آموزش سریع زبان اسمبلی میکروکنترلر ۸۰۵۱

گرد آوری : امید طالبی

Omidtalebi2003@yahoo.com

مقدمه:

در این جزوه آموزشی سعی شده است که طریقه کار و برنامه نویسی میکرو کنترلر به ساده ترین زبان و با کمترین پیش نیاز آموزش و توضیح داده شود ، مفاهیمی که یک کاربر نیاز دارد در فصل اول توضیح داده شده است اما در این بین فرض بر این بوده که کاربر مفاهیم اولیه مانند ولتاژ و جریان و مقاومت و ... را بداند و تا حدی با منطق های مختلف از جمله باینری یا دودویی آشنا باشد و از نظر الگوریم نویسی و برنامه ریزی نیز مشکلی نداشته باشد توصیه می شود در هنگام مطالعه فصل ۲ را به صورت سطحی یک بار مطالعه نموده و پس از اتمام جزوه نیز یک بار دیگر این فصل را برای درک بهتر مطالعه نمایید امید است با مطالعه این جزوه بتوانید تا حد مورد نیاز با میکروکنترلر کار کرده و پس از اتمام این جزوه یک کاربر میکرو کنترلر باشید در این جزوه در حد بسیار کمی دو نرم افزار **FRANKLIN** و **PROTEUS** که قویترین نرم افزار ها در زمینه ی میکرو کنترلر هستند تشریح شود اما باید دانست که موفقیت بوسیله شبیه سازی با نرم افزار تمام راه نیست و بستن مدار های عملی میکرو کنترلر نیاز به مهارت های عملی بیشتری دارد.

فصل اول :

مفاهیم اولیه ای که باید قبل از مطالعه این جزوه بدانیم :

مبنای دهدهی یا دسیمال : مبنای معمولی اعداد که همواره مورد استفاده قرار می دهیم و جهت ساخت هر عددی در این مبنا از ترکیبات ده جز آن که ۰ ، ۱ ، ۲ ، ۳ ، ٤ ، ٥ ، ٦ ، ۷ ، ۸ و ۹ می باشند استفاده می کنیم.

مبنای دودویی یا باینری : این مبنا فقط ۲ عضو ۰ و ۱ را دارد و هر عددی را تنها با این دو عضو می سازند.

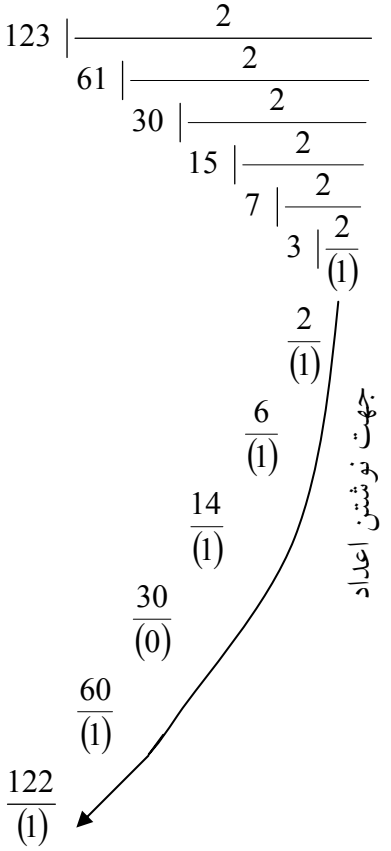
مبنای شانزده یا هگزادسیمال : این مبنا از ١٦ عضو تشکیل شده است که عبارتند از ۰ ، ۱ ، ۲ ، ۳ ، ٤ ، ٥ ، ٦ ، ۷ ، ۸ ، ۹ ، **a** ، **b** ، **c** ، **d** ، **e** و **f** . توجه شود در این مبنا **a** به اندازه ۱۰ واحد دسیمال و **b** به اندازه ۱۱ واحد دسیمال و ... **f** به اندازه ۱۵ واحد دسیمال ارزش دارد .

بیت : هر رقم از اعداد در مبنای ۲ ×

بایت : به مجموعه ۸ رقم از اعداد در مبنای دو ×××× ××××

نیبل : به هر نیم بایت یا چهار بیت یک نیبل گویند. ××××

تبدیل مبنا های مختلف به یکدیگر : جهت تبدیل مبنای دسیمال به باینری باید انرا مرتبا بر دو تقسیم نمود بـه مثال زیر توجه کنید :



$(123)_{10} = (1111011)_2$

در این الگوریتم خارج قسمت هر تقسیم مقسوم تقسیم بعدی است و باقیمانده هر تقسیم یکی از ارقام عدد مورد نظر ما است .

جهت استفاده از کامپیوتر برای تبدیل مبنا ها می توانید از ماشین حساب ویندوز اسـتفاده نماییـد لذا بـه منـوی start رفتـه و در Run کلمـه ی calc را تایپ نمایید تـا ماشـین حـساب ظـاهر شود سپس از گزینه ی view در ماشین حساب scientific را انتخاب کنید تا ماشـین حـساب مهندسی داشته باشید .

حال هر عددی را کـه در ماشـین حـساب وارد کنید به راحتی با استفاده از گزینـه هـای Hex ، Dec ، Bin به یکدیگر تبدیل کنید.

جهت تبدیل اعداد مبنای شانزده به مبنای دو و بر عکس بهترین راه تبدیل آن بـه نیبـل و اسـتفاده از مقـادیر دهدهی آنهاست به عنوان مثال عدد f به مقدار ۱۵ در مبنای ۱۰ ارزش دارد و عدد ۱۵ نیز در مبنـای ۲ مقـدار ۱۱۱۱ می شود توجه شوی که هر رقم مبنای ۱۶ یک نیبل در مبنای ۲ یا چهار بیت است پس آنها را به همـین صورت به یکدیگر تبدیل می کنیم .

مثال :

$$(AF4C)_{16} = (1010\ 1111\ 0100\ 1100)_2$$

برای تبدیل باینری به هگز از سمت راست چهار بیت به چهار بیت جدا می کنیم (از محل فلش ها)

$$(10000101100)_2 = (42C)_{16}$$

هر فردی که با سیستم های دیجیتال کار می کند جهت راحتی کار خود باید بر مقادیر زیر کاملا مسلط بوده و بتواند به راحتی آنها را به کار گیرد پس یاد گیری آنها امر بسیار مهمی می باشد . توجه شود حـروف B و H به ترتیب به معنای باینری و هگز می باشند.

0 = 0000 B = 0 H	1 = 0001 B = 1 H	2 = 0010 B = 2 H
3 = 0011 B = 3 H	4 = 0100 B = 4 H	5 = 0101 B = 5 H
6 = 0110 B = 6 H	7 = 0111 B = 7 H	8 = 1000 B = 8 H
9 = 1001 B = 9 H	10 = 1010 B = a H	11 = 1011 B = b H
12 = 1100 B = c H	13 = 1101 B = d H	14 = 1110 B = e H
	15 = 1111 B = f H	

اعداد BCD : به اعداد ۰ تا ۹ که به مبنای ۲ برده شوند اعداد BCD گویند.

کیلو بایت : ۲ به توان ۱۰ بایت (۱۰۲٤ بایت)

مگابایت : ۲ به توان ۱۰ کیلو بایت (۱۰۲٤ کیلو بایت)

RAM یا حافظه موقت : به حافظه ای می گویند که اطلاعات را به راحتی می توان روی آن قرار داد و به راحتی نیز پاک می شوند این حافظه جهت نگهداری اطلاعات نیاز به تغذیه دارند و در صورتی که تغذیه آنها قطع شود اطلاعات روی آنها پاک می شود.

ROM یا حافظه دایمی : به حافظه ای می گویند که اطلاعات آن دایمی است و با قطع برق از بین نمی رود به همین جهت برنامه یا اطلاعاتی که دایما مورد نیاز است داخل این حافظه ریخته می شوند.

ROM انواع مختلفی دارد که عناوین مهمترین آنها که در میکرو ها کاربرد دارند عبارت است از:

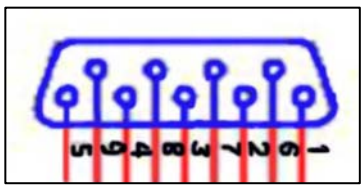
PROM که یک بار و فقط در زمان تولید برنامه ریزی می شود و دیگر قابل برنامه ریزی نیست.

EPROM (Erasable Programable ROM) که با استفاده از اشعه ماورا بنفش پاک شده و دوباره می توان آن را برنامه ریزی نمود.

EEPROM (Electricall Erasable Programable ROM) که به وسیله جریان الکتریکی پاک شده و قابل برنامه ریزی مجدد است.

FLASH ROM که بوسیله جریان الکتریکی قابل برنامه ریزی و پاک شدن است از خواص این نوع سرعت بالای تبادل اطلاعات و برنامه ریزی می باشد.

پورت سریال : سریال سازی اطلاعات یعنی اینکه اطلاعات یک بایت را به صورت یک بسته به ترتیب پشت سر هم قرار داده و جهت انتقال از آن استفاده کنیم که پورت سریال عمل انتقال و بسته بندی اطلاعات را بر عهده دارد مثلا جهت تبادل اطلاعات بین کامپیوتر و میکرو کنترلر از پورت سریال استفاده می کنیم که پورت سریال کامپیوتر یا COM در پشت کامپیوتر و ۹ پایه دارد.



البته غیر از دو نوع آخر انواع قبلی میکرو تقریبا منسوخ شده و دیگر تولید نمی شود.

عملگر الکتریکی یا الکترونیکی : به وسیله ای می کویند که با گرفتن سیگنال الکتریکی عملی خاص را انجام می دهد که این عملگر ها انواع متفاوتی دارند برخی از آنها فقط خاموش یا روشن می شوند برخی دیگر میزان نور یا سعت یا گرما یا ... در آنها به صورت پیوسته کم یا زیاد می شود به بیان دیگر عملگر های آنالوگ و دیجیتال را داریم دقت داشته باشیم که تقریبا هر وسیله ای میتواند عملگر باشد حتی یک لامپ.

سنسور : به قطعه ای گفته می شود که شرایط محیط را به سیگنال های الکترونیکی تبدیل می کند و از آن طریق می توان آنرا به عملگر ها و اندازه گیر ها داد سنسور ها انواع مختلفی برای اندازه گیری کمیت های مختلف دارند مانند سنسور های دما ، رطوبت ، نور و.... که بسته به نوع مصرف در رنج های مختلف به کار برده می شوند.

منطق دیجیتال : منطقی که بر اساس صفر و یک (باینری) یا به عبارت دیگر قطع و وصل یا خاموش و روشن کار می کند به عبارت دیگر نیز دیجیتال به معنای جدا از هم یا گسسته می باشد.

آنالوگ : به هر چیزی یا سیستمی که مقدار آن به صورت پیوسته تغییر کند مثل نور یک لامپ که با ولتاژ های مختلف نور متفاوتی دارد یا میزان دمای یک اتاق که می توان آنرا یا یک سنسور به ولتاژ یا جریان الکتریکی تبدیل کرد اما توجه شود که این ولتاژ یا جریان را می توان یا یک A/D به مقادیر دیجیتال تبدیل کرد و عکس این عمل نیز با یک D/A امکان پذیر است.

A/D : تبدیل کننده ی ولتاژهای آنالوگ به دیجیتال ، آین قطعه انواع مختلفی دارد و از نظر تعدادبیتهای خروجی تقسیم بندی می شود که می تواند ۸ یا ۱۲ یا ١٦ بیتی باشد این قطعه به این صورت عمل می کند که یک ولتاژ مرجع بالا و یک ولتاژ مرجع پایین از ما گرفته و خروجی را بر آن اساس و بر اساس تعداد بیتهای خروجی تقسیم بندی می کند مثلا فرض کنید ولتاژ مرجع مثبت را ٥+ و ولتاژ مرجع منفی را ٥- در نظر بگیریم برای یک A/D که هشت بیت خروجی دارد چون A/D هشت بیتی است ۲ به توان ۸ به اضافه یک حالت می توانیم در خروجی داشته باشیم یعنی ٢٥٦ حالت ، پس از ٥- تا ٥+ را به ٢٥٦ قسمت مساوی تقسیم می کند یعنی ۱۰ تقسیم بر ٢٥٦ پس به ازا هر۰,۰۳۹۰٦۲٥ یا تقریبا هر ۰,۰٤ ولت افزایش در ورودی در خروجی یک واحد دیجیتال افزایش داریم برا فهم بهتر به مثال دیگری در این زمینه توجه کنید :

A/D داریم که ۱۲ بیتی بوده و ولتاژ مرچع مثبت آن را ٨+ و ولتاژ مرجع منفی آنرا ٦- ولت داده ایم پس داریم :

$$2^{12} = 4096 \qquad 8-(-6) = 14 \Rightarrow 14/4096 \approx 0.00342$$

پس به ازای هر ۰,۰۰۳٤۲ ولت افزایش ولتاژ در ورودی یک واحد دیجیتال به خروجی افزوده می شود و توجه شود که ولتاژ مر جع منفی ما ٦- ولت است پس ولتاژ ۳- ولت در ورودی عدد ۰۰۱۱۰۱۱۰۱۱۰۱ را در خروجی ایجاد می کند به این صورت که :

$$-3-(-6) = 3 \Rightarrow 3/0.00342 \approx 877 \Rightarrow 001101101101\ \mathbf{B}$$

D/A : عکس عمل **A/D** را انجام می دهد یعنی یک ورودی دیجیتال گرفته و خروجی آنالوگ را با توجه به ولتاژ های مرجع به ما می دهد خروجی **D/A** معمولا جریان است که بایک مقاومت مناسب می توان آنرا به ولتاژ تبدیل نمود.

رجیستر یا ثبات : به قسمتی در میکرو گفته می شود که به مانند یک بایت از رم عمل می کند و می توان اطلاعات را روی آن قرار داد اما رجیستر ها اسامی خاصی دارند و در برنامه نوژسی از آنها استفاده می شود مانند ثبات **A** یا **B** یا **R0** و ...

باس : به بخشی گفته می شود که معمولا از سیم یا کانکتور تشکیل شده و وظیفه آن رساندن اطلاعات یا فرمان از جایی به جای دیگر است.

دیتا باس : بخشی در داخل میکرو کنترلر که اطلاعات از آن طریق حمل و به یکدیگر منتقل می شود این بخش در یک میکرو کنترلر ۸ بیتی متشکل از ۸ سیم فرضی در داخل میکرو است که **RAM** و **ROM** و **I/O** ها رجیستر ها و ... را به هم متصل می کند و اطلاعات را در میان آنها منتقل می کند.

میکرو کنترلر : برای اینکه به زبان ساده تر بفهمیم یک میکروکنترلر چیست باید بگوییم این یک قطعه الکترونیکی است که می توان با نوشتن برنامه ای خاص عملی را از آن بخواهیم گفتیم که میکرو تعدادی ورودی و خروجی دارد که از آن طریق می تواند با دنیای بیرون از میکرو ارتباط برقرار کند و بفهمد مثلا فلان کلید زده شده پس باید فلان عمل را انجام دهد و این عمل را با استفاده از پایه های خروجی انجام می دهد میکرو از این نظر که قابلیت برنامه نویسی دارد و به راحتی می توان آنرا به صورت دلخواه به کار برد وسیله بسیار قوی و پر کاربرد در زمینه های دیجیتال و حتی انالوگ دارد.

میکروکنترلر قطعه ای الکترونیکی است که با منطق دیجیتال کار می کند و از یک پروسسور یا پردازشگر مرکزی و تعدادی **Port** یا درکاه ورودی و خروجی اطلاعات و **RAM** یا حافظه موقت و ... تشکیل شده است .میکرو کنترلر ها معمولا دارای **ROM** یا حافظه دائمی داخلی هستند که به مقادیر معینی گنجایش دارد.

میکروکنترلر معمولا جهت کاربرد هایی است که فقط یک کار از آن می خواهند به عنوان مثال جهت ساخت وسایل موسیقی ، کنترل دور موتور **DC** یا **AC** ، قفل الکترونیکی ، دزدگیر ، تابلو های نویسنده ، ربات های با عملکرد محدود و ... از میکرو کنترلر بهره گرفته می شود .

