بکار می رود. دستورالعملهای *MOVX* نیز برای کار با حافظه *RAM* خارجی و نوشتن یا خواندن از آن استفاده می شود.

جدول ۳ـ۶ـ دستورالعملهای عملیات انتقال

نماد	توضیح	تعداد بایت	تعداد پالس
MOV A,Rn	محتویات رجستر *Rn* به انباره انتقال می یابد	1	12
MOV A,direct	محتویات مکانی از حافظه *RAM* که در دستور مشخص شده به انباره انتقال می یابد	2	12
MOV A,@Ri	محتویات مکانی از حافظه *RAM* که *Ri* به آن اشاره می کند به انباره انتقال می یابد	1	12
MOV A,#data	عدد مشخص شده به انباره انتقال می یابد	2	12
MOV Rn,A	محتویات انباره به *Rn* انتقال می یابد	1	12
MOV Rn,direct	محتویات مکانی از حافظه *RAM* به انباره انتقال می یابد	2	24
MOV Rn,#data	عدد مشخص شده به رجستر *Rn* انتقال می یابد	2	12
MOV direct,A	محتویات انباره به مکانی از حافظه *RAM* انتقال می یابد	2	12
MOV direct,Rn	رجستر *Rn* به مکانی از حافظه *RAM* انتقال می یابد	2	24
MOV direct,direct	محتویات مکانی از حافظه *RAM* به مکان دیگری از حافظه *RAM* انتقال می یابد	3	24
MOV direct,@Ri	محتویات مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند به مکانی از حافظه *RAM* انتقال می یابد	2	24
MOV direct,#data	عددی مشخص به مکانی از حافظه *RAM* انتقال می یابد	3	24
MOV @Ri,A	محتویات انباره به مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند انتقال می یابد	1	12
MOV @Ri,direct	محتویات مکانی از حافظه *RAM* را به مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند انتقال می یابد	2	24
MOV @Ri,#data	عددی مشخص به مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند انتقال می یابد	2	12
MOV DPTR,#data16	یک عدد ۱۶ بیتی به رجستر *DPTR* انتقال می یابد	3	24
MOVC A,@A+DPTR	مجموع انباره و رجستر *DPTR* یک آدرس ۱۶ بیتی را می سازند محتویات آن مکان حافظه برنامه به انباره انتقال می یابد	1	24
MOVC A,@A+PC	مجموع انباره و شمارنده برنامه *(PC)* یک آدرس ۱۶ بیتی را می سازند محتویات آن مکان حافظه برنامه به انباره انتقال می یابد	1	24
MOVX A,@Ri	محتویاتی از مکان *RAM* خارجی که رجستر *Ri* به آن اشاره می کند در انباره قرار می گیرد	1	24
MOVX A,@DPTR	محتویاتی از مکان *RAM* خارجی که رجستر *DPTR* به آن اشاره می کند در انباره قرار می گیرد	1	24

MOVX @Ri,A	محتویات انباره در مکانی از حافظه خارجی که رجستر *Ri* به آن اشاره می کند قرار می گیرد	1	24
MOVX @DPTR,A	محتویات انباره در مکانی از حافظه خارجی که رجستر *DPTR* به آن اشاره می کند قرار می گیرد	1	24
PUSH direct	محتویات مکانی از حافظه *RAM* در پشته ذخیره می گردد	2	24
POP direct	از پشته یک بایت خوانده شده و در مکانی از حافظه *RAM* ذخیره می گردد	2	24
XCH A,Rn	محتویات رجستر *Rn* را با انباره جابجا می کند	1	12
XCH A,direct	محتویات رجستر *Rn* را با مکانی از حافظه *RAM* جابجا می کند	2	12
XCH A,@Ri	محتویات رجستر *Rn* را با مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند جابجا می کند	1	12
XCHD A,@Ri	چهار بیت کم ارزش انباره با مکانی که رجستر *Ri* به آن اشاره می کند تعویض می گردد	1	12

۴ـ۳ـ۶ـ دستورالعملهای متغیر بولی

این دسته از دستورالعملها که در جدول ۴ـ۶ آورده شده اند به سخت افزار پردازنده بولی تک بیتی موجود در ۸۰۵۱ مربوط می شوند که شامل دستورهای یک کردن، صفر کردن، *AND* و *OR* کردن و متمم کردن بیتها هستند. همچنین دستورالعملهای انتقال بیتی و پرشهای شرطی که ارزش یک بیت را امتحان می کنند، جزء این دسته می باشند. در نتیجه این دسته از دستورالعملها کاربرد خوبی می توانند در کنترل داشته باشند.

جدول ۴ـ۶ـ دستورالعملهای عملیات بیتی

نماد	توضیح	تعداد بایت	تعداد پالس
CLR C	بیت نقلی صفر می گردد	1	12
CLR bit	بیت مشخص شده صفر می گردد	2	12
SETB C	بیت نقلی یک می گردد	1	12
SETB bit	بیت مشخص شده یک می گردد	2	12
CPL C	بیت نقلی متمم می گردد	1	12
CPL bit	بیت مشخص شده متمم می گردد	2	24
ANL C,bit	بیت مشخص شده با بیت نقلی *AND* شده و نتیجه در بیت نقلی قرار می گیرد	2	24
ANL C,/bit	بیت مشخص شده با متمم بیت نقلی *AND* شده و نتیجه در بیت نقلی قرار می گیرد	2	24

ORL C,bit	بیت مشخص شده با بیت نقلی *OR* شده و نتیجه در بیت نقلی قرار می گیرد	2	24
ORL C,/bit	بیت مشخص شده با متمم بیت نقل *OR* شده و نتیجه در بیت نقلی قرار می گیرد	2	24
MOV C,bit	وضعیت بیت مشخص شده به بیت نقلی انتقال می یابد	2	12
MOV bit,C	وضعیت بیت نقلی به بیت مشخص شده انتقال می یابد	2	24
JC rel	اگر بیت نقلی یک باشد به آدرس مشخص شده پرش می کند	2	24
JNC rel	اگر بیت نقلی صفر باشد به آدرس مشخص شده پرش می کند	2	24
JB bit,rel	اگر بیت مشخص شده یک باشد به آدرس مشخص شده پرش می کند	3	24
JNB bit,rel	اگر بیت مشخص شده صفر باشد به آدرس مشخص شده پرش می کند	3	24
JBC bit,rel	اگر بیت مشخص شده یک باشد به آدرس مشخص شده پرش می کند و سپس آن بیت صفر می گردد	3	24

۵ـ۳ـ۶ـ دستورالعملهای انشعاب برنامه

این دسته از دستورالعملها که در جدول ۵ـ۶ آورده شده اند شامل زیرروالهای فراخوانی و برگشت و انواع پرشهای شرطی و بدون شرط می باشند. پرشهای شرطی نسبت به اولین بایت دستورالعمل بعدی صورت می گیرند.

دستورالعملهای پرش به ۳ نوع اصلی پرش کوتاه *(SJMP)* ، پرش بلند *(LJMP)* و پرش مطلق *(AJMP)* تقسیم بندی می شوند. پرش کوتاه از آدرس دهی نسبی، استفاده می کند در این نوع پرش مقدار پرش با یک عدد ۸ بیتی متمم ۲ علامتدار مشخص می شود، لذا پرش در هر دو سمت جلو و عقب در محدوده ۱۲۸- تا ۱۲۷ بایت امکان پذیر خواهد بود. پرش بلند از یک آدرس ۱۶ بیتی بعنوان جزئی از دستورالعمل استفاده می کند لذا دستورالعمل سه بایتی بوده و می توان به هر مکانی از فضای حافظه برنامه (*ROM* یا *EPROM*) یعنی ۶۴ کیلوبایت دسترسی داشت.

پرش مطلق از یک آدرس ۱۱ بیتی، که از دو قسمت ۸ بیت پایین و ۳ بیت بالا تشکیل یافته است استفاده می کند. سه بیت بالایی آدرس با ۵ بیت مشخص کننده عملیات ترکیب شده و یک کد عملیاتی ۸ بیتی را تشکیل می دهند. بنابراین کل دستورالعمل از ۲ بایت تشکیل می شود. در طول اجرای دستورالعمل، ۱۱ بیت آدرس به ۱۱ بیت پایینی رجستر شمارنده (*PC*) منتقل می شود، یعنی مکانی از حافظه که آدرس ۱۱ بیتی به آن اشاره می کند، در محدوده ۲ کیلوبایتی بعد از دستورالعمل *AJMP* قرار خواهد داشت.

یکی از دستورالعملهای مهم و پرکاربرد این دسته *CJNE* می باشد که ترکیبی از دو دستورالعمل مقایسه و پرش می باشد. در این دستورالعمل محتویات رجستر اول با عدد دوم یا محتویات مکان هایی از حافظه

RAM بـا یکدیگـر مقایســه و در صورت نابرابری پرش به مکان مشخص صورت می گیرد در غیر اینصورت دسـتورالعمل بعـدی اجـرا می گردد این دستوالعمل روی فنگ *CY* تأثیر گذاشته و از بررسی آن می توان تشـخیص داد کـه عـدد اول بزرگتر بوده است یا عدد دوم بعنوان مثال دستورالعمل زیر در نظر گرفته می شود.

CJNE A,#40h,LOC1

در این دستورالعمل محتویات انباره با عدد *40h* مقایسه می گردد اگر برابر بودند دستورالعمل بعدی اجرا خواهـد شـد ولـی اگر مساوی نبودند به مکانی که با *LOC1* مشخص شده است و ماکزیمم می تواند ۱۲۸ بایـت قـبل یا ۱۲۷ بایت بعد باشد پرش می کند پس از پرش در برنامه می توان بیت نقلی را بررسی و در صـورت صـفر بودن تشخیص داد *A* از *40h* بزرگتر و در صورت یک بودن *A* از *40h* کوچکتر بوده است. در دسـتورالعمل فـوق بجـای *A* مـی تـوان *Rn* و یا *@Ri* و بجای *#40h* عددی بین *00h* الی *FFh* قرار داد که مشخص کننده مکانی از حافظه *RAM* داخلی یا *SFR* می باشد.

دستورالعمل *NOP* کار بخصوصی انجام نداده و از آن می توان به عنوان تأخیر استفاده کرد.

جدول ۵ـ۶ـ دستورالعملهای انشعاب دار

نماد	توضیح	تعداد بایت	تعداد پالس
ACALL addr11	یک زیر برنامه بصورت مطلق فراخوانی می گردد	2	24
LCALL addr16	یک زیر برنامه بصورت طولانی فراخوانی می گردد	3	24
RET	برگشت از زیر برنامه	1	24
RETI	برگشت از زیر روال وقفه	1	24
AJMP addr11	پرش به آدرسهایی با جابجایی کمتر از ۲ کیلوبایت	2	24
LJMP addr16	پرش به آدرسهایی بدون محدودیت	3	24
SJMP rel	پرش به آدرسهایی با جابجایی ۱۲۷ تا ۱۲۸-	2	24
JMP @A+DPTR	پرش به جایی که مجموع انباره و *DPTR* اشاره می کند	1	24
JZ rel	در صورت صفر بودن انباره به آدرس نسبی پرش می کند	2	24
JNZ rel	در صورت صفر نبودن انباره به آدرس نسبی پرش می کند	2	24
CJNE A,direct,rel	محتویات انباره با مکانی از حافظه *RAM* مقایسه می گردد در صورت نابرابری به آدرس نسبی پرش می کند	3	24
CJNE A,#data,rel	محتویات انباره با عددی مشخص مقایسه می گردد در صورت نابرابری به آدرس نسبی پرش می کند	3	24
CJNE Rn,#data,rel	محتویات رجستر *Rn* با عددی مشخص مقایسه می گردد در صورت نابرابری به آدرس نسبی پرش می کند	3	24
CJNE @Ri,#data,rel	محتویات مکانی از حافظه *RAM* که رجستر *Ri* به آن اشاره می کند با عددی مشخص مقایسه می گردد در صورت نابرابری به آدرس نسبی پرش می کند	3	24

24	2	از رجستر *Rn* یک واحد کم می گردد در صورتی که صفر نشده باشد به آدرس نسبی پرش می کند	*DJNZ Rn,rel*
24	3	از مکان مشخصی از حافظه *RAM* یک واحد کم می گردد در صورتی که صفر نشده باشد به آدرس نسبی پرش می کند	*DJNZ direct,rel*
12	1	کاری انجام نمی دهد	*NOP*

۴ـ۶ـ دستورالعملهایی که روی فلگها تأثیر می گذارند

جدول ۶ـ۶ دستورالعملهایی را که رو فلگ کری، روی فلگ سرریز و فلگ کری کمکی تأثیر می گذارند را نشان می دهد. همانطور که مشاهده می شود تنها محاسبات حسابی و منطقی روی فلگهای سرریز و کری کمکی تأثیر می گذارند.

جدول ۶ـ۶ـ دستورالعملهایی تأثیر گذار روی فلگها

دستورالعمل	*CY*	*OV*	*AC*
ADD	*X*	*X*	*X*
ADDC	*X*	*X*	*X*
SUBB	*X*	*X*	*X*
MUL	*0*	*X*	
DIV	*0*	*X*	
DA	*X*		
RRC	*X*		
RLC	*X*		
SETBC	*1*		
CLR C	*0*		
CPL C	*X*		
ANL C,bit	*X*		
ANL C,/bit	*X*		
ORL C,bit	*X*		
ORL C,/BIT	*X*		
MOV C,bit	*X*		
CJNE	*X*		

۵ـ۶ـ دستورات اسمبلر[1]

این دستورات، دستورالعملهای زبان اسمبلی نبوده و توسط میکروپروسسور قابل اجرا نیستند بلکه فرمانهایی هستند که عملکرد اسمبلر را در بعضی از حالات تعیین می کنند. و بجز دستورهای *DB* و *DW* سایر دستورات تأثیری بر محتوای حافظه ندارند. بعضی از این دستورات در حالت عادی کاربردی ندارند و تنها با دستورات *DB*، *DW*، *END* و *EQU* می توان هربرنامه ای را نوشت. اما هنگامی که حافظه های برنامه و دیتای استفاده شده در سخت افزار پیوسته نبوده و مدیریت حافظه داده داخلی توسط کاربر مشکل بنظر می رسد و یا برنامه بخاطر حجم زیاد آن به چند فایل جداگانه تقسیم شده است از این

---

[1] *Assembler Directive*

دستورات استفاده می گردد. توسط این دستورات مشخص می گردد هر حافظه برای چه منظوری استفاده گردد.

۶ـ۵ـ۱ ORG

قالب دستور *ORG* (شروع) به صورت زیر است:

ORG expression

دستور *ORG* به شمارنده اعلام می کند که با آغاز برنامه جدید که در ادامه آن می آید، بار شود. از برچسب استفاده نمی شود. به دو مثال زیر توجه کنید:

```
ORG 100h
ORG ($+1000h) AND 0F000h ; Set To Next 4K Boundary
```

دستور *ORG* را می توان در هر سگمنتی بکار برد. اگر سگمنت جاری، مطلق باشد، مقدار آن یک آدرس مطلق در سگمنت جاری است. اگر سگمنت با جایگزینی مجدد باشد، مقدار عبارت *ORG* به عنوان افست آدرس پایه تلقی می شود.

۶ـ۵ـ۲ END

قالب دستور *END* بصورت زیر است:

END

END باید آخرین جمله فایل منبع باشد. هیچ برچسب و دستورالعملی بعد از آن مورد پردازش اسمبلر قرار نمی گیرد.

۶ـ۵ـ۳ EQU (مساوی)

قالب دستور *EQU* بصورت زیر است:

symbol EQU expression

دستور *EQU* یک مقدار عددی را به یک نام نماد مشخص، منتسب می کند. نماد، باید نمادی معتبر بوده و در عبارت باید قواعد نام گذاری رعایت شده باشد مثلاً با عدد شروع نگردد، کلمات کنترلی داخل آن استفاده نشده باشد. استفاده از نماد بجای اعداد خوانایی برنامه را افزایش می دهد.

مانند مثالهای زیر:

```
N27     EQU     27
HERE    EQU     $
CR      EQU     0Dh
```

۶ـ۵ـ۴ DS (تعریف حافظه)

قالب دستور *DS* (تعریف حافظه) به صورت زیر است:

[label:] DS expression

دستور *DS* فضا را به صورت واحد بایت ذخیره می کند. از آن می توان در هر نوع سگمنتی بجز *BIT* استفاده کرد. عبارت باید یک عبارت زمان اسمبل معتبر، بدون مرجع های مستقیم و مرجع های خارجی یا با جایگزینی مجدد باشد. هنگامی که از یک دستور *DS* در یک برنامه استفاده شود، شمارنده محل

سگمنت های جاری به اندازه مقدار عبارت افزایش می یابد. مجموع شمارنده محل و عبارت مشخص شده نباید از محدوده فضای آدرس جاری تجاوز نماید.

دستورهای زیر یک بافر ۴۰ بایتی در سگمنت داده داخلی تولید می کنند:

```
            DSEG AT 30h
LENGTH      EQU  40
BUFFER      DS   LENGTH
```

برچسب *BUFFER* نماینده آدرس اولین محل حافظه ذخیره شده است. در این مثال بافر از آدرس *30h* شروع می شود.

برای ایجاد یک بافر ۱۰۰۰ بایتی در *RAM* خارجی از آدرس *4000h*، می توان از دستورهای زیر استفاده کرد:

```
XSTART      EQU         4000h
XLENGTH     EQU         1000
            XSEG  AT    XSTART
XBUFFER     DS    XLENGTH
```

۶_۵_۵ DBIT

قالب دستور *DBIT* (تعریف بیت) به صورت زیر است:

```
[label:]        DBIT        expression
```

دستور *DBIT* فضا را به صورت واحد بیت ذخیره می کند، و از آن می توان تنها در مورد سگمنت *BIT* استفاده کرد. عبارت باید یک عبارت زمان اسمبل بدون مرجع های مستقیم باشد. هنگامی که در یک برنامه از دستور *DBIT* استفاده شود، شمارنده محل سگمنت جاری (*BIT*) به اندازه عبارت افزایش می یابد. دقت کنید که در سگمنت *BIT*، واحد پایه شمارنده محل به جای بایت بیت است. دستورات زیر، سه پرچم در سگمنت بیت مطلق ایجاد می کنند:

```
            BSEG
KBFLAG:     DBIT  1
PRFLAG:     DBIT  1
DKFLAG:     DBIT 1
```

۶_۵_۶ DB (تعریف بایت)

قالب دستور *DB* (تعریف بایت) به شکل زیر است:

```
[label:]        DB      expression      [,expression][...]
```

دستور *DB* حافظه کد را با مقادیر بایت، مقدار دهی اولیه می کند. از آنجا که این دستور ثوابت داده را در حافظه کد قرار می دهد، سگمنت *CODE* باید فعال باشد. فهرست عبارت، یک سری از یک، یا چند مقدار بایت است (که هر کدام ممکن است یک عبارت باشد) و توسط ویرگول از یکدیگر جداشده اند. از دستور *DB* برای تعریف رشته کاراکتری که بین علامت نقل قول قرار گرفته است نیز استفاده می شود.

```
            CSEG          AT      100h
SQUARES: DB         0,1,4,9,16,25
MESSAGE     DB 'WAHT IS YOUR NAME'
```

۷ـ۵ـ۶ـ DW (تعریف کلمه)

قالب دستور *DW* (تعریف کلمه) به شکل زیر است:

```
[label:]    DW    expressiorn    [,expression][...]
```

دستور *DW* همان دستور *DB* است با این تفاوت که دو محل حافظه (۱۶ بیت) برای انتساب هر یک از داده ها بکار می روند. برای مثال، دستورهای:

```
            CSEG        AT      200h
DW          $,'A',1234h,2,'BC'
```

۸ـ۵ـ۶ـ PUBLIC

قالب دستور *PUBLIC* (نماد عمومی) بصورت زیر است:

```
PUBLIC symbol [,symbol][...]
```

دستور *PUBLIC* به فهرست نمادهای مشخص شده اجازه می دهد که بیرون از مدول اسمبل شده جاری، شناخته شده و به کار روند. نمادی که با *PUBLIC* همراه باشد، باید در مدول جاری تعریف شده باشد. همراه کردن نماد *PUBLIC* به نماد اجازه می دهد که در یک پیمانه دیگر، قابل استفاده باشد. برای مثال:

```
PUBLIC INCHAR,OUTCHAR,INLINE,OUTSTR
```

۹ـ۵ـ۶ـ EXTERN

قالب دستور (نماد خارجی) عبارت است از:

```
EXTERN segment_type (symbol [,symbol][...],...)
```

دستور *EXTERN* نمادهایی را که در پیمانه های دیگر تعریف شده باشند، طوری فهرست می کند که در نماد جاری مرجع باشند، فهرست نمادهای خارجی باید دارای نوع سگمنت متحد با هر نماد در فهرست باشد.

دستورهای *PUBLIC* و *EXTERN* با یکدیگر کار می کنند. دو فایل نشان داده شده در زیر یعنی *MAIN.SRC* و *MESSAGE.SRC* را در نظر بگیرید. زیر روالهای *HELLO* و *GOOD_BYE* در پیمانه *MESSAGE* تعریف شده اند. اما با استفاده از دستور *PUBLIC* در دسترس سایر پیمانه ها قرار می گیرند. زیر برنامه ها در پیمانه *MAIN* فراخوانی می شوند، حتی اگر در آنجا تعریف نشده باشند. دستور *EXTERN* بیانگر این امر است که نمادها در یک پیمانه دیگر تعریف شده اند.

```
MAIN.SRC
            EXTERN      CODE(HELLO,GOOD_BYE)
            ...
            CALL  HELLO
            ...
            CALL  GOOD_BYE
            ...
            END

MESSAGE.SRC
            PUBLIC      HELLO,GOOD_BYE
            ...
```

```
HELLO:
        ...
        RET
GOOD_BYE:
        ...
        RET
        END
```

هیچ کدام از پیمانه های *MAIN.SRC* و *MESSAGE.SRC* یک برنامه کامل نیستند و باید بطور جداگانه اسمبل شده و با یک برنامه قابل اجرا به یکدیگر ملحق شوند. مرجع های خارجی با آدرس صحیح به عنوان مقصد دستورالعملهای *CALL* بین دستورها قرار می گیرند.

۱۰ـ۵ـ۶ـ سگمنت

قالب دستور *SEGMENT* به صورت زیر است:

```
symbol      SEGMENT      segment_type
```

موارد زیر انواع سگمنت (*segment_type*) تعریف شده در ۸۰۵۱ هستند.

CODE : (سگمنت کد) مکانهایی که کدهای برنامه در آن قرار می گیرند.

XDATA : (فضای داده خارجی) مکانهایی از حافظه *RAM* خارجی که داده ها را می توان در آنها ذخیره کرد.

DATA : فضای داده داخلی قابل دسترسی با آدرس دهی مستقیم

IDATA : فضای داده داخلی قابل دسترسی با آدرس دهی غیر مستقیم

BIT : (فضای بیت) مکانهایی از فضای داده داخلی با آدرسهای *20h* الی *2Fh*

بعنوان مثال:

```
EPROM       SEGMENT      CODE
```

دستور فوق نماد *EPROM* را به صورت یک *SEGMENT* از نوع *CODE* بیان می کند. در این دستور نوع سگمنت مشخص شده است ولی شروع آن مشخص نشده است. برای تعیین آدرس شروع سگمنت از دستور *RSEG* استفاده می شود.

۱۱ـ۵ـ۶ـ دستورهای انتخاب سگمنت

هنگامی که اسمبلر با یک دستور انتخاب سگمنت مواجه می شود، کد یا داده بعدی را متوجه سگمنت انتخاب شده می کند تا سگمنت دیگری توسط دستور انتخاب سگمنت، انتخاب شود. دستور ممکن است سگمنت با جایگزینی مجددی را که قبلاً تعریف شده است انتخاب نماید یا بطور دلخواه سگمنتهای مطلق را تولید و انتخاب نماید.

۱ـ۱۱ـ۵ـ۶ـ *RSEG* (سگمنت با جایگزینی مجدد)

قالب دستور *RSEG* به صورت زیر است:

```
RSEG  segment_name
```

که در آن *segment_name* نام سگمنت با جایگزینی مجددی است که قبلاً با دستور *SEGMENT* تعریف شده باشد. *RSEG* یک دستور انتخاب سگمنت است که کد یا داده بعدی را متوجه سگمنت نامیده شده می کند تا این که دستور انتخاب سگمنت دیگری اجرا شود.

۲ـ۱۱ـ۵ـ۶ـ انتخاب سگمنتهای مطلق

یک سگمنت مطلق با یکی از دستورهای زیر انتخاب می شود:

```
CSEG [AT    address]
DSEG [AT    address]
ISEG  [AT   address]
BSEG [AT    address]
XSEG [AT    address]
```

این دستورها بترتیب یک سگمنت مطلق را در کد، داده داخلی، داده داخلی غیر مستقیم، بیت و فضاهای آدرس داده خارجی انتخاب می کند. اگر یک آدرس مطلق فراهم شده باشد اسمبلر آخرین سگمنت آدرس مطلق از هر نوع سگمنت مشخص شده ای را پایان داده و یک سگمنت جدید را در آن آدرس شروع می کند. اگر یک آدرس مطلق مشخص نشده باشد، آخرین سگمنت مطلق مشخص شده ادامه می یابد. اگر هیچ سگمنت مطلقی از این نوع قبلاً انتخاب نشده باشد و آدرس مطلق حذف شود، سگمنت جدید از محل *0* شروع می شود. مرجع های مستقیم مجاز نبوده و آدرسهای شروع باید مطلق باشند.

هرسگمنت دارای شمارنده محل مخصوص به خود است که همیشه مقدار اولیه آن *0* است. سگمنت پیش فرض یگ سگمنت کد مطلق است، بنابراین حالت اولیه اسمبلر محل *0000h* در سگمنت کد مطلق می باشد. هنگامی که سگمنت دیگری برای اولین بار انتخاب می شود، شمارنده محل سگمنت قبلی آخرین مقدار فعال را حفظ می کند. هنگام انتخاب مجدد سگمنت قبلی، شمارنده، محل آخرین مقدار فعال را خواهد داشت. دستور *ORG* ممکن است برای تغییر شمارنده محل داخل سگمنت انتخاب شده جاری به کار رود. در مثال زیر مثالهایی از تعریف و آغاز سگمنت های مطلق و با جایگزینی مجدد را نشان می دهد.

در دو خط اول نمادهای *ONCHIP* و *EPROM* بترتیب از نوع سگمنت *DATA* و سگمنت *CODE* تعیین شده اند. خط سوم یک سگمنت بیت مطلق را از آدرس بیت *70h* شروع می کند. سپس *FLAG1*و *FLAG2* به عنوان برچسبهای متناظر محلهای بیت آدرس پذیر *70h* و *71h* تولید می شوند. دستور *RSEG* در خط ۶ ، سگمنت *ONCHIP* با جایگزینی مجدد را برای *RAM* داخلی آغاز می کند. *TOTAL* و *COUNT* برچسبهای متناظر با محلهای بایت هستند. *SUM16* یک برچسب متناظر با محل یک کلمه (۲ بایت) است. استفاده بعدی *RSEG* در خط ۱۰ سگمنت *EPROM* با جایگزینی مجدد را برای حافظه کد آغاز می کند. برچسب *BEGIN* در آدرس اولین دستورالعمل در این مثال مربوط به *EPROM* می باشد. بایستی توجه کرد که تعیین آدرس برچسب های *TOTAL*، *COUNT*، *SUM16* و *BEGIN* امکان پذیر نیست و این وظیفه لینکر است که این آدرسها را بعداً مشخص نماید. اما *FLAG1* و *FLAG2* همیشه منطبق برآدرسهای بیت *70h* و *71h* می باشند چون در یک سگمنت *BIT* مطلق تعریف شده اند.

```
ONCHIP    SEGMENT    DATA
EPROM     SEGMENT    CODE
          BSEG       AT 70h
FLAG1:    DBIT       1
FLAG2:    DBIT       1
          RESEG  ONCHIP
TOTAL:    DS         1
COUNT:    DS         1
SUM16:    DS         2
          RSEG       EPROM
BEGIN:
          MOV        TOTAL,#0
          .
          .
          .
          END
```

۶ـ۶ مثالهای برنامه نویسی با اسمبلی

در مثالهای زیر برنامه هایی مورد بحث قرار می گیرد که به میکروکنترلر بعنوان یک میکروپروسسور ۸ بیتی نگرسته و به قابلیتهای میکرو کنترلر جزپینهای ورودی خروجی توجه ندارد.