داخل میکرو کنترلر تعدادی ثبات یا رجیستر وجود دارد که بسته به نوع میکرو ۸ یا ١٦ بیتی باشد که اکثر میکرو های پر کاربرد ۸ بیتی می باشند مثلا میکرو کنترلر ۸۰۵۱ یک میکروی با دیتا باس ۸ بیتی و چهار درگاه ورودی و خروجی ۸ بیتی می باشداین میکرو ٤٠ پایه دارد که ۳۲ عدد از آنها ورودی خروجی یا **I/O** هستند.

امروزه میکرو ها از شرکتهای مختلفی در بازار موجود می باشد که عملکرد برخی از آنها واقعا درخور توجه است مانند میکرو های **PIC** یا **AVR** که نسبت به سری ۸۰۵۱ عملکرد های بسیار زیاد تری

دارند و برنامه نویسی آنها نیز بسیار راحت تر شده است اما علت اینکه سری ۸۰۵۱ هنوز به صورت بی رقیب در بازار باقی مانده است این است که اولا از نظر قیمت بسیار ارزان قیمت تر از انواع دیگر است و برای کارهای ساده و عادی نیز عملکرد مناسبی دارد و هر کسی که کار با ۸۰۵۱ را بلد باشد به راحتی می تواند کار با دیگر میکرو ها را یاد بگیرد چون این میکرو از میکرو های اولیه است و بقیه میکرو ها افزودن امکانات به این سری است.

میکروکنترلر ۸۰۵۱ را برای اولین بار در سال ۱۹۸۱ شرکت **Intel** ساخت و به شرکتهای دیگر نظیر **Atmel** ، **AMD** ، **Siemens** ، **Motorela** و ... امتیاز آنرا در صورتی که از زبان اسمبلی ۸۰۵۱ استفاده کنند فروخت یکی از معیار های انتخاب میکروکنترلر در دسترس بودن پروگرمر آن است و امروزه اکثر میکروکنترلرها نظیر **AVR** و **AT89S52** از شرکت **Atmel** و **PIC** فقط نیاز به یک پروگرمر بسیار ساده یا به عنوان دیگر یک **Download Cable** یا سوکت برنامه ریزی دارند.

مشخصات ۸۰۵۱ : ٤ درگاه موازی ۸ بیتی ، ۱۲۸ بایت **RAM** ، ٤ کیلو بایت **EPROM** دو تایمر و ٦ منبع وقفه دارد.

۸۰۵۲ همان ۸۰۵۱ با ۲٥٦ بایت **RAM** و سه تایمر و ۸ کیلوبایت **EPROM**

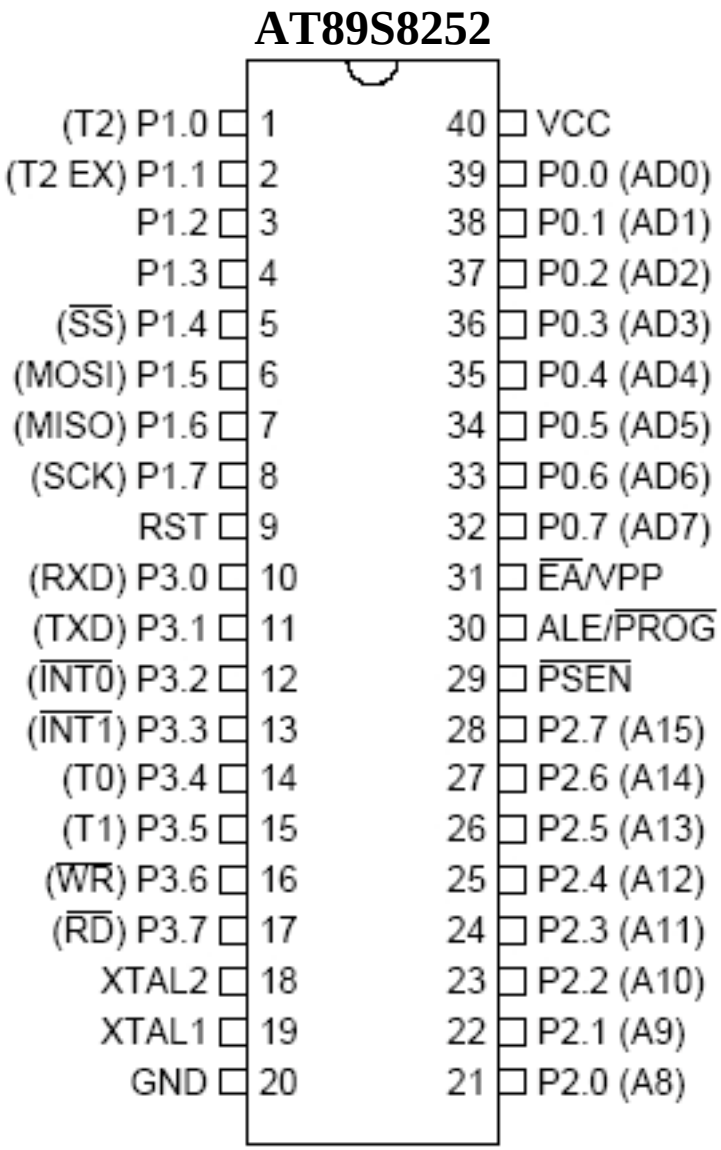
۸۰۳۱ امروزه کاربردی ندارد چون **ROM** ندارد.

میکرو های ساخت **Atmel** :

	ROM (K)	RAM (B)	I/O	TIMER	VCC
AT89C51	٤	۱۲۸	۳۲	۲	٥
AT89C52	۸	۲٥٦	۳۲	۳	٥
AT89LV51	٤	۱۲۸	۳۲	۲	۳
AT89C2051	۲	۱۲۸	۱٥	۲	۳
AT89LV52	۸	۱۲۸	۳۲	۳	۳
AT89S52	۸	۲٥٦	۳۲	۳	٥
AT89S8252	۲K EEPROM DATA MEMORY ۸K FLASH	۲٥٦	۳۲	۳	٥

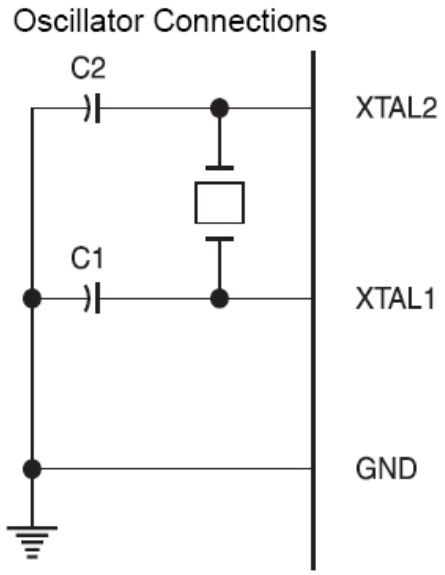
فصل دوم :

پایه ها و مدار داخلی ۸۰۵۱ :



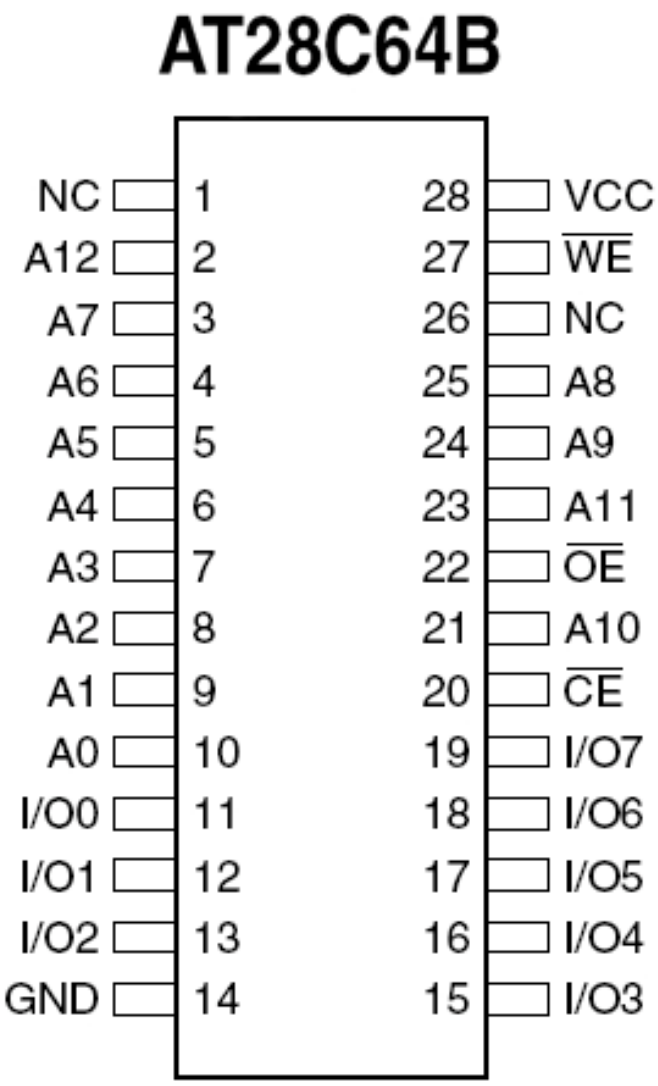
همانگونه که مشاهده می شود **AT89S8252** که درباره مشخصات آن قبلا توضیح دادیم یک میکروی ٤٠ پایه است که از نظر ساختار میکرو با اکثر میکرو های سری ۸۰۵۱ که٤٠ پایه دارند ساختمان مشابهی دارد.

پورت صفر ویک و دو و سه که ۳۲ پایه را تشکیل می دهند ، پایه **VCC** که باید به ولتاژ ٤+ تا ٦+ وصل شود پایه **GND** که باید به ۰ ولت یا زمین مدار وصل شود ، پایه **RST** که باید که برای ریست کردن و راه اندازی مجدد میکرو است که باید به صفر متصل باشد مگر زمانی که می خواهیم میکرو را ریست کنیم این پایه باعث می شود میکرو برنامه خود را از خط اول اجرا کند ، پایه های **XTAL 1,2** که باید به کریستال متصل شوند ، کریستال قطعه ای است که برای تولید کلاک مورد نیاز برای میکرو به کار می رود و همانگونه که در شکل مشاهده می شود باید دو سر آنرا با دو خازن ۳۳ پیکو فاراد به زمین متصل نمود کریستال می تواند فرکانسهای متفاوتی ایجاد کند که معمولا برای میکرو کنترلر ۸۰۵۱ از کریستال با فرکانس ۱۱,۰۵۹۲ مگاهرتز استفاده می شود توجه شود که میکرو برای انجام هر دستور یک زمانی نیاز دارد که این زمان برای هر دستور ممکن است یک یا دو یا حتی سه سیکل باشد باید دقت کرد که سیکل ماشین در ۸۰۵۱ برابر دوازده تقسیم بر فرکانس اسیلاتور (کریستال) است این به این معنی است که میکرو فرکانس کریستال را تقسیم بر ۱۲ می کند و



این به ساختمان داخلی میکرو مربوط می شود یعنی اگر ما کریستال ۱۲ مگا هرتز به میکرو متصل کنیم فرکانس داخلی میکرو ۱ مگا هرتز و هر سیکل میکرو ۱ میکرو ثانیه خواهد بود پس میکرو دستوراتی که در یک سیکل ماشین انجام می شوتد را در یک میکرو ثانیه انجام می دهد.

پایه **$\overline{EA}$/VPP** یا **(External Access)** یک پایه فعال صفر است یعنی در صورتی که به صفر یا زمین متصل شود عمل می کند و خط بالای **$\overline{EA}$** نیز حاکی از همین امر است (در زمینه الکترونیک دیجیتال همه ی پایه های فعال صفر با خط بالای سرشان مشخص می شوند) این پایه به معنای دسترسی خارجی زمانی استفاده می شود که بخواهیم از **ROM** خارجی استفاده کنیم یعنی میکروی ما **ROM** ندارد یا این **ROM** داخلی گنجایش برنامه ما را ندارد و **VPP** نیز فقط توسط پرگرامر برای برنامه ریزی میکرو استفاده می شود پس در صورتی که **ROM** خارجی برای برنامه میکرو نداریم باید این پایه به **VCC** وصل باشد ، پایه **$\overline{PSEN}$** یا **(Program Store Enable)** فعال ساز حافظه خارجی است این پایه زمانی به کار میرود که ما برای برنامه میکرو از حافظه خارجی بهره گرفته ایم در این صورت این پایه باید به **$\overline{OE}$** یا **$\overline{CE}$** از آن **ROM** متصل بوده تا زمانی که لازم است خروجی **ROM** را فعال کرده و اطلاعات را از آن بخواند برای فهم بهتر لازم است تا پایه های یک **ROM** که از نوع **EEPROM** می باشد را توضیح دهیم این سری یعنی **EEPROM** با عدد ۲۸ شروع می شود مانند **28C64** در صورتی که **EPROM** با ۲۷ شروع می شود در بالا توجه شود که حرف **C** نمایش گر این است که در این **IC** از تکنولوژی **CMOS** استفاده شده است و عدد ٦٤ نیز نمایشگر این است که ٦٤ کیلو بیت حافظه داریم و چون این **ROM** خروجی ۸ بیتی دارد پس ٦٤ تقسیم بر ۸ یعنی ۸ کیلو بایت حافظه داریم به همین صورت می توان دید که **28C16** یک **EEPROM** با ۲ کیلو بایت ظرفیت حافظه است.



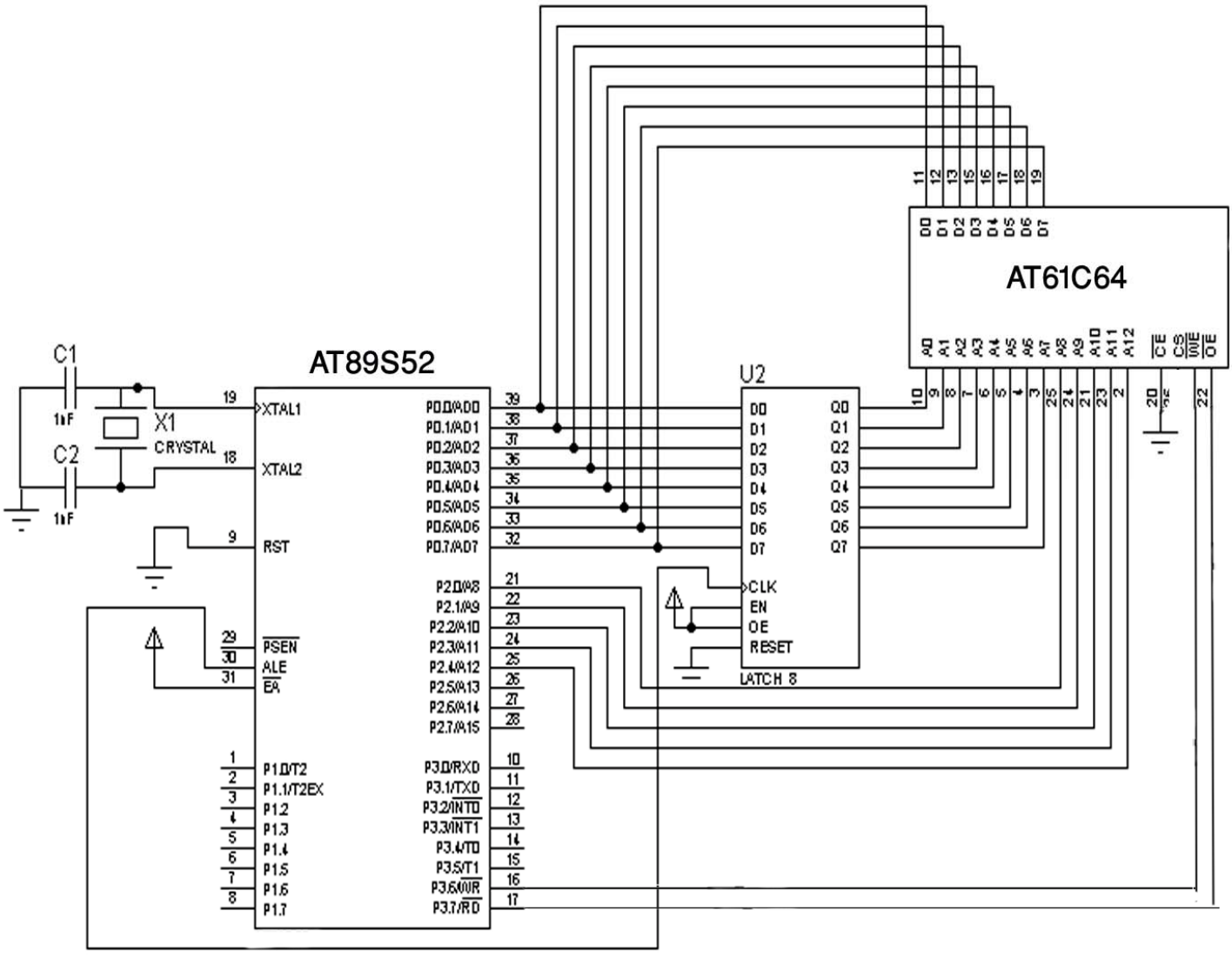
حال به معرفی پایه های آن می پردازیم این **IC** یک **EEPROM** با ۸ کیلو بایت حافظه است باید بدانیم که هر بایت نیاز با یک آدرس خاص است تا در زمانی که می خواهیم به آن دسترسی داشته باشیم بتوانیم از آن آدرس استفاده کنیم ما ۸ کیلو بایت یعنی ۸۱۹۲ بایت داریم پس نیاز به ۸۱۹۲ آدرس داریم این آدرس ها توسط پایه های آدرس باس یعنی **A0** تا **A12** مشخص میشود حال باید بدانیم چرا ۱۳ پایه برای آدرس نیاز

داریم ، چونکه به ۸۱۹۲ آدرس مختلف نیاز داریم و ۱۳ پایه مختلف می توانند ۲ به توان ۱۳ حالت مختلف برای ما ایجاد کنند یعنی ۸۱۹۲ حالت به همین روش دیده می شود که یک **ROM** دو کیلو بایتی به ۱۱ خط آدرس نیاز دارد زیرا ۲ به توان ۱۱ برابر ۲۰۴۸ می شود. پایه های **I/O0** تا **I/O7** نیز ۸ بیت دیتا باس خروجی ، ورودی **ROM** ما هستند که اطلاعات را از آن می خوانیم و در موقع پروگرم کردن اطلاعات روی آن می ریزیم ، پایه $\overline{\text{WE}}$ یا **(Write Enable)** در مو قع پروگرم کردن میکرو به کار می رود و در مواقع دیگر کاربردی ندارد پس باید آنرا به **VCC** وصل نمود ، پایه های **VCC** و **GND** تغذیه های **ROM** هستند و باید به ترتیب به ۵+ و ۰ ولت متصل شوند ، پایه های **NC** یا **(No Connect)** به هیچ جایی متصل نیست و برای حفظ شکل ظاهری در **IC** گذاشته شده است ، پایه های $\overline{\text{OE}}$ یا **(Output Enable)** و **CE** یا **(Chip Select)** برای فعال سازی **IC** و خروجی آن به کار می روند و در صورتی که به **ROM** نیازی نداریم می توان آنها را به **VCC** متصل نمود.

به سراغ آخرین پایه از میکرو کنترلر $\overline{\text{ALE/PROG}}$ یا **(Address Latch Enable/PROGram)** برمی گردیم این پایه فعال ساز رجیستر آدرس است یعنی وقتی که داریم از حافظه خارجی چه برای برنامه میکرو و چه برای ذخیره کردن اطلاعاتی روی حافظه خارجی استفاده می کنیم این پایه به اندازه یک پالس فعال شده و آدرس را روی ثبات آدرس قرار می دهد برای فهم بهتر این مطلب نیاز داریم که چند چیز را بدانیم یکی اینکه ما در میکرو به دو صورت از حافظه استفاده می کنیم اول به صورت **Code Memory** که این حافظه برای ریختن برنامه بر روی آن استفاده می شود دوم به صورت **Data Memory** که برای ریختن اطلاعات روی آن مورد استفاده قرار می گیرد به عنوان مثال فرض کنیم دستگاهی می خواهیم که هر ۱۰ دقیقه دمای یک اتاق را گرفته وآنرا در جایی ذخیره می کند این دستگاه یک میکرو کنترلر دارد که برنامه ای را برای آن نوشته ایم و بر روی آن ریخته ایم که ما برنامه را روی حافظه کد ریخته ایم اما این که برنامه اطلاعات را برای ما ذخیره می کند نیاز به یک **RAM** یا حافظه قابل نوشتن داخلی یا خارجی دارد که به آن حافظه اطلاعات می گوییم پس مشاهده می شود که **RAM** داخلی میکرو نیز یک حافظه اطلاعات است خوب حال باید ببینیم که اگر ما یک حافظه اطلاعات خارجی داشتیم چگونه باید آنرا به میکرو متصل کنیم ، ما می توانیم دیتا باس آنرا به یک پورت متصل کرده و آدرس باس آنرا به دو پورت دیگر و پایه های کنترلی نظیر **RD** و **WR** را هم به یک پورت دیگر از میکرو ، اما همان طور که دیده می شود این عمل بسیاری از پایه های میکرو ما را مشغول کرده است و اگر برای عملی دیگر به پایه ای نیاز داشته باشیم به کمبود مواجه خواهیم شد پس چه باید کرد ؟ دیده می شود که در کنار پایه های پورت صفر میکرو کنترلر نوشته شده است **AD0** تا **AD7** این به چه معنی است در کنار پایه های پورت ۲ نیز نوشته شده **A8** تا **A15** ، اینها پایه های آدرس یاس و دیتا باس داخلی میکرو هستند که به وسیله پایه های پورت صفر و پورت دو که دو منظوره هستند به بیرون آورده شده اند تا ما بتوانیم در مواقعی که به آنها احتیاج داریم از آن استفاده کنیم پایه های پورت ۲ که مشخصند که پایه های آدرس باسند اما پایه های پورت صفر آدرس باس و دیتا باس به صورت

مشترکند یعنی زمانی که لازم است دیتا باسند و زمانی که لازم است آدرس باسند و در این زمان است که **ALE** فعال می شود و به یک **IC** خارجی فرمان می دهد که آدرس را نگه دارد و وقتی که آن ثبات آدرس را در خود نگه داشت پورت صفر به دیتا باس تبدیل می شود پس پایه **ALE** باید به کلاک یک ثبات ۸ بیتی مانند ۷٤۱۹۹ وصل شده و پایه های پورت صفر به ورودی ۷٤۱۹۹ و ورودی دیتاباس حافظه اطلاعات و پایه های پورت دو به میزان مورد نیاز به آدرس باس حافظه اطلاعات.

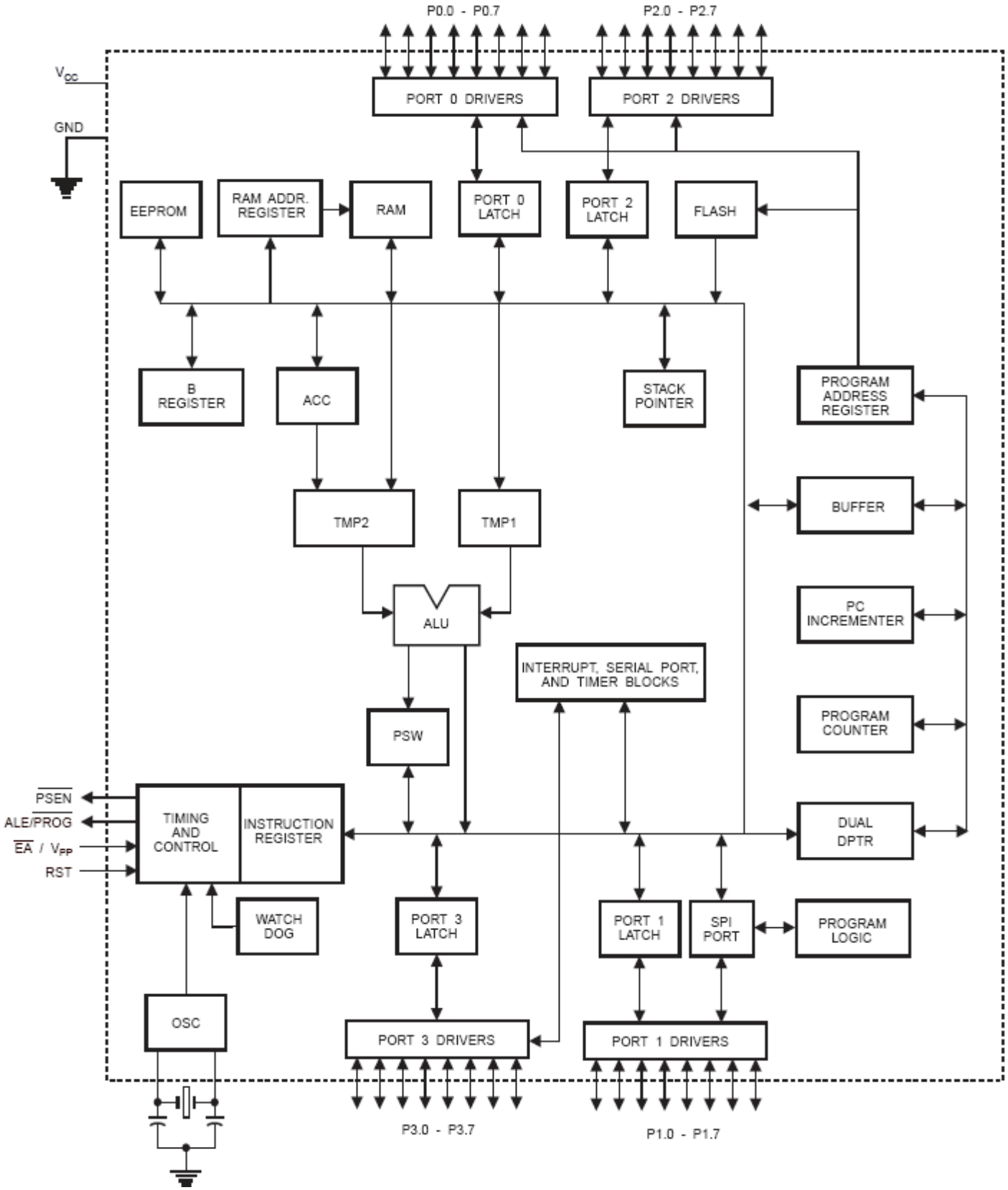
شکل زیر اتصال یک **RAM** هشت کیلو بایتی **AT61C64** را به میکرو نشان می دهد.



در شکل بالا دقت شود که پایه های $\overline{CE}$ فعال شده و پایه **ALE** به کلاک رجیستر آدرس متصل شده و پایه های $\overline{OE}$ به $\overline{RD}$ و $\overline{WE}$ به $\overline{WE}$ از پورت ۳ میکروکنترلر متصلند دیده می شود که پایه های پورت ۳ نیز پایه هایی دو منظوره هستند که **RXD** و **TXD** به ترتیب برای دریافت و ارسال اطلاعات به صورت سریال می باشند ، پایه های $\overline{INT0}$ و $\overline{INT1}$ وقفه های خارجی هستند که بعدا درباره وقفه توضیح داده خواهد شد

، پایه های **T0** و **T1** به ترتیب خروجی های تایمر صفر و یک هستند که در فصول بعدی توضیح داده خواهد شد کاربرد پایه های $\overline{\mathbf{RD}}$ و $\overline{\mathbf{WR}}$ نیز در هنگام استفاده از حافظه ی خارجی است که پایه $\overline{\mathbf{RD}}$ باید به $\overline{\mathbf{RD}}$ یا $\overline{\mathbf{OE}}$ از حافظه و $\overline{\mathbf{WR}}$ باید به $\overline{\mathbf{WR}}$ یا $\overline{\mathbf{WE}}$ از حافظه متصل شود.

در شکل زیر می توانید ساختمان داخلی میکرو کنترلر را مشاهده و آنرا بررسی نمایید.



فصل سوم

ثبات ها و دستورات ۸۰۵۱ :

در ۸۰۵۱ ما چندین ثبات داریم که مهمترین آنها **A** یا **ACC** میباشد ، ثبات های دیگر ۸۰۵۱ که با آنها به صورت متغیر عادی برخورد میکنیم عبارتند از **B** ، **P0** تا **P3** که همان پورت های صفر تا سه ما هستند و **R0** تا **R7** که توجه شود که این ثبات ها مکانی در **RAM** داخلی میکرو هستند و ثبات های **R0** تا **R7** در چهار بانک موجود می باشند به این معنی که در ثباتی به نام **PSW** ما ۲ بیت داریم که با آنها می توانیم شماره بانک متغیر های **R0** تا **R7** را مشخص کنیم پس چهار بانک صفر و یک و دو و سه را داریم ، در میکرو کنترلر به دو صورت می توانیم با یک متغیر کار کنیم یکی بیتی و یکی بایتی ، به این صورت که بعضی از دستورات میکرو بیتی هستند و فقط با یک بیت کار می کنند مانند **SETB** که فقط بر یک بیت اثر می کند و آنرا یک می کند ، اکثر ثبات های میکرو آدرس پذیر بیتی می باشند و اگر بخواهیم بیت مثلا اول **A** را یک کنیم می نویسیم **SETB ACC.0** یا اگر بخاهیم بیت هشتم پورت ۱ را یک کنیم می نویسیم **SETB P1.7** یعنی با یک نقطه بعد از نام متغیر و سپس شماره بیت آن بیت را معین می کنیم البته باید توجه داشت که همه ی ثبات ها در میکرو آدرس پذیر بیتی نیستند و آنهایی که هستند غیر از ثبات های **A** و **P0** تا **P3** معمولا برای هر بیت نام معینی دارند مانند بیت **C** که بیت **CARRY** در محاسبات جمع و ضرب و ... است بیت **CARRY** همان بیت سر ریز می باشد یعنی مثلا ما اگر دو عدد ۲۵۰ را با ۱۰ جمع کنیم جواب آن ۲۶۰ است که از ۲۵۵ یعنی ۸ بیت یک (۱۱۱۱۱۱۱۱) بیشتر است پس بیت **C** یک می شود و ۲۵۵ واحد از جواب کم می کند و آنر نشان می دهد پس ۲۶۰ را به صورت ۰۰۰۰۰۱۰۰ با بیت سر ریز یک نشان می دهد به عبارت دیگر این بیت نهم می باشد که به این صورت نشان داده شده است یعنی اگر ۲۶۰ را در ماشین حساب ویندوز به صورت باینری ببینیم می بینیم ۱۰۰۰۰۰۱۰۰=۲۶۰ که دیده می شود بیت نهم همان سرریز ما است حال به سراغ دستورات میکرو می رویم.

MOV dest,source : ای دستور مقدار **source** را در **dest** می ریزد ، این دستور یک دستور بایتی است و **source** و **dest** هر ثباتی از میکرو و هر عددی می توانند باشند.

مثال :

MOV R0,A : که بر روی دو ثبات غیر هم جنس انجام می شود و مقدار **A** را در **R0** می ریزد باید دقت کرد که متغیر ها نباید هم جنس باشند به این معنی که نمی توانیم بنویسیم **MOV R3,R1** چونکه هر دو از یک نوعند برای چنین کاری باید از یک متغیر واسطه استفاده نمود و بنویسیم **MOV A,R1** و سپس **MOV R3,A** که به مانند دستور بالا عمل می کند.

MOV B,#3 : که مقدار ۳ را در **B** می ریزد علامت # به این معنی است که یک مقدار عددی را در **B** می ریزیم دقت شود که این عدد به صورت دسیمال است و اگر مثلا بنویسیم **MOV A,#5AH** مقدار **5A** را به صورت هگزا دسیمال در **A** می ریزد (علامت **H** به معنای هگزا دسیمال است) همچنین اگر بنویسیم

MOV A,#11011001B مقدار ۱۱۰۱۱۰۰۱ را به صورت باینری در **A** می ریزد البته هیچ تفاتی بین حالت های بالا وجود ندارد و دادن اعداد به صورت هگز یا باینری یا دسیمال بستگی به این دارد که استفاده کدام ساده تر می باشد پس همانگونه که ما ۱۱۰۱۱۰۰۱ باینری را در **A** ریختیم می توانستیم معادل دسیمال یا هگزادسیمال آنرا یعنی مقدار ۲۱۷ دسیمال یا مقدار **D9** هگز را در **A** بریزیم که همگی به یک اندازه هستند. البته مشکلی که اسمبلر ها دارند این است که نباید بعد از علامت # حرف گذاشته شود یعنی نمی توانیم بنویسیم **MOV A,#D9H** و باید بنویسیم **MOV A,#0D9H** یعنی یک صفر قبل از **D** بگذاریم که اسمبلر اشکالی نگیرد.