مثال۱: برنامه ای بنویسید که روی پورت *P1* موج مربعی ایجاد نماید.

```
          ORG 0h
          MOV P1,#0FFh
LOC0:
          MOV R3,#20
LOC1:
          DJNZ R3,LOC1
          CPL    P1
          LJMP LOC0
          END
```

مثال ۲: برنامه ای بنویسید که روی پورت *P1* موج مربعی ایجاد نماید بنحوی که فرکانس آن یک دهم فرکانس قبلی باشد.

```
          ORG 0h
          MOV P1,#0FFh
LOC0:
          MOV R2,#10
LOC1:
          MOV    R3,#20
LOC2:
          DJNZ R3,LOC2
          DJNZ R2,LOC1
          CPL    P1
          LJMP LOC0
          END
```

مثال قبلی را با استفاده از زیر برنامه بنویسید.

```
          ORG 0h
          MOV P1,#0FFh
LOC0:
          LCALL DELAY
          CPL    P1
          LJMP LOC0
```

```
DELAY:
        MOV R2,#10
LOC1:
        MOV   R3,#20
LOC2:
        DJNZ R3,LOC2
        DJNZ R2,LOC1
        RET
        END
```

مثال۳: برنامه ای بالا را به نوحی تغییر دهید که نسبت فرکانسها براحتی قابل تغییر باشد.

```
        ORG 0h
        MOV P1,#0FFh
LOC0:
        MOV   A,#10
        LCALL DELAY
        CPL   P1
        LJMP LOC0
DELAY:
        MOV R2,A
LOC1:
        MOV   R3,#0FFh
LOC2:
        DJNZ R3,LOC2
        DJNZ R2,LOC1
        RET
        END
```

تمرین ۱: برنامه ای بنویسید که یک عدد صحیح را که در رجسترهای *R0* و *R1* قرار گرفته است را با عدد صحیحی دیگر که در رجسترهای *R2* و *R3* قرار گرفته اند جمع و نتیجه را در رجسترهای *R4* و *R5* ذخیره نماید. (*R0*، *R2* و *R3* بایتهای کم ارزش باشند)

تمرین ۲ : برنامه ای بنویسید که یک بایت را که در انباره قرار گرفته است به کد اسکی تبدیل و در رجستر های *R0* و *R1* برگرداند.

تمرین ۳ : برنامه ای بنویسید که پورت *P1* را خوانده آنرا با عدد *80h* مقایسه کند اگر عدد خوانده شده از *80h* کوچکتر بود پین *P3.2* را مکمل نماید و اگر از *80h* بزرگتر بود *P3.3* را مکمل نماید و اگر مساوی بود کار بخصوص انجام ندهد و یا عمل مکمل شدن انجام گرفت.

تمرین ۴: برنامه ای بنویسید که تمامی محتوی *RAM* داخلی را در ۸۰۵۱ صفر نماید.

تمرین ۵ : برنامه ای بنویسید که وضعیت پایه سوم پورت یک را بررسی نموده و در صورت یک شدن آن به رجستر *DPTR* یک واحد اضافه نماید.

تمرین ۶: برنامه ای بنویسید که محتویات حافظه *RAM* داخلی از آدرس *00h* الی *40h* را برای پورت سریال ارسال نماید.

فصل هفتم

برنامه نویسی به زبان C

۱ـ۷ـ مقدمه

هر میکروپروسسور یا میکروکنترلری که مد نظر قرار گیرد زبان اسمبلی خاص خود را دارد که هنگام استفاده از آن پروسسور بایستی برنامه مورد نظر را به آن زبان اسمبلی خاص نوشته و پس از غلط گیری کد آنرا استخراج و توسط امکانات سخت افزاری پیش بینی شده به آن اعمال کرد. بعنوان مثال اگر میکروپروسسور از خانواده *X86* باشد بایستی برنامه را با زبان اسمبلی *80x86* نوشته پس از غلط گیری و استخراج فایل اجرایی آنرا اجرا کرد. اجرای این چنین برنامه ای روی کامپیوترهای موجود بسادگی ممکن است اما اگر میکروپروسسور غیر از آن باشد بایستی برنامه را با اسمبلی مربوط به آن میکروپروسسور نوشته و پس از غلط گیری و استخراج کد برنامه، توسط کامپیوتر آنرا به سخت افزار منتقل و اجرا نمود. نوشتن برنامه با اسمبلی بسیار وقت گیر بوده و عیب یابی آنها بسادگی مقدور نیست به این دلیل استفاده از مترجمها رایج شده و می توان برنامه را به زبان سطح بالا نوشت و سپس آنرا به زبان ماشین مربوطه تبدیل و پس از انتقال به سخت افزار مربوطه اجرا کرد.

در طراحی سیستمهای سخت افزاری لازم است طراح با نرم افزاری قدرت مندی همچون *C* آشنایی داشته باشد و این بدلایل زیادی لازم است که از آنجمله می توان به موارد زیر اشاره کرد:

۱- نوشتن برنامه با اسمبلی برای یک میکرو پروسسور هنگامی که وظیفه محوله به میکرو پروسسور زیاد است بسیار وقت گیر و مشکل است در این مواقع برنامه نویس لازم است برنامه را با زبان سطح بالا همچون *C* نوشته و توسط کامپایلرهایی که بدین منظور نوشته شده اند به زبان قابل فهم برای میکرو پروسسور تبدیل کند.

۲- گاهی اوقات به جهت سادگی و عیب یابی ساده تر بهتر است ابتدا الگوریتم مورد نظر را که قرار است روی میکروپروسسور یا میکروکنترلر خاص پیاده گردد با یک زبان سطح بالا نوشته و آنرا روی کامپیوتر اجرا کرد و یک جواب معقول و پسندیده ایی گرفت سپس با اصلاحاتی آنرا روی سخت افزار مربوطه منتقل و نتیجه مطلوب را با دردسر کمتری دریافت کرد.

۳ـ انجام عملیات ضرب و تقسیم اعداد صحیح و اعشاری با نوشتن برنامه به زبان اسمبلی بسیار وقت گیر است استفاده از کامپایلرها کمک خوبی می تواند باشد.

۵ـ علاوه بر اینکه نوشتن برنامه با زبان سطح بالا همچون *C* ساده بوده و در وقت و هزینه صرفه جویی می کند بلکه خوانایی برنامه را افزایش داده و عیب یابی آنرا بسیار ساده تر می کند.

لذا به دلایل متعدد فراگیری زبان C لازم بوده و بایستی از آن در طراحی و ساخت مخصوصاً هنگامی که کار محوله نسبتاً زیاد است استفاده کرد. در مورد میکروکنترلر ۸۰۵۱ و استفاده از کامپایلر C51 فراگیری چند دستور برای نوشتن برنامه کفایت کرده و لزومی به یادگیری کامل زبان C نمی باشد. در ادامه انواع داده ها، چگونگی تعریف متغیرها، عملگرها و غیره در زبان C استاندارد بیان شده سپس اختلافات کامپایلر C51 با آن بیان می گردد و در فصل آینده با مثالهای متعدد نحوه برنامه نویسی با اسمبلی و C نشان داده می شود.

۲ـ۷ـ برنامه نویسی به زبان C

هدف از بیان این مطالب توضیح برنامه نویسی و دستورات مورد استفاده در زبان C نبوده بلکه بیان خلاصه ای از دستورات مورد استفاده در برنامه نویسی برای میکروکنترلرها می باشد، لذا ابتدا C استاندارد بصورت خلاصه بیان شده و سپس اختلافات کامپایلر C51 با آن بیان می گردد.

۳ـ۷ـ عملگرهای زبان C

در برنامه نویسی به زبان C دانستن عملگرهای محاسباتی، منطقی، بیتی، مقایسه ای و غیره ضروری بوده که در ادامه این عملگرها بصورت جدول و با توضیح مختصر بیان می شوند.

۱ـ۳ـ۷ـ عملگرهای یکانی

عملگر	مفهوم	مثال	نتیجه
!	Not منطقی	*!(0)*	مقداری مخالف صفر
~	مکمل ۱	*~(0x55)*	*0xAA*
+	مثبت یگانی	*b=+a*	
-	منفی یگانی	*b=-a*	
++	افزایش به اندازه ۱	*a++*	*a=a+1*
--	کاهش به اندازه ۱	*a--*	*a=a-1*
&	آدرس	*b=&a*	*b* برابر آدرس *a* می شود
*	محتوای آدرس	*b=*addr*	*b* برابر مقدار موجود در حافظه آدرس *addr* می شود
sizeof()	اندازه عملوند برحسب بایت	*sizeof(int)*	2

۲ـ۳ـ۷ـ عملگرهای حسابی

عملگر	مفهوم	مثال	نتیجه
*	ضرب	*2*3*	6
/	تقسیم	*6/2*	3
%	باقیمانده تقسیم	*17%3*	2
+	جمع	*2+3*	5
-	تفریق	*3-2*	1

۳ـ۳ـ۷ـ عملگرهای شیفت و مقایسه

عملگر	مفهوم	مثال	نتیجه
<<	شیفت به چپ	*0x01<<2*	*0x04*
>>	شیفت براست	*0x80>>2*	*0x20*
<	کمتر از	*3<2*	*0*
<=	کوچکتر یا مساوی	*2<=2*	مخالف صفر
>	بیشتر از	*3>2*	مخالف صفر
>=	بزرگتر یا مساوی	*2>=3*	*0*
==	برابر است با	*2==3*	*0*
!=	مخالف است با	*2!=3*	مخالف صفر

۴ـ۳ـ۷ـ عملگرهای بیتی و منطقی

عملگر	مفهوم	مثال	نتیجه
&	بیتی *AND*	*0xFF & 0x22*	*0x22*
^	بیتی *XOR*	*0xFF ^ 0xFF*	*0*
|	بیتی *OR*	*0x0a | 0x20*	*0x2a*
&&	منطقی *AND*	*2 && 3*	مقدار غیر صفر
||	منطقی *OR*	*0||1*	مقدار غیر صفر

۴ـ۷ـ انواع داده های در C

در C مشابه هر زبان برنامه نویسی انواع داده هایی که کامپایلر می تواند آنها را شناخته و از آنها استفاده کند وجود دارد. برای استفاده از انواع داده ها بایستی متغیرهایی را تعریف کرد.

برای معرفی متغیرها از فرمت زیر استفاده می شود:

data_type variable_list;

مثال:

char a,b;

data_type: مشخص کننده نوع داده می باشد که می تواند یکی از انواع *char*، *int*، *float*، *short int*، *long int* و *double* باشد که در ادامه هریک از آنها مختصراً توضیح داده می شود.

۱ـ ***char***: برای تعریف یک متغیر که بایستی یک بایت یا ۸ بیت را در خود ذخیره کند استفاده می شود. و می تواند با علامت (*signed*) و یا بدون علامت (*unsigned*) باشد.

۲ـ ***int*** : توسط این کلمه کلیدی می توان یک متغیر از نوع عدد صحیح که ۱۶ بیت طول دارد استفاده کرد. این نوع اعداد نیز می توانند با علامت یا بدون علامت باشند. بایستی در نظر داشت حداکثر عددی که در حالت با علامت می توان نشان داد ۳۲۷۶۷ تا ۳۲۷۶۸- است و اگر عدد بدون علامت در نظر گرفته شود اعداد از ۰ الی ۶۵۰۰۰ را می توان نمایش داد.

۳ـ ***long int***: نشان دهنده آنست که متغیر از نوع عدد صحیح بلند و در محدوده اعداد ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۸ می باشد.

float و ***double***: برای تعریف یک متغیر از نوع اعشاری از این کلمه کلیدی استفاده می گردد. استفاده از این کلمه کلیدی در صورتی مؤثر است که در کامپایلر مربوطه عملیات ممیز شناور پشتیبانی شده باشد.

با قرار دادن کلمه کلیدی *unsigned* قبل از هریک از انواع داده فوق، نوع داده بدون علامت می شود و محدوده عدد مثبت آن تا دو برابر بعلاوه یک افزایش می یابد.

۱ـ۴ـ۷ـ کلمات کلیدی ***const*** و ***volatile***

می توان هنگام تعریف یک متغیر، پیش از نوع داده *data_type* از کلمات کلیدی *const* و *volatile* استفاده کرد.

const: این کلمه کلیدی باعث می شود تا تغییر محتوای متغیری که به این شکل تعریف شده است در برنامه غیر ممکن گردد. لذا می توان از آن برای تعریف ثابتها استفاده کرد.

volatile: این کلمه کلیدی برای متغیرهایی که امکان تغییر آنها بدون اطلاع برنامه وجود دارد، بکار می رود.

۲ـ۴ـ۷ـ کلاس ذخیره سازی متغیرها

در تعریف یک متغیر، می توان کلاس ذخیره سازی آنرا قبل از *data_type* معین کرد. بدین ترتیب، شکل کلی تعریف متغیر بصورت زیر می باشد:

<storage_class> <data_type> variable_list;

مثال:

Satatic int a,b,c;

<storage_class> می تواند یکی از انواع *auto static extern* و *register* باشد.

auto : نشان دهنده آنست که متغیر از نوع خودکار می باشد. این نوع متغیر فقط در داخل تابع شناخته شده است و با برگشت از تابع مقدار آن از بین می رود. معمولاً این کلمه کلیدی در اعلان متغیر ذکر نمی شود زیرا چنانچه متغیری در داخل تابع بدون ذکر کلاس ذخیره سازی اعلان شود، کامپایلر آنرا از نوع خودکار فرض خواهد کرد.

static : نشان دهنده آنست که متغیر از نوع ایستا می باشد. چنانچه این اعلان در داخل تابع انجام شود، متغیر فقط در داخل تابع شناخته شده است، ولی مقدار آن از بین نمی رود. بعبارت دیگر چنانچه تابع دوباره فراخوانده شود، متغیر مقدار قبلی را داراست و چنانچه این اعلان در خارج از تابع انجام شود، متغیر فقط در کامپایل جاری شناخته شده است و از این متغیر نمی توان در فایلهای دیگر (در برنامه های چند فایلی) استفاده کرد.

extern :نشان دهنده آنست که تعریف اصلی (اختصاص حافظه) و مقدار اولیه متغیر، در فایل دیگری انجام شده است و در فایل جاری، می توان از آن متغیر استفاده کرد.

register: نشان دهنده آنست که مکان متغیر در یکی از رجسترهای میکروپروسسور یا میکروکنترلر می باشد.

۳ـ۴ـ۷ـ آرایه ها و رشته ها

یک آرایه، مجموعه ای از متغیرهای بهم مرتبط است که بوسیله یک نام مشترک، مورد رجوع قرار می گیرند. آرایه ها می توانند از یک تا چندین بعد داشته باشند. نحوه تعریف یک آرایه یک بعدی بصورت زیر می باشد.

data_type var_name[size];

مثال

char A[20];

در این تعریف *data_type* نوع داده، *var_name* نام آرایه مورد نظر و *size* تعداد عناصر این آرایه می باشد. برای دسترسی به عنصر آرایه از شماره آن عنصر به عنوان زیرنویس آن آرایه استفاده می شود. زیر نویس یک آرایه با صفر شروع می شود. بطور نمونه در مثال فوق آرایه *A* دارای عناصر *A[0]* تا *A[19]* می باشد.

علاوه بر آرایه یک بعدی می توان آرایه دو یا چند بعدی نیز ساخت.

از متداولترین کاربرد آرایه های یک بعدی، تعریف رشته ها می باشد. یک رشته در حقیقت آرایه ای از کاراکترها میباشد که به یک کاراکتر تهی (*NULL*) ختم می شود. کاراکتر تهی نشان دهنده پایان رشته می باشد. بعنوان مثال آرایه زیر می تواند رشته ای با طول حداکثر ۱۰ کاراکتر را در خود نگهدارد که عضو یازدهم آرایه، کاراکتر تهی می باشد:

```
Char name[11];
```

همانند دیگر متغیرها، آرایه ها را نیز می توان در هنگام تعریف مقدار دهی اولیه نمود و برای این منظور از قالب بندی زیر استفاده می شود.

```
data_type Array_name={value_list};
```

مثال:

```
char A[5]={1,3,4,6,7};
```

۴ـ۴ـ۷ـ اشاره گرها

اشاره گر یک متغیر خاص است که آدرس متغیرها و ثابتهای دیگر را در خود نگهمیدارد. بعنوان مثال در زبان اسمبلی ۸۰۵۱ ، رجسترهای *R0*، *R1*، *DPTR* و *SP* برای نگهداشتن آدرس داده های قابل دستیابی با آدرس دهی غیر مستقیم بکار می روند، لذا طبق تعریف، آنها اشاره گرهایی هستند برای داده هایی که دستیابی به آنها صورت گرفته است.

کاربرد اصلی اشاره گرها دسترسی غیر مستقیم به داده هاست. با توجه به اینکه همه متغیرها در نهایت در *RAM* داخلی یا خارجی مقیم می شوند، لذا همه آنها از طریق اشاره گرها قابل دستیابی هستند. برخلاف زبان اسمبلی، *C* این امکان را برای برنامه نویسان فراهم می کند که با ایجاد متغیرهای اشاره گری، از آنها برای اشاره به متغیرهای دیگر یا بخشهای داده ای همچون اعداد صحیح و ممیز شناور استفاده کنند.

۵ـ۴ـ۷ـ ساختار ها

قالب بندی یک ساختار بصورت زیر است:

```
struct type_name{
        type member1;
        type member2;
        ...
        type memberN;
        } variable_list;
```

مثال:

```
struct date {
        int month;
        int day;
        int year;
        } birthday;
```

کلمه کلیدی *struct* به کامپایلر تفهیم می کند که یک نوع داده ساختاری در حال تعریف شدن می باشد. در این قالب بندی هر *type* بیانگر یک نوع داده مجاز *C* می باشد. *type_name*، نام نوع داده ای

است که توسط ساختار تعریف می شود و *variable_list*، متغیرهایی می باشند که از نوع ساختار تعریف می شوند. چنانچه از این نوع ساختار، دیگر استفاده نمی شود، می توان *type_name* را از تعریف حذف کرد. همچنین می توان با حذف *variable_list* ، تعریف متغیرهای ساختار را به بعد موکول کرد. اگر نوع داده ساختاری قبلاً تعریف شده باشد، می توان به شکل زیر متغیرهایی را از آن نوع معرفی نمود.

Struct type_name variable_list;

برای دسترسی به اعضای یک ساختار، می توان از عملگر "." استفاده کرد. مثال زیر نمونه ای از این دسترسی را نشان می دهد:

Birthday.month=10;

اگر اشاره گری به یک ساختار تعریف شده باشد، می توان از عملگر "->" برای دسترسی به اعضای آن ساختار استفاده کرد. مثال زیر این نحوه دسترسی را نشان می دهد:

```
Struct date *ptr;
Ptr->day=21;
```

۵ـ۷ـ حلقه ها و تصمیم گیریها

۱ـ۵ـ۷ـ حلقه های *while* و *do while*

ساده ترین کنترل حلقه ای بوسیله ترکیب حلقه *while* که در زیر آمده است صورت می گیرد.

while(شرط)

{ یک یا چند دستور *C*}

در ترکیب فوق شرط ارزیابی می گردد در صورتی که شرط برقرار باشد (عبارت داخل پرانتز مخالف صفر باشد) یک یا چند دستور *C* اجرا می شود سپس شرط مورد ارزیابی مجدد قرار می گیرد و این چرخه تا زمانیکه شرط صفر نشده است ادامه می یابد. اگر شرط برقرار نباشد دستورات بعد از ساختار *while* اجرا خواهند شد.

بعنوان مثال هنگام استفاده از پورت سریال با قرار دادن یک بایت در بافر پورت سریال، بیت *TI* صفر شده و تا این بایت بطور کامل ارسال نگردد این بیت یک نمی گردد. می توان از این دستور مانند زیر استفاده و شرط فوق را مورد ارزیابی قرار داد تا یک بایت بطور کامل از طریق پورت سریال ارسال گردد.

```
SBUF=0x55;
While(TI==0);
```

حلقه *do while* کاملاً شبیه حلقه *while* است با این تفاوت که در آن شرط حلقه بعد از اجرای دستورات موجود در داخل حلقه ارزیابی می شود در حالی که در حلقه *while* این کار قبل از دستورات صورت می گرفت. بنابراین در حلقه *do while* حلقه حداقل یکبار اجرا می گردد. ترکیب حلقه *do while* بصورت زیر است:

do{

یک یا چند دستور *C*

}while(شرط)

با توجه به ساختار این دو، کاربرد *do while* از *while* کمتر است.

۲ـ۵ـ۷ـ حلقه *for*

حلقه *for* با جزئیات و ترکیب زیر تعریف می شود:

for (تجدید مقدار متغیر; شرط ; مقدار دهی اولیه)

{*C* یک یا چند دستور};

حلقه *for* برای پیاده سازی حلقه های کنترل شده با شمارش طراحی می شود. حلقه *for* بطور صریح شامل یک عبارت برای مقدار دهی اولیه ، شرط و عبارتی برای تجدید مقدار متغیر شمارنده به منظور نگهداشتن تعداد اجرای حلقه می باشد. حلقه *for* از نظر نحوه شمارش کاملاً شبیه برنامه زیر است:

مقدار دهی اولیه;

while(شرط)

{*C* یک یا چند دستور

تجدید مقدار متغیر;}

۳ـ۵ـ۷ـ دستورات تصمیم گیری و کنترل

دستور *if* ، ساده ترین کنترل با تصمیم گیری را در *C* ارائه می دهد. این دستور یک عبارت شرطی را بررسی می کند، اگر شرط درست باشد، عملیاتهای مشخص شده ای را انجام می دهد.

if(شرط)

C یک دستور

اگر تعداد دستورات بیش از یکی باشد بایستی بصورت زیر عمل کرد:

if(شرط)

{

C یک یا چند دستور

}

دستور *else* امکان اجرای یک عملیات دیگر را در صورت درست نبودن شرط فراهم می کند.

if(شرط)

{

دستورات گروه ۱

}

else

{

دستورات گروه ۲

}

دستور *else if* نیز وجود دارد که اگر شرط قبلی برقرار نباشد شرط دیگری را بررسی می کند و در صورت درست بودن دستورات را اجرا می کند.

۴ـ۵ـ۷ـ دستور *break*

این دستور برای خارج شدن بدون شرط از حلقه، مورد استفاده قرار می گیرد. هرگاه برنامه به دستور *break* برسد از حلقه خارج شده و ادامه برنامه از اولین دستور بعد از حلقه دنبال می شود.

۵ـ۵ـ۷ـ دستور *switch*

شکل کلی ایجاد یک دستور *switch* بصورت زیر می باشد:

```
Switch (expression) {
        Case constant1:
                Statement sequence
                Break;
        Case constant2:
                Statement sequence
                Break;
        …
        default:
                Statement sequence
}
```

این دستور برای انتخاب یکی از چند گزینه ممکن در اجرای برنامه بکار می رود. نحوه کار آن به این صورت است که مقدار عبارت *expression* متوالیاً با فهرستی از ثابتهای صحیح یا کاراکتری مقایسه می شود. زمانیکه مورد انطباقی یافت شود، دستور یا دستورهای مربوط به آن انطباق اجرا می شود و اگر هیچ انطباقی یافت نشد، دستورات مربوط به قسمت *default* اجرا می شود. بکار بردن *default* دلخواه می باشد و اگر این قسمت استفاده نشود، در اینصورت با یافت نشدن مورد انطباق هیچ کاری صورت نمی گیرد. از طرفی دیگر اگر دستور *break* حذف شود، آنگاه برنامه پس از اجرای مورد انطباق، بقیه انتخابها ها را بهمراه *default* اجرا کرده تا به انتهای دستور *switch* برسد مگر آنکه سر راه اجرای خود به یک دستور *break* دیگر برسد که در اینصورت دستور از مکان *break* قطع شده و ادامه کار از دستور بعد از *switch* دنبال می شود.

۶ـ۷ـ توابع، ماژول و برنامه ها

توابع، ارکان اصلی یک برنامه *C* را تشکیل می دهند. در حقیقت یک برنامه *C* عبارت از چندین تابع *C* می باشد. ممکن است یک برنامه از یک یا چند تابع که هر کدام وظیفه أی را انجام می دهند تشکیل شده باشد. در برنامه نویسی به جهت سادگی در نوشتن و عیب یابی سعی می گردد برنامه به اجزاء کوچکتر تقسیم گردد. هنگامی که یک برنامه شروع می گردد، اولین تابعی که اجرا می شود تابع *main()* می باشد، بدین ترتیب هر برنامه باید دارای تابع *main()* باشد.

یک تابع زیر برنامه ایست که شامل یک یا چند دستور *C* بوده و عمل بخصوصی را انجام می دهد. شکل کلی تعریف یک تابع بصورت زیر می باشد:

```
return_type function(parameter list)
{

/* body of the function */
}
```

مثال:

```
char max(char a, char b) {
if( a<b) reurn b;
else return a;
}
```

در قالب بندی فوق *return_type* نوع داده برگشتی تابع می باشد که بطور پیش فرض *int* است این مقدار می تواند یکی از انواع داده های ذکر شده باشد. چنانچه تابعی مقدار برگشتی نداشته باشد، نوع برگشتی آن باید *void* تعریف شود.

برای برگرداندن مقدار در داخل بدنه تابع، از دستور *return* استفاده می شود.این دستور باعث قرار دادن مقدار برگشتی در مکان مخصوص و خارج شدن از تابع می شود. اگر نوع داده برگشتی *void* باشد، دیگر احتیاج به دستور *return* نبوده و با پایان یافتن بدنه تابع. برنامه از تابع خارج می شود. *parameter list* آرگومان های تابع می باشند که هریک بصورت *data_type parameter_name* تعریف شده و با عملگر ",," از هم جدا می شوند. از آرگومانها برای ارسال مقدار به تابع استفاده می شود. اگر یک تابع فاقد آرگومان باشد، *parameter list* از تعریف تابع حذف می شود ولی پرانتزها باقی می مانند.

فراخوانی یک تابع بصورت *function_name(parameter name)* می باشد. بعنوان مثال می توان در متن برنامه از جمله *number=max(i,2)* استفاده کرد. در اینصورت *a* مقداری برابر *i* و *b* برابر ۲ می شود و طبق تعریف تابع *max* ، مقدار بزرگتر به متغیر *number* نسبت داده می شود.

۷ـ۷ـ کامپایلر *C51*

کامپایلر *C51* همچون کامپایلرهای دیگر زبان *C*، قواعد *ANSI C* را رعایت می کند. بعبارت دیگر کاربر مجاز است کلیه دستورات *ANSI C* را بکار برد. علاوه بر این بخاطر آنکه *C51* مخصوص میکروکنترلرهای خانواده *MCS-51* می باشد، لذا تعدادی کلمه کلیدی به آن اضافه شده است که به کاربر اجازه می دهند تا بتواند در نحوه کار این کامپایلر دخالت کند.