MOV R3,20H : همانطور که دیده می شود این دستور علامت # را قبل از عدد نگذاشته است به این صورت از دستور آدرس دهی مسقیم می گویند یعنی مقداری که در آدرس **20H** از **RAM** وجود دارد را در **R3** می ریزد یا مثلا دستور **MOV B,A2H** مقداری که در آدرس **A2** هگزادسیمال از **RAM** وجود دارد را در **B** می ریزد.

MOV 34H,A : این دستور مقدار **A** را در آدرس **34** هگز از **RAM** می ریزد.

MOV 22H,#10H : این دستور مقدار ۱۰ هگز را در آدرس ۲۲ هگز از **RAM** می ریزد.

MOV A,@R0 : فرض کنیم از قبل مقدار **35H** را در **R0** ریخته ایم این دستور باعث می شود که مقداری که در آدرس **35H** از **RAM** است در **A** ریخته شود ، به این روش آدرس دهی غیر مستقیم گفته می شود زیرا اول آدرس را در یک ثبات دیگر ریخته و سپس به وسیله آن از آدرس خوانده ایم علامت **@** نیز اشاره به آدرس می کند البته این نوع نوشتن فقط برای **R0** و **R1** عمل می کند و هیچ ثبات دیگری را نمی توان به این صورت به کار برد.

MOVX A,132FH : این دستور برای خواندن یا نوشتن بر روی حافظه خارجی است به این ترتیب که همانگونه که در فصل قبل توضیح داده شد اگر خواستیم از حافظه داده خارجی بخوانیم یا بر روی آن بنویسیم باید اول آدرس بر روی پورت های صفر و دو قرار گرفته و سپس **ALE** فعال شده و ... که اگر ما از دستور **MOV A,132FH** استفاده کنیم این اعمال انجام نشده و میکرو تصور می کند که آدرس **132FH** آدرسی از **RAM** داخلی میکرو است پس در مواقعی که حافظه داده خارجی داریم باید از دستور **MOVX** استفاده کنیم که به این وسیله هم می توانیم از حافظه خارجی بخوانیم و هم می توانیم بر روی آن بنویسیم. توجه شود اگر در میکرو کنترلری مانند **AT89S8252** حافظه داده داخلی نیز داشتیم باید برای خواندن و نوشتن بر روی آن از این دستور استفاده کنیم به این ترتیب از حافظه ی داده در داخل این میکرو می توانیم به عنوان **RAM** استفاده کنیم با این تفاوت که برای ریختن در این **RAM** باید از دستور **MOVX** استفاده کنیم. این دستور به مانند **MOV** هم به صورت آدرس دهی مستقیم و هم به صورت آدرس دهی غیر مستقیم به کار رود با این تفاوت که در دستور **MOV** جون **RAM** حداکثر ۲۵٦ خانه دارد یک ثبات ۸ بیتی برای آدرس دهی غیر مستقیم کافی بود اما اینجا آدرس های ما بیشتر از ۲۵٦ خانه می باشد و برای آدرس دهی نیاز

به یک ثبات ۱٦ بیتی داریم این ثبات که برای این کار ها در میکرو در نظر گرفته شده **DPTR** نامیده می شود و مثلا اگر بخاهیم از آدرس **2D31H** اطلاعات بخوانیم بایب بنویسیم :

```
MOV   DPTR,#2D31H
MOVX A,@DPTR
```

باید دقت کرد با اینکه دستور **MOV** دستور بایتی است می توان آنرا برای ۲ بایت نیز به کار برد.

MOVC A,6F4H : این دستور برای خواندن از حافظه کد برنامه است مثلا اگر در زمان برنامه نویسی در آدرسی از حافظه برنامه مقداری را گذاشته با این دستور می توانیم آنرا بخوانیم همچنین اگر حافظه خارجی برای کد برنامه داشته باشیم (**EA** زمین شده باشد) برای خواندن از حافظه کد خارجی باید از این دستور استفاده کنیم.

ADD A,source : این دستور جمع دو عدد ۸ بیتی است و **A** و **source** را با هم جمع می کند و جواب آنرا در **A** می ریزد و در صورتی که بیشتر از ۲٥٦ شد بیت سرریز **(C)** یک می شود. این دستور به صورت های زیادی وجود دارد مانند **ADD A,#52** یا **ADD A,@R1** و همچنین دستوری به نام **ADDC** نیز داریم که با توجه به مقداری که **C** قبلا بوده آنرا نیز با مقایری که داریم جمع می کنیم جمع می کند مثلا **ADDC A,B** مقدار **A** و **B** را باهم و با **C** که یک بیت است جمع می کند و مقدار آنرا در **A** می ریزد و در صورتی که سرریز داشته باشیم آنرا در **C** می ریزد.

SUBB A,source : که مقدار **source** را از **A** کم می کند و آنر ا در **A** می ریزد توجه شود این دستور بیت **C** را (به مانند **C** در جمع) در تفریق اثر می دهد و این بیتی است که ممکن است در تفریقهای قبلی به علت منفی شدن **A** یک شده باشد.

MUL AB : این دستور **A** و **B** را در هم ضرب می کند و همانطور که می دانیم ضرب دو عدد ۸ بیتی ۱٦ بیتی می باشد پس ۸ بیت کم ارزش را در **A** و ۸ بیت پر ارزش را در **B** می ریزد. (۸ بیت کم ارزش ۸ بیت پایینی یا ۸ بیت سمت راست جواب می باشند)

DIV AB : این دستور **A** را بر **B** تقسیم کرده و خارج قسمت را در **A** و باقیمانده را در**B** می ریزد.

INC source : این دستور یک واحد به **source** اضافه می کند.

DEC source : این دستور یک واحد از **source** کم می کند.

ANL dest,source : این دستور تک تک بیت های **source** را با **dest** مقایسه می کند و با هم **AND** می کند یعنی اگر یکی از آنها صفر بود مقدار صفر را در **dest bit** و اگر هر دوی آنها یک بود مقدار یک را در بیت مقصد (**dest**) می ریزد پس این دستور تک تک بیت های **dest** و **source** را با هم **AND** می کند و پاسخ را در **dest** می ریزد.

ORL dest,source : این دستور تک تک بیت های **dest** و **source** را با هم **OR** می کند یعنی اگر هر دوی آنها صفر بود صفر را صفر را در **dest bit** و در غیر این صورت یک را در آن می ریزد پس به این ترتیب این دستور **dest** و **source** را با هم **OR** می کند و پاسخ آنرا در **dest** می ریزد.

ANL C,source bit : این دستور صورت بیتی **AND** است و فقط برای بیت سرریز کا ربرد دارد مانند **ANL C,P1.0** که بیت اول پورت صفر را با بیت سرریز **AND** می کند و نتیجه را در **C** می ریزد.

ORL C,source bit : صورت بیتی دستور **OR** برای بیت سرریز

SETB source bit : دستور **SETB** بیتی را که در جلوی آن باشد یک می کند مانند **SETB ACC.2** ، **SETB P3.5** ، **SETB PSW.7** و ... این دستور برای متغیر هایی که آدرس پذیر بیتی هستند به کار می رود.

CLR source bit : این دستور بیت جلوی خود را صفر می کند مانند **CLR P3.5**

CPL A : این دستور همه ی بیتهای **A** را عکس می کند یعنی اگر صفر بود یک می کند و اگر یک بود صفر می کند. مثلا اگر **A=10000111B** با اجرای این دستور داریم **A=01111000B**

CPL source bit : صورت بیتی دستور **CPL** که بر روی هر بیتی اجرا شود آنرا عکس می کند.

SWAP A : جای چهار بیت پایین و بالای **A** را با هم عوض می کند.

RR A : این دستور بیت های **A** را یکی به سمت راست شیفت می دهند و کم ارزش ترین بیت را به جای پر ارزش ترین بیت می گذارد ، به عنوان مثال **10001110B** بعد از اجرای این دستور به **01000111B** تبدیل می شود.

RL A : این دستور بیت های **A** را یکی به سمت چپ شیفت می دهد به این ترتیب **10001110B** پس از اجرای این دستور به **00011101B** تبدیل می شود.

RRC A : این دستور همان **RR A** می باشد اما بیت سرریز **C** به عنوان بیت نهم در این عمل شرکت می کند و بیت کم ارزش پس از اجرای این دستور به داخل **C** میرود.

RLC A : این دستور همان دستور **RL A** با دخالت بیت **C** به عنوان بیت نهم می باشد.

NOP : این دستور برای تلف کردن زمان می باشد و هیچ عملی انجام نمی دهد .

دستورات **JUMP** : این دستورات دستورات پرشی هستند و به دو صورت وجود دارند : شرطی و غیر شرطی که **JUMP** های شرطی با چک کردن صحیح بودن یک شرط برای پرش به یک خط دیگر تصمیم می گیرند (که آیا باید پرش انجام شود یا خیر) اما **JUMP** های غیر شرطی آنهایی هستند که هر وقت در برنامه به آنها رسیدیم باید پرش را حتما انجام دهیم ، برای بررسی دستورات ابتدا **JUMP** های غیر شرطی را مورد بررسی قرار می دهیم.

SJMP label : با اجرای این دستور به **label** پرش می کنیم اما باید توجه داشت این دستور پرش کوتاه **(short jump)** است و حداکثر تا ۱۲۷+ و ۱۲۸- خط نسبت به مکان فعلی در حافظه کد می تواند پرش کند.

LJMP label : این دستور پرش بلند است و با اجرای این دستور می توانیم به هر جایی از حافظه پرش کنیم این دستور سه بایت از حافظه را اشغال می کند در صورتی که دستور پرش کوتاه دو بایت از حافظه را می گیرد به همین علت زمانی که کمبود حافظه داریم استفاده از این دستور توصیه نمی شود.

AJMP label : این دستور نیز تا ۲ کیلو بایت بعد و قبل نسبت به آدرس می تواند پرش کند و کاربرد زیادی در برنامه نویسی ندارد.

حال به سراغ پرش های شرطی می رویم :

JNB source bit,label : این دستور **(jump not bit)** در صورتی که بیت مورد نظر یک نبود پرش می کند مثلا **JNB P1.0,LOOP1** چک می کند که اگر **P1.0** صفر بود به **LOOP1** که آنرا در برنامه مشخص کرده ایم ، پرش می کند و در غیر این صورت پرش نمی کند.

JB source bit,label : این دستور بیت مورد نظر را چک می کند و در صورتی که یک بود پرش می کند مثلا **JB P3.1,L2** چک می کند اگر **P3.1** یک بود به **L2** پرش می کند و در صورتی که صفر بود به خط بعدی می رود.

JC label : این دستور **(jump carry)** در صورتی که بیت سرریز **(C)** یک بود پرش می کند و در غیر اینصورت به خط بعدی می رود.

JNC label : این دستور در صورتی که بیت سرریز صفر بود پرش می کند مثلا **JNC DELAY** در صورتی که بیت سرریز صفر باشد به **DELAY** پرش می کند و در غیر اینصورت به خص بعدی می رود.

CJNE source,data,label : این دستور **(compare jump not equal)** مقدار منبع را با داده مقایسه می کند و در صورتی که مساوی نبودند پرش می کند البته این دستور باعث می شود که اگر **source>data** باشد بیت سرریز **C=0** شود و اگر **source<data** باشد **C=1** شود برای مثال دستور زیر **CJNE A,#12,OVER** در صورتی که **A** برابر ۱۲ باشد به خط بعد می رود و در غیر اینصورت به برچسب **OVER** پرش می کند و در صورتی که **A>12** باشد **C=0** و در غیر اینصورت **C=1** می شود.

DJNZ source,label : این دستور **(decrease jump not zero)** یک واحد از **source** کم می کند و اگر صفر نشده بود به **label** پرش می کند مثلا **DJNZ A,LOOP** از **A** یک واحد کم می کند اگر صفر شد به خط بعدی می رود و در غیر اینصورت به **LOOP** پرش می کند..

LCALL subroutine : این دستور در هر جای برنامه که باشد یک زیر روال (تابع) را فرا می خواند و به آن زیر برنامه پرش می کند تفاوت این دستور با **LJMP** در این است که با این دستور پس از اجرای زیر برنامه به خط بعد از این دستور بر می گردیم اما با **LJMP** برگشتی در کار نیست ، مثلا فرض کنیم در برنامه

خود به یک ثانیه تاخیر نیاز داریم که هر بار آنرا در مکان های مختلف باید اجرا کنیم اگر از LJMP استفاده کنیم یک بار به آن تاخیر می رویم و دیگر از آن نمی توانیم خارج شویم و برگردیم اما هنگامی که از LCALL استفاده کنیم پس از اجرای زیر برنامه ی تاخیر به خط بعد از LCALL بر می گردیم مثلا LCALL DELAY تاخیر را صدا زده و آنرا اجرا می کند البته این دستور نیز مانند AJMP صورت ACALL نیز دارد که کاربرد کمی دارد.

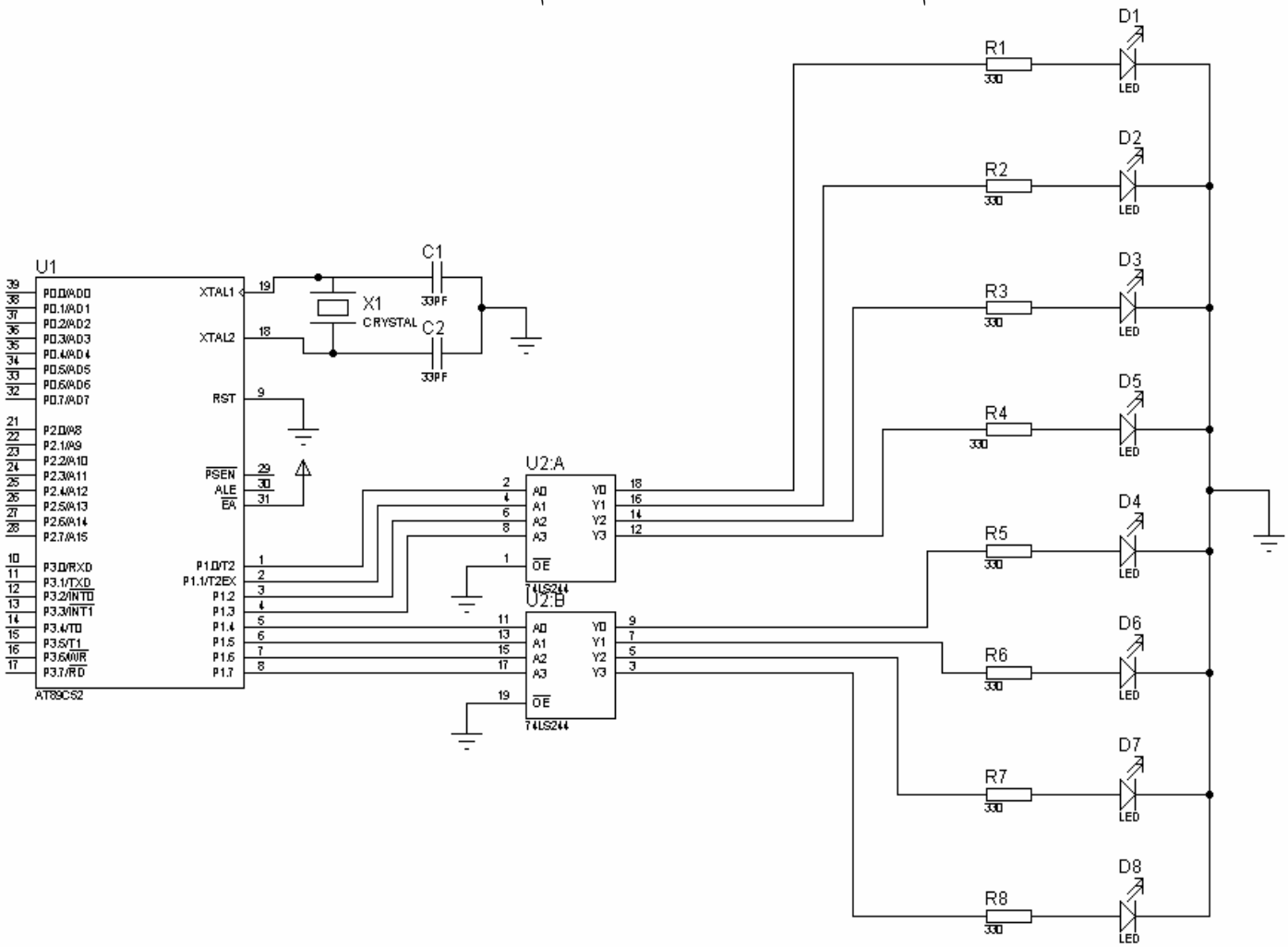
RET : این دستور باید در آخر زیر برنامه گذاشته شود تا به این وسیله بتوانیم به خط بعد از LCALL برگردیم در واقع دستور LCALL مکان فعلی را در جایی ذخیره می کند و به زیر برنامه پرش می کند و دستور RET آن آدرسی را که ذخیره کرده ایم برداشته و به خص بعد از آن می رود (خط بعد از LCALL) در صورتی که در LJMP چنین نیست و آدرس فعلی در جایی ذخیره نمی شود و نکته دیگری نیز که در اینجا باید مورد توجه قار گیرد تفاوت label با subroutine است که همانگونه که دیدیم باید در آخر subroutine حتما RET بگذاریم اما برای label خیر.

RETI : این دستور برای خارج شدن از زیر برنامه وقفه می باشد در مورد وقفه به صورت گسترده بعدا توضیح داده خواهد شد اما بهتر است بدانیم که وقفه در هر میکرو از چند چیز ناشی می شود مثلا وقفه ی خارجی که فعال شدن یکی از پایه ها است این امر سبب می شود که ما در هر مکانی از اجرای برنامه که باشیم فقط همان دستور را اجرا کرده و سپس آدرس محل فعلی در جایی ذخیره شده و ما به آدرس خاصی از حافظه که مخصوص همان وقفه است برویم و زیر برنامه ای را اجرا کنیم و دستور RETI ما را از زیر برنامه وقفه خارج کرده و به یک خط بعد از خطی که برنامه در حال اجرا بود باز می گرداند.

فصل چهارم

مثال های کاربردی از بر نامه اسمبلی AT89S8252 :

برای شروع ابتدا می خواهیم یک برنامه ی ساده بنویسیم که ۸ خروجی پورت یک را که بـه ۸ لامـپ LED متصل کرده ایم به تر تیب دو تا دوتا روشن کرده و همین روال را ادامه دهد ، توجه شود که خروجی جریـان میکرو کنترلر سری ۸۰۵۱ بسیار کم و در حد میکرو آمپر می باشد پس ما نمی توانیم مستقیما آنرا به LED ها متصل کنیم زیرا LED جریانی در حد چند میلی آمپر می کشد پس نیاز به یک IC بـافر یـا تقویـت کننـده جریان داریم آی سی ۷٤۲٤٤ یک ای سی بافر ۸ تایی می باشد که تا حد ۲۰ میلی آمپر می تواند جریان بدهد و چون ۸ تایی می باشد برای یک پورت ما بسیار مناسب می باشد پس این ای سی را در سر راه بـین پـورت یک میکروکنترلر (P1) و LED ها قرار می دهیم توجه داشته باشید که ولتاژ لازم برای روشن شـدن LED حدود ۲ ولت است اما خروجی آی سی ۷٤۲٤٤ (یک منطقی برای سری TTL)ولتاژ ۵+ ولت می باشـد کـه به LED های ما آسیب زده و آنها را می سوزاند پس یک مقامت باید با آنها سری کنیم حـال ببینـیم چگونـه باید مقدار این مقاومت تعیین شود ، فرض کنیم LED های ما در حالت روشن ۱۰ میلـی آمپـر جریـان لازم داشته باشد و ولتاژ دو سر آن نیز ۲+ ولت بیفتد و خروجی آی سی ۷٤۲٤٤ نیز برای حالت یک ۵+ ولت است پس باید ۳=۲-۵ ولت دو سر مقاومت ما بیفتد و ۱۰ میلی آمپر نیز از آن عبور کند پس طبق رابطه V=RI مـا ۳+ ولت داریم و ۱۰ میلی آمپر جریان پس ۳ تقسیم بر ۱۰ میلی آمپر برابر ۳۰۰ اهم می شود مقامت لازم بـرای ما که ما مقامت ۳۳۰ اهم را سری با LED ها قرار می دهیم.



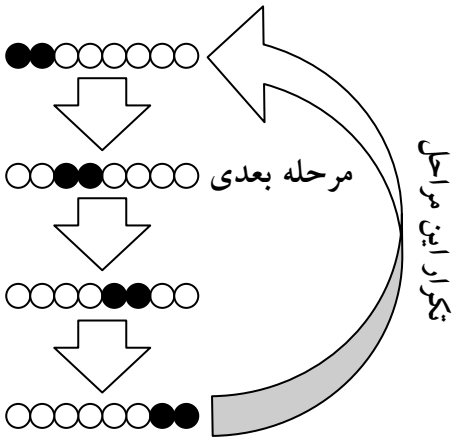
دقت شود که IC ۷٤۲٤٤ دو بافر چها تایی است که باید $\overline{OE}$ هر دوی آنها را زمین کرد.

```
ORG 0000H          ← از آدرس ۰۰۰۰ هگز شروع به نوشتن در رام می کند
L1:  MOV P1,#11000000B
     LCALL DELAY   ;200MS DELAY  ← زیر برنامه ای را صدا می زند که ۲۰۰ میلی ثانیه تاخیر می سازد
     MOV P1,#00110000B
     LCALL DELAY
     MOV P1,#00001100B
     LCALL DELAY
     MOV P1,#00000011B
     LCALL DELAY
     LJMP L1       ← برنامه به اول برنامه باز می گردد
```

در زیر برنامه تاخیر جند نکته باید توضیح داده شود اول اینکه هم زیر برنامه و هم برچسب با علامت : مشخص می شوند مانند DELAY: یا L1: دوم اینکه تاخیر را در میکرو کنترلر به ۲ روش می توان ایجاد کرد به وسیله تایمر و به وسیله ایجاد حلقه های تو در تو که ایجاد به وسیله تایمر در مثال های بعدی توضیح داده می شود ما در اینجا به روش حلقه های تو در تو تاخیر ایجاد کرده ایم یعنی به R3 مقدار ۲۵۰ دسیمال می دهیم و آنقدر آنرا کم می کنیم تا صفر شود (DJNZ R3,D1) و سپس R2 را که مقدار آنرا ۲۰۰ داده ایم یکی کم می کنیم و دوباره R3 را مقرار ۲۵۰ می دهیم و آنرا دایما کاهش می دهیم تا صفر شود و دو باره R2 را کاهش می دهیم و به همین تر تیب ادامه می دهیم تا R1 نیز صفر شود و سپس به دستور RET می رسیم که ما را از این زیر برنامه خارج می کند اگر بدانیم که دستور DJNZ دو سیکل ماشین طول می کشد و سیکل ماشین ما ۱ میکرو ثانیه باشد خواهیم دید که تقریبا ۲۰۰۰۰۰=۲*۲۰۰*۲۵۰ میکروثانیه برابر با ۲۰۰ میلی ثانیه اجرای این زیر برنامه طول خواهد کشید که در بالا ۲۵۰ مقدار R3 و ۲۰۰ مقدار R2 و ۲ مقدار R1 و ۲ بعدی زمان انجام هر دستور DJNZ می باشد. نکته سوم اینکه اسمبلر در هر خط بعد از علامت ; را نمی خواند و برای توضیح مانند مثال بالا (200MS DELAY;) می توان از آن استفاده کرد.

```
DELAY:  MOV  R1,#2
D3:     MOV  R2,#200
D2:     MOV  R3,#250
D1:     DJNZ  R3,D1
        DJNZ  R2,D2
        DJNZ  R1,D3
        RET
END
```

این خط برای نشان دادن انتهای برنامه است



شکل بالا طریقه روشن شدن LED ها را به ترتیب در ٤ مرحله با اختلاف ۲۰۰ میلی ثانیه نشان می دهد.

در مثال بالا فرض کنیم می خواهیم تاخیر زمانی را با تایمر بسازیم پس ابتدا به توضیح طریقه کار با تایمر می پردازیم.

در میکرو کنترلر ۸۰۵۱ دو تایمر داریم تایمر صفر و یک اما به علت محدودیت هایی که این دو تایمر داشتند در سری ۸۹ تعداد تایمر ها به ۳ تایمر افزایش یافت و به این ترتیب تایمر دو نیز به این تایمر ها افزوده شد. در این بخش ابتدا به توضیح تایمر صفر و یک که کاملا دارای طرز کا یکسانی هستند می پردازیم سپس به سراغ تایمر دو می رویم. تایمر های صفر و یک داری دو ثبات می باشند که ۸ بیت پایین و بالای تایمر می باشند این ثبات ها **TL0** و **TH0** برای تایمر صفر و **TL1** و **TH1** برای تایمر یک نامیده می شوند که به ترتیب ۸ بیت پایینی و بالایی تایمر هستند. تایمر صفر و یک چهار مد کاری دارد مد صفر که مد ۱۳ بیتی تایمر میباشد مد یک که مد ١٦ بیتی تایمر می باشد مد دو که مد ۸ بیتی با بار شدن خودکار می باشد و مد ۳ که مد نصف شدن تایمر میباشد که با مد ۳ معمولا کاری نداریم در اینجا لازم است توضیح دهیم که یک تایمر چگونه کار می کند ؟ ما باید برای تایمر یک مد کاری انتخاب نماییم سپس مقدار مورد نیاز را در ثبات های تایمر ریخته و تایمر را راه اندازی نماییم ، تایمر با فعال شدن بیت تایمر روشن می شود و این بیت **TR0** یا **TR1** یا **TR2** به ترتیب برای تایمر صفر و یک و دو نامیده می شود و فرض کنیم می خواهیم تایمر یک را راه اندازی نماییم باید بنویسیم **SETB TR1** که از این به بعد تایمر شروع به شمارش می کند و هر شمارش تایمر یک سیکل ماشین طول می کشد و فرضا اگر تایمر ما ۱۲ مگا هرتز باشد هر سیکل ماشین ۱ میکرو ثانیه می باشد پس مقدار ثبات های تایمر به ازای هر یک میکرو ثانیه یک واحد افزایش می یابر در این حالت بسته به این که مد تایمر ما ۸ یا ١٦ بیتی باشد مدت زمانی طول می کشد تا تایمر تا مقدار ماکزیمم شمارش کرده و سرریز کند که مقدار ماکزیمم برای مد ۸ بیتی **FF** در دستگاه هگزادسیمال و برای مد ١٦ بیتی **FFFFH** می باشد در اینجا زمانی که می گوییم تایمر سرریز کرده است منظور این است که بیت سرریز تایمر که **TF0** برای تایمر صفر و **TF1** برای تایمر یک و **TF2** برای تایمر دو فعال شده است حال مد های کاری تایمرهای صفر ویک را توضیح می دهیم از مد ۱۳ بیتی نیز معمولا استفاده نمی شود پس به سراغ مد یک می رویم این مد ١٦ بیتی برای تایمر می باشد که جون این مد ١٦ بیتی است حداکثر تا **FFFFH** یا ٦٥٥٣٥ دسیمال سیکل ماشین به اضافه یک را می توانیم بسازیم اگر سیکل ماشین یک میکرو ثانیه باشد ما تا ٦٥٥٣٦ میکرو ثانیه را می توانیم با این تایمرها مستقیما بسازیم حال فرض کنیم ما می خواهیم هر بار ۱۰ میلی ثانیه را با تایمر یک بسازیم باید بدانیم که چه اعدادی را باید در ثبات های تایمر **(TH1,TL1)** بریزیم ۱۰ میلی ثانیه یعنی ۱۰۰۰۰ میکرو ثانیه پس تایمر ما باید ۱۰۰۰۰ با بشمارد و سپس بیت سرریز ما یک شود اما باید بدانیم که تایمر صفر و یک فقط به صورت بالارونده می توانند شمارش کنند پس عددی که باید در ثبات های تایمر بریزیم ٦٥٥٣٦-۱۰۰۰۰=٥٥٥٣٦ است که با ریختن این عدد درثبات ها (که البته می دانیم مقدار هگزا دسیمال آنرا باید در ثبات ها ریخت یعنی **D8F0H** که این عمل را به این صورت انجام می دهیم که می نویسیم **MOV TL1,#0F0H** و **MOV TH1,#0D8H**) تایمر از

۵۵۵۳۶ شروع به شمارش می کند و افزایش می یابر تا به ۶۵۵۳۵ برسد و یک میکرو ثانیه بعد بیت TF1 فعال شده و و مقدار ثبات های تایمر 0000H می شود با فعال شدن این بیت (TF1) می توانیم بفهمیم که ما ۱۰ میلی ثانیه زمان ساخته ایم و کار شمارش را باید تمام کرد در اینجا اگر بخواهیم دوباره ۱۰ میلی ثانیه را بسازیم باید بیت TF1 را پاک کرده و دوباره ثبات های تایمر را که صفر شده اند مقدار دهی کنیم و تایمر را فعال کنیم. اکنون زیر برنامه DELAY برای تولید ۲۰۰ میلی ثانیه تاخیر را برای مثال بالا با استفاده از تایمر صفر در مد ۱۶ بیتی می سازیم (توجه شود که تعیین مد کاری تایمر با استفاده از ثبات TMOD میباشد که بعدا در مورد آن توضیح داده خواهد شد در اینجا فرض کنیم ثبات TMOD طوری مقدار دهی شده که تایمر صفر در مد یک باشد)

```
DELAY:  MOV R0,#4
D2:     MOV TH0,#3CH
        MOV TL0,#0B0H
        SETB TR0
D1:     JNB TF0,D1
        CLR TR0
        CLR TF0
        DJNZ R0,D2
        RET
```

در اینجا می خواهیم ۵۰ میلی ثانیه را با تایمر بسازیم و چهار مرتبه آنرا انجام می دهیم تا ۲۰۰ میلی ثانیه را بسازیم برای اینکه تایمر ۵۰ میلی ثانیه بشمارد ۱۵۵۳۶=۵۰۰۰۰-۶۵۵۳۶ که مقدار هگز آن برابر 3CB0H است را در ثبات های تایمر صفر بریزیم که بایت کم ارزش (B0) باید در TL0 و بایت پرارزش (3C) باید در TH1 ریخته شود توجه شود که در خط D1: ما ۵۰ میلی ثانیه منتظر می شویم تا TF0 فعال شود و سپس تایمر را خاموش کرده و R0 را یکی کم می کنیم و تا چها ر مرتبه این کار را انجام می دهیم.