۸ـ۷ـ کلمات کلیدی کامپایلر *C51*

کلمات کلیدی، کلماتی هستند که برای کامپایلر معنای مشخصی دارند و برنامه نویس نمی تواند از این کلمات برای نامگذاری متغیرها، توابع و ... استفاده کند. لیست این کلمات در جدول زیر آورده شده است:

float	*double*	*long*	*char*	*int*
unsigned	*union*	*struct*	*void*	*short*
register	*extern*	*auto*	*enum*	*signed*
return	*goto*	*typedef*	*alien*	*static*
else	*if*	*continue*	*break*	*sizeof*

case	*Switch*	*while*	*do*	*for*
volatile	*const*	*using*	*interrupt*	*dafault*
idata	*xdata*	*sfr*	*sbit*	*bit*
bdata	*sfr16*	*code*	*data*	*pdata*
small	*large*	*compact*		

همانطور که ملاحظه می شود کلمات کلیدی *data, code, bit, bdata, alien, sfr, sbit,idata,* *interrupt, large, pdata, small, using, xdata, compact* و *sfr16* در کامپایلر *C51* اضافه شده اند که کاربردهای خاص خود را برنامه نویسی برای *MCS-51* دارند.

۹ـ۷ـ نحوه تعیین ناحیه حافظه در *C51*

حافظه میکروکنترلر ۸۰۵۱ شامل چهار ناحیه می باشد که دسترسی به هریک از آنها توسط روش آدرس دهی مربوط به خود امکان پذیر است. این چهار ناحیه عبارتند از: حافظه برنامه (*code*) ، حافظه دیتای داخلی (*bdata, idata, data*) ، حافظه دیتای خارجی (*pdata, xdata*) و حافظه ***sfr*** ها.

C51 امکان تعیین ناحیه حافظه مورد نیاز برای متغیرها و آرگومانها را به دو روش فراهم کرده است. در روش اول روش پیش فرض و روش دوم تخصیص دهی حافظه براساس مدل حافظه انتخاب شده توسط پارامترهای کنترلی (*large, Compact, Small*) می باشد. علاوه بر این امکان تغییر این رویه برای متغیرهای مختلف را نیز با اضافه کردن کلمات کلیدی *bit, xdata, pdata, bdata, idata, data* و *sfr* فراهم کرده است. توسط این کلمات کلیدی می توان ناحیه ذخیره سازی متغیر در حافظه را نیز مشخص کرد.

بجز در موارد بسیار خاص، می توان از مدل پیش فرض *small* استفاده کرد. این مدل باعث تولید کدهای بهینه تر و سریعتر می شود.

۱ـ۹ـ۷ـ تعیین ناحیه ذخیره سازی متغیر در حافظه بطور صریح

کلمات کلیدی که برای تعیین کردن ناحیه حافظه متغیر بکار می روند عبارتند از: *xdata, pdata,* *idata, data* و *bdata* .

data : نشان دهنده آنست که مکان ذخیره متغیر در *RAM* داخلی می باشد و دسترسی به آن از طریق آدرس دهی مستقیم امکان پذیر است.

```
data char i;
i=2;
```

معادل اسمبلی دستورات مثال فوق عبارتست از:

```
MOV i,#02h
```

idata : نشان دهنده آنست که مکان متغیر در *RAM* داخلی می باشد و دسترسی به آن از طریق آدرس دهی غیر مستقیم می باشد قابل ذکر است در ۸۰۵۲ برای دسترسی به ۱۲۸ بایت بالای حافظه داخلی باید از آدرس دهی غیر مستقیم استفاده کرد.

```
idata char i;
i=2;
```

معادل اسمبلی دستورات مثال فوق عبارتست از:

```
MOV R0,#i
MOV @R0,#02h
```

pdata : نشاندهنده آنست که مکان متغیر در یک *page* (۲۵۶ بایت) از حافظه *RAM* خارجی می باشد.

```
pdat char i;
i=2;
```

معادل اسمبلی دستورات مثال فوق عبارتست از:

```
MOV A,=02h
MOV R0,#i
MOVX @R0,A
```

xdata : نشاندهنده آنست که مکان متغیر در ۶۴ کیلوبایت حافظه *RAM* خارجی می باشد.

```
xdata char i;
i=2;
```

معادل اسمبلی دستورات مثال فوق عبارتست از:

```
MOV DPRT,#i
MOV A,#02h
MOVX @DPTR,A
```

code : نشان دهنده آنست که عدد بصورت ثابت در حافظه *ROM* یا *EPROM* بوده و قابل تغییر نیست. از این حالت می توان برای ذخیره جداول ثابت استفاده کرد.

```
code char i=2;
data char c;
c=i;
```

معادل اسمبلی دستورات مثال فوق عبارتست از:

```
MOV DPTR,#i
CLR A
MOVC A,@A+DPTR
MOV c,A
```

۱۰ـ۷ـ انواع داده ها در مترجم *C51*

مترجم فرانکلین نوع *C51* از انواع داده های زیر پشتیبانی می کند همانطور که مشاهده می شود بعضی از این نوع داده ها در *C* استاندارد که بیان شد وجود ندارد. تعاریف آنچه که تاکنون بیان شد در هردو یکسان است.

نوع داده	تعداد بیت	تعداد بایت	محدوده مقدار
signed char	*8*	*1*	۱۲۷- تا ۱۲۸+
unsigned char	*8*	*1*	۰ تا ۲۵۵
enum	*16*	*2*	۳۲۷۶۸- تا ۳۲۷۶۷+
signed short	*16*	*2*	۳۲۷۶۸- تا ۳۲۷۶۷+
unsigned short	*16*	*2*	۰ تا ۶۵۵۳۵
signed int	*16*	*2*	۳۲۷۶۸- تا ۳۲۷۶۷+

unSigned int	*16*	*2*	۰ تا ۶۵۵۳۵
signed long	*32*	*4*	+۲۱۴۷۴۸۳۶۴۸ تا -۲۱۴۷۴۸۳۶۴۸
unsigned long	*32*	*4*	۰ تا ۴۲۹۴۹۶۷۲۹۵
float	*32*	*4*	۱/۱۷۵۴۹۴ E-۳۸ تا ۱/۱۷۵۴۹۴ E+۳۸
bit	*1*		۰ تا ۱
sbit	*1*		۰ تا ۱
sfr	*8*	*1*	۰ تا ۲۵۵
sfr16	*16*	*2*	۰ تا ۶۵۵۳۵

۱ـ۱۰ـ۷ـ داده نوع *bit*

داده نوع *bit* را می توان برای اعلان متغیرها، آرگومان ها و مقادیر برگشتی توابع استفاده کرد. اعلان یک متغیر نوع *bit* درست مانند اعلان دیگر انواع داده ها می باشد. برای مثال:

```
static bit done_flage=0;
bit test_func(bit flage1,bit flage2)
{
...
return(0);
}
```

تمامی متغیرهای نوع *bit* در یک سگمنت *BIT* در ناحیه حافظه دیتای داخلی میکروکنترلر ۸۰۵۱ ذخیره می شوند. از آنجایی که این ناحیه فقط ۱۶ بایت است، ماکزیمم تعداد متغیرهای نوع بیت قابل اعلان ۱۲۸ بیت می باشد. همچنین همراه اعلان نوع *bit* می توان نوع حافظه را مشخص کرد ولی بدلیل آنکه متغیر نوع در ناحیه دیتای داخلی ۸۰۵۱ می باشد، لذا فقط انواع حافظه *data* و *idata* را در اعلان می توان مشخص کرد و اعلان هر نوع حافظه دیگر نادرست می باشد.

متغیرها و اعلان های نوع بیت دارای محدودیت های زیر می باشد:

ـ توابعی که از پارامتر کنترلی *disable* (*#pragma disable*) استفاده می کنند و توابعی که با استفاده از تعیین بانک رجستری اعلان شده اند (*using n*)، نمی توانند مقدار بیت را برگردانند.

یک *bit* نمی تواند بعنوان یک اشاره گر اعلان شود و تعریف آرایه ای از بیتها نیز امکان پذیر نیست. بطور مثال اعلان های زیر غیر قابل قبول می باشند:

```
bit *ptr;
bit arr[5];
```

۲ـ۱۰ـ۷ـ تعیین کننده ناحیه حافظه *bdata*

تعیین کننده ناحیه حافظه *bdata* برای متغیرهایی بکار می رود که آدرس دهی آنها هم بصورت بایت و هم بصورت بیت امکان پذیر باشد. کامپایلر *C51* متغیرهایی که توسط *bdata* اعلان می شوند را در ناحیه قابل قبول آدرس دهی بیتی (*bit addressable*) حافظه داخلی میکروکنترلر ۸۰۵۱ قرار می دهد. بطور مثال این نوع متغیرها را می توان بصورت زیر اعلان کرد:

```
int bdata ibase;          /* bit addressable int */
```

```
char bdata baray[4];          /* bit addressable array */
```

در ایـن مـثال متغـیرهای *ibase* و *baray* قابل آدرس دهی بیتی می باشند. بنابراین بیت های مجزای هـریک از ایـن متغـیرها بطـور مسـتقیم قـابل دسترسـی و تغییر می باشند. برای این منظور باید از کلمه کلـیدی *sbit* اسـتفاده کـرد تـا متغیر جدیدی تعریف گردد که به بیتهای نوع *bdata* دسترسی پیدا کند. برای مثال:

```
sbit mybit0=ibase^0;  /* 0th bit of ibase */
sbit mybit15= ibase^15;  /* 15th bit of ibase */
sbit ary07=baray[0]^7;  /* 7th bit of baray[0] */
sbit ary37= baray[3]^7;  */ 7th bit of baray[3] */
```

دقـت شـود در مثال فوق، فضای جدیدی برای *mybit0*, *mybit15*, *ary07* و *ary37* اختصاص داده نمی شـود و ایـن متغـیرها در همـان فضای متغیرهای *ibase* و *baray* که ۴ بایت می باشد قرار دارند. در این مـثال عـبارت بعـد از کاراکـتر '^' مشـخص کننده محل بیت برای دسترسی در این اعلان می باشد. این عـبارت بـاید یـک مقـدار ثابـت باشـد و محدوده آن به نوع متغیر پایه بستگی دارد. محدوده ۰ تا ۷ برای متغـیرهای *char* و *unsigned char* و محـدوه ۰ تـا ۱۵ برای متغیرهای نوع *int*، *unsigned int*، *short* و *unsigned short* و محـدوده ۰ تـا ۳۱ بـرای متغـیرهای نوع *long* و *unsigned long* می باشد. همچنین بـرای دسترسـی به متغیر نوع *sbit* در ماژول های دیگر، می توان آنها را بصورت *external* نیز اعلان کرد. بطور مثال:

```
external bit mybit0;          /* 0th bit of ibase */
```

در اعلان نوع *sbit*، متغیر پایه باید با حافظه نوع *bdata* اعلان شده باشد.

براحتی می توان محتویات بیتها را تغییر داد.

```
ary37=0;       /* clear bit 7 in baray[3] */
```

بایستی دقت کرد که مجموعه فضای *bdata* از ۱۶ بایت تجاوز نکند.

۳ـ۱۰ـ۷ـ داده نوع *sfr*

sfr ها درست مانند متغیرهای دیگر *C* اعلان می شوند. تنها تفاوت آنها در آنست که نوع دیتای مشخص شده بجای *char* یا *int*، *sfr* می باشد. بعنوان مثال:

```
sfr p0=0x80;  /* port-0 , address 80h */
sfr p3=0xB0;  /* port-3 , address B0h */
```

p0 و *p3* نـام هـای اعـلان *sfr* مـی باشـد. نـامگذاری برای متغیرهای *sfr* درست مانند اعلان های دیگر متغـیرهای *C* می باشد و هر نام سمبلیک را می توان در اعلان *sfr* بکار برد. آدرس بعد از علامت '=' باید یک ثابت عددی باشد (عبارات با عملگر مورد قبول نیستند). این عبارات ثابت باید در محدوده آدرس *sfr* ها (*0x80* تا *0xFF*) قرار گیرند.

تذکـر: اعـلان هـای مـورد نـیاز بـرای *sfr* هـای هـر یک از میکروکنترلرهای خانواده ۸۰۵۱، در فایلهای *header* موجـود مـی باشـند و نـیازی بـه تعریف آنها توسط کاربر نمی باشد مگر آنکه این فایلها به برنامه ضمیمه نشده باشند.

۴ـ۱۰ـ۷ـ داده نوع *sfr16*

بسیاری از محصولات جدید خانواده ۸۰۵۱، برای مشخص کردن یک مقدار ۱۶ بیتی از دو *sfr* با آدرس پشت سرهم استفاده می کنند. برای مثال میکروکنترلر ۸۰۵۲، از آدرسهای *0xCC* و *0xCD* برای بایتهای کم ارزش و پرارزش تایمر ۲ استفاده می کنند.

دسترسی به *sfr* ها بصورت ۱۶ بیتی تنها زمانی امکان پذیر است که بایت کم ارزش درست قبل از بایت پرارزش باشد. ضمناً از آدرس کم ارزش برای اعلان *sfr16* استفاده می شود. بعنوان مثال:

```
sfr16 T2=0xCC;       /* Timer2 : T2L is 0CCh , T2H is 0CDh */
sfr16 RCAP2=0xCA;  /* RCAP2L is 0CAh , RCAP2H is 0CBh */
```

در این مثال *T2* و *RCAP2* بعنوان *sfr* های ۱۶ بیتی اعلان شده اند. اعلان *sfr16* از همان قواعد اعلان *sfr* پیروی می کند.

۱۱ـ۷ـ اشاره گرهای *C* در *8051*

چون اشاره گرها خود از نوع متغیر یا ثابت هستند، سؤالی که پیش می آید این است که یک اشاره گر کجا باید ذخیره شود، زیرا وقتی اشاره گر وارد سیستم ۸۰۵۱ می شود با نواحی مختلف حافظه (*xdata*, *pdata*, *idata*, *data*, *code*) مواجه می شود. علاوه براین، یک اشاره گر به منظور اشاره کردن به بخشهای داده ای یا متغیرهایی که خود آنها می توانند در نواحی مختلف حافظه جای بگیرند، تعریف می شود. بعبارت دیگر، هنگام تعریف کردن یک اشاره گر، نه تنها مکانی که اشاره گر در آن ذخیره خواهد شد باید تعیین گردد بلکه محل ذخیره شدن داده مورد اشاره قرار گرفته نیز باید مشخص گردد.

```
unsigned char data *xdata x_ptr=0x4a;
```

x_ptr را بعنوان یک اشاره گر که در ناحیه حافظه *xdata* مقیم می شود، تعریف می کند، این اشاره گر به یک بخش داده ای یا یک متغیر از نوع کاراکتری که در ناحیه حافظه *data* ذخیره می شود اشاره می کند. اشاره گر *x_ptr* مقدار دهی اولیه شده و مقدار *0x4a* را که آدرس یک بایت در ناحیه حافظه *data* است، نگه می دارد.

۱۲ـ۷ـ توابع در *C51*

قالب بندی تعریف یک تابع در *C51* از همان قواعد *C* استاندارد پیروی می کند. کامپایلر *C51* علاوه بر این تعداد کلمه کلیدی در تعریف تابع اضافه کرده است که توسط آنها می توان:

- یک تابع را بعنوان روتین وقفه معرفی کرد.
- بانک رجیستری مورد استفاده را مشخص کرد.
- مدل حافظه را برای تابع مورد استفاده مشخص کرد.
- یک تابع را از نوع بازگشتی معرفی کرد.
- یک تابع را از نوع *alien* (مطابق با *PL/M-51*) معرفی کرد.

در اعلان تابع می توان هریک از این مشخصه ها یا ترکیبی از آنها را مشخص کرد. قالب بندی استاندارد اعلان تابع در کامپایلر *C51* مطابق زیر است:

```
return_type func_name(args) [small | compact | large] [reentrant] [ interrupt num] [using n]
```

در این اعلان:

return_type : نوع داده برگشتی تابع را مشخص می کند و چنانچه مشخص نشود، داده برگشتی از نوع *int* در نظر گرفته می شود.

func_name : نام تابع می باشد.

args : لیست آرگومانهای تابع است.

compact, small و ***large*** : مدل حافظه بکار رفته برای تابع مزبور را مشخص می کند.

reentrant : نشان می دهد که تابع از نوع ***reentrant*** یا ***recursive*** است.

interrupt : نشان می دهد که تابع از نوع وقفه است.

using : مشخص می کند که تابع از کدام بانک رجستری استفاده می کند.

لازم بذکر است که علامت '[] ' در این اعلان نشان می دهد که استفاده از عبارت داخل این علامت اختیاری می باشد.

۱ـ۱۲ـ۷ـ تعیین مدل حافظه یک تابع

توابع کامپایلر *C51*، بطور عادی از مدل حافظه پیش فرض برای اختصاص فضای حافظه به آرگومانهای و متغیرها استفاده می کند. علاوه براین می توان مدل حافظه را برای یک تابع، با اضافه کردن کلمات کلیدی *compact*, *small* یا *large* مشخص کرد. بعنوان مثال:

```
#pragma small  /* default to small model */
extern int calc(char i,int b) large reentrant;
extern int func(int i, float f) large;
extern void *tcp(char xdata *xp,int ndx) small;
int mtest(int i,int y)
{
return(i+y+y*i+func(-1,4.75));
}
int large_func(int i,int k) large;
{
return(mtest(i,k)+2);
}
```

در این مثال از آنجایی که مدل پیش فرض *small* می باشد، لذا تابع *mtest* از مدل *small* حافظه بهره می برد و تابع *large_func* بدلیل تعیین مشخصه مدل *large* برای تابع، از این مدل استفاده می کند.

۲ـ۱۲ـ۷ـ تعیین بانک رجستری برای یک تابع

کامپایلر *C51* برای انجام محاسبات و تبادل داده ها بین توابع از رجسترهای *R0* تا *R7* استفاده می کند. از طرفی میکروکنترلرهای خانواده *MCS-51* دارای چهار بانک رجستری *R0* تا *R7* می باشند که انتخاب هریک از این بانکها از طریق رجستر *PSW* امکان پذیر است. *C51* بطور پیش فرض بانک رجستری صفر را فعال می کند. برای تغییر بانک رجستری در توابع، از کلمه کلیدی *using* استفاده می شود. ثابت ***n*** که بعد از کلمه کلیدی *using* در تعریف تابع می آید، معرف بانک رجستری مورد استفاده است و مقدار آن

مـی توانـد عـددی بین ۰ تا ۳ باشد. مثال زیر مشخص می کند که تابع *func()* از بانک رجستری ۲ برای انجام محاسبات خود استفاده می کند:

```
void func(void) using 2
{
...
}
```

تذکـر: چـنانچه در تعـریف تـابع از عـبارت *using* استفاده نشود، کامپایلر بصورت پیش فرض از هربانک رجستری که قبلاً فعال بوده است استفاده می کند.

در اسـتفاده از کلمـه کلـیدی *using* باید نهایت دقت بشود تا اجرای کد تولید شده در نواحی مختلف با بـانک هـای رجسـتری مخـتلف کاملاً کنترل شده باشد. هرگونه ایراد در این قسمت باعث نتایج اشتباه و عدم اجرای درست برنامه می شود.

کلمـه کلـیدی *using* در توابـع وقفـه (توابع همراه با کلمه کلیدی *interrupt*) بیشترین کاربرد را دارند. عموماً بـرای هـر سـطح وقفـه، یـک بانک رجستری در نظر گرفته می شود. بنابراین می توان یک بانک رجسـتری را بـرای توابـع از نـوع غـیر وقفـه، بانک رجستری دیگر را برای وقفه با اولویت بالا و یک بانک رجستری را برای وقفه با اولویت پایین در نظر گرفت.

در استفاده از توابع با کلمه کلیدی *using* محدودیتهایی وجود دارد که عبارتند از:

ـ تابع با کلمه کلیدی *using* نمی تواند داده نوع *bit* را برگرداند.

ـ پـیش الگـوی تـابع (*prototype*) تـابع نمـی توانـد شامل کلمه کلیدی *using* باشد. بنابراین در پیش الگوی تابع باید این کلمه کلیدی را حذف کرد.

۳ـ۱۲ـ۷ـ توابع *reentrant*

یک تابع *reentrant* در یک زمان می تواند بین چند فرایند، مشترک باشد. زمانیکه یک تابع *reentrant* در حال اجراست، پروسه دیگر می تواند اجرای آنرا متوقف ساخته و آنرا دوباره از ابتدا شروع کند. توابع معمولی *C* چنین خصوصیتی را ندارند زیرا برای آرگومان ها و متغیرهای محلی، از مکانهای حافظه ثابت استفاده می کنند. برای آنکه تابعی بتواند خصوصیت فراخوانی حین اجرا[1] را داشته باشد و در نتیجه بتوان از آن بعنوان تابع *recursive* بهره برد، باید از کلمه کلیدی *reentrant* استفاده کرد. بعنوان مثالی از توابع *reentrant* تابعی برای محاسبه فاکتوریل *n* در زیر آورده شده است:

```
char factorial(char n) reentrant {
if(n==1) return1;
else
return(n*factorial(n-1);
}
```

برای هر تابع *reentrant* یک ناحیه پشته *reentrant* شبیه سازی می شود این ناحیه، بسته به مدل حافظه می تواند در حافظه داخلی یا خارجی باشد.

---

[1] *_Reentrancy*

۴ـ۱۲ـ۷ـ استفاده از انتراپتها در C ۸۰۵۱

در *C51* برای پیاده سازی روالهای وقفه، از توابع استفاده می شود. این نوع از توابع با توابع معمولی دو تفاوت اساسی دارند.

۱ـ زمان صدا زدن و اجرای آنها مشخص نیست. بعبارت دیگر با آمدن وقفه این توابع فعال می شوند، لذا بایستی تابع وقفه بصورت خودکار در روال وقفه، رجسترهای ضروری را در پشته ذخیره کند.

۲ـ با توجه به نوع وقفه مشخص شده در تعریف تابع ، کامپایلر بطور خودکار تعیین می کند که این تابع به کدام بردار وقفه تعلق دارد.

۳ـ کد خروجی توابع وقفه با توابع عادی فرق دارد.

در *C51* برای آنکه تابعی از نوع وقفه شود، کافی است که از کلمه کلیدی *interrupt* در تعریف تابع استفاده کرد. ثابت *num* بعد از کلمه کلیدی *interrupt* مشخص کننده نوع وقفه می باشد که براساس نوع میکروکنترلر خانواده *MCS-51* می تواند عددی بین ۰ تا ۳۱ را دارا باشد.

جدول زیر مقدار *num* را برای وقفه های مختلف میکروکنترلر *8052* نشان می دهد.

نوع وقفه (*interrupt type*)	*num*	آدرس بردار وقفه (*vector address*)
External0	*0*	*0003h*
Timer0	*1*	*000Bh*
External1	*2*	*0013h*
Timer1	*3*	*001Bh*
Serial port	*4*	*0023h*
Timer2	*5*	*002Bh*

در مورد توابع وقفه می توان به نکات زیر اشاره کرد:

ـ در پیش الگوی توابع وقفه نمی توان از کلمه کلیدی *interrupt* استفاده کرد. بنابراین در پیش الگوی تابع باید این کلمه کلیدی را حذف کرد.

ـ تابع وقفه نمی تواند دارای آرگومان یا مقدار برگشتی باشد.

ـ کامپایلر برای تابع وقفه، تولید یک بردار وقفه می کند. کد تولید شده برای این بردار، یک پرش به ابتدای تابع وقفه می باشد.

چنانچه در تابع وقفه، محاسبات ممیز شناور (*floating point*) انجام می شود باید وضعیت رجسترهای شناور ذخیره شود. در صورتیکه در قسمتهای دیگر برنامه محاسبات ممیز شناور صورت نگیرد، ذخیره سازی رجسترهای ممیز شناور ضروری نخواهد بود. برای ذخیره سازی و بازیابی وضعیت رجیسترهای ممیز شناور می توان از توابع کتابخانه ای *fpsave* و *fprestore* استفاده کرد. برای مثال:

```
#include <math.h>
struct FPBUF intsave;
float x,y,z;
void intfunc(void) interrupt 1 using 2
{
fpsave(&intsave);
x=y*z;
fprestore(&intsave);
}
```

۱۳ـ۷ـ فایلهای سرآمد[۲]

فایلهای سرآمد حاوی معرفی پیش الگوهای (*prototype*) توابع کتابخانه ای و معرفی متغیرهای استفاده شده در آنها می باشند. در ادامه خلاصه ای از آنها آورده شده است.

۱ـ۷ـ۱۳ ***REGXX.h***

این مجموعه فایلها حاوی معرفی رجسترهای میکروکنترلرهای خانواده *MCS-51* می باشند. این فایلها عبارتند از:

میکروکنترلر	فایل	میکروکنترلر	فایل	میکروکنترلر	فایل
80515	*REG515.h*	*8052*	*REG52.h*	*8051*	*REG51.h*
80452	*REG452.h*	*80451*	*REG451.h*	*80517*	*REG517.h*
80552	*REG552.h*	*80152*	*REG152.h*	*80252*	*REG252.h*
8051F	*REG51F.h*	*8051GB*	*REG51GB.h*	*8051G*	*REG51G.H*

بعنوان نمونه یکی از اعلان های موجود در فایل *REG51.h* در زیر آورده شده است:

```
sfr P0=0x80;
```

برای اطلاع از لیست کلیه اعلان های هریک از فایلهای مذکور می توان با یک ویرایشگر، فایلهای مورد نظر که در زیر فهرست مثلاً *C:\C51\INC* می باشند را خوانده و محتویات آنها را ملاحظه نمود.

۲ـ۷ـ۱۳ ***CTYPE.h***

این فایل حاوی پیش الگوهای توابع و ماکروهای مفید برای امتحان و تبدیل کاراکترها می باشد. لیست توابع اعلان شده در این فایل به قرار زیر است:

isalpha	*isalnum*	*iscntrl*	*isdigit*	*isgraph*
isprint	*ispunct*	*islower*	*isupper*	*isspace*
isxdigit	*tolower*	*toupper*	*toint*	

۳ـ۷ـ۱۳ ***math.h***

این فایل حاوی اعلان توابع کتابخانه ای ریاضی موجود در کتابخانه *C51* می باشد. لیست این توابع به قرار زیر است:

cabs	*abs*	*labs*	*fabs*	*sqrt*	*exp*	*log*
log10	*sin*	*cos*	*tan*	*asin*	*acos*	*atan*
sinh	*cosh*	*tanh*	*ceil*	*pow*	*modf*	*atan2*
floor	*fpsave*	*fprestore*				

در فایل سرآمد *math.h* ساختار *FPBUF* برای ذخیره سازی و بازیابی وضعیت محاسبات اعشاری بصورت زیر تعریف شده است:

```
struct FPBUF {
```

---

[2] *Header file*

```
unsigned char save[16];
}
```

توابع کتابخانه ای اعشاری *C51*، برای محاسبات خود از یک فضای *RAM* ، ۱۶ بایتی استفاده می کنند و چنانچه به دلیلی نیازی به ذخیره سازی و بازیابی این فضای ۱۶ بایتی باشد، می توان با تعریف یک اشاره گر از نوع ساختار *FPBUF*، از توابع *fpsave* و *fprestore* بهره برد. پیش الگوی این توابع بصورت زیر است:

```
extern void fpsave(struct FPBUF *);
extern void fprestore(struct FPBUF *);
```

از موارد کاربرد این توابع می توان توابع وقفه را نام برد.

۴ـ۱۳ـ۷ـ stdlib.h

این فایل حاوی معرفی پیش الگوهای توابع با کاربرد عمومی زیر است:

atof	*atol*	*atoi*	*rand*	*srand*
malloc	*free*	*realloc*	*calloc*	*init_mempool*

۵ـ۱۳ـ۷ـ string.h

کتابخانه *C51* توابعی را برای ساده تر کردن کار با رشته ها را فراهم کرده است. فایل سرأمد *string.h* حاوی پیش الگوهای (*prototype*) لازم برای این توابع می باشد. لیست این توابع به قرار زیر است:

strcat	*strncat*	*strcmp*	*strncmp*	*strcpy*
strncpy	*strlen*	*strchr*	*strrchr*	*strrpos*
strspn	*strcspn*	*strpbrk*	*strrpbrk*	*memcmp*
memcpy	*memchr*	*memmove*	*memset*	

۶ـ۱۳ـ۷ـ stdio.h

کتابخانه *C51* برای آسانتر کردن کار با استاندارد ورودی/خروجی (در اینجا منظور پورت سریال می باشد) توابعی را فراهم کرده است. فایل *stdio.h* حاوی پیش الگوهای مورد نیاز برای استفاده از این توابع می باشد. فهرست این توابع در زیر آورده شده است:

getchar	*ungetchar*	*putchar*	*printf*	*sprintf*
scanf	*sscanf*	*_getkey*	*gets*	*puts*

۷ـ۱۳ـ۷ـ absacc.h

این فایل حاوی تعریف ماکروهایی می باشد که برای دسترسی به آدرس های مطلق در حافظه کاربرد دارند. این ماکروها عبارتند از:

DBYTE اشاره گری به کاراکتر در آدرس *0h* از فضای حافظه *data* می باشد.

PBYTE : اشاره گری به کاراکتر در آدرس *0h* از فضای حافظه *pdata* می باشد.

XBYTE : اشاره گری به کاراکتر در آدرس *0h* از فضای حافظه *xdata* می باشد.

CBYTE : اشاره گری به کاراکتر در آدرس *0h* از فضای حافظه *code* می باشد.

DWORD : اشاره گری به عدد صحیح در آدرس *0h* از فضای حافظه *data* می باشد.

PWORD : اشاره گری به عدد صحیح در آدرس *0h* از فضای حافظه *pdata* می باشد.

XWORD : اشاره گری به عدد صحیح در آدرس *0h* از فضای حافظه *xdata* می باشد.

CWORD : اشاره گری به عدد صحیح در آدرس *0h* از فضای حافظه *code* می باشد.

بطور مثال برای دسترسی به بایت ذخیره شده در مکان دهم حافظه *RAM* داخلی، می توان از *DBYTE[10]* استفاده کرد. یا برای نوشتن عدد *45h* در آدرس *FF00h* حافظه *RAM* خارجی می توان از دستور زیر استفاده کرد.

```
XBYTE[0xff00]=0x45;
```

و یا برای خواندن از این آدرس می توان از دستور زیر استفاده کرد.

```
c=XBYTE[0xff00];
```

فصل هشتم

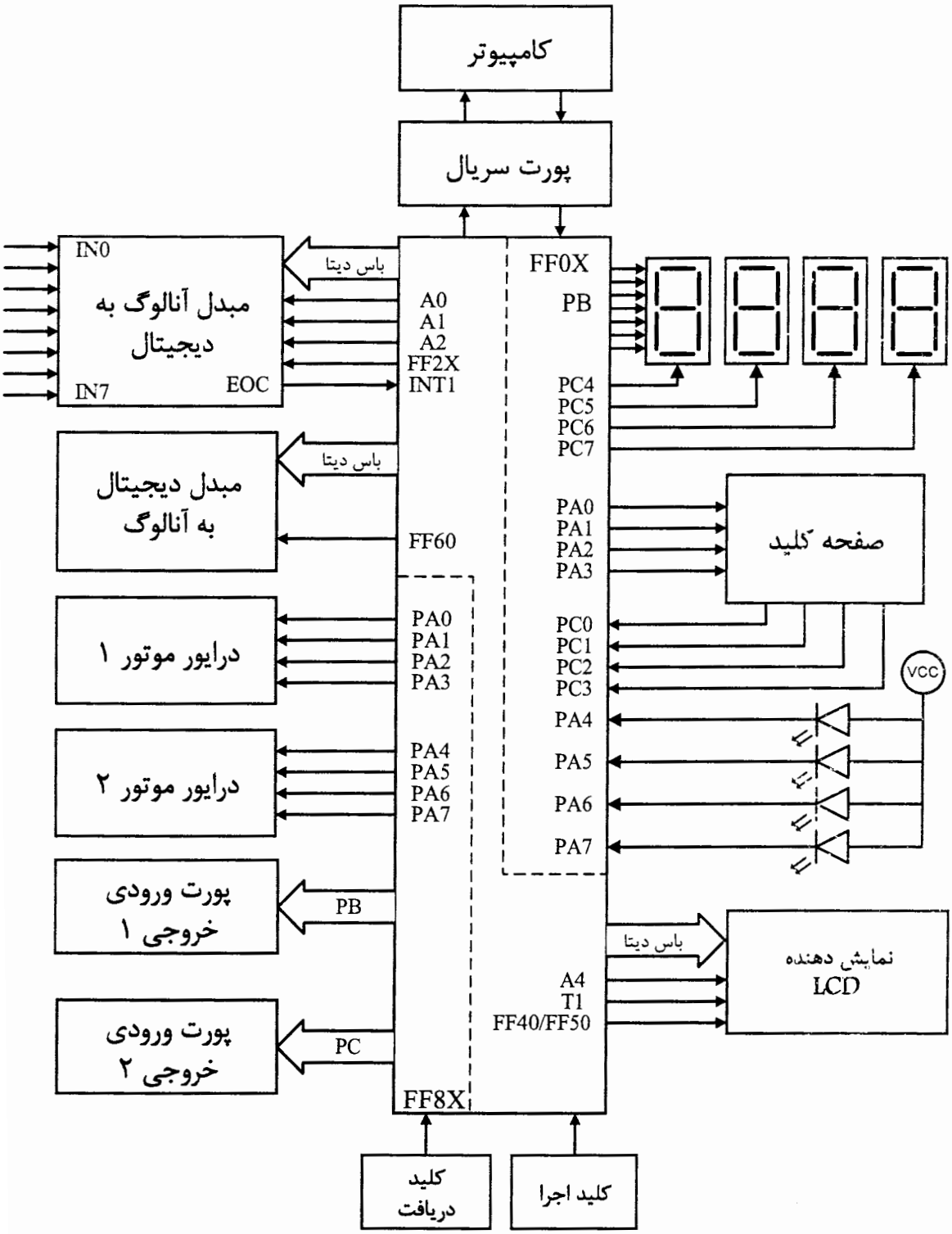
مثالهای برنامه نویسی به زبان اسمبلی و C

۸ـ۱ـ مقدمه

در ایـن فصـل جهت آموزش چگونگی استفاده از برد آموزشی و کار با قسمتهای مختلف سخت افزاری و نحـوه نوشـتن بـرنامه هـای مخـتلف مـثالهای متعددی آورده شده است و به جهت آموزش بیشتر و بیان ارتـباط بیـن اسمبی و *C* برای هر دو یک مسئله به همراه فلوچارت لازم آورده شده است. برای هریک از این مثالها، آموزشی خاص مد نظر قرار گرفته است. در این مثالها نحوه برنامه ریزی پورت ورودی خروجی *8255*، عملکـرد *7_Segment* ، *LCD*، صـفحه کلـید، موتور *DC*، موتور پله ای، مبدل آنالوگ به دیجیتال، مـبدل دیجیـتال بـه آنـالوگ، نحـوه عملکرد پورت سریال و نحوه عملکرد تایمرها در میکروکنترلر ۸۰۵۱ نشان داده شده است.

۸ـ۲ـ بلوک دیاگرام سیستم

در شکل ۸ـ۱ ارتباط هسته سخت افزار با پورتهای ورودی خروجی در سیستم آموزشی بصورت بلوک دیاگرام نشان داده شده است. در این شکل هسته سخت افزاری شامل میکروکنترلر، حافظه ها، دیکدر و پورتهای ورودی خروجی بوده و آدرس هر قسمت ورودی و خروجی نشان داده شده است. در جدول ۸ـ۱ آدرس هر قسمت نشان داده شده است. بعنوان مثال آدرس پایه برای پورت ورودی خروجی ۸۲۵۵ که *7_segment* ها، صفحه کلید، *LED* ها به آن متصل شده اند *FF0X* می باشد که با توجه به پورت مورد استفاده مقدار *X* تعیین می گردد.



شکل ۱ـ۸ ـ بلوک دیاگرام کلی سیستم آموزشی

جدول ۱ـ۸ـ آدرس پورتهای ورودی خروجی سیستم

توضیح	سخت افزار مورد استفاده	آدرس
پورت A ۸۲۵۵	۴ بیت کم ارزش ردیف های صفحه کلید و ۴ بیت پرارزش LED ها	$FF00h$
پورت B ۸۲۵۵	اتصال LED های $7\_segment$ ها	$FF01h$
پورت C ۸۲۵۵	۴ بیت کم ارزش ستونهای صفحه کلید و ۴ بیت پرارزش فعال و غیر فعال سازی $7\_segment$ ها	$FF02h$
کلمه کنترلی ۸۲۵۵	---	$FF03h$
---	کانال صفر مبدل A/D	$FF20h$
---	کانال یک مبدل A/D	$FF21h$
---	کانال دو مبدل A/D	$FF22h$
---	کانال سه مبدل A/D	$FF23h$
فتوسل	کانال چهار مبدل A/D	$FF24h$
ترمیستور	کانال پنجم مبدل A/D	$FF25h$
اندازه گیری جریان	کانال ششم مبدل A/D	$FF26h$
اندازه گیری جریان	کانال هفتم مبدل A/D	$FF27h$
---	ارسال فرمات برای LCD	$FF40h$
---	ارسال کاراکتر برای LCD	$FF50h$
---	ارسال داده برای مبدل دیجیتال به آنالوگ	$FF60h$
پورت A ۸۲۵۵	چهار بیت پر ارزش برای کنترل $L298$ دوم	$FF80h$
پورت B ۸۲۵۵	پورت ورودی خروجی قابل استفاده	$FF81h$
پورت C ۸۲۵۵	پورت ورودی خروجی قابل استفاده	$FF82h$
کلمه کنترلی ۸۲۵۵	---	$FF83h$

در ادامه نحوه برنامه ریزی و استفاده از قسمتهای مختلف سخت افزار آورده شده است.

۳ـ۸ـ فایلهای سرآمد[1]

هدر فایلهایی که به مبین آدرس پورتهای ورودی خروجی هستند و به ابتدای برنامه اسمبلی یا C ضمیمه می گردد در زیر آورده شده است (البته حتماً لازم نیست این فایلها به برنامه ضمیمه گردند بلکه می توان از آدرس های مشخص شده در اصل برنامه نیز استفاده کرد). منظور از $ONBORD$ پورت ورودی خروجی ۸۲۵۵ است که به $7\_segment$ ها، صفحه کلید و LED ها وصل شده است و منظور از

[1] $\_Header\ File$

OUTBORD پورت ورودی خروجی ۸۲۵۵ است که به موتورهای پله ای و *DC* و پین های ورودی خروجی اضافی وصل می شود.

```
PA_ONBORD   EQU   0FF00h
PB_ONBORD   EQU   0FF01h
PC_ONBORD   EQU   0FF02h
CN_ONBORD   EQU   0FF03h
ATODC0      EQU   0FF20h
ATODC1      EQU   0FF21h
ATODC2      EQU   0FF22h
ATODC3      EQU   0FF23h
ATODC4      EQU   0FF24h
ATODC5      EQU   0FF25h
ATODC6      EQU   0FF26h
ATODC7      EQU    0FF27h
LCD_COMMAND EQU 0FF40h
LCD_DATA    EQU    0FF50h

DTOAC       EQU    0FF60h

PA_OUTBORD EQU    0FF80h
PB_OUTBORD EQU    0FF81h
PC_OUTBORD EQU   0FF82h
CN_OUTBORD EQU   0FF83h
```

هدر فایل ضمیمه شده به برنامه های اسمبلی

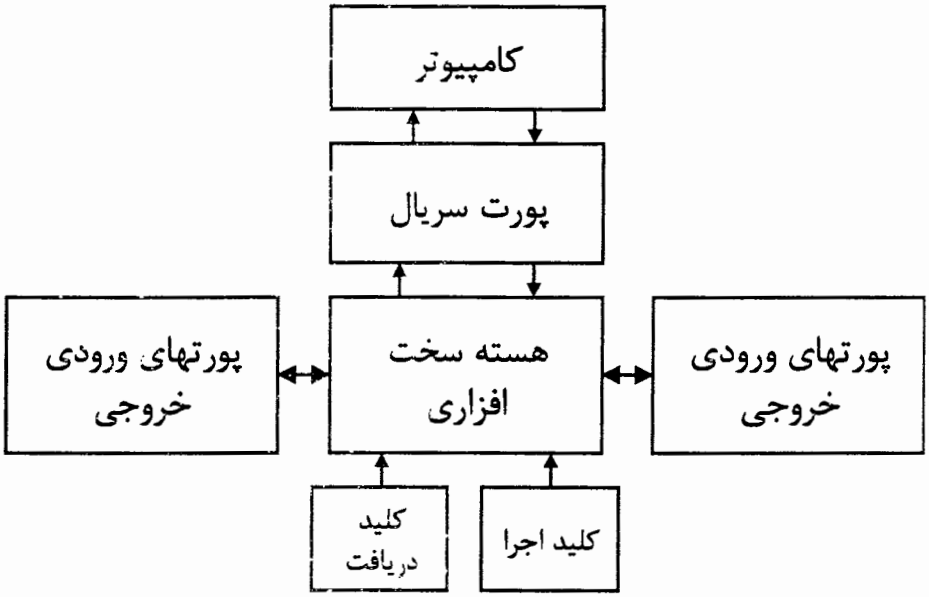
```
#define PB_ONBORD   0xff01
#define PC_ONBORD   0xff02
#define CN_ONBORD   0xff03
#define ATODC0      0xff20
#define ATODC1      0xff21
#define ATODC2      0xff22
#define ATODC3      0xff23
#define ATODC4      0xff24
#define ATODC5      0xff25
#define ATODC6      0xff26
#define ATODC7      0xff27
#define LCD_COMMAND 0xff40
#define LCD_DATA    0xff50

#define DTOAC       0xff60

#define PA_OUTBORD 0xff80
#define PB_OUTBORD 0xff81
#define PC_OUTBORD 0xff82
#define CN_OUTBORD 0xff83
```

فایل سرآمد ضمیمه شده به برنامه های *C*

علاوه بر این فایل سرآمد فایلهای سرآمد *reg51.h* یا *reg52.h*، *math.h* و *absacc.h* نیز به برنامه های زیر ضمیمه می گردند و مورد استفاده قرار می گیرند.

۸ـ۴ـ پورت سریال

همانطور که در شکل ۸ـ۲ نشان داده شده است پورت سریال سیستم آموزشی از یک طرف به کامپیوتر و از طرف دیگر با هسته سخت افزاری در ارتباط است. از پورت سریال به دو منظور استفاده شده است. در حالت اول از آن به منظور دریافت فایل و ذخیره آن و در حالت دوم که زمان اجرای برنامه می باشد وسیله ای جهت ارتباط با کامپیوتر و ارسال و دریافت کاراکترهایی که در برنامه پیش بینی شده است می باشد. در این حالت می توان از پورت سریال بمنظور عیب یابی برنامه ها که بعداً تشریح می گردد استفاده کرد. لذا به جهت اهمیت پورت سریال و بکار گیری آن در بقیه آزمایشها ابتدا نحوه برنامه ریزی و استفاده از آن بیان می گردد.



شکل ۸ـ۲ ـ بلوک دیاگرام ارتباط پورت سریال با سخت افزار

۸ـ۴ـ۱ـ برنامه ریزی پورت سریال

همانطور که قبلاً بیان شد پورت سریال ۸۰۵۱ چند مد کاری دارد. مد کاری یک آن که بصورت سریال آسنکرون ۸ بیتی می باشد در مثالهای زیر مورد استفاده قرار گرفته است. نحوه برنامه ریزی آن در برنامه های زیر بصورت اسمبلی و C آورده شده است.

```
MOV SCON,#50h          ; setup serial port
MOV TMOD,#20h          ; Used  Timer 1
MOV TH1,#0E8h          ; Baud Rate 1200 bit/s  11.0592 MHZ
SETB TR1
```

زیر برنامه برنامه ریزی پورت سریال در ۸۰۵۱ بصورت اسمبلی

```
SCON=0x50;          /* setup serial port */
TMOD=0x20;          /*      Used   Timer 1 */
TH1=0xE8;           /* Baud Rate 1200 bit/s  11.0592 MHZ*/
TR1=1;
```

زیر برنامه برنامه ریزی پورت سریال در ۸۰۵۱ بصورت C

۲ـ۴ـ۸ـ برنامه ریزی پورت سریال در ۸۰۵۲

در میکروکنترلر ۸۰۵۲ علاوه بر استفاده از تایمر یک می توان از تایمر ۲ نیز جهت ارتباط سریال بصورت آسنکرون کمک گرفت. در این حالت بایستی رجسترهایی را با اعدادی که از فرمول زیر استخراج می گردند بارگذاری نمود.

$$RCAP2H, RCAP2L = 65536 - \frac{\text{Oscillator Frequency}}{32 * \text{Baud rate}}$$

نحوه برنامه ریزی آن در برنامه زیر بصورت اسمبلی و *C* آورده شده است.

```
MOV SCON,#50h          ; setup serial port
MOV T2CON,#34h         ; Used  Timer 2
MOV RCAP2H,#0FEh       ; Baud Rate 1200 bit/s 11.0592 MHZ
MOV RCAP2L,#0E0h
```

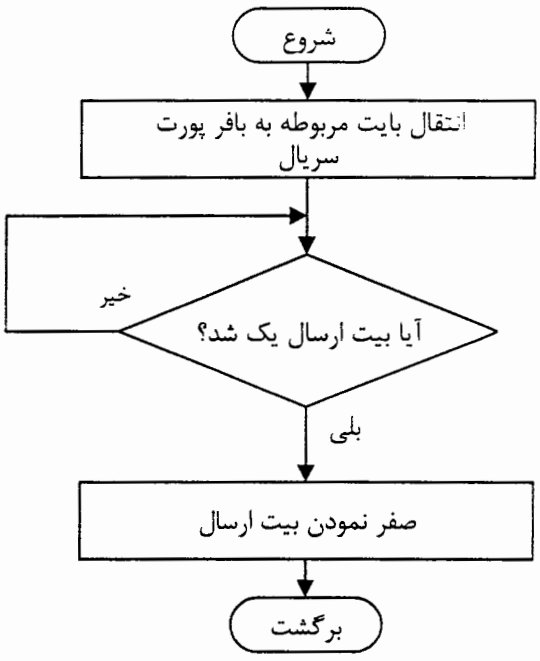
زیر برنامه برنامه ریزی پورت سریال در ۸۰۵۲ بصورت اسمبلی

```
SCON=0x050;        /* setup serial port */
T2CON=0x34;        /*      Used    Timer 2 */
RCAP2H=0xFE;       /* Baud Rate 1200 bit/s 11.0592 MHZ */
RCAP2L=0xE0;
```

زیر برنامه برنامه ریزی پورت سریال در ۸۰۵۲ بصورت *C*

۳ـ۴ـ۸ـ ارسال یک کاراکتر

فلوچارت شکل ۳ـ۸ نحوه ارسال کاراکتری را که دریافت می کند نشان می دهد. برنامه نوشته شده با اسمبلی و *C* نیز آورده شده است.



شکل ۳ـ۸ ـ فلوچارت نحوه ارسال یک کاراکتر

```
SEND_CHAR:
        MOV         SBUF,A
SEND1:
        JNB         TI,SEND1
        CLR         TI
        RET
```

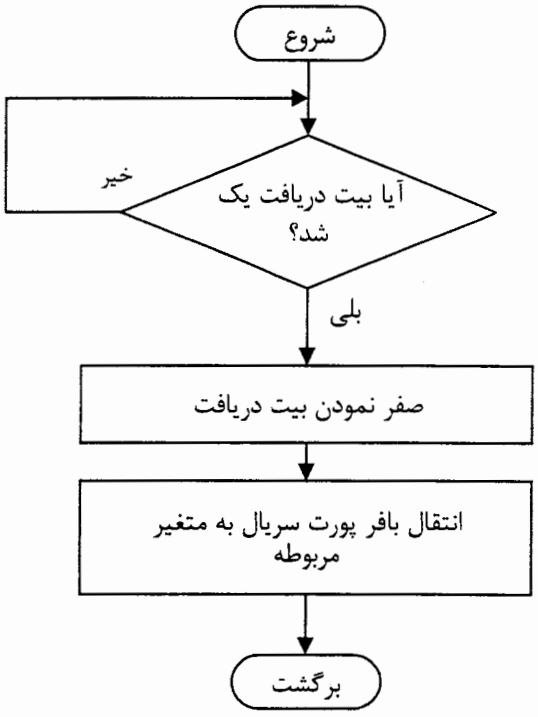
ارسال یک کاراکتر بصورت اسمبلی

```
void    send_char(char c)
{
        SBUF=c;
        while(TI==0);
        TI=0;
}
```

ارسال یک کاراکتر با *C*

۴ـ۴ـ۸ـ دریافت یک کاراکتر

شکل ۴ـ۸ نحوه دریافت یک کاراکتر را از پورت سریال بصورت سرکشی نشان می دهد.



شکل ۴ـ۸ ـ فلوچارت نحوه دریافت یک کاراکتر

```
READ_CHAR:
LOOP:
        JNB         RI,LOOP
        CLR         RI
        MOV         A,SBUF
        RET
```

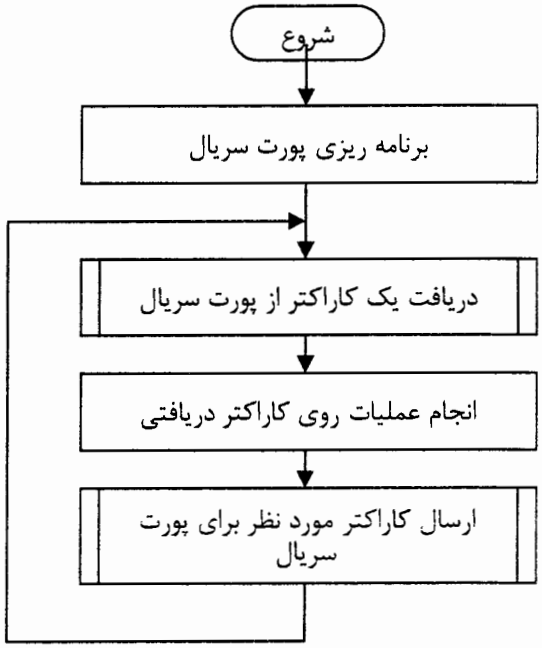
دریافت یک کاراکتر بصورت اسمبلی

```
char    read_char()
        {
        while(RI==0);
                RI=0;
        return SBUF;
        }
```

دریافت یک کاراکتر با *C*

۵ـ۴ـ۸ـ دریافت و ارسال یک کاراکتر بصورت سرکشی

در فلوچارت شکل ۵ـ۸ نحوه دریافت و ارسال یک کاراکتر توسط پورت سریال نشان داده شده است. در این فلوچارت از زیر برنامه های قبلی استفاده شده است. نحوه پیاده سازی این برنامه بصورت اسمبلی و *C* در ادامه آورده شده است. در این برنامه ها جهت سادگی فرض شده است به بایت دریافتی یک واحد اضافه گردد.



شکل ۵ـ۸ ـ فلوچارت نحوه دریافت و ارسال یک کاراکتر بصورت سرکشی

```
$INCLUDE (AIO_ADD.h)

        org 000h
        LJMP   START
        org 100h
START:
        MOV    SCON,#50h     ;setup serial port
        MOV    TMOD,#20h     ;Used   Timer 1
        MOV    TH1,#0E8h     ; Baud rate 1200 bit/s 11.0592 MHZ
        SETB   TR1
LOOP:
        LCALL READ_CHAR
        INC    A
        LCALL SEND_CHAR
        LJMP   LOOP
```

```
SEND_CHAR:
        MOV   SBUF,A
SEND1:
        JNB   TI,SEND1
        CLR   TI
        RET
READ_CHAR:
LOOP2:
        JNB   RI,LOOP2
        CLR   RI
        MOV   A,SBUF
        RET

        END
```

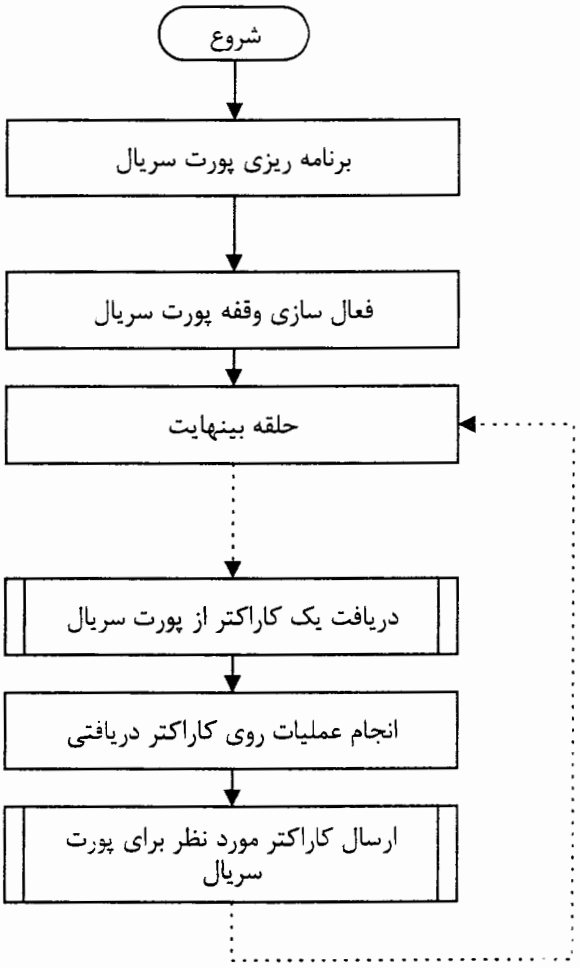
دریافت و ارسال یک کاراکتر با اسمبلی بصورت سرکشی

```
#include      <REG52.H>
#include      <absacc.h>
#include      <CIO_ADD.h>
void    send_char(char);
char  read_char();
void main ()
{
char    ch;
        SCON=0x50;          /* setup serial port */
        TMOD=0x20;          /*      Used    Timer 1 */
        TH1=0xE8;           /* 1200 */
        TR1=1;
        for(;;)
                {
                ch=read_char();
                ch=ch+1;        //for example
                send_char(ch);
                }
}
void    send_char(char c)
{
        SBUF=c;
        while(TI==0);
        TI=0;
}
char    read_char()
        {
        while(RI==0);
                RI=0;
        return SBUF;
        }
```

دریافت و ارسال یک کاراکتر با C بصورت سرکشی

۶ـ۴ـ۸ـ دریافت و ارسال یک کاراکتر بصورت اینتراپتی

فلوچارت نحوه دریافت و ارسال یک کاراکتر بصورت اینتراپتی در شکل ۶ـ۸ نشان داده شده است. در این حالت پس از برنامه ریزی پورت سریال بصورت اینتراپتی، برنامه در یک حلقه بینهایت قرار گرفته، هرگاه یک بایت از طریق پورت سریال دریافت شد اینتراپت مربوطه فعال می گردد کاراکتر مربوطه خوانده شده، عملیات لازم روی آن که افزایش به اندازه یک واحد می باشد انجام، و سپس کاراکتر بدست آمده برای پورت سریال ارسال می گردد.