حال به سراغ مد ۲ می رویم این مد ۸ بیتی با بار شدن خودکار میباشد همانطور که می دانیم چون این تایمر ۸ بیتی است حداکثر تا ۲۵۶ سیکل ماشین یا ۲۵۶ میکرو ثانیه را می توان مستقیما با این تایمر ساخت اما با اینکه این مد زمان بسیار کوتاهی قابلیت زمان گیری دارد ویژگی دیگری دارد که استفاده از این مد را بیشتر کرده است و آن بار کردن خودکار می باشد. در مثال بالا می بینیم که در مد یک برای هر بار استفاده از تایمر باید ثبات های آنرا مقدار دهی نمود و پرچم تایمر TF را پاک کرد اگر دقت شود در این روش مانند روش استفاده از حلقه های تودر تو دقت کمی داریم زیرا در محاسبه زمان تاخیر زمان های لازم برای انجام دستوراتی مانند MOV TH0,#3CH و CLR TR0 و ... را به علت کم بودن و در حد چند میکرو ثانیه بودن در نظر نمی گیریم اما اگر ما بخواهیم با تایمر مثلا یک صدم ثانیه را برای کرنومتر یا ساعت با دقت بالا بسازیم دیده می شود که این زمان های زیاد مانند چند ساعت یا چند روز در ساعت خطای بسیار زیادی به حساب می آیند مد ۲ در تایمر صفر و یک قابلیت جالبی دارد و آن اینکه ثبات های این تایمر را باید فقط یک بار مقدار دهی کرد به این صورت که ما یکمرتبه باید مقدار مورد نیاز برای شمارش را در TH0 یا TH1 ریخته و در هر مرحله با فعال شدن TF0 یا TF1 این مقادیر به صورت خودکار در TL0

یا **TL1** ریخته می شوند در اینجا می خواهیم با استفاده از تایمر یک زمان ۲۰۰ میلی ثانیه را برای زیر برنامه **DELAY** در مثال قبلی بسازیم :

```
DELAY:   SETB TR1
         MOV R0,#4
D2:      MOV R1,#250
D1:      JNB TF1,D1
         CRL TF1
         DJNZ R1,D1
         DJNZ R0,D2
         CLR TR1
         RET
```

در این روش باید دقت کرد که مقداردهی ثبات های تایمر را باید در ابتدای برنامه و خارج از زیر برنامه **DELAY** انجام داد به این ترتیب ما در اینجا ۲۰۰ میکرو ثانیه را با تایمر ساخته ایم پس باید ۵۶=۲۰۰-۲۵۶ یا **(38H)** را در **TH1** بریزیم پس در ابتدای برنامه باید بنویسیم **MOV TH1,#38H** که نکته ی کوچکی در اینجا وجود دارد و آن مربوط به اولین مرتبه شمارش تایمر میباشد زیرا برای اولین بار باید مقدار **MOV TL1,#38H** را نیز بدهیم اما در مراحل بعدی نیازی به این عمل نیست.

در اینجا همانگونه که میبینیم در ابتدای برنامه **TR1** فعال شده و در طول برنامه و همواره فعال بوده و هیچگاه جز در انتهای برنامه آنرا غیر فعال نکرده ایم زیرا به مقدار دهی نیازی نداشته ایم و در همه حال و در همه جای برنامه در حال شمارش بوده ایم پس این روش برای زمانگیری با دقت بسیار بالایی به کار می آید زیرا تایمر هیچگاه از شمارش باز نمی ایستد.

حال به بررسی ثبات **TMOD** می پردازیم و به وسیله آن مد های مختلف را ایجاد می کنیم شکل زیر بیت های مختلف ثبات **TMOD** را نشان می دهد.

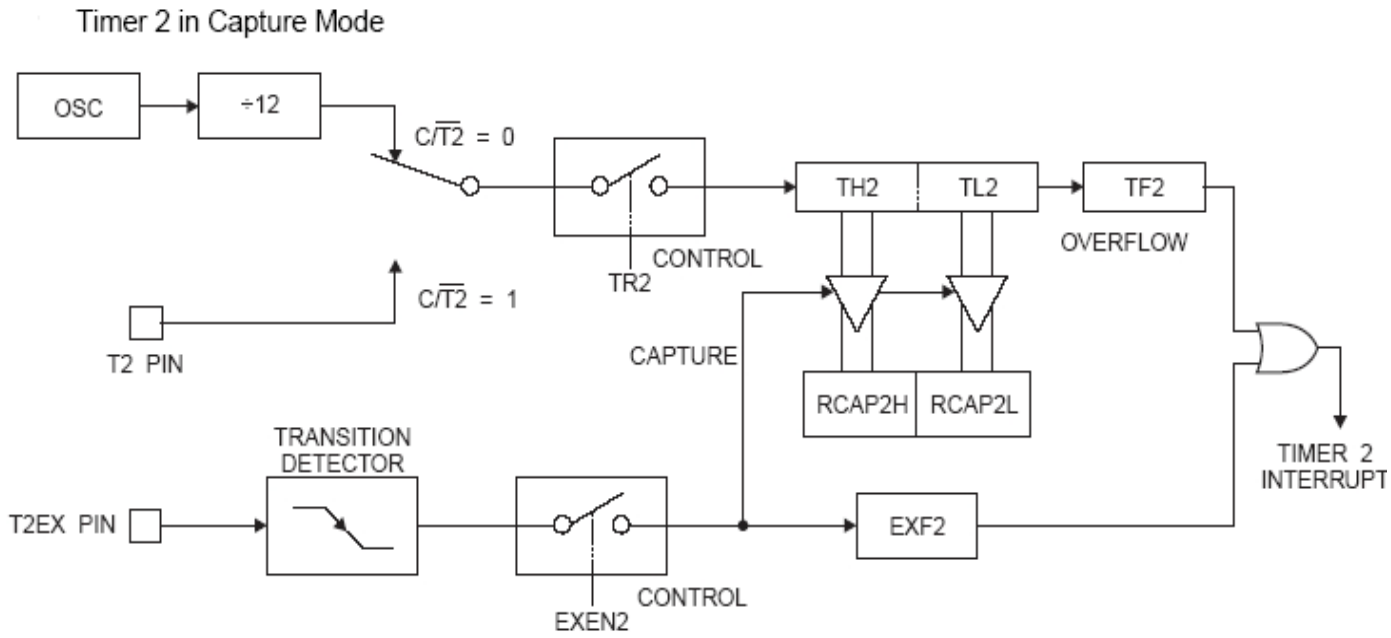
ثبات **TMOD**

GATE	C/T	M1	M0	GATE	C/T	M1	M0
تایمر یک				تایمر صفر			

ثبات **TMOD** متشکل از دو نیبل بالا و پایین می باشد که چهار بیت بالا برای تایمر یک و چهار بیت پایین برای تایمر صفر می باشند. **M0** و **M1** برای تعیین مد تایمر می باشند که تعیین مد کاری برای آنها در زیر توضیح داده شده است **C/T** برای تعیین تایمر یا کانتر (شمارنده) بودن می باشد با فعال شدن این بیت **(C/T=1)** تایمر تبدیل به شمارنده می شود به این ترتیب با وارد شدن هر پالس به پایه ی **P3.4** برای تایمر صفر و **P3.5** برای تایمر یک ، یک واحد به مقدار ثبات های تایمر افزوده شده و دقیقا شبیه تایمر عمل می کند فقط کلاک ورودی از خارج از میکرو به آن داده می شود . بیت **GATE** نیز یکی دیگر از بیتهای ثبات **TMOD** میباشد هر گاه این بیت صفر باشد (که ما معمولا آنرا صفر می گذاریم) ما به راحتی با استفاده از دستورات **CLR TRX** و **SETB TRX** به راه اندازی و غیر فعال سازی تایمر می پردازیم اما با فعال کردن بیت **GATE=1** فعال و غیر فعال سازی تایمر با استفاده از سخت افزار بیرونی می باشد که معمولا از آن استفاده نمی کنیم.

مد سه	مد دو	مد یک	مد صفر
M0=1	M0=0	M0=1	M0=0
M1=1	M1=1	M1=0	M1=0

تایمر ۲ که بعدا به میکرو های ۸۹ افزوده شد قابلیت های بسیار زیادی دارد که تقریبا استفاده از تایمر های صفر و یک را به فراموشی سپرد. این تایمر دارای سه مد اصلی کاری می باشد که یک مد ۱۶ بیتی با بار شدن خودکار میباشد و یکی مد ۱۶ بیت **Capture** م مد بعدی مد **Baud Rate Generator** مـی باشـد که برای تولید کلاک جهت ارتباط سریال است که بعدا توضیح داده خواهد شد. همانگونه که از اسم آن نیـز پیدا است تایمر ۲ مد کاری ۱۶ بیتی با بار شدن خودکار دارد که این مد بسیار دقیق و کار آمد می باشـد بـه این ترتیب که ما برای تایمر ۲ ، چهار ثبات ۸ بیتی (یا ۲ ثبات ۱۶ بیتی داریم) که نـام آنهـا **TL2** و **TH2** که همان ۱۶ بیت اصلی تایمر می باشند و **RCAP2L** و **RCAP2H** که ۱۶ بیت مورد نیاز برای بـار شـدن خودکار می باشند به این ترتیب که ما زمان مـورد نیـاز بـرای شـمارش را محاسـبه و مقـدار هگـز آنـرا در **RCAP2L** و **RCAP2H** می ریزیم و به این ترتیب با هر بار فعال شـدن **TF2** ایـن مقـدار بـه صـورت خودکار در **TH2** و **TL2** ریخته می شود مد **Capture** نیز به این صورت می باشد که هنگامی که در ایـن مد می باشیم و **EXEN2** که بیت فعال ساز خروجی تایمر ۲ است نیز فعال باشد با هر پـالس بـا لبـه پـایین رونده بر روی پایه **T2EX** (که پایه **P1.1** می باشد) هـر مقـداری در **TL2** و **TH2** باشـد بـه ترتیـب در **RCAP2L** و **RCAP2H** ریخته می شود. شکل این مد را در زیر می بینید.



برای تایمر دو ۲ ثبات کنترلی نیز وجود دارند به نام های **T2CON** و **T2MOD** که سـاختار آنهـا در زیـر توضیح داده شده است.

T2CON Address = 0C8H

Bit Addressable

	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$
Bit	7	6	5	4	3	2	1	0

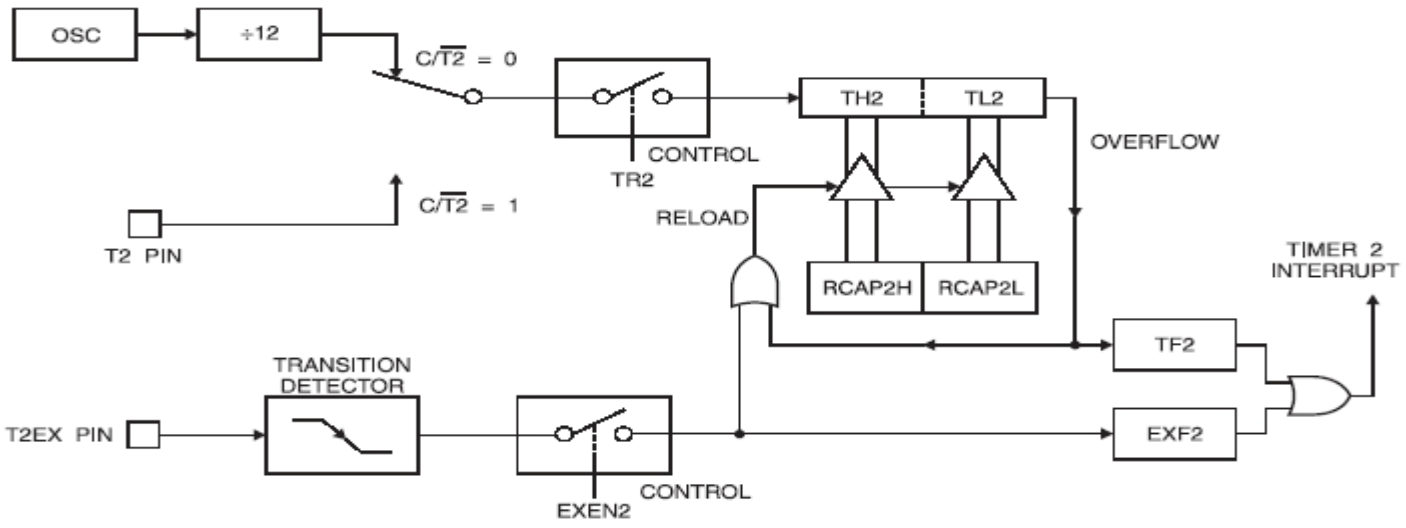
T2MOD Address = 0C9H

Not Bit Addressable

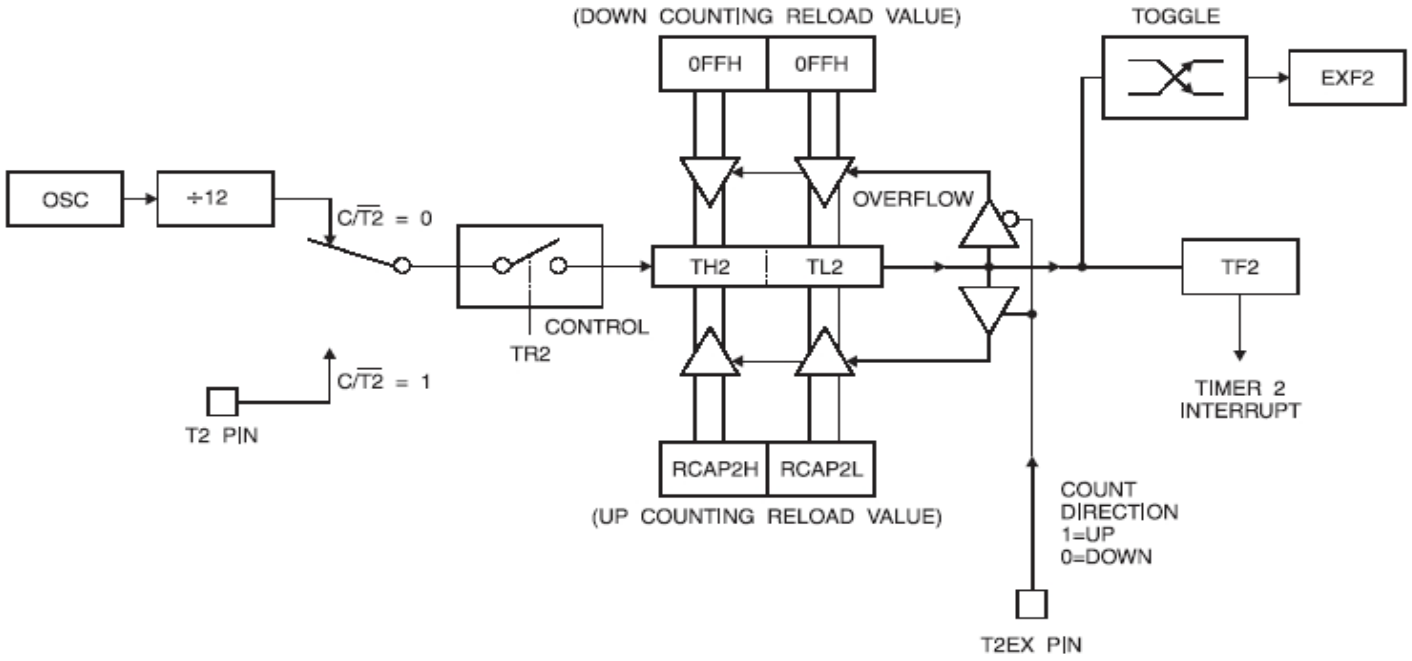
	–	–	–	–	–	–	T2OE	DCEN
Bit	7	6	5	4	3	2	1	0

ابتدا از ثبات T2CON شروع می کنیم بیت CP/$\overline{RL2}$ مشخص می کند که که در مد Capture باشیم یا مد بار شدن خودکار ، C/$\overline{T2}$ بیان کننده ی تایمر یا شمارنده بودن تایمر ۲ می باشد ، TR2 که بیت فعال ساز تایمر و روشن کننده ی تایمر می باشد ، EXEN2 همانگونه که قبلا گفته شد ودر شکل مشاهده می شود فعال ساز ورودی پایه T2EX می باشد ، TCLK فعال سازی این پایه باعث می شود که درگاه سریال از تایمر ۲ برای ارسال داده استفاده کند و در غیر اینصورت از تایمر یک استفاده می کند ، RCLK فعال سازی این پایه باعث می شود از تایمر ۲ برای دریافت از پورت سریال استفاده کنیم و در غیر اینصورت (RCLK=0) از تایمر یک استفاده می کند که دو بیت اخیر مد Baud Rate Generator را می سازند ، EXF2 زمانی فعال می شود که EXEN2 فعال بوده و لبه ی پایین رونده روی T2EX دیده شود در این صورت این بیت یک می شود و باعث ایجاد وقفه ی تایمر ۲ می شود که باید آنرا به صورت نرم افزاری پاک کرد ، TF2 که همان بیت سرریز تایمر ۲ ما می باشد. حال به سراغ ثبات T2MOD می رویم ، فعال سازی بیت DCEN (یا فعال ساز پایین شمار) باعث می شود که تایمر ما به سمت پایین بشمارد یا شمارنده ما کاهنده باشد به این ترتیب نیازی به قرار دادن مکمل یعنی کم کردن مقدار مورد نیاز از ٦٥٥٣٦ نیست پس می توان اگر فرضا خواستیم ۵۰ میلی ثانیه را بسازیم در مد پایین شمار باید 3C50H را در ثبات ها بریزیم زمانی که DCEN=0 باشد همانند تایمر صفر و یک در حالت بالا شمار هستیم ، T2OE فعالسازی این بیت باعث می شود که ما بتوانیم کلاک مورد نیاز خود را به صورت موج مربعی بدون نیاز به هیچگونه برنامه نویسی با استفاده از تایمر ۲ روی پایه P1.0 ایجاد کنیم که شکل های زیر به راحتی طریقه کار تایمر ۲ را در هر مد نشان میدهد.