شکل ۶ـ۸ ـ فلوچارت نحوه دریافت و ارسال یک کاراکتر بصورت انتراپتی

```
$INCLUDE (AIO_ADD.h)
        org 000h
        LJMP  START
        org     23h
        CLR     RI
        LCALL READ_CHAR
        RETI
        org 100h
START:
        MOV   SCON,#50h      ;setup serial port
        MOV   TMOD,#20h      ;Used   Timer 1
        MOV   TH1,#0E8h      ; Baud rate 1200 bit/s 11.0592 MHZ
        SETB  ES
        SETB  EA
```

```
        SETB   TR1
LOOP:
        LJMP   LOOP
SEND_CHAR:
        MOV    SBUF,A
SEND1:
        JNB    TI,SEND1
        CLR    TI
        RET
READ_CHAR:
        MOV    A,SBUF
        INC    A
        LCALL SEND_CHAR
        RET
        END
```

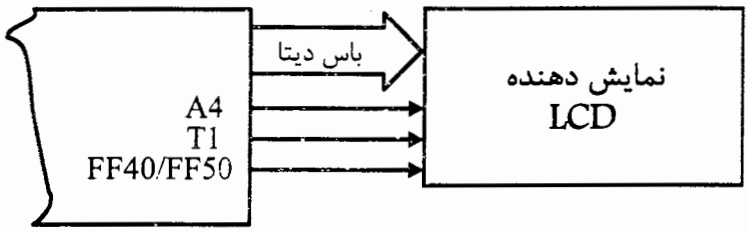
دریافت و ارسال یک کاراکتر با اسمبلی بصورت انتراپتی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    send_char(char);
void    read_char();
serial_port() interrupt 4 using 0
{
        RI=0;
        read_char();
}
char    ch;
void main ()
{
        SCON=0x50;              /* setup serial port */
        TMOD=0x20;              /*      Used    Timer 1 */
        TH1=0xE8;               /* 1200 bit/s */
        TR1=1;
        ES=1;
        EA=1;
        for(;;);
}
void    send_char(char c)
{
        SBUF=c;
        while(TI==0);
        TI=0;
}
void    read_char()
{
        ch=SBUF;
        ch=ch+1;        // for example
        send_char(ch);
        return;
}
```

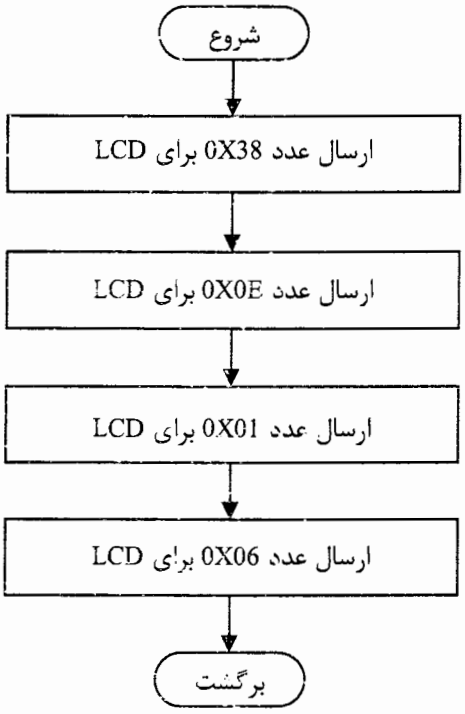
دریافت و ارسال یک کاراکتر با *C* بصورت انتراپتی

۸ـ۵ *LCD*

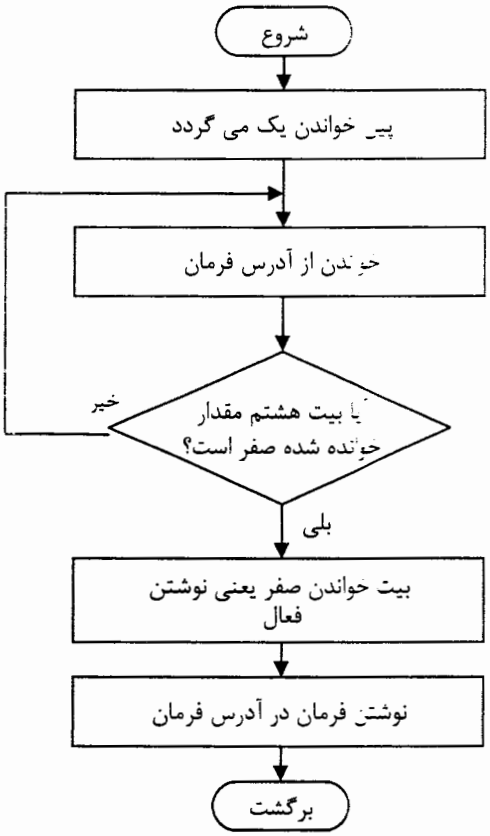
شکل ۷_۸ ارتباط بین هسته سخت افزاری و *LCD* را با جزئیات لازم نشان می دهد و فلوچارت شکل ۸ ـ۸ نحوه برنامه ریزی *LCD* را که در برگه مشخصات *LCD* ها آورده شده است نشان می دهد. در *LCD* ها پس از اعمال هر فرمان مدتی بایستی صبر کرد و سپس فرمان بعدی را فرستاد. این تأخیر را می توان در برنامه گنجاند و یا اطلاعات لازم را از *LCD* خوانده و هرگاه آماده بود فرمان بعدی را اعمال کرد. در برنامه های زیر از روش دوم استفاده شده است. همانطور که در شکل نشان داده شده است برای کنترل *LCD* از پین *T1* میکروکنترلر استفاده شده است که بایستی بصورت مناسب فعال و غیر فعال گردد. لازم بذکر است برنامه ریزی *LCD* ها بصورت مشخص بوده و فرق زیادی با یکدیگر نمی کنند.



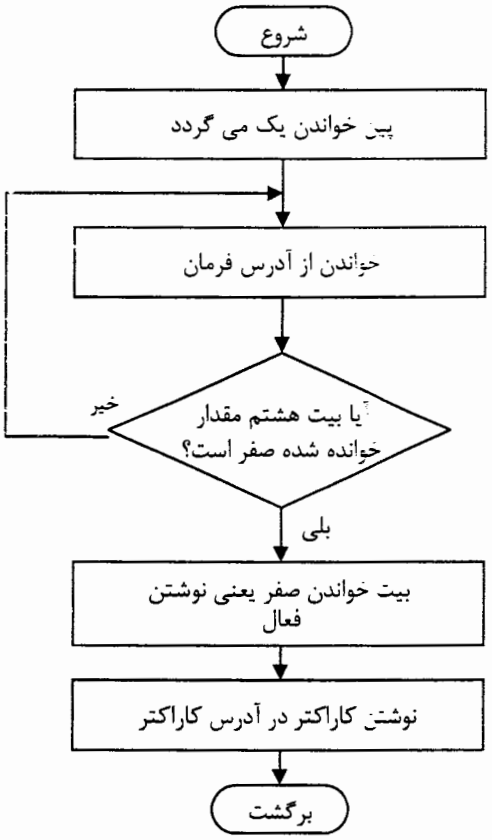
شکل ۷ـ۸ ـ *LCD* و ارتباط آن با سخت افزار



شکل ۸ـ۸ ـ فلوچارت برنامه ریزی *LCD*



شکل ۹ـ۸ ـ فلوچارت ارسال فرمان برای *LCD*



شکل ۱۰ـ۸ ـ فلوچارت ارسال کاراکتر برای *LCD*

```
INIT_LCD:
        MOV   A,#38h
        LCALL WRITE_COMM
        MOV   A,#0Eh
        LCALL WRITE_COMM
        MOV   A,#01h
        LCALL WRITE_COMM
        MOV   A,#06h
        LCALL WRITE_COMM
        RET
```

برنامه ریزی *LCD* با اسمبلی

```
WRITE_COMM:
        MOV   R0,A
        SETB  T1
LOC1:
        MOV   DPTR,#LCD_COMMAND
        MOVX  A,@DPTR
        ANL   A,#80h
        CJNE  A,#0h,LOC1
        CLR   T1
        MOV   DPTR,#LCD_COMMAND
        MOV   A,R0
        MOVX  @DPTR,A
        RET
```

ارسال فرمان برای *LCD* با اسمبلی

```
WRITE_CHAR:
        MOV   R0,A
        SETB  T1
LOC2:
        MOV   DPTR,#LCD_COMMAND
        MOVX  A,@DPTR
        ANL   A,#80h
        CJNE  A,#0h,LOC2
        CLR   T1
        MOV   DPTR,#LCD_DATA
        MOV   A,R0
        MOVX  @DPTR,A
        RET
```

ارسال یک کاراکتر برای *LCD* با اسمبلی

```
void    init_lcd()
{
        write_comm(0x38);
        write_comm(0x0E);
        write_comm(0x01);
        write_comm(0x06);
        return;
}
```

برنامه ریزی *LCD* با *C*

```
void    write_comm(char cx)
{
char    c;
        T1=1;
        for(;;)
        {
        c=XBYTE[LCD_COMMAND];
        if((c & 0x80) == 0 )
                break;
        }
        T1=0;
        XBYTE[LCD_COMMAND]=cx;
        return;
}
```

ارسال فرمان برای *LCD* با *C*

```
void    write_char(char cx)
{
char    c;
        T1=1;
        for(;;)
        {
        c=XBYTE[LCD_COMMAND];
        if((c & 0x80) == 0 )
                break;
        }
        T1=0;
        XBYTE[LCD_DATA]=cx;
        return;
}
```

ارسال یک کاراکتر برای *LCD* با *C*

مثال: برنامه ای بنویسید که دو عبارت "*HELLO*" و "*SYSTEM IS OK* " را روی *LCD* نمایش دهد.

```
$INCLUDE (AIO_ADD.h)
        org 000h
        LJMP   START
        org 100h
START:
        MOV    SP,#60h
        LCALL INIT_LCD
        MOV    A,#86h
        MOV    DPTR,#STR1
        LCALL LCD_PRINT
        MOV    A,#0C0h
        MOV    DPTR,#STR2
        LCALL LCD_PRINT
        LJMP   $
INIT_LCD:
        MOV    A,#38h
        LCALL WRITE_COMM
        MOV    A,#0Eh
        LCALL WRITE_COMM
```

```
        MOV   A,#01h
        LCALL WRITE_COMM
        MOV   A,#06h
        LCALL WRITE_COMM
        RET
WRITE_COMM:
        MOV   R0,A
        SETB  T1
LOC1:
        MOV   DPTR,#LCD_COMMAND
        MOVX  A,@DPTR
        ANL   A,#80h
        CJNE  A,#0h,LOC1
        CLR   T1
        MOV   DPTR,#LCD_COMMAND
        MOV   A,R0
        MOVX  @DPTR,A
        RET
WRITE_CHAR:
        MOV   R0,A
        SETB  T1
LOC2:
        MOV   DPTR,#LCD_COMMAND
        MOVX  A,@DPTR
        ANL   A,#80h
        CJNE  A,#0h,LOC2
        CLR   T1
        MOV   DPTR,#LCD_DATA
        MOV   A,R0
        MOVX  @DPTR,A
        RET
LCD_PRINT:
        PUSH  DPH
        PUSH  DPL
        LCALL WRITE_COMM
        POP   DPL
        POP   DPH
LCD0:   MOV   A,#0
        MOVC  A,@A+DPTR
        CJNE  A,#'$',LCD1
        RET
LCD1:   PUSH  DPH
        PUSH  DPL
        LCALL WRITE_CHAR
        POP   DPL
        POP   DPH
        INC   DPTR
        LJMP  LCD0
STR1:
        DB    'HELLO$'
STR2:
        DB    '  SYSTEM IS OK  $'
        END
```

نوشتن پیغام روی *LCD* با اسمبلی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    lcd_print(char cr,char *s);
void    init_lcd();
void    write_comm(char);
void    write_char(char);
void main ()
{
        init_lcd();
        lcd_print(0x86,"HELLO");
        lcd_print(0xC0,"  SYSTEM IS OK  ");
        for(;;);
}
void    init_lcd()
{
        write_comm(0x38);
        write_comm(0x0E);
        write_comm(0x01);
        write_comm(0x06);
        return;
}
void    write_comm(char cx)
{       char    c;
        T1=1;
        for(;;)
        {
        c=XBYTE[LCD_COMMAND];
        if((c & 0x80) == 0 )
                break;
        }
        T1=0;
        XBYTE[LCD_COMMAND]=cx;
        return;
}
void    write_char(char cx)
{       char    c;
        T1=1;
        for(;;)
        {
        c=XBYTE[LCD_COMMAND];
        if((c & 0x80) == 0 )
                break;
        }
        T1=0;
        XBYTE[LCD_DATA]=cx;
        return;         }
void    lcd_print(char cr,char *s)
{       char    tmp;
        write_comm(cr);
        for(tmp=0;s[tmp]!=0;tmp++)
                write_char(s[tmp]);
        return;         }
```

نوشتن پیغام روی *LCD* با *C*

۸ـ۶ تایمرها

میکروکنترلر ۸۰۵۱ دارای دو عدد تایمر یا کانتر و میکروکنترلر ۸۰۵۲ سه عدد تایمر یا کانتر دارند. از آنجایی که میکروکنترلر انتخابی برای هسته سخت افزاری ۸۰۵۲ پیش بینی شده است و از تایمر دو آن می توان جهت ارتباط با پورت سریال استفاده کرد، لذا در برنامه های کاربردی می توان تایمر صفر و یک را در اختیار داشت و از آنها برای مقاصد مورد نظر استفاده کرد. برنامه ریزی و استفاده از این تایمرها ساده است. آنچه که در این بین اهمیت دارد تعیین حالت شمارش و مقدار دهی لازم به رجسترهای مربوطه است که در زیر آورده شده است. همانطور که قبلاً اشاره شد با توجه به فرکانس کریستال که تقسیم بر ۱۲ شده است تعداد کلاک های لازم جهت تعیین زمان مورد نظر محاسبه می گردد بعنوان مثال اگر کریستال مورد استفاده ۱۱/۰۵۹۲ مگاهرتز باشد کلاکی با فرکانس ۹۲۱۶۰۰ هرتز به شمارنده اعمال می گردد در نتیجه با هر کلاک ۱/۰۸۵۰۶۹ میکروثانیه سپری می گردد و برای یک میلی ثانیه بایستی ۹۲۱ (*399h*) کلاک به شمارنده اعمال گردد در نتیجه بایستی عدد *FFFFh-399h=FC66h* به رجسترها *TH0*و *TL0* و یا *TH1* و *TL1* اعمال گردد.

```
/* setup timer0 */
TMOD=0x1;
TH0=0xFC;     /* 1.00043 ms */
TL0=0x65;
TR0=1;
ET0=1;
EA=1;
```

برنامه ریزی تایمر صفر بصورت ۱۶ بیتی جهت انتراپتهای یک میلی ثانیه ای

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    timer0();
void    init();
void    send_char(char);

timer0_trn() interrupt 1 using 1
{
        TR0=0;
        timer0();
}
int             TIMER;
bit             TIME_1SEC;
void main ()
{
        init_8255();
        init();
        TIMER=0;
        TIME_1SEC=0;
        for(;;)
                {
                if( TIME_1SEC == 1 )
                        {
```

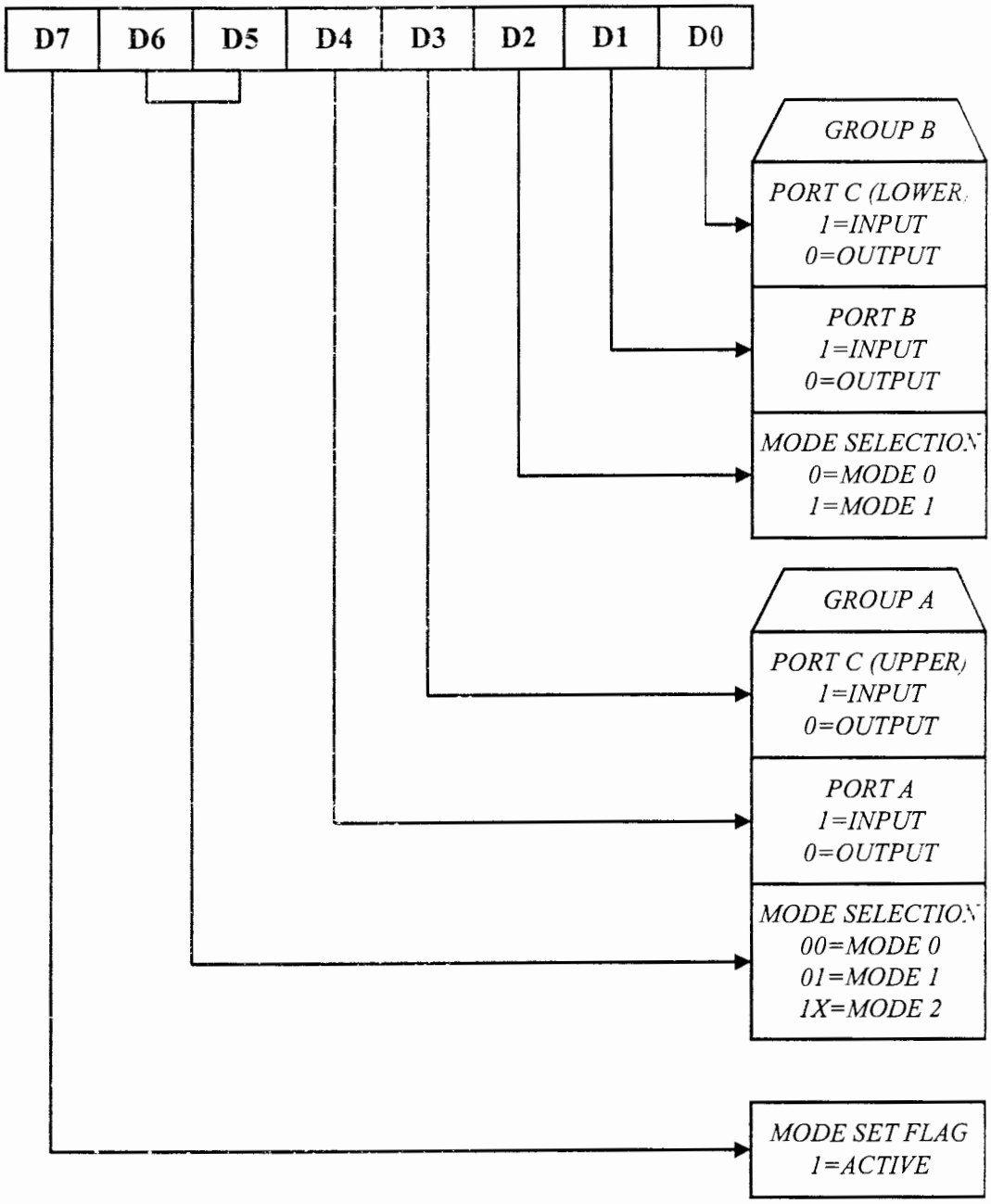
```
                        TIME_1SEC=0;
                        send_char('U');
                        }
                }
}
void    init_8255()
{
        XBYTE[CN_ONBORD]=0x81;
        return;
}
void    timer0()
{
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        TIMER++;
        if( TIMER >= 1000 ) // one secend
                {
                TIMER=0;
                TIME_1SEC=1;
                }
        return;
}
void    init()
{
        /* setup timer0 */
        TMOD=0x1;
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        ET0=1;
        SCON=0x50;              /* setup serial port */
        TMOD=0x20;              /*       Used    Timer 1 */
        TH1=0xE8;               /* Baud Rate 1200 bit/s  11.0592 MHZ*/
        TR1=1;
        EA=1;
}
void    send_char(char c)
{
        SBUF=c;
        while(TI==0);
        TI=0;
}
```

ارسال کاراکتر U برای پورت سریال هر ثانیه یک بار

۸ـ۷ـ برنامه ریزی پورت ورودی خروجی 8255

پورت ورودی خروجی ۸۲۵۵ کاربردهای متنوعی دارد. همانطور که قبلاً نشان داده شد این آی سی دارای ۲۴ پین ورودی خروجی بوده که بنامهای پورت A (PA)، پورت B (PB) و پورت C (PC) که هریک ۸ پین ورودی خروجی دارند شناخته می شود اینکه کاربر با کدامیک از این پورتها کار دارد خطوط آدرس $A0$ و $A1$ تعیین می کنند و اینکه هرپورت چه وظیفه ای را انجام دهد برنامه ریزی کلمه کنترلی

مشخص می کند. اگر تغییر عمکرد این پورتها مد نظر نباشد تنها کافی است یکبار عمل برنامه ریزی انجام گیرد. ولی اگر قرار باشد عملکرد آنها تغییر کند بایستی برنامه ریزی را تغییر داد. برای استفاده بصورت ورودی خروجی ساده حالت عملکرد مد صفر انتخاب می شود.



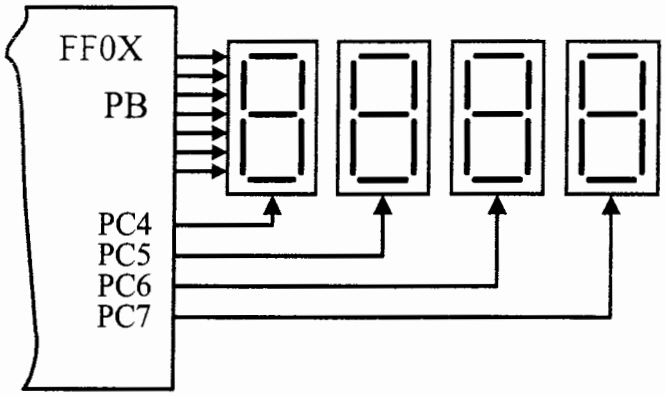
۱۱ـ۸ـ کلمه کنترلی در ۸۲۵۵

مثال: ۸۲۵۵ را بنحوی برنامه ریزی کنید که پورت A و قسمت پایین پورت C بعنوان ورودی و پورت B و قسمت بالای پورت C بعنوان خروجی برنامه ریزی گردد.

با توجه به کلمه کنترلی عددی که باید در این رجستر قرار گیرد $A1h$ می باشد.

۸ـ۸ *7_Segment* یا *LED* هفت قسمتی

همانطور که در شکل ۱۲ـ۸ نشان داده شده است سگمنتهای a تا g ، همه *7_segment* ها به ترتیب به پینهای پورت B ۸۲۵۵ با آدرس پایه *FF0X* وصل شده است. با توجه به این شکل می توان جدول ۲ـ۸ را تشکیل و کد اعداد *0h* الی *Fh* را استخراج کرد.



شکل ۱۲ـ۸ـ *7_Segment* ها و ارتباط آن با سخت افزار

جدول ۲ـ۸ـ کد های *7_segment* اعداد *0h* الی *Fh*

	PB7	*PB6*	*PB5*	*PB4*	*PB3*	*PB2*	*PB1*	*PB0*	
عدد	*dp*	*g*	*f*	*e*	*d*	*c*	*b*	*a*	کد
0h	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*3Fh*
1h	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*06h*
2h	*0*	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*5Bh*
3h	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	*4Fh*
4h	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*66h*
5h	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*1*	*6Dh*
6h	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*	*7Dh*
7h	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*07h*
8h	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*7Fh*
9h	*0*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*6Fh*
Ah	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*77h*
Bh	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*7Ch*
Ch	*0*	*0*	*1*	*1*	*1*	*0*	*0*	*1*	*39h*
Dh	*0*	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*5Eh*
Eh	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*1*	*79h*
Fh	*0*	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*71h*

کد این اعداد بصورت اسمبلی و C بصورت زیر ذخیره می شود.

```
CODE_7SEG:
        DB      3fh,06h,5bh,4fh,66h,6dh,7dh,07h
        DB      7fh,6fh,77h,7Ch,39h,5Eh,79h,71h
```

نحوه ذخیره کد ها در اسمبلی

```
char    code  CODE_7SEG[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
                              0x7f,0x6f,0x77,0x7C,0x39,0x5E,0x79,0x71};
```

نحوه ذخیره کد ها در C

```
$INCLUDE (AIO_ADD.h)

        org 000h
        LJMP   START
        org 100h
START:
        MOV    SP,#60h
        MOV    DPTR,#CN_ONBORD
        MOV    A,#81h
        MOVX   @DPTR,A
        MOV    R2,#0
LOC1:
        MOV    DPTR,#CODE_7SEG
        MOV    A,R2
        MOVC   A,@A+DPTR
        MOV    DPTR,#PB_ONBORD
        MOVX   @DPTR,A
        MOV    DPTR,#PC_ONBORD
        MOV    A,#80h
        MOVX   @DPTR,A
        LCALL  DELAY
        LCALL  DELAY
        INC    R2
        CJNE   R2,#16d,LOC1
        MOV    R2,#0
        LJMP   LOC1
DELAY:
        MOV    R0,#0FFh
DELAY1:
        MOV    R1,#0FFh
DELAY2:
        DEC    R1
        CJNE   R1,#0,DELAY2
        DEC    R0
        CJNE   R0,#0,DELAY1
        RET

CODE_7SEG:
        DB     3fh,06h,5bh,4fh,66h,6dh,7dh,07h
        DB     7fh,6fh,77h,7Ch,39h,5Eh,79h,71h

        END
```

نوشتن اعداد 0 الی FH روی 7_Segment با اسمبلی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
char    code  CODE_7SEG[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
                                                0x7f,0x6f,0x77,0x7C,0x39,0x5E,0x79,0x71};
void main ()
{
int     i,k;
        init_8255();
        for(;;)
                for(k=0;k<=15;k++)
                        {
                        XBYTE[PB_ONBORD]=CODE_7SEG[k];
                        XBYTE[PC_ONBORD]=0x80;
                        for(i=0;i<32000;i++);           // DELAY
                        }
}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
```

نوشتن اعداد 0 الی FH روی 7_Segment با C

```
$INCLUDE (AIO_ADD.h)
SEG0  EQU   30h
SEG1  EQU   31h
SEG2  EQU   32h
SEG3  EQU   33h

        org 000h
        LJMP  START
        org 0Bh
        CLR   TR0
        LCALL TIMER0
        RETI
        org 100h
START:
        MOV   SP,#60h
        LCALL INIT
        MOV   DPTR,#CN_ONBORD
        MOV   A,#81h
        MOVX  @DPTR,A
        MOV   A,#04h
        MOV   DPTR,#CODE_7SEG
        MOVC  A,@A+DPTR
        MOV   SEG0,A
        MOV   A,#05h
        MOV   DPTR,#CODE_7SEG
        MOVC  A,@A+DPTR
        MOV   SEG1,A
        MOV   A,#06h
        MOV   DPTR,#CODE_7SEG
```

```
        MOVC A,@A+DPTR
        MOV  SEG2,A
        MOV  A,#07h
        MOV  DPTR,#CODE_7SEG
        MOVC A,@A+DPTR
        MOV  SEG3,A
        MOV  R0,#SEG0
        MOV  R1,#80h
LOOP:
        LJMP LOOP
INIT:
        ; setup timer0
        MOV  TMOD,#1
        MOV  TH0,#0FCh     ; 1.00043 ms XTAL=11.0592 MHZ
        MOV  TL0,#065h
        SETB TR0
        SETB ET0
        SETB EA
        RET
TIMER0:
        MOV  TH0,#0FCh
        MOV  TL0,#65h
        SETB TR0
        MOV  A,#0
        MOV  DPTR,#PC_ONBORD
        MOVX @DPTR,A                    ; off all segment
        MOV  A,@R0
        MOV  DPTR,#PB_ONBORD
        MOVX @DPTR,A                    ; data for segment
        MOV  A,R1
        MOV  DPTR,#PC_ONBORD
        MOVX @DPTR,A                    ; on one segment
        INC  R0
        CJNE R0,#SEG0+04h,LOC1
        MOV  R0,#SEG0
LOC1:
        MOV  A,R1
        RR   A
        MOV  R1,A
        CJNE R1,#08h,LOC2
        MOV  R1,#80h
LOC2:
        RET
CODE_7SEG:
        DB   3fh,06h,5bh,4fh,66h,6dh,7dh,07h
        DB   7fh,6fh,77h,7Ch,39h,5Eh,79h,71h

        END
```