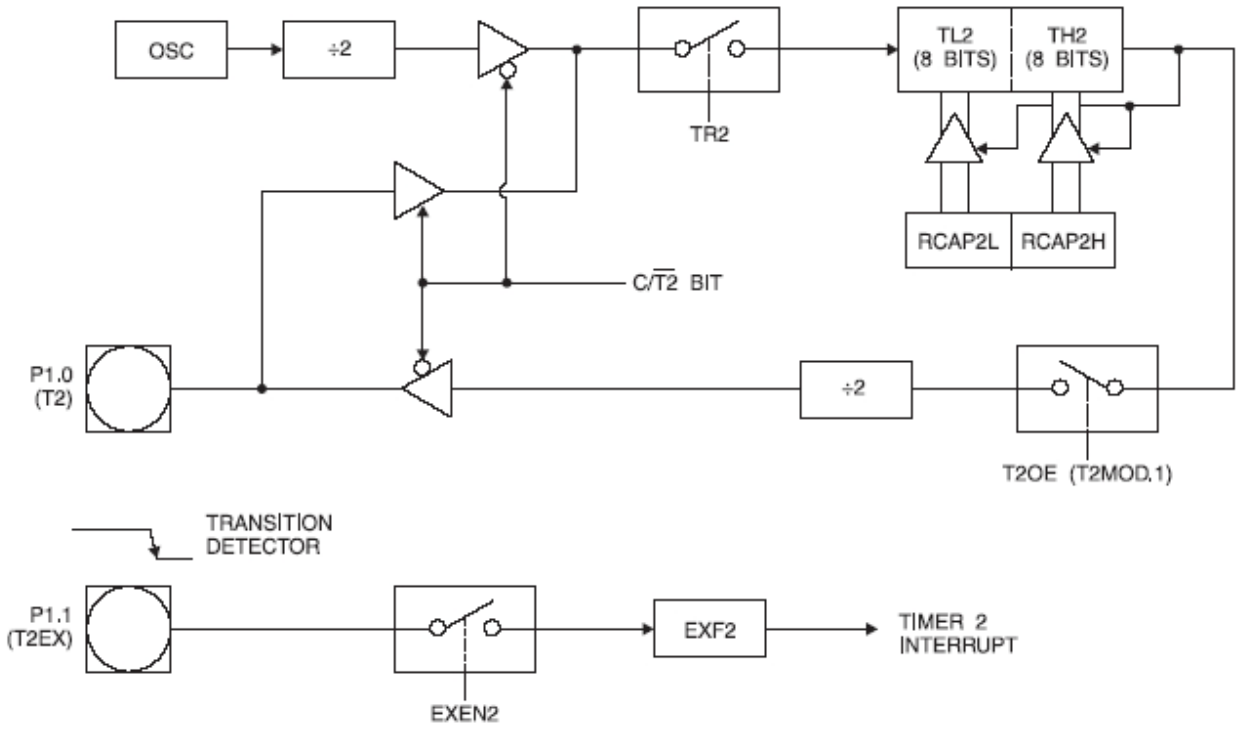
Timer 2 in Auto Reload Mode (DCEN = 0)



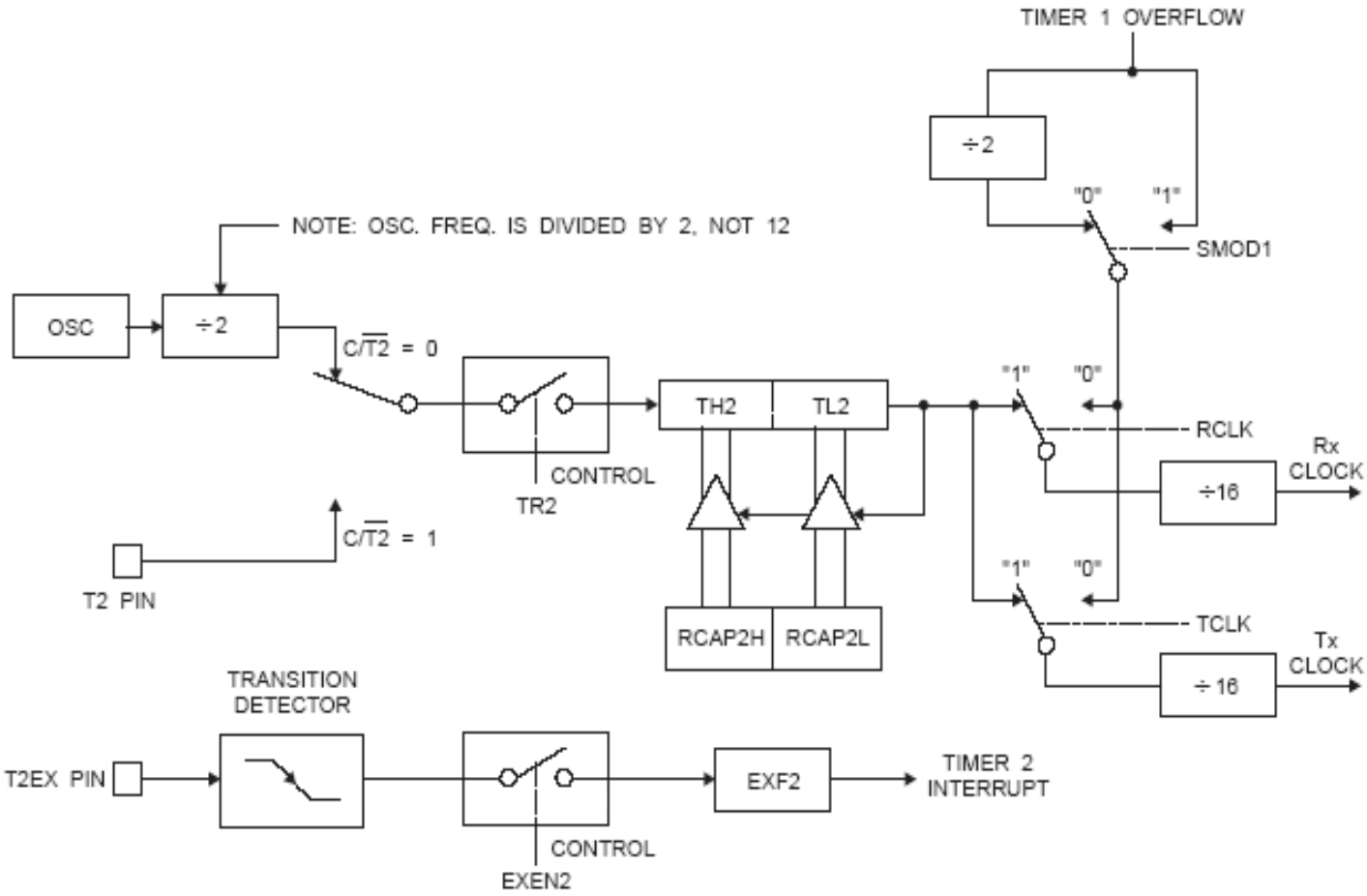
Auto Reload Mode (DCEN = 1)



Timer 2 in Clock-out Mode



Timer 2 in Baud Rate Generator Mode



برای مطالعه بیشتر بر روی این اشکال و مد های تایمر ۲ به صورت کاملتر

برگه ی اطلاعات AT89S8252 را از سایت

HTTP://WWW.ATMEL.COM

دانلود کرده و از آن استفاده نمایید.

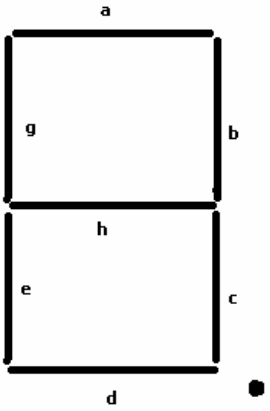
حال زیر برنامه DELAY ۲۰۰ میلی ثانیه را برای مثال قبلی می نویسیم.

```
DELAY:  SETB TR2
        MOV R0,#4
D1:     JNB TF2,D1
        CLR TF2
        DJNZ R0,D1
        CLR TR2
        RET
```

در این زیر برنامه توجه داشته باشید برای برنامه ریزی تایمر در مد بار شدن خودکار در ابتدای برنامه ی اصلی باید بنویسیم MOV RCAP2L,#0B0H و MOV RCAP2H,#3CH که به این ترتیب باید در مد بالا شمار باشیم تا ۵۰ میلی ثانیه توسط تایمر تولید شده و چهار مرتبه این عمل تکرار شود و همانگونه که دیده می شود دقت این روش بسیار زیاد می باشد.

پروژه ساخت ساعت با میکرو کنترلر AT89S8252 :

ساعت ۳ بخش ثانیه شمار دقیقه شمار و ساعت شمار دارد و هر بخش نیاز به ۲ نمایشگر دارد این نمایشگر هایی که معمولا برای نمایش اعداد به کار می روند 7SEG نام دارند که از هفت LED تشکیل شده اند که به صورت شکل روبرو تشکیل شده است و برای نمایش هر عدد باید تعدادی از LED ها روشن باشد مثلا برای نمایش عدد ۸ باید تمام LED های a تا h روشن باشند تا عدد به صورت نمایش داده شود یب برای نمایش عدد ۲ به صورت انگلیسی باید LED های a و b و d و e و h به صورت روشن باشند. حال اگر ما بخواهیم برای هر عددی که که می خواهیم نمایش دهیم به ترتیب این LED ها را روشن کنیم کار نسبتا مشکلی می باشد به همین علت یک IC مخصوص دیکد کردن و تبدیل اعداد BCD (اعداد باینری که فقط از صفر تا نه را در آن داریم و به راحتی آنها را می توانیم روی 7SEG نمایش دهیم) به کد های مخصوص 7SEG به نام ۷٤٤۷ و ۷٤٤۸ تولید کرده اند و به راحتی کار می کنند این IC ها ورودی ٤ بیتی دارند و از ۰۰۰۰ تا ۱۰۰۱ را به عنوان ورودی میگیرند و اگر خروجی های a تا h از آی سی را به ورودیهای 7SEG متصل کنیم اعداد ۰۰۰۰ تا ۹ را برای ما نمایش می دهند . تفاوت ۷٤٤۷ و ۷٤٤۸ در این است که ۷٤٤۷ برای 7SEG ها Common Anode (مثبت مشترک) و ۷٤٤۸ برای 7SEG های Common Cathode (منفی مشترک) به کار می روند تفاوت 7SEG های مثبت مشترک و منفی مشترک همانطور که از اسم آنها پیداست در پایه مشترک آنها است در 7SEG ها مثبت مشترک سر مثبت همه ی LED ها به هم متصل شده و به نام پایه مثبت بیرون می آید و ۷ سر دیگر LED ها نیز بیرون می آید به نام های a تا h و یک پایه دیگر نیز برای نقطه ی کناری یا ممیز بیرون می آید توجه داشته باشی که این 7SEG مثبت مشترک است و برای اینکه یک LED روشن شود باید روی پایه ی آن صفر گذاشت و اگر یک بگذاریم خاموش می شود به هر حال آی سی ۷٤٤۷ همه ی این کار ها را برای ما انجام می دهد و کافی است مثلا برای نمایش عدد ۲ بر روی ورودی ۷٤٤۷ عدد ۰۰۱۰ را بگذاریم. حال به سراغ سخت افزار این مدار می رویم در این مدار توجه داشته باشید که از وقفه استفده شده است از وقفه ی صفر برای تنظیم ساعت و از وقفه ی یک برای تنظیم دقیقه ی ساعت. گفتیم که در AT89S8252 نه منبع وقفه داریم . وقفه چیزی است که هرگاه اتفاق بیفتد و بیت وقفه ی مورد نظر نیز فعال باشد ما را به آدرس خاصی از حافظه ی کد می برد که زیر برنامه ی مخصوصی در آنجا نوشته شده است و آنرا اجرا می کند بری فعال سازی وقفه ما ثباتی به نام IE داریم این ثبات برای فعال سازی انواع وقفه های در میکرو است در زیر این ثبات را میبینید.