نوشتن چهار عدد مختلف روی ۴ عدد *7_Segment* بصورت مالتی پلکس با اسمبلی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    timer0();
void    init();

timer0_trn() interrupt 1 using 0
{
        TR0=0;
        timer0();
}
char    code  CODE_7SEG[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,

        0x7f,0x6f,0x77,0x7C,0x39,0x5E,0x79,0x71};
char    SEG[4];
int     k;
unsigned char   ch;
void main ()
{
        init();
        init_8255();
        SEG[0]=CODE_7SEG[4];
        SEG[1]=CODE_7SEG[5];
        SEG[2]=CODE_7SEG[6];
        SEG[3]=CODE_7SEG[7];
        ch=0x80;
        k=0;
        for(;;);

}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
void    timer0()
{
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        XBYTE[PC_ONBORD]=00;     // Off All Segment
        XBYTE[PB_ONBORD]=SEG[k];
        XBYTE[PC_ONBORD]=ch;     // On One Segment
        k=k+1;
        if( k>=4 )
                k=0;
        ch=ch >> 1;
        if(ch == 0x8)
                ch=0x80;
        return;
}
void    init()
{
        /* setup timer0 */
```

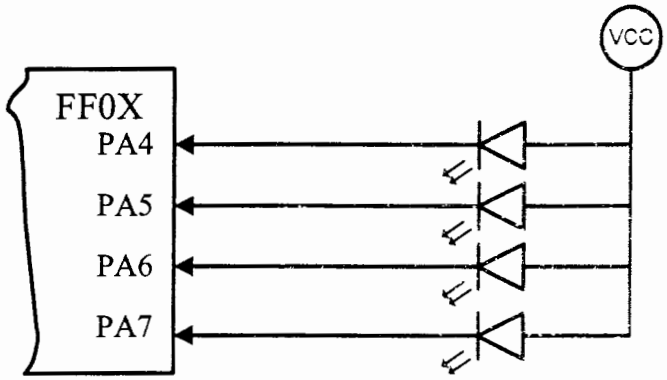
```
    TMOD=0x1;
    TH0=0xFC;     /* 1.00043 ms */
    TL0=0x65;
    TR0=1;
    ET0=1;
    EA=1;
}
```

نوشتن چهار عدد مختلف روی ۴ عدد *7_Segment* بصورت مالتی پلکس با *C*

۸ـ۹ *LED*

همانطور که در شکل ۱۳ـ۸ نشان داده شده است چهار عدد *LED* روی سخت افزار پیش بینی شده است که برای تست و آموزش برنامه نویسی می توان از آنها استفاده کرد. این چهار عدد *LED* به پینهای *PA4* الی *PA7* ۸۲۵۵ به آدرس پایه *FF0Xh* وصل شده اند. با صفر کردن هر پایه *LED* مربوطه روشن می گردد. هنگام کار با *LED* ها بایستی دقت کرد که وضعیت پایه های *PA0* الی *PA3* تحت تأثیر نامطلوب قرار نگیرند.

به جهت سادگی عملکرد *LED* ها در برنامه های زیر سعی شده است عملکرد تایمر نرم افزاری و تایمر سخت افزاری نیز نمایش داده شود.



شکل ۱۳ـ۸ *LED* ها و ارتباط آن با سخت افزار

```
$INCLUDE (AIO_ADD.h)

        org 000h
        LJMP   START
        org 100h
START:
        MOV    DPTR,#CN_ONBORD
        MOV    A,#81h
        MOVX @DPTR,A
LOOP:
        MOV    DPTR,#PA_ONBORD
        MOV    A,#0F0h
        MOVX  @DPTR,A
        LCALL DELAY
        MOV    DPTR,#PA_ONBORD
        MOV    A,#00h
        MOVX @DPTR,A
        LCALL DELAY
```

```
        LJMP   LOOP

DELAY:
        MOV    R0,#60h
DELAY1:
        MOV    R1,#0FFh
DELAY2:
        DEC     R1
        CJNE    R1,#0,DELAY2
        DEC    R0
        CJNE   R0,#0,DELAY1
        RET

        END
```

روشن و خاموش نمودن *LED* ها با استفاده از تأخیر نرم افزاری با اسمبلی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void main ()
{
int     i,k;
        init_8255();
        for(;;)
                {
                XBYTE[PA_ONBORD]=0xF0;
                for(i=0;i<32000;i++);
                XBYTE[PA_ONBORD]=0x00;
                for(i=0;i<32000;i++);
                }
}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
```

روشن و خاموش نمودن *LED* ها با استفاده از تأخیر نرم افزاری با *C*

```
$INCLUDE (AIO_ADD.h)
CH      EQU    32h

        org 000h
        LJMP   START
        org     0Bh
        CLR     TR0
        LCALL TIMER0
        RETI
        org 100h
START:
        MOV    SP,#60h
        MOV    DPTR,#CN_ONBORD
        MOV    A,#81h
```

```
        MOVX @DPTR,A
        LCALL INIT
        MOV   R0,#0
        MOV   R1,#0
        MOV   CH,#0
LOOP:
        LJMP  LOOP

INIT:
        ; setup timer0
        MOV   TMOD,#01h
        MOV   TH0,#0FCh     ; 1.00043 ms
        MOV   TL0,#65h
        SETB  TR0
        SETB  ET0
        SETB  EA
        RET

TIMER0:
        MOV   TH0,#0FCh     ;1.00043 ms
        MOV   TL0,#065h
        SETB  TR0
        INC   R0
        CJNE  R0,#0E8h,LOC1
        MOV   R0,#0
        INC   R1
        CJNE  R1,#03h,LOC1
        MOV   R1,#0
        MOV   DPTR,#PA_ONBORD
        MOV   A,CH
        MOVX  @DPTR,A
        CJNE  A,#0F0h,LOC2
        MOV   CH,#0
        RET
LOC2:
        MOV   CH,#0F0h
LOC1:
        RET
        END
```

روشن و خاموش نمودن *LED* ها با استفاده از تایمر صفر با اسمبلی

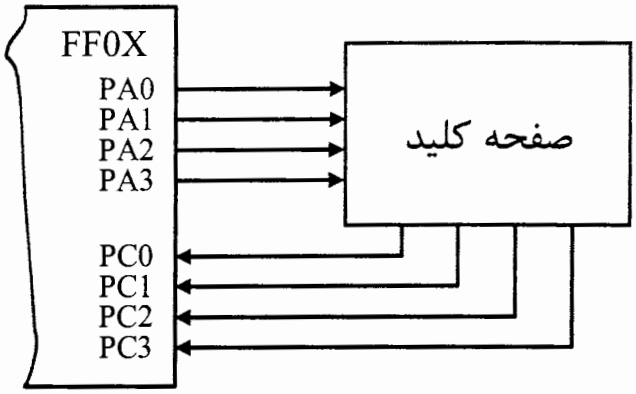
```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    timer0();
void    init();

timer0_trn() interrupt 1 using 0
{
        TR0=0;
        timer0();
}
int             TIMER;
```

```
char    ch;
void main ()
{
int     i;
        init_8255();
        init();
        TIMER=0;
        for(;;);
}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
void    timer0()
{
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        TIMER++;
        if( TIMER >= 1000 ) // one secend
                {
                TIMER=0;
                XBYTE[PA_ONBORD]=ch;
                if( ch ==0xF0 )
                        ch=00;
                else
                        ch=0xF0;
                }
        return;
}
void    init()
{
        /* setup timer0 */
        TMOD=0x1;
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        ET0=1;
        EA=1;
}
```

روشن و خاموش نمودن *LED* ها با استفاد از تایمر صفر با *C*

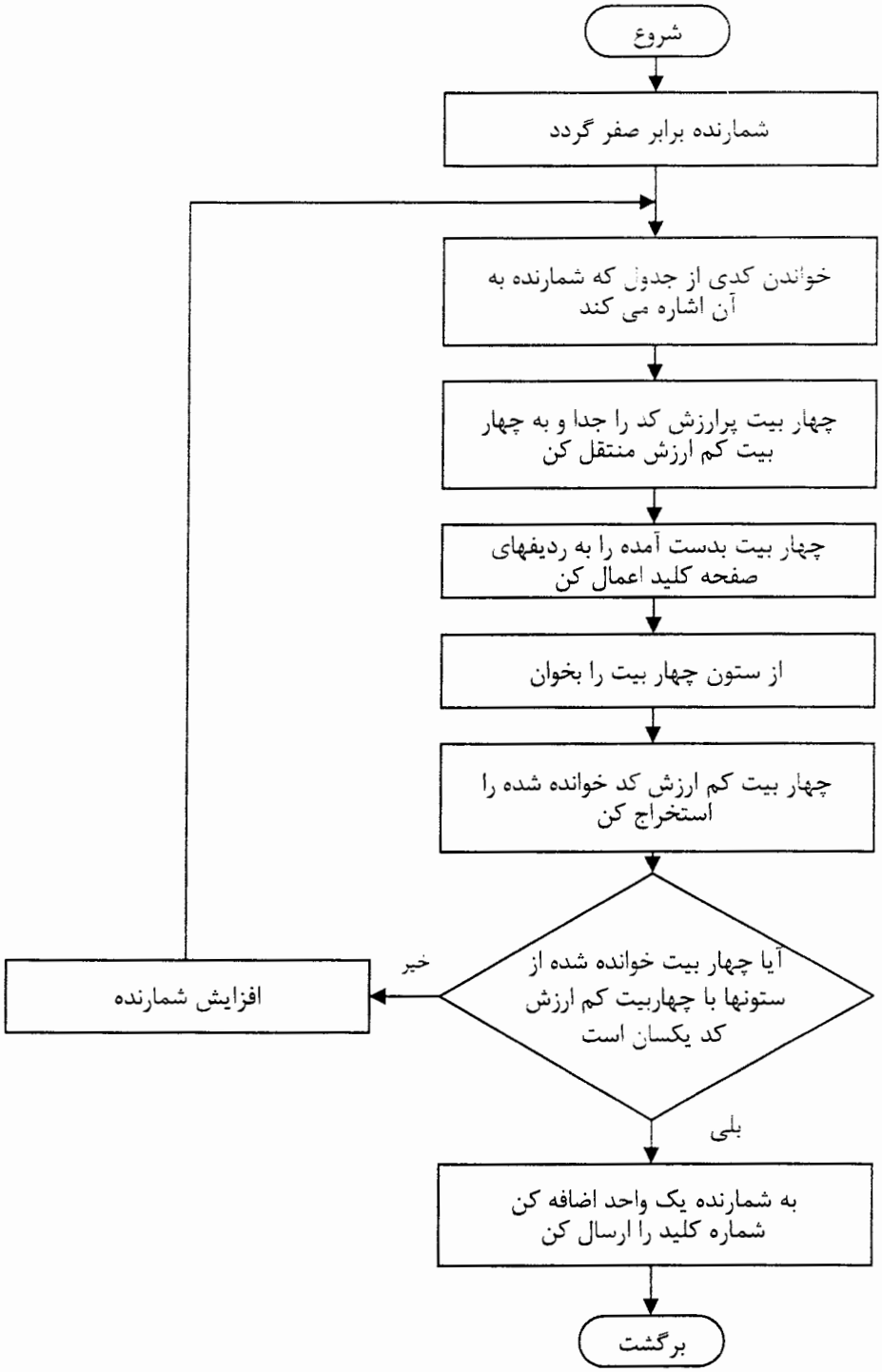
۱۰ـ۸ـ صفحه کلید

برای سخت افزار یک صفحه کلید *4x4* که نحوه ارتباط آن در شکل نشان ۱۴ـ۸ داده شده است پیش بینی شده است.



شکل ۱۴ـ۸ـ صفحه کلید و ارتباط آن با سخت افزار

جهت استخراج کد کلید فشرده شده از فلوچارت نشان داده شده در شکل ۱۵ـ۸ استفاده شده است. این الگوریتم از روش ساده ای کمک گرفته است. در این روش کد اعمالی به ردیفها (چهار بیت) و کد دریافتی از ستونها (چهار بیت) در یک بایت قرار گرفته و مجموعاً ۱۶ (*4X4*) بایت را تشکیل می دهند که در جدولی ذخیره می گردد. شمارنده از صفر شروع نموده کد مربوطه را خواندن چهار بیت اعمالی به ردیفها را جدا و به ردیفها اعمال می نماید سپس کد دریافتی از ستونها را دریافت و با چهار بیت کد خوانده شده مقایسه می کند اگر برابر بودند شمارنده کلید فشرده شده را نمایش می دهد در غیر اینصورت شمارنده اضافه شده و عملیات مجدداً انجام می گیرد اگر شمارنده به عدد ۱۶ رسید و تطابقی صورت نگرفت مبین این است که کلیدی فشرده نشده است.



شکل ۱۵ـ۸ـ فلوچارت استخراج کد کلید فشرده شده

```
$INCLUDE (AIO_ADD.h)

        org 000h
        LJMP   START
        org 100h
START:
        MOV    DPTR,#CN_ONBORD
        MOV    A,#81h
        MOVX   @DPTR,A
LOOP:
```

```
        LCALL READ_KEY
        CJNE  A,#0h,LOOP1
        LJMP  LOOP
LOOP1:
        MOV   DPTR,#CODE_7SEG
        MOVC  A,#A+DPTR
        MOV   DPTR,#PB_ONBORD
        MOVX  @DPTR,A
        MOV   DPTR,#PC_ONBORD
        MOV   A,#80h
        MOVX  @DPTR,A
        LJMP  LOOP

READ_KEY:
        MOV   R2,#0
READ1:
        MOV   A,R2
        MOV   DPTR,#CODE_KEY
        MOVC  A,@DPTR
        MOV   R0,A
        ANL   A,#0F0h
        SWAP  A
        MOV   DPTR,#PA_ONBORD
        MOVX  @DPTR,A
        MOV   DPTR,#PC_ONBORD
        MOVX  A,@DPTR
        ANL   A,#0Fh
        ANL   R0,#0Fh
        CJNE  A,R0,READ2
        MOV   A,R2
        RET
READ2:
        INC   R2
        CJNE  R2,#16d,READ1
        MOV   A,#00h
        RET

CODE_7SEG:
        DB    3fh,06h,5bh,4fh,66h,6dh,7dh,07h
        DB    7fh,6fh,77h,7Ch,39h,5Eh,79h,71h
CODE_KEY:
        DB    0EEh,0EDh,0EBh,0DEh,0DDh,0DBh,0BEh,0BDh,0BBh,07Dh
        DB    0E7h,0D7h,0B7h,07Eh,07Bh,077h

        END
```

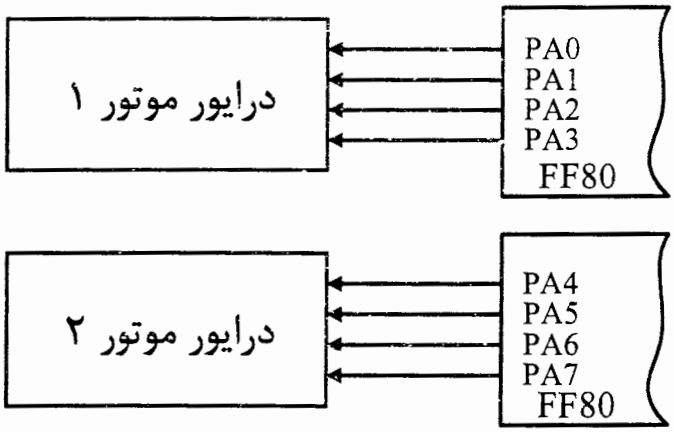
نحوه استخراج کلید فشرده با اسمبلی

```
#include        <REG52.H>
#include        <absacc.h>
#include        <IO_ADD.h>
void    init_8255();
char    read_key();
char    code  CODE_KEY[16]={0xEE,0xED,0xEB,0xDE,0xDD,0xDB,0xBE,0xBD,0xBB,0x7D

        ,0xE7,0xD7,0xB7,0x7E,0x7B,0x77};//1,2,3,F1,...,stop,enter
char    code  CODE_7SEG[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,

        0x7f,0x6f,0x77,0x7C,0x39,0x5E,0x79,0x71};
void main ()
{
int     k;
        init_8255();
        for(;;)
                {
                k=(int) read_key();
                if( k !=0 )
                        {
                        XBYTE[PB_ONBORD]=CODE_7SEG[k];
                        XBYTE[PC_ONBORD]=0x80;
                        }
                }
}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
char    read_key()
{
char    c,c0,d,k;
        for(k=0;k<16;k++)
        {
        d=CODE_KEY[k];
        c=d & 0xF0;
        c=c>>4;
        XBYTE[PA_ONBORD]=c;
        c=XBYTE[PC_ONBORD];
        c=c & 0x0f;
        if( c==(d & 0x0f) )
                return (k+1);
        }
        return 0;
}
```

نحوه استخراج کلید فشرده با C

۱۱ـ۸ـ موتور *DC* و موتور پله ای

در سخت افزار مورد نظر دو عدد آی سی *L298* پیش بینی شده است و مطابق شکل ۱۶ـ۸ به پینهای پورت A در ۸۲۵۵ متصل شده است. در نتیجه توسط سخت افزار فوق می توان ۴ موتور *DC* و یا دو موتور پله ای را کنترل کرد.



شکل ۱۶ـ۸ـ موتور پله ای یا *DC* و ارتباط آن با سخت افزار

```
#include    <REG52.H>
#include    <absacc.h>
#include    <CIO_ADD.h>
void  init_8255_IO();

void main ()
{
int   i,j;
      init_8255_IO();
      for(;;)
            {
            XBYTE[PA_OUTBORD]=0x55;
            for(i=0;i<10;i++)
                  for(j=0;j<100;j++);
            XBYTE[PA_OUTBORD]=0xaa;
            for(i=0;i<10;i++)
                  for(j=0;j<10;j++);
            send_char(0x55);
            }
}
void  init_8255_IO()
{
      XBYTE[CN_OUTBORD]=0x80;
      return;
}
```

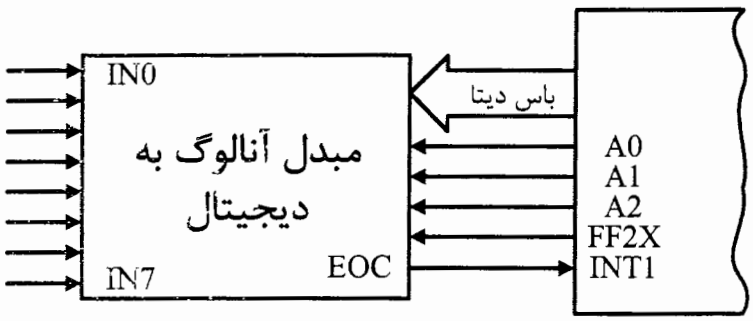
کنترل موتور *DC* بصورت تایمر نرم افزاری

۱۲ـ۸ـ انتراپت سخت افزاری

همانطور که قبلاً بیان شد میکروکنترلر ۸۰۵۱ دارای دوپایه اینتراپت سخت افزاری خارجی است. پایه *INT1* میکروکنترلر مطابق شکل ۱۷ـ۸ به پایه *EOC* مبدل آنالوگ به دیجیتال وصل شده است. در نتیجه می توان برای تبدیل از آنالوگ به دیجیتال از قابلیت وقفه سخت افزاری علاوه برتأخیر نرم افزاری استفاده کرد. پایه *INT0* میکروکنترلر بهمراه پایه *T0* به کانکتور سه پایه ای وصل شده است در نتیجه می توان با اتصال یک کلید یا سخت افزاری خاص قابلیت اینتراپت سخت افزاری و یا کانتر را در این میکروکنترلر مورد آزمایش قرار داد.

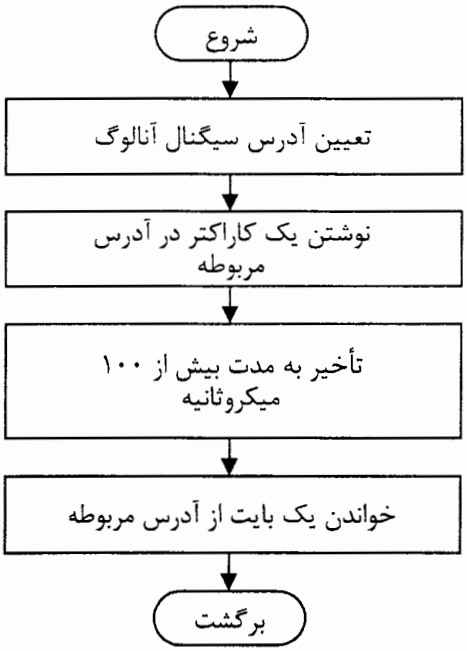
۱۳ـ۸ـ مبدل آنالوگ به دیجیتال

جهت تبدیل سیگنالهای آنالوگ به دیجیتال از مبدل *ADC0809* کمک گرفته شده است. این مبدل قادر است ۸ سیگنال آنالوگ را بصورت مالتی پلکس (یعنی در یک لحظه خاص فقط یک سیگنال) به دیجیتال در مدت حدود ۱۰۰ میکروثانیه تبدیل و در اختیار قرار دهد. اتصالات لازم این مبدل به هسته سخت افزاری در شکل ۱۷ـ۸ نشان داده شده است. فلوچارت نحوه تبدیل سیگنال در شکل ۱۹ـ۸ بااستفاده از تأخیر نرم افزاری و در شکل ۲۰ـ۸ بااستفاده از اینتراپت سخت افزاری نشان داده شده است.

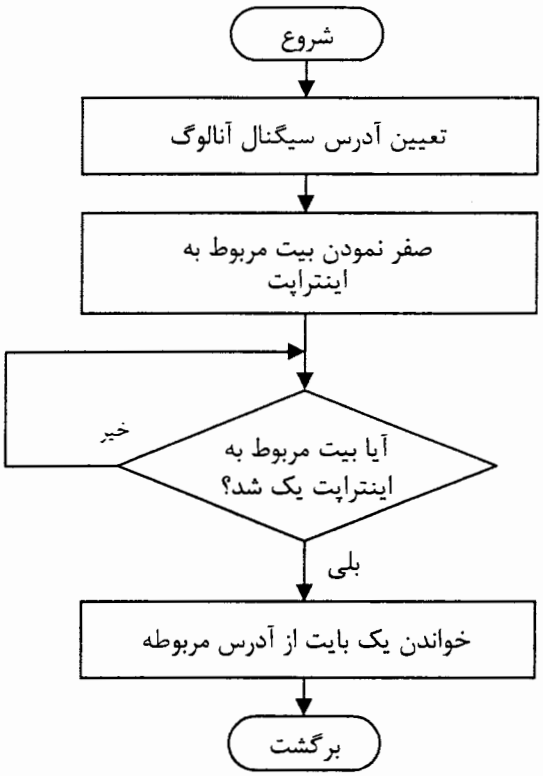


شکل ۱۷ـ۸ـ مبدل آنالوگ به دیجیتال و ارتباط آن با سخت افزار

از کانالهای هفتم و هشتم این مبدل جهت اندازه گیری جریان موتورهای *DC* یا پله ای و به کانالهای پنجم و ششم دو سنسور نوری و گرمایی متصل شده است که برای تست و آزمایش می توان از آنها استفاده نمود.



شکل ۱۹ـ۸ـ فلوچارت نحوه تبدیل سیگنال آنالوگ به دیجیتال با استفاده از تأخیر



شکل ۲۰ـ۸ـ فلوچارت نحوه تبدیل سیگنال آنالوگ به دیجیتال با استفاده از اینتراپت سخت افزاری

```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    timer0();
void    init();

timer0_trn() interrupt 1 using 0
{
        TR0=0;
        timer0();
}
char    code  CODE_7SEG[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,

        0x7f,0x6f,0x77,0x7C,0x39,0x5E,0x79,0x71};
char    SEG[4];
int     k,i;
unsigned char   ch;
void main ()
{
char    c;
unsigned char   m;
        init();
        init_8255();
        ch=0x80;
        k=0;

        for(;;)
                {
                XBYTE[ATODC0]=c;   // Start AD IN0
                for(i=0;i<15000;i++); // Delay
                c=c;
                c=XBYTE[ATODC0];   // Read Result
                m= c & 0x0F;               // Show Result
                SEG[0]=CODE_7SEG[(int) m];
                m= c & 0xF0;
                m=m >> 4;
                SEG[1]=CODE_7SEG[(int) m];
                }
}
void    init_8255()
{
        XBYTE[CN_ONBORD]=0x81;
        return;
}
void    timer0()
{
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        XBYTE[PC_ONBORD]=00;     // Off All Segment
        XBYTE[PB_ONBORD]=SEG[k];
        XBYTE[PC_ONBORD]=ch;     // On One Segment
        k=k+1;
        if( k>=4 )
```

```
                k=0;
        ch=ch >> 1;
        if(ch == 0x8)
                ch=0x80;
        return;
}
void    init()
{
        /* setup timer0 */
        TMOD=0x1;
        TH0=0xFC;       /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        ET0=1;
        EA=1;
}
```

تبدیل سیگنال آنالوگ به دیجیتال و نمایش آن روی دو عدد *7_segment* بصورت C

```
$INCLUDE (AIO_ADD.h)
T2CON           EQU 0C8h
RCAP2H          EQU 0CBh
RCAP2L          EQU 0CAh
INT1_OK EQU 1

        org 000h
        LJMP   START
        org 13h
        SETB INT1_OK
        RETI
        org 100h

START:
        LCALL init_8255

        SETB EX1                ; Setup INT1
        SETB IT1
        CLR INT1_OK
        SETB EA

        MOV SCON,#50h                   ; setup serial port
        MOV T2CON,#34h
        MOV RCAP2H,#0FEh  ; 1200
        MOV RCAP2L,#0E0h
        MOV    A,#55h
        LCALL SEND_CHAR
LOOP:
        MOV    DPTR,#ATODC5
        MOVX @DPTR,A
LOC1:
        JNB INT1_OK,LOC1
        CLR INT1_OK
        MOV    DPTR,#ATODC5
        MOVX A,@DPTR
        LCALL SEND_CHAR
```

```
        LJMP LOOP

init_8255:
        MOV    DPTR,#CN_ONBORD
        MOV A,#81h
        MOVX @DPTR,A
        RET
SEND_CHAR:
        MOV SBUF,A
LOC2:
        JNB TI,LOC2
        CLR TI
        RET
        END
```

تبدیل سیگنال آنالوگ به دیجیتال با استفاده از اینتراپت و ارسال آن برای پورت سریال با C

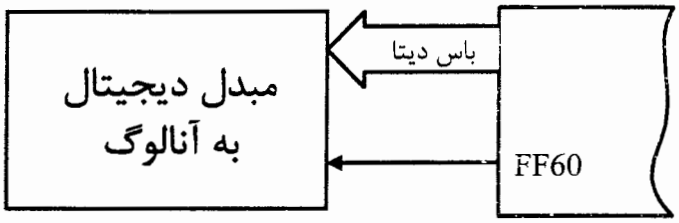
```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    send_char(char);

bit             INT1_OK;
int1_trn() interrupt 2 using 0
{
        INT1_OK=1;
}
void main ()
{
char    c;
        init_8255();

        EX1=1;          /* Setup INT1    */
        IT1=1;
        INT1_OK=0;
        EA=1;

        SCON=0x050;             /* setup serial port */
        T2CON=0x34;
        RCAP2H=0xFE;            /* 1200 */
        RCAP2L=0xE0;
        send_char(0x55);
        send_char(0x55);
        for(;;)
                {
                XBYTE[ATODC5]=c;   // Start AD IN4
                while(INT1_OK==0);
                INT1_OK=0;
                c=XBYTE[ATODC5];   // Read Result
                send_char(c);
                }
}
void    init_8255()
{
```

```
        XBYTE[CN_ONBORD]=0x81;
        return;
}
void    send_char(char cx)
{
        ES=0;
        SBUF=cx;
        while(TI==0);
        TI=0;
        ES=1;
}
```

تبدیل سیگنال آنالوگ به دیجیتال با استفاده از اینتراپت و ارسال آن برای پورت سریال با *C*

۱۴ـ۸ـ مبدل دیجیتال به آنالوگ

نحوه اتصال هسته سخت افزاری به مبدل دیجیتال به آنالوگ در شکل۲۱ـ۸ نشان داده شده است. آدرس نگهدار بایت ارسال شده FF60h می باشد و نتیجه تبدیل روی کانکتور *J7* ظاهر می گردد. لذا در برنامه نویسی کافی است عدد مورد نظر در حافظه خارجی و به آدرس FF60h نوشته شود.



شکل ۲۱ـ۸ـ مبدل دیجیتال به آنالوگ و ارتباط آن با سخت افزار

```
$INCLUDE (AIO_ADD.h)

        org 000h
        LJMP   START
        org    0Bh
        CLR    TR0
        LCALL TIMER0
        RETI
        org 100h
START:
        MOV    DPTR,#CN_ONBORD
        MOV    A,#81h
        MOVX   @DPTR,A
        MOV    R4,#0
LOOP:
        LJMP   LOOP
INIT:
        ; setup timer0
        MOV    TMOD,#01h
        MOV    TH0,#0FCh      ; 1.00043 ms
        MOV    TL0,#65h
        SETB   TR0
        SETB   ET0
        SETB   EA
```

```
        RET
TIMER0:
        MOV   TH0,#0FCh     ;1.00043 ms
        MOV   TL0,#065h
        SETB  TR0
        MOV   DPTR,#DTOAC
        MOV   A,R4
        MOVX  @DPTR,A
        INC   R4
        RET

        END
```

تولید یک موج دندان اره ای با استفاده از مبدل دیجیتال به آنالوگ با اسمبلی با فرکانس مشخص

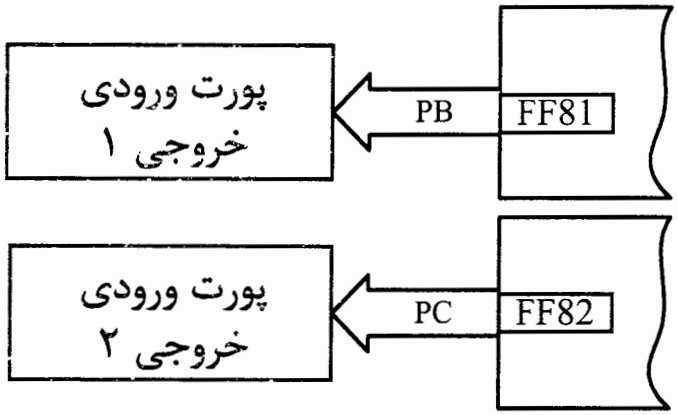
```
#include        <REG52.H>
#include        <absacc.h>
#include        <CIO_ADD.h>
void    init_8255();
void    timer0();
void    init();

timer0_trn() interrupt 1 using 0
{
        TR0=0;
        timer0();
}
char    ch=0;
void main ()
{
        init();
        init_8255();

        for(;;);
}
void    init_8255()
{
        XBYTE[CT_ONBORD]=0x81;
        return;
}
void    timer0()
{
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
        TR0=1;
        XBYTE[DTOAC]=ch;   // Convert to Analoge
        ch++;
        return;
}
void    init()
{
        /* setup timer0 */
        TMOD=0x1;
        TH0=0xFC;     /* 1.00043 ms */
        TL0=0x65;
```

```
    TR0=1;
    ET0=1;
    EA=1;
}
```

تولید یک موج دندان اره ای با استفاده از مبدل دیجیتال به آنالوگ با C و با استفاده از تایمر

۱۵ـ۸ـ پورتهای ورودی خروجی اضافی

ازاین پورتها می توان برای توسعه کاربردهای سیستم، اتصال پورت موازی کامپیوتر و غیره استفاده کرد. آدرس این پورتها مطابق شکل ۲۱ـ۸ *FF81h* و *FF82h* می باشدو بترتیب به کانکتورهای *J9* و *J8* وصل شده اند. لذا می توان ۱۶ پین ورودی خروجی اضافی علاوه بر پینهای ورودی خروجی میکروکنترلر که به جایی وصل نشده اند در اختیار داشت در نتیجه ۲۶ (۱۶+۱۰) پایه ورودی خروجی می توان در سخت افزار در اختیار گرفت و از آنها برای منظورهای خاص استفاده کرد.



شکل ۲۲ـ۸ـ پورتهای اضافی بکار رفته در سخت افزار

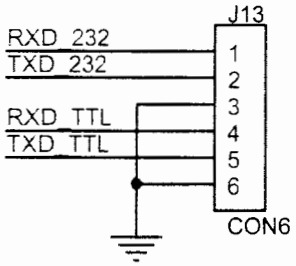
۱۶ـ۸ـ قابلیتهای اضافی سخت افزار

۱ـ قابلیت دریافت تمامی پایه های میکروکنترلر

با این قابلیت که توسط کانکتور *J1* در اختیار قرار می گیرد می توان هر سخت افزاری را طراحی و به سیستم مورد نظر متصل و آزمایشها و تستهای مختلف را انجام و نتایج لازم را استخراج کرد.

۲ـ وجود مبدل *TTL* به *RS232* و بالعکس اضافی

با این قابلیت که توسط کانکتور *J13* در اختیار قرار می گیرد می توان سیگنالهای *TTL* را به *RS232* و یا *RS232* را به *TTL* تبدیل و مورد استفاده قرار داد. نحوه اتصال آن در شکل ۲۳ـ۸ نشان داده شده است.



شکل ۲۳ـ۸ـ کانکتور اضافی پورت سریال جهت بعضی آزمایشها

۳ـ وجود تغذیه ۵ ولت جهت بعضی آزمایشها

در سخت افزار فوق کانکتوری پیش بینی شده است (*J14*) که توسط آن می توان ۵ ولت را دریافت و در موقع لزوم از آن استفاده کرد.

۴ـ برنامه ریزی حافظه های *EEPROM*

در سخت افزار فوق می توان بجای حافظه های *RAM* که هر یک ۳۲ کیلوبایت ظرفیت دارند حافظه *EEPROM* از نوع *28C258* قرار داد و برنامه ریزی لازم را انجام داد و در سخت افزارهای دیگر مورد استفاده قرار داد.

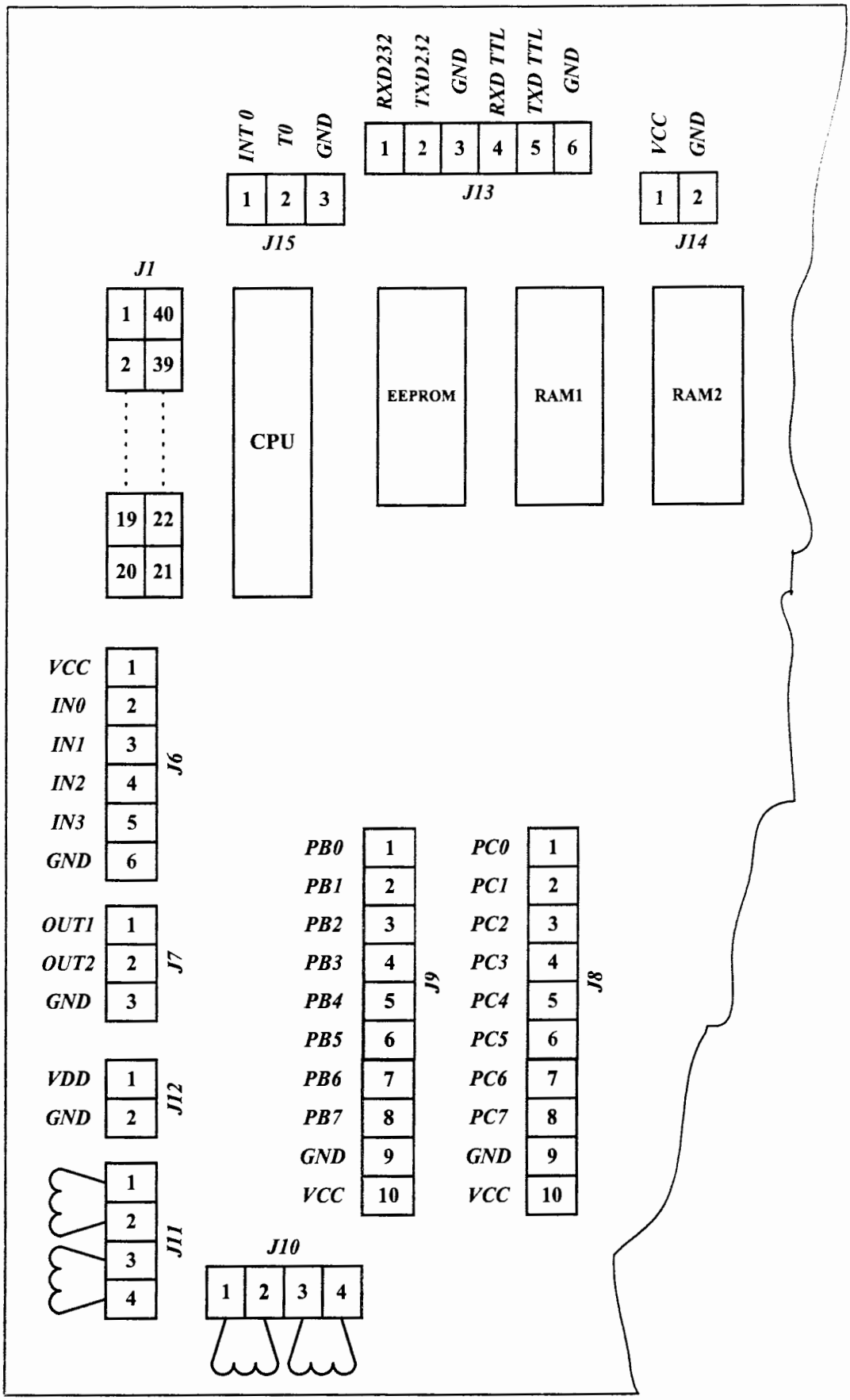
موقعیت هریک از کانکتورهای مختلف در شکل ۲۴ـ۸ نشان داده شده است.

تمرین ۱: با استفاده از سخت افزار فوق یک فرکانس متر طراحی کنید.

تمرین ۲: با استفاده از سخت افزار فوق یک ساعت دیجیتال طراحی کنید.

تمرین ۳: با استفاده از سخت افزار فوق یک سیگنال ژنراتور سینوسی و مربعی طراحی کنید.

تمرین ۴: با استفاده از سخت افزار فوق یک کنترل کننده دیجیتال طراحی کنید.



شکل ۲۴ـ۸ـ موقعیت کانکتورها روی سخت افزار

فصل نهم

عیب یابی برنامه ها به زبان اسمبلی و C

۱ـ۹ـ مقدمه

عیب یابی از دو جنبه سخت افزاری و نرم افزاری حائز اهمیت است. هنگامی که یک سخت افزار طراحی می گردد روند طولانی بایستی طی گردد تا به یک مدار بدون عیب و نقص تبدیل گردد. مشکلاتی چون اشکال طراحی، نبود قطعات با کیفیت مناسب، اشکال در مدارچاپی، اشکال در نرم افزار و ... باعث می گردند طراحی انجام شده در مراحل اولیه جواب مطلوب نداده و لازم باشد روی طرح وقت بیشتری صرف نمود. علاوه بر اشکالات اولیه، اشکالات دیگری پس از ساخت و طراحی و بکارگیری سیستم بروز خواهند کرد که لازم است این اشکالات تشخیص داده شده و رفع عیب گردند اشکالاتی چون سوختن قطعات، ترک برداشتن مدار چاپی، اثرات حرارتی روی سخت افزار و ... باعث می گردند سیستم بلااستفاده باشد لذا لازم است سخت افزار مورد نظر عیب یابی شده و اشکالات مرتفع گردد.

لذا طراح هنگام اجرای یک پروژه میکروپروسسوری با دو مشکل عمده سخت افزاری و نرم افزاری درگیر می باشد معمولاً اشکالات سخت افزاری در ابتدای طراحی خود را نشان می دهند و استفاده از امکاناتی چون اسیلوسکوپ، لاجیک آنالایزر و ... می توانند کمک خوبی باشد. در مقابل، مشکلات عدیده ای وجود دارند که منشأ نرم افزاری دارند. عیب یابی اینگونه برنامه ها مخصوصاً هنگامی که حجم برنامه زیاد شود و روند درستی در برنامه ریزی رعایت نشده باشد بسیار مشکل است برای حل این مشکل تاکنون سیمولاتورهای نرم افزاری مختلف ارائه شده اند که توسط آنها می توان برنامه مورد نظر را روی کامپیوتر به فرمهای مختلف اجرا و نتیجه را مشاهده کرد و اشکال نرم افزار را مشخص و عیب یابی نمود. اگر میکروپروسسور از نوع *80x86* باشد و لازم باشد برنامه نوشته شده روی کامپیوتر اجرا گردد از آنجایی که در کامپایلرها امکانات اجرای برنامه بصورت قدم بقدم پیش بینی شده است عیب یابی آن خیلی مشکل نیست ولی هنگام کار با میکروپروسسوری غیر از سری *X86* کار به این راحتی نیست و بایستی از شبیه سازها کمک گرفت. از طرفی کار با شبیه سازهای نرم افزاری نیز نسبتاً سخت بوده و دادن ورودیهای آنالوگ و دیجیتال و انتراپتها ساده نمی باشد استفاده از آنها گرچه تااندازه أی می تواند کمک طراح باشد ولی خیلی نمی تواند مؤثر باشد و دلیلی وجود ندارد نرم افزار جواب گرفته با سیمولاتور روی سخت افزار واقعی جواب مطلوب بدهد. لذا باتوجه به تجارب گذشته و کار با چند میکروپروسسور و میکروکنترلر روش زیر برای عیب یابی برنامه ها با مثالهایی ارائه می گردد.

۲ـ۹ـ روش پیش نهادی جهت عیب یابی برنامه ها

این روش بر اساس نمایش نتایج روی کامپیوتر بنا شده است. بعبارت دیگر در این روش از کامپیوتر بعنوان یک وسیله نمایش استفاده می گردد اما آنچه که بایستی نمایش داده شود شکل موج و منحنی نبوده بلکه اطلاعاتی است که برنامه نویس در مکانهای مشخص برنامه آنها را بفرم مناسب برای کامپیوتر ارسال می کند. بعبارت دیگر در این روش برنامه نویس نتایج حاصل از یک عملیات را که ممکن است درست یا غلط باشند برای پورت سریال ارسال می کند هنگام اجرای برنامه و نمایش نتایج بدست آمده از پورت سریال روی کامپیوتر کاربر این نتایج را با نتایج مورد انتظار مقایسه و به وجود خطا پی می برد.

بعنوان مثال پس از فشردن کلید ریست برنامه ای بایستی اجرا و از پورت *P1* اطلاعاتی را دریافت و آنرا با *80h* مقایسه کند اگر عدد خوانده شده بزرگتر از *80h* بود زیر برنامه یک و اگر کوچکتر یا مساوی *80h* بود زیر برنامه ۲ اجرا گردد. آیا سخت افزار سالم است و از پورت P1 چه اعدادی دریافت می کند؟

برای مطمئن شدن از اجرای برنامه و سالم بودن سخت افزار می توان در ابتدای برنامه پس از برنامه ریزی پورت سریال و دیگر پورتها دو عدد مشخص مثلاً 0x54 و 0x55 را برای پورت سریال فرستاد و هنگامی که وارد قسمت اصلی برنامه شد داده خوانده شده از پورت P1 را برای پورت سریال ارسال کند و در زیر برنامه یک عدد 00h و در زیر برنامه دو عدد FFh را ارسال کند. در نتیجه اگر هنگام اجرای برنامه اعداد درست و منطقی دریافت نشدند می توان به برنامه برگشت و اصلاحات لازم را انجام داد.

```
#include    <REG52.H>
void  send_char(char);
void  sub1(),sub2();
void main ()
{
      SCON=0x50;          /* setup serial port */
      TMOD=0x20;          /*    Used  Timer 1 */
      TH1=0xE8;           /* 1200 */
      TR1=1;
      send_char(0x54);
      send_char(0x55);
      for(;;)
            {
            c=P1; // read P1
            if ( c > 0x80 )
                  sub1();
            else
                  sub2();
            }
}
void  send_char(char c)
{
      SBUF=c;
      while(TI==0);
      TI=0;
}
void  sub1()
{
      send_char(0);
      // other statement
      return;
}
void  sub2()
```

```
{
        send_char(0xff);
        // other statemet
        return;
}
```

بعنوان مثالی دیگر در یک سخت افزار مبدل آنالوگ به دیجیتال پیش بینی شده است و قرار است از آن برای کنترل درجه حرارت محیط استفاده شود برنامه لازم نوشته شده اما جواب مطلوب دریافت نمی گردد در این حالت چه بایستی کرد؟

برای این منظور روش زیر پیش بینی می گردد. برنامه ای نوشته می شود که سیگنال آنالوگ را به دیجیتال تبدیل و نتیجه بدست آمده را علاوه بر استفاده در برنامه، برای پورت سریال نیز ارسال نماید. با انجام این تست و تغییر درجه حرارت کاربر براحتی به وجود یا عدم وجود اشکال در این قسمت پی می برد. اگر اشکال در این قسمت وجود نداشت ممکن است در محاسبات کنترل کننده باشد بدین جهت محاسبات لازم انجام و نتایج آن برای پورت سریال ارسال می گردد با مقایسه نتایج و تغییر آن به ازای شرایط مختلف می توان به وجود مشکل در این قسمت پی برد.

به عبارت دیگر با این روش می توان کلیه زیر برنامه ها، نتایج حاصل از محاسبات، عملکرد مناسب سخت افزار و ... را بسادگی مورد ارزیابی قرار داد و در کوتاه مدت عیب مورد نظر را استخراج کرد و به وسایلی چون اسیلوسکوپ نیازی نداشت.

۳ـ۹ـ پیش بینی پورت سریال در سخت افزار

به دلایل متعدد وجود پورت سریال در بعضی از سخت افزارها ضروری بوده و بایستی طراحی لازم را برای این منظور انجام داد در این حالت اگر مقدور بود و پورت سریال برای کار ضروری پیش بینی نشده بود می توان از آن به منظور عیب یابی استفاده کرد اما اگر نتوان از آن استفاده کرد بایستی تنها از خروجی پورت سریال (*TXD*) استفاده کرد و یا دو عدد پورت سریال در آن پیش بینی کرد.

ممکن است در سخت افزار واقعی گذاشتن پورت سریال امری ضروری نبوده و باعث بالا رفتن هزینه گردد در این حالت می توان بصورت زیر اقدام کرد.

در این صورت دو حالت وجود دارد حالت اول آنست که در میکروپروسسور یا میکروکنترلر پورت سریال پیش بینی نشده باشد و حالت دوم آن است که پورت سریال وجود داشته باشد. در حالت اول کار چندانی نمی توان انجام داد و یا اگر امکانات اجازه دهد پورت سریال با استفاده از مبدلهای پارالل به موازی و بالعکس همچون *2651* پیش بینی کرد.

در حالت دوم برد سخت افزاری مطابق خواسته طراحی می گردد به نحوی که براحتی بتوان به پایه های *TXD* و *RXD* میکرو کنترلر دسترسی داشت (دو نقطه تست گذاشته می شود) سپس بردی سخت افزاری کوچک که تنها مبدل *TTL* به *RS232* و بالعکس داشته باشد طراحی و ساخته می گردد که از یک طرف توسط گیره های سوسماری به سخت افزار و از طرف دیگر به کامپیوتر قابل اتصال باشد در این صورت براحتی می توان تستهای لازم را انجام و به نتایج دلخواه رسید.

فصل دهم

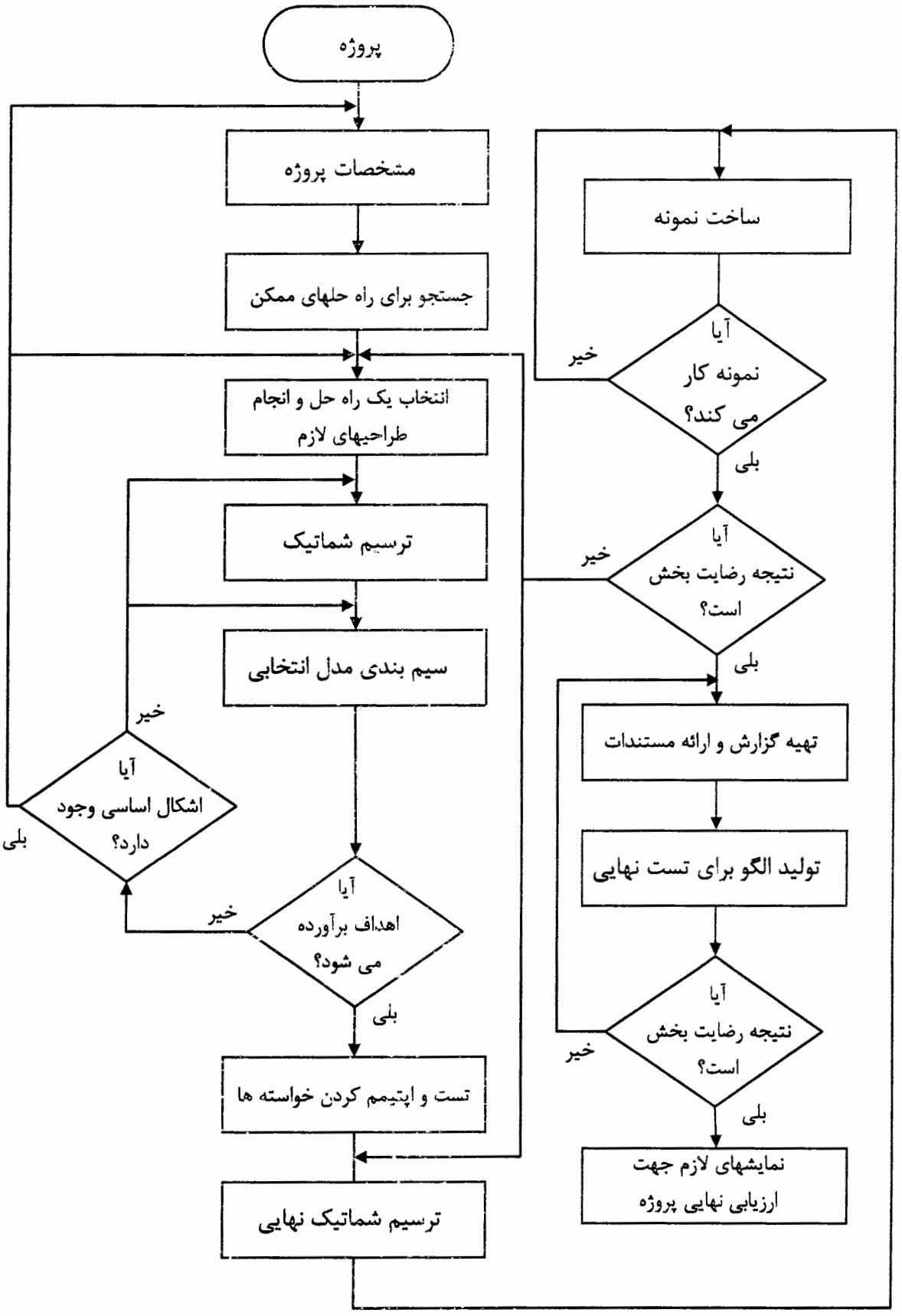
مراحل انجام پروژه و نتیجه گیری

۱ـ۱۰ـ مقدمه

از هنگامی که یک پروژه از نوع طراحی و ساخت بصورت خام تعریف می گردد تا به مرحله بهره برداری برسد مراحل زیادی بایستی طی گردد اجرای درست و مرحله به مرحله آنها باعث به نتیجه رسیدن سریعتر پروژه می گردد. هدف از بیان این فصل یاد آوری مراحل مختلف و توضیحات لازم برای هریک است. همانطور که مشاهده شد بعضی از مراحل مختلف این پروژه قبلاً مورد بررسی قرار گرفت و توضیحات لازم به تفصیل آورده شد. در این فصل بخاطر جمع بندی و بدست آوردن نتیجه بهتر این مراحل با توضیحات مختصر آورده شده است.

۲-۱۰- مراحل اجرای پروژه

برای اجرای هرپروژه تحقیقاتی و کاربردی بایستی مراحل مختلفی را مطابق بلوک دیاگرام شکل ۱ـ۱۰ طی کرد تا به نتیجه مطلوب رسید. این بلوک دیاگرام نشان می دهد ابتدا بایستی مشخصات پروژه تعیین، راه حلهای ممکن تعیین و از بین آنها بهترین راه انتخاب و طراحی های لازم انجام و پس از بستن مدار و انجام تستها و عیب یابیهای مختلف، آنرا به نتیجه رساند. برای پروژه مورد نظر نیز این مراحل که در ادامه با جزئیات بیشتری آورده شده اند طی شده و جواب نهایی که طراحی و ساخت سیستم شبیه ساز خانواده ۸۰۵۱ بوده است حاصل شده است.



شکل ۱-۱۰ - فلوچارت کلی نحوه انجام پروژه

۳-۱۰- تعریف دقیق پروژه و اهداف مورد نظر

در پروژه های کاربردی و عملی یکی از موارد مشکل و وقت گیر تعریف صورت مسئله است. اگر پروژه از جانب فرد، گروه یا شرکتی پیشنهاد شده باشد و تمامی جزئیات در آن منعکس شده باشد طراح بدون هیچ دغدغه ای به طراحی سخت افزار و نوشتن نرم افزار پرداخته و تنها مشکل او بعد علمی و عملی بوده و کار او مشخص و روتین وار می باشد. اما همیشه اینطور نیست گاهی صورت مسئله کامل نبوده و تنها کلیات در آن بیان شده است در این صورت کار طراح یا برنامه نویس مشکل بوده و بایستی با پرس و جوهای مختلف، با در نظر گرفتن محدودیتها، با در نظر گرفتن امکانات و وقت و هزینه پیش بینی شده کلیه جزئیات لازم را استخراج و در طراحی و برنامه نویسی لحاظ نماید. علاوه بر این گاهی اوقات صورت مسئله یا پروژه در ابتدای امر بظاهر مناسب تعریف شده است و مشخصات لازم تعیین شده اند اما با گذشت زمان و تجربه بیشتر، صورت مسئله دچار تغییراتی می گردد که گاهی نمی توان از آن اجتناب کرد بعنوان مثال طرح قبلی اعتبار خود را از دست داده و بایستی آنرا با قابلیتهای بیشتری جهت جلب مشتری ارائه کرد و یا پیش فرضهای نامناسبی در صورت مسئله منظور شده است که بایستی اصلاح گردند. در هر صورت با در نظر گرفتن موارد بالا طراح بایستی بداند جهت انجام یک پروژه نرم افزاری یا سخت افزاری دانستن کلیه جزئیات لازم و ضروری بوده و بایستی قبل از هر اقدامی به مشخص کردن آنها اقدام کند. وجود این نوع کم و کاستی ها هنگام اجرای یک پروژه عملی کاملاً طبیعی بوده که تنها با مطالعه و تحقیق و انجام پرس و جوهای متعدد می توان به قسمتی از آنها پاسخ گفت. لذا قبل از هر اقدامی بایستی صورت مسئله دقیقاً تشریح و خواسته ها مشخص گردند.

یکی از مشکلات موجود در اجرای این پروژه نیز تعیین مشخصات لازم برای سخت افزار و تعیین قابلیتهای لازم برای نرم افزار بود که پس از بررسی های فراوان سخت افزار و نرم افزار مورد نظر که کمبودها و اشکالاتی نیز خواهد داشت ارائه گردید.

۴-۱۰- انجام مطالعات و تحقیقات لازم

پس از اینکه صورت مسئله و خواسته های پروژه مشخص گردید بایستی راه حلهای مختلف و ابزار لازم جهت رسیدن به هدف تعیین گردند. تعیین راه حلهای ممکن و ابزار لازم و اینکه چگونه پروژه پیاده سازی گردد ساده نبوده و به عوامل زیادی وابسته است در این بین بایستی اطلاعات موجود، وجود قطعات و عناصر لازم، هزینه سخت افزار و نرم افزار و ... مد نظر قرار گیرد. در اولین مرحله بایستی اطلاعات لازم در آن زمینه را حد امکان تکمیل نمود و اگر کار یا فعالیتی قبلاً انجام شده است شناسایی و مطالعه گردد و پس از بدست آوردن اطلاعات لازم تصمیم گیری لازم انجام گیرد.

در پروژه مورد نظر نیز این مطالعات انجام شده و پاسخ مناسبی برای آنها دریافت گردید. این مطالعات بطور کلی روی دو موضوع مبانی و ساخت متمرکزند که هریک بایستی بدقت مورد بررسی و مطالعه قرار گیرد و مناسب ترین انتخاب صورت گیرد.

۵-۱۰- تعیین راه حلهای ممکن و انتخاب بهترین راه

برای حل یک مسئله راه حلهای مختلفی ممکن است وجود داشته باشد. بایستی ابتدا این راه حلها شناسایی و باتوجه به بررسی های انجام شده مناسب ترین راه با توجه به هزینه، قابلیت اجرا و سادگی انتخاب، و اقدامات بعدی صورت گیرد در این پروژه می بایست ساختار سخت افزار، نوع میکروکنترلر، ساختار نرم افزار و کامپایلرهای مناسب و امکانات لازم برای سخت افزار و نرم افزار مشخص گردند و سپس از بین پاسخهای مختلف موجود، مناسب ترین راه با توجه به امکانات موجود انتخاب گردد.

۶-۱۰- تهیه الگوریتم حل مسئله

نتیجه حاصل از مطالعات و تحقیقات به دو بخش سخت افزار و نرم افزار تقسیم می شود. بخش سخت افزار موضوع بعدی بوده و الگوریتم خاصی برای آن نمی توان در نظر گرفت. آنچه که در این بین اهمیت دارد انتخاب راه حلهای مناسب جهت حل مسائل مربوط به نرم افزار است. در این پروژه مسائل متعدد نرم افزاری وجود داشت که بایستی حل می شد. در این ارتباط فلوچارتهای مختلفی برای حل این مسائل ترسیم گردید. گاه پیش می آمد فلوچارتهای مختلف برای حل مسئله ارائه می گردید بهترین آنها انتخاب می گردید و گاه پس از مدتی کار و فعالیت و پیشرفت در امور مختلف این نتیجه عاید می شد که بهتر است نوع نگاه به حل مسئله تغییر کرده و فلوچارت متناسب با آن نوشته شود. قابل ذکر است با توجه به تجارب و نتایج بدست آمده لازم نیست ابتدا تمامی فلوچارتها ترسیم و سپس برنامه نویسی شروع گردد می توان یک فلوچارت را نوشت آنرا مورد ارزیابی قرار داد و در صورت نیاز آنرا اصلاح کرد و سپس روی فلوچارت بعدی در صورت مرتبط نبودن با یکدیگر فکر کرد.

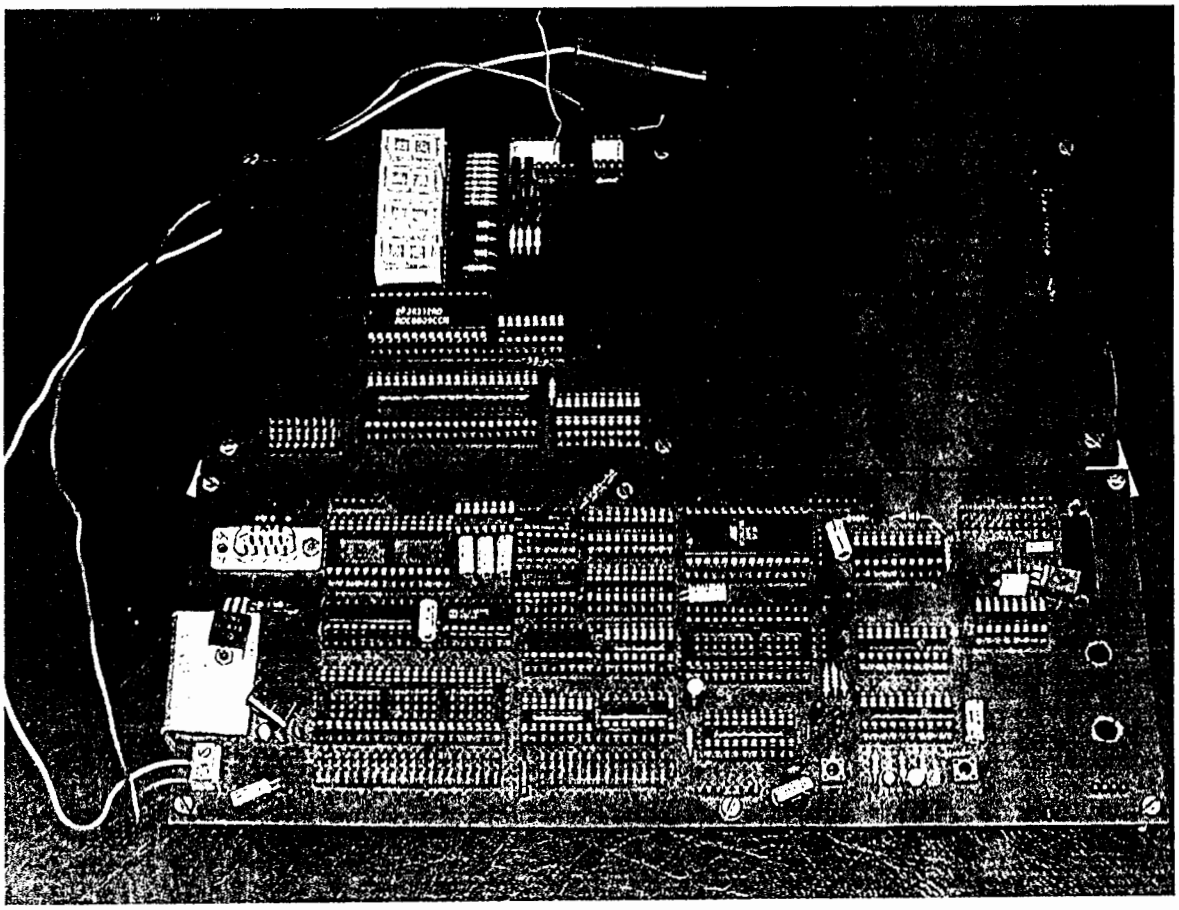
۷-۱۰- تهیه قطعات و طراحی سخت افزار

پس از انجام مطالعات مفصل و تعیین راه حلهای ممکن برای سخت افزار و دریافت پاسخ مناسب برای هرسؤال بایستی قطعات و عناصر مورد استفاده تهیه گردند. طراحی و تهیه قطعات بایستی توأم انجام گیرد بدون در اختیار بودن قطعات و عناصر لازم، طراحی قابلیت اجرا نخواهد داشت. این مسئله کار را مشکل نموده و باعث به عقب افتادن طراحی و ساخت می گردد. در اجرای پروژه های منجر به ساخت با موارد زیاد برخورد می شود که قطعات مورد استفاده در طراحی در بازه زمانی مشخص در اختیار نیست لذا بایستی از قطعات و عناصری استفاده کرد که از منابع مختلف بتوان آنها را تهیه کرد و یا موارد زیادی پیش می آید که در حین طراحی و ساخت به عناصر و قطعات جدیدتر و بهتری می توان دست یافت که به بهبود طراحی بینجامد. مجموعه این عوامل کار طراحی و ساخت و در اختیار بودن قطعات را مشکل می سازد و بایستی در هر صورت آنرا در نظر داشت.

۸-۱۰- ساخت بردهای وایرپ

پس از اینکه قسمت عمده ای از طراحی انجام شد بایستی طراحی انجام شده مورد تست و ارزیابی قرار گیرد. یکی از روشهای مؤثر جهت ارزیابی طراحی انجام شده استفاده از نرم افزارهای شبیه ساز که

روزبـروز بـرتعداد و قابلیـتهای آنهـا افـزوده می شود می باشد. گرچه این نرم افزارها کمک فراوانی به حل مسـئله مـی کنـند امـا تمامـی اشـکالات و نواقص را نمی توان توسط آنها برطرف نمود. از طرف دیگر در اختـیار گرفتـن نـرم افزار شبیه ساز که برای سیستمهای میکروپروسسوری و میکروکنترلری قابل استفاده باشـد براحتی ممکن نبوده و تنها شرکتهای معتبر قادر به تهیه آنها می باشند لذا بهترین راه این است که سـخت افـزار بطـور کـامل پیاده سازی گردد برنامه های لازم نوشته شود و نواقص آن با بکارگیری آن در محیط واقعی نمایش داده شود و برطرف گردند. د راین مرحله است که بسیار اتفاق می افتد طراحی انجام شـده جوابگـو نـبوده و بایسـتی تغیـیرات عمـده ای در آن ایجـاد گردد. برای پروژه فوق سخت افزار لازم بصـورت وایـرپ آمـاده شـد کـه در شـکل ۲-۱۰ نشان داده شده است. قابل ذکر است تغییرات زیادی در سخت افزار برد وایرپ اعمال می گردد تا نتیجه مطلوب بدست آید.



شکل ۲-۱۰- برد وایرپ طراحی شده

۹-۱۰- نوشتن برنامه های تست

پس از ساخت بردهای وایرپ بایستی برنامه های مختلفی جهت تست قسمتهای مختلف سیستم همچون پورت سریال، تایمر، 7_segment ها، صفحه کلید، راه اندازی موتور DC و پله ای، حافظه های *EEPROM*، نوشتن اطلاعات روی *LCD* و ... نوشت. در این زیر برنامه ها هدف تنها تست قسمتی از سیستم است که نشان می دهد ارتباط آن با سخت افزار واقعی برقرار شده است. از آنجایی که طراح در این مرحله با دو مشکل همزمان یعنی درست بودن اتصال سخت افزار و برنامه تست نوشته شده درگیر

می باشد بایستی سعی کرد تا جایی که ممکن است برنامه تست از ساختار ساده و مطمئنی برخوردار باشد بنحوی که از صحت آن مطمئن و عیب سخت افزار را بدرستی نشان دهد. در طراحی سیستمهای میکروپروسسوری بسیار پیش می آید که طراح اشکال را در سخت افزار جستجو می کند در حالی که اشکال در نرم افزار است و همچنین مواقعی پیش می آید که اشکال در نرم افزار است و عیب را در نرم افزار جستجو می کند. این مشکل بزرگی است که تنها با تجربه و سعی و خطا و نوشتن برنامه تست ساده ممکن است قابل حل باشد.

۱۰-۱۰- نوشتن برنامه های آموزشی

پس از ساخت بردهای وایرپ و تست قسمتهای مختلف سیستم و عیب یابی سخت افزار و رفع نواقص، نوشتن برنامه اصلی با توجه به مطالعات انجام شده و فلوچارتهای ترسیم شده شروع می گردد. در نوشتن این برنامه ها از برنامه های تست نوشته شده کمک گرفته می شود. انتخاب زبان برنامه نویسی و رعایت قواعد برنامه نویسی از اهمیت خاصی برخوردار است. نوشتن برنامه های آموزشی به دو بخش برنامه نویسی با اسمبلی و برنامه نویسی با C تقسیم می گردد که قسمت مهمی از کار رابخود اختصاص می دهد. در فصول گذشته این برنامه ها و وظایف آنها به تفصیل آورده شد.

۱۰-۱۱- طراحی *Layout* مدار چاپی

پس از اینکه طراحی انجام شد و با ساخت برد وایرپ، طراحی انجام شده مورد تست و آزمایش قرار گرفت و برنامه های تست و کاربردی نوشته و نواقص برطرف شد و طراحی تقریباً به مرحله نیمه نهایی نزدیک شد بایستی اقدام به ساخت مدار چاپی نمود. ساخت مدار چاپی هنگامی که تعداد قطعات و عناصر زیاد باشد و طراحی پیچیده باشد مشکل است و به تجارب خاصی نیاز دارد. در این پروژه با توجه به سوابق قبلی و اینکه طراحی آن با دست بسیار مشکل و وقت گیر است تصمیم بر این گرفته شد که از نرم افزاری استفاده گردد که بتوان با آن براحتی مدار چاپی را بصورت خود کار طراحی کرد. لذا از نرم افزار *Layout Plus* استفاده شد و بردهای مدار چاپی طراحی گردید.

۱۰-۱۲- سفارش ساخت برد مدار چاپی

پس از طراحی مدار چاپی با نرم افزار *Layout Plus* بایستی اقدامات لازم جهت ساخت آنرا انجام داد. این بردها بصورت دورو و متالیزه بوده و بایستی برای نمونه نهایی ترتیبی اتخاذ کرد که در محیطهای مرطوب مشکلی برای آنها پیش نیامده و اکسید نگردند. ساخت این بردها نسبتاً وقت گیر و هزینه بر بوده و در زمان اجرای پروژه بایستی مورد نظر قرار گیرد. همچنین ممکن است بردهای ساخته شده اشکالاتی نیز داشته باشند و لازم باشد آن اشکالات مرتفع گردند.

۱۰-۱۳- مونتاژ برد مدار چاپی و تست آن

پس از ساخت برد مدار چاپی و تهیه قطعات لازم بایستی قطعات و عناصر مورد استفاده در مکانهای خود نصب گردند. هنگام طراحی و ساخت برد مدار چاپی اشکالات متعددی ممکن است وجود داشته

باشد که تنها با مونتاژ و تست برد ساخته شده مشخص خواهند شد این اشکالات می تواند ۱ـ قطر سوراخهای در نظر گرفته شده برای قطعات مناسب نباشد. ۲ـ جای در نظر گرفته شده برای قطعات کافی نباشد و لازم باشد قطعات جابجا گردند. ۳ـ ترتیب پایه های در نظر گرفته شده برای قطعات مناسب نباشد. ۴ـ نقشه شماتیک اشکالاتی داشته باشد و متناسب با آن برد مدار چاپی درست نباشد مثلاً تغذیه برای تمامی آی سی ها اعمال نشده باشد. ۵ـ طراحی نقص داشته باشد و مواردی در آن لحاظ نشده باشد.

۱۰-۱۴- اصلاح نقشه های شماتیک و مدار چاپی

پس از مونتاژ و مشخص شدن اشکالات که به احتمال زیاد وجود خواهند داشت نقشه های شماتیک و مدار چاپی اصلاح و برد نهایی ساخته می شود. در این برد سخت افزاری تاجایی که ممکن است بایستی از فضای روی برد استفاده کرد و فضای خالی روی برد کم باشد. تجربه طراح و نرم افزار مورد استفاده در این رابطه تعیین کننده است. هرچقدر سطح برد کمتر باشد هزینه ساخت کمتر بوده و به توجیه اقتصادی طرح کمک می کند. اگر تعداد بردهای درخواستی زیاد باشد بایستی یک یا چند نمونه از برد مدارچاپی اصلاح شده تهیه و مونتاژ شود و تستهای لازم انجام گیرد و در صورت وجود هیچگونه نقصی سفارش به تعداد مورد نیاز صورت گیرد.

۱۰-۱۵- ساخت برد نهایی

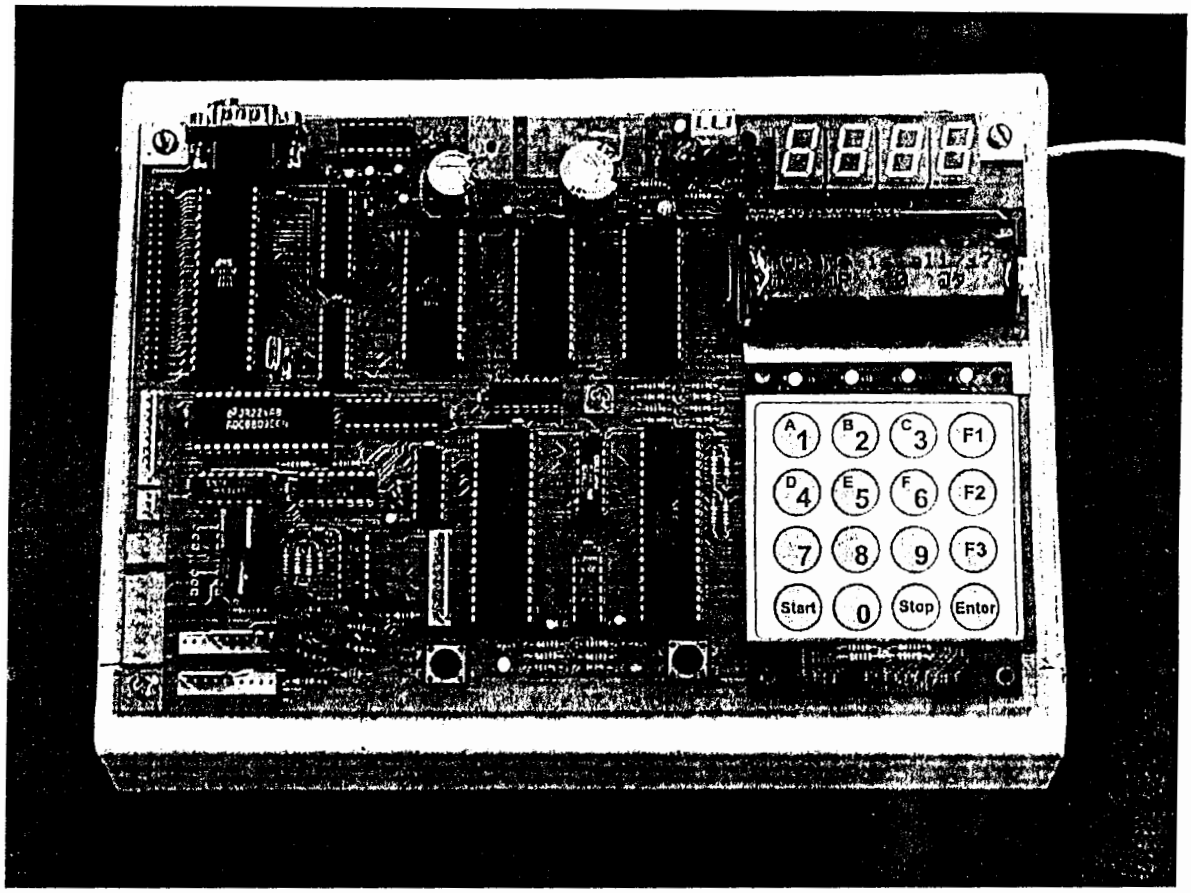
پس از اصلاح نقشه شماتیک و اصلاح مدارچاپی بایستی برد مورد نظر را جهت سفارش نهایی آماده کرد.

۱۰-۱۶- تست و ارزیابی نهایی

پس از مونتاژ برد نهایی بایستی آنرا مورد ارزیابی و تست قرار داده و جواب مطلوب را از آن استخراج کرد. بدلایل متعددی ممکن است برد ساخته شده جوابگوی نیازها نباشد و لازم باشد به مراحل قبل برگشته و روند را تکرار نمود در هر صورت بایستی بردنهایی با مشخصات خواسته شده ساخته شده و مورد تأیید قرار گیرد.

۱۰-۱۷- تهیه و یا ساخت جعبه مناسب

پس از مونتاژ و ساخت برد نهایی بایستی برای برد مورد نظر جعبه مناسب آماده و برد یا بردهای مورد نظر با در نظر گرفتن دیگر شرایط در آن قرار گیرد. برای برد ساخته شده و مونتاژ و تست شده این کار انجام شده و نتیجه در شکلهای ۳-۱۰ نشان داده شده است.



شکل ۳-۱۰- سیستم آموزشی ساخته شده

۱۸-۱۰- نوشتن برنامه های لازم روی کامپیوتر

پس از تهیه سخت افزارها و نوشتن برنامه های لازم در محیط *DOS* و انجام تستهای مختلف، نوشتن برنامه لازم روی کامپیوتر جهت برنامه ریزی و مدیریت در محیط مناسب شروع می گردد. این برنامه با *tourbo C* و ویژوال بیسیک که فلوچارتهای مختلف آن در بخش نرم افزار بیان شد نوشته شده است.

۱۹-۱۰- مستند سازی و گزارش نهایی

پس از اینکه برنامه کامل و جواب مورد نظر حاصل شد ممکن است برنامه موجود برای مدت کوتاه یا طولانی جوابگوی نیازها باشد اما با گذشت زمان انتظارات و خواسته ها تغییر نموده و ممکن است لازم باشد تغییراتی در برنامه ایجاد گردد. برای این منظور بایستی شرح برنامه، نمودار ساختار برنامه، منطق برنامه، لیست برنامه همراه با توضیحات لازم بهمراه طراحی قسمتهای مختلف سخت افزار مورد نقد و بررسی قرار گرفته و دسترس باشد و این ممکن نیست مگر اینکه پس از تکمیل سخت افزار و برنامه ها اقدامات لازم جهت مستند سازی صورت گیرد.

۲۰ـ۱۰ـ نتایج بدست آمده

برای هر پروژه تحقیقاتی اهداف اولیه خاصی مد نظر قرار می گیرد که در مدت اجرای آن سعی می گردد آن اهداف تعقیب گردند. همانطور که قبلاً اشاره شد طراحی و ساخت سیستمی مد نظر بود که آموزش میکروکنترلرهای خانواده *MCS51* با سرعت و هزینه کمتری انجام گیرد. در این بین کاربردی بودن سیستم، هزینه کم، قابل اجرا بودن از اهمیت خاصی برخوردار بود. در این راستا در این گزارش به تعریف پروژه، مزایا و قابلیتها، نحوه عملکرد سیستم، قسمتهای مختلف، مشکلات و راه حلهای ممکن پرداخته شد و از بین راه حلهای ممکن برای قسمتهای مختلف، راه حل مناسب انتخاب و متناسب با آن، طراحی ها بصورت بلوک دیاگرام و سپس با جزئیات کامل انجام شد. پس از طراحی، قسمتهای مختلف سیستم ساخته شد، نرم افزارهای تست و آموزشی روی سخت افزار و کامپیوتر نوشته شد و جواب لازم دریافت گردید. حال سؤالی که مطرح می گردد این است که نتایج حاصل از اجرای این پروژه چیست؟ چه نقاط قوت و ضعفی دارد و برای اجرای آن چه توجیهی وجود دارد؟

نتایج بدست آمده از اجرای پروژه به دو مقوله نتایج پژوهشی و نتایج عملی تقسیم بندی می گردد. در نتایج پژوهشی ابتدا مطالعات و تحقیقات لازم در مورد میکروکنترلرهای خانواده *MCS51* انجام شده، مشخصات شبیه سازهای نرم افزاری و سخت افزاری مطالعه شده و طراحیهای لازم جهت ساخت سیستم آموزشی انجام می گیرد و نتایج حاصل از آن بصورت خلاصه در گزارش ارائه می گردد و علاوه بر این، محیطی مناسب برای انجام آزمایشها و تحقیقات مختلف با ساخت این سیستم فراهم می گردد. اما در نتایج عملی بیشتر به بررسی افزایش تجارب عملی دست اندرکاران با اجرای این پروژه می پردازد و در این بین می توان به موارد زیر اشاره کرد:

۱ـ آشنایی با حافظه های *EEPROM* موازی: قبل از اجرای این پروژه آشنایی کمی حافظه های *EEPROM* موازی وجود داشت. در اجرای این پروژه لازم بود از این قطعات استفاده گردد در ابتدا هدف این بود اطلاعات مربوط به برنامه در حافظه های *RAM* ذخیره گردد برای این منظور سخت افزار لازم طراحی و نحوه استفاده از آن فراگرفته شد اما پس از آشنایی با حافظه های *EEPROM* موازی همچون *AT28C256* ترجیح داده شد از این حافظه ها نیز استفاده گردد. این حافظه ها جهت برنامه ریزی تنها به ولتاژ ۵ ولت نیاز دارند و کار با آنها ساده است و به جهت حفاظت اطلاعات نحوه نوشتن در آنها از الگوریتم خاصی تبعیت می کند. سرعت نوشتن در آنها بحدی است که می توان اطلاعات را از پورت سریال که از کامپیوتر می آید خواند و ذخیره کرد.

۲ـ آشنایی با نحوه طراحی مدار چاپی با نرم افزار ارکد بصورت خودکار: از جمله تجارب با ارزش بدست آمده در حین اجرای پروژه کار با نرم افزار *Layout Plus* از برنامه های نرم افزار ارکد ۹ است. این نرم افزار *NETLIST* حاصل از نرم افزار شماتیک را دریافت و قطعات مورد استفاده را مشخص می کند سپس کاربر می تواند با تعیین اندازه برد و جایگزینی قطعات در مکانهای مناسب از قابلیت خوب طراحی *Layout* بصورت اتومات استفاده کند و مدار چاپی مورد نظر را با صرف وقت کم تهیه نماید. البته لازم بتوضیح است فراگیری این نرم افزار و بکارگیری آن وقت زیادی می گیرد.

۳ـ آشنایی با نرم افزار ***Visual Basic***: برای اجرای نرم افزار روی کامپیوتر لازم بود از نرم افزاری *Visual* استفاده گردد. با توجه به خواسته های مورد نظر و سادگی و در اختیار بودن امکانات لازم تشخیص داده از این نرم افزار استفاده گردد. لذا این نرم افزار مطالعه و مورد استفاده قرار گرفت.

مراجع:

[1] Muhammad Ali Mazidi , Janice Gillispie Mazidi " The 8051 Microcontroller and Embedded Systems " Prentice Hall 2000

[2] I.Scott Makenzi " The 8051 Microcontroller"

[3] James W.Stewart "The 8051 Microcontroller, Hardware, Software and Interfacing"

[4] http://www.atmel.com

[5] http://www.intel.com

[6] http://www.bipom.com/simulator51.htm "8051 Simulator for Micro-IDE"

[7] http://www.crossware.com/datasheets/ s8051nt.htm "8051 Simulator for Windows"

[8] http://www.rotgradpsi.de/mc/8051dev/emu.html "ROM and RAM Emulator for 8051"

[9] http://www.designnotes.com/emulate.htm "8051 Microcontroller Real Time In-Circuit Emulator"

[10] http://scmstore.com/english/micros/ Emulator/8051/default.asp "*8051 In-Circuit Emulator"*

[11] Muhammad Ali Mazidi , Janice Gillispie Mazidi " The 80X86 IBM PC and Compatible Computers" VOL. II Prentice Hall 1998

[۱۲] برنامه نویسی با *C-8051* ، دکتر امیر حسین رضایی، دانشگاه صنعتی امیر کبیر، ۱۳۸۰

[۱۳] آموزش گام به گام ویژوال بیسیک، مهندس عین الله جعفرنژاد قمی، علوم رایانه، ۱۳۸۱