EA	–	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

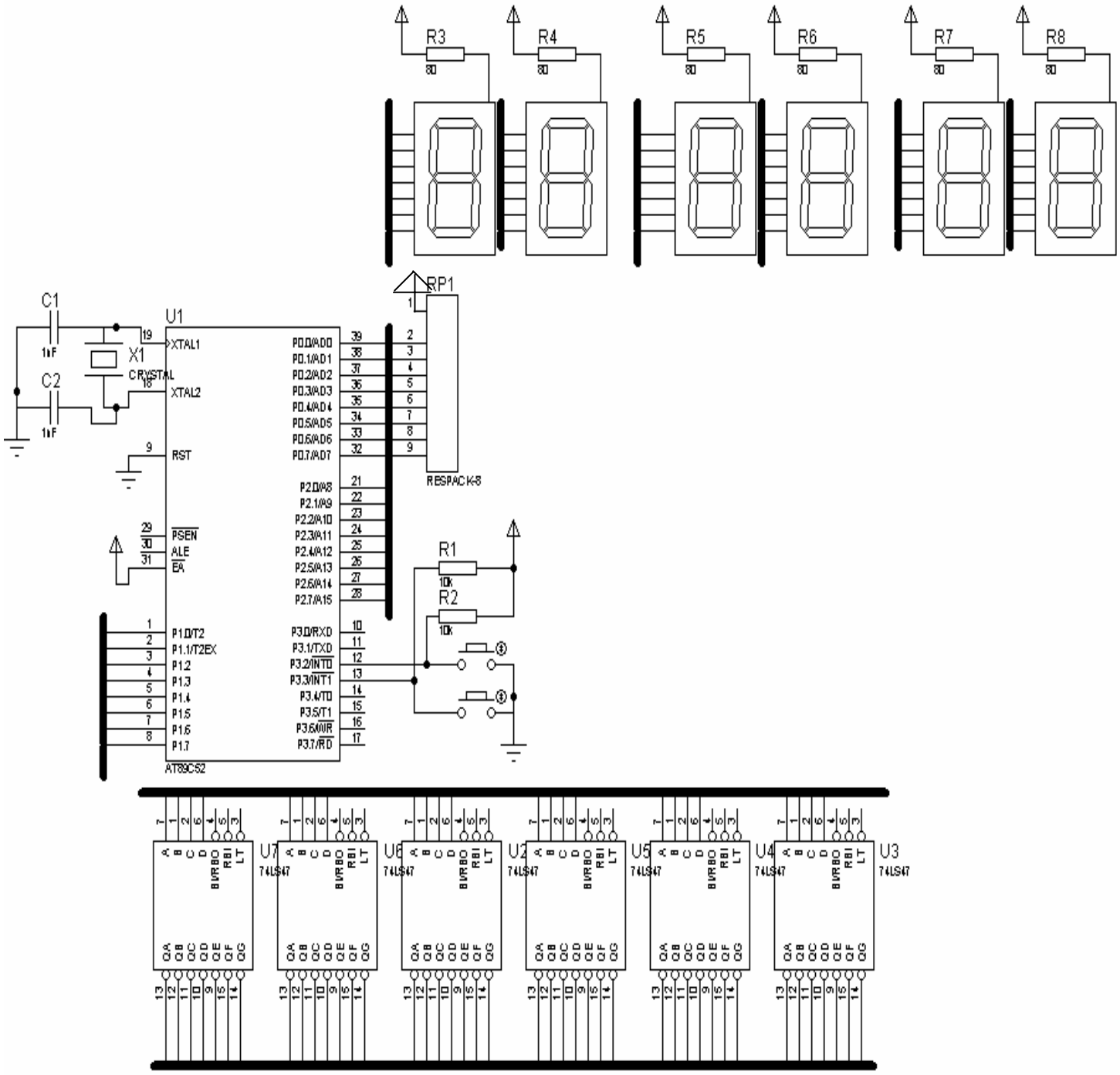
بیت **EA** فعال ساز وقفه در میکرو است و اگر این بیت صفر باشد به هیچ وقفه ای پاسخ داده نمی شود و برای اینکه به وقفه ها ی فعال پاسخ داده شود باید این بیت یک شود ، بیت **ET2** فعال ساز وقفه ی تایمر ۲ است همانطور که در شکل های بالا مربوط به تایمر ۲ می بینید این تایمر در مد های مختلف می تواند وقفه یا **Interrupt** دهد که در همه ی شکل ها با **TIMER 2 INTERRUPT** مشخص شده است و در صورتی که بیت **ET2** یک باشد به این وقفه پاسخ داده می شود ، بیت **ES** وقفه ی ارتباط سریال است و اگر این بیت فعال باشد به وقفه ی ارتباط سریال پاسخ داده می شود که بعدا در مورد آن بحث خواهد شد ، **ET1** وقفه ی تایمر یک می باشد که همانند تایمر ۲ در زمان فعال شدن **TF1** اتفاق می افتد ، **EX1** فعال ساز وقفه ی خارجی یک است وقفه ی خارجی یک وا بسته به فعال شدن پایه ی **INT1** که همانگونه که مشاده می شود این پایه یک پایه فعال صفر است و در صورتی که صفر روی این پایه قرار و **EX1=1** باشد به این وقفه پاسخ داده می شود البته وقفه های خارجی را حساس به لبه نیز میتوان نمود ، با فعال کردن این حالت با دیدن هر لبه ی پایین رونده بر روی پایه ی وقفه ی خارجی (منظور از لبه ی پایین رونده این است که پایه ابتدا یک بوده و سپس صفر شود در این صورت لبه ی پایین رونده روی آن پایه داشته ایم) ما به وقفه می رویم ، باید بدانیم که با فعال شدن وقفه های خارجی ، پرچم هایی در داخل میکرو با نام های **IE0** و **IE1** فعال می شوند که این پرچمها در ثباتی به نام **TCON** هستند ، اما اگر ما بخواهیم وقفه های خارجی خود را حساس به لبه کنیم بیتی به نام **IT0** یا **TCON.0** برای وقفه ی خارجی صفر و **IT1** یا **TCON.2** برای وقفه ی خارجی یک را فعال کنیم به این ترتیب اگر بخواهیم وقفه ی خارجی یک حساس به لبه باشد باید بنویسیم **SETB TCON.2** و به همین ترتیب برای وقفه ی خارجی صفر ، **ET0** فعال ساز وقفه ی تایمر صفر ، **EX0** فعال ساز وقفه ی خارجی صفر . برای فعال کردن هر وقفه باید بیت مربوطه آنرا فعال کنیم . حال ببینیم اگر هر وقفه اتفاق بیفتد به چه آدرسی می رویم.

آدرس وقفه	منبع وقفه
0003H	IE0
000BH	TF0
0013H	IE1
001BH	TF1
0023H	RI & TI
002BH	TF2 & EXF2

همانگونه که دیده می شود **IEX** نشان دهنده ی وقفه ی خارجی **X** و **TFX** پرچم وقفه ی تایمر **X** و **TI,RI** نشان دهنده ی وقفه ی سریال و **TF2,EXF2** پرچم های وقفه ی تایمر ۲ هستند.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

ثبات **IE**



در شکل بالا توجه داشته باشید که P1.0 تا P1.3 به ۷٤٤۷ که به بخش کم ارزش ساعت (اولین از سمت چـپ) متصل است و P1.4 تا P1.7 به پر ارزش ساعت ، و P2.0 تا P2.3 کم ارزش دقیقه P2.4 تا P2.7 به پـر ارزش ساعت ، P0.0 تا P0.3 کم ارزش ثانیه P0.4 تا P0.7 پر ارزش ثانیه را از طریق IC ها ۷٤٤۷ تغذیه می کنند در این زمان کار با میکرو دقت کنید که پورت صفر همواره باید با ۸ مقاومت ۱۰ کیلو اهم به VCC متـصل باشـد و این به علت ساختمان داخلی میکرو کنترلر است در اینجا از مقاومت ۸ تایی استفاده شده است . در این شکل دقت شود که اگر فرض کنیم هر LED حدود ۱۰ میلی آمپر جریان بکشد با توجه به اینکه ۷٤٤۷ خودش بافر و تقویـت کننده جریان نیز هست جریان توسط آن تامین می شود ام نیاز به کاهش ولتاژ جهت سالم ماندن LED ها داریم به همین علت با فرض روشن بودن ۵ LED در هر 7SEG به طور متوسط یک مقاومت در سر راه مثبـت هـای هـر 7SEG قرار داده ایم در حدود ۸۰ اهم که حدود ۳ ولت افت ولتاژ ایجاد می کند.

حال به سراغ خود برنامه می رویم :

این خط به معنای نوشتن در آدرس ۰۰۰۰ است

```
.ORG 0000H
        LJMP MAIN
```
به زیر برنامه MAIN پرش می کند زیرا در بین آدرس صفر تا ۳ برای MAIN جا به اندازه کافی نیست

```
.ORG 0003H
        LJMP INTER0
```
به زیر برنامه وقفه ی صفرو یک پرش میکند البته می توان زیر برنامه وقفه را همین جا نیز نوشت زیرا بسیار کوچک هستند اما برای اطمینان آنها را در آدرس دیگری از حافظه می نویسیم

```
.ORG 0013H
        LJMP INTER1
```

در یک آدرس دلخواه که با بالا تداخل نداشته باشد می نویسیم

```
.ORG 0030H
MAIN:   MOV IE,#10000101B
        MOV TMOD,#02H
        MOV TH0,#6
        MOV TL0,#6
        MOV P0,#0
        MOV P1,#0
        MOV P2,#0
        SETB TCON.2
        SETB TCON.0
        MOV R0,#0
        MOV R1,#0
        MOV R2,#0

BACK:   LCALL SECOND
        MOV A,R0
        LCALL HEX_BCD
        MOV P0,A
        MOV A,R1
        LCALL HEX_BCD
        MOV P2,A
        MOV A,R2
        LCALL HEX_BCD
        MOV P1,A
        INC R0
        CJNE R0,#60,BACK
        MOV R0,#0
        INC R1
        CJNE R1,#60,BACK
        MOV R1,#0
        INC R2
        CJNE R2,#24,BACK
        MOV R2,#0
        LJMP BACK
```
فعال کردن وقفه ی خارجی صفر ویک وEA

تایمر صفر در مد ۲

تایمر صفرباید ۲۵۰ میکرو ثانیه تولید کند

پورت صفر ثانیه پورت یک ساعت و پورت دو دقیقه را نشان می دهد

وقفه های خارجی را حساس به لبه می کنیم

R0 ثانیه و R1 دقیقه و R2 ساعت را در خود می ریزد و با ریختن آن روی پورت آنرا نشان می دهد

زیر برنامه ای که یک ثانیه را برای ما می سازد

زیر برنامه ای که عدد هگز A را به دسیمال تبدیل ودر A می ریزد و برای نمایش اماده می کند

اعداد آماده را روی پورت می ریزیم

ثانیه را یکی اضافه می کند و اگر ٦٠ ثانیه شده بود آنرا صفر می کند و دقیقه را یکی اضافه می کند ، در غیر اینصورت ادامه می دهد

دقیقه را یکی اضافه می کند و اگر ٦٠ دقیقه شده بود آنرا صفر می کند و ساعت را یکی اضافه می کند ، در غیر اینصورت ادامه می دهد

ساعت را یکی اضافه می کند و اگر ٢٤ ساعت شده بود آنرا صفر می کند و باز می گردد ، در غیر اینصورت ادامه می دهد

```
SECOND:  SETB TR0
         MOV R3,#20
LOOP2:   MOV R4,#200
LOOP1:   JNB TF0,LOOP1
         CLR TF0
         DJNZ R4,LOOP1
         DJNZ R3,LOOP2
         CLR TR0
         RET
```

همانگونه که قبلا توضیح دادیم این زیر برنامه به وسیله ی تایمر صفر و در مد ۲ یک ثانیه را می سازد تایمر صفر ۲۵۰ میکرو ثانیه می شمارد اما اگر دقت کنیم می بینیم که ما زمان دستورات بالا مانند زیر برنامه HEX_BCD و غیره را محاسبه نکرده ایم با کمی زمانگیری تجربی و دادن مقدار ۲٤۹ یا کمتر به تایمر صفر می توان دقت ساعت را بالا برد

```
HEX_BCD: MOV B,#0AH
         DIV AB
         ANL A,#00001111B
         RL A
         RL A
         RL A
         RL A
         ANL B,#00001111B
         ORL A,B
         RET
```

این زیر برنامه عدد هگزا دسیمال A را که ممکن است ثانیه یا ساعت یا دقیقه باشد را می گیرد و آنرا در A می ریزد مثلا اگر عدد داخل A=1DH باشد باید مقدار هگزادسیمال آن A=29 شود تا بتوانیم آنرا نمایش دهیم این عمل به این ترتیب صورت می گیرد که A را بر 0AH تقسیم می کنیم خارج قسمت رقم پر ارزش دسیمال و باقیمانده نیز رقم کم ارزش دسیمال می شود و در زیر برنامه ی روبرو پس از تقسیم ، کلیه اعمال برای بردن خارج قسمت بخش پر ارزش یا سمت چپ A و ریختن عدد B یا باقیمانده در بخش کم ارزش A می باشد .

```
.ORG 0200H
INTER0:  INC R1
         CLR IE0  ;TCON.1
         CJNE R1,#60,L1
         MOV R1,#0
L1:      RETI
```

این زیر برنامه وقفه ی خارجی صفر می باشد که برای تنظیم دقیقه می باشد و چون و این وقفه حساس به لبه است با هر بار فشردن کلید مربوط یک دقیقه اضافه می شود و در صورتی که به ٦۰ رسیده باشیم آنرا صفر می کند ، توجه شود که در اینجا ما در آدرس 0200H زیر برنامه وقفه ی صفر را نوشته ایم که با جایی تداخل نداشته باشیم.

```
.ORG 0250H
INTER1:  INC R2
         CLR IE1  ;TCON.3
         CJNE R2,#24,L2
         MOV R2,#0
L2:      RETI
```

این زیر برنامه وقفه ی خارجی یک می باشد که برای تنظیم ساعت می باشد و چون و این وقفه حساس به لبه است با هر بار فشردن کلید مربوط یک ساعت اضافه می شود و در صورتی که به ۲٤ رسیده باشیم آنرا صفر می کند ، توجه شود که در اینجا ما در آدرس 0250H زیر برنامه وقفه ی یک را نوشته ایم که با جایی تداخل نداشته باشیم.

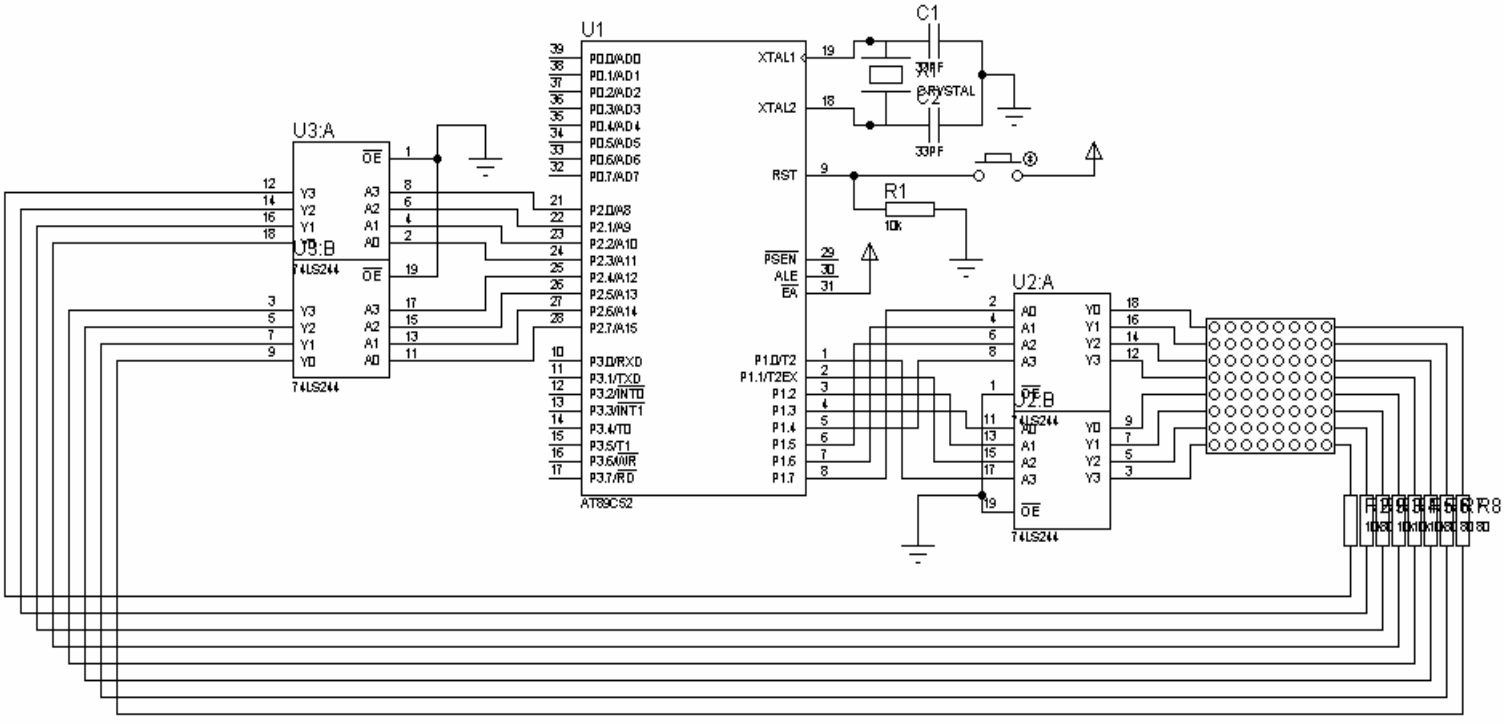
```
.END
```

END. نشان دهنده ی انتهای برنامه است . به نقطه قبل از **END.** توجه کنید در زمان نوشتن **ORG.** نیز آنرا می نویسیم توجه شود که در بعضی اسمبلر ها مانند نرم افزار **FRANKLIN** نباید این نقطه ها را گذاشت و نوشتن **END** و **ORG** کافی می باشد.

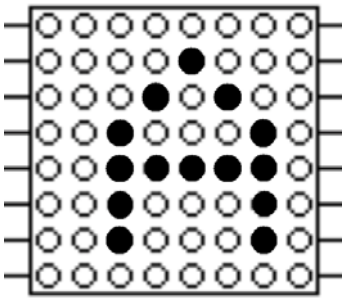
پروژه ساخت تابلو LED :

شاید در رفت و آمد های روزانه ی خود تابلو هایی را در خیابان دیده باشید که نوشته های مختلف بر روی آنها در حرکت است یا نوشته می شود و به شکل های مختلف این نوشته ها پاک می شوند و نوشته های دیگری جای آنها را می گیرند و این چرخه پس از چندی تکرار می شود این تابلو ها را در مغازه های بزرگ اتوبوس ها ترمینال ها و ... حتما دیده اید آری این تابلو ها ، تابلو LED هستند که از تعدا زیادی LED تشکیل شده اند . حتما با خود فکر می کنید که این تابلو ها چگونه کار می کنند و مثلا یک تابلوی ۳۲×۸ چگونه به وسیله ی یک میکرو کنترلرکار می کند در صورتی که این تابلو ۲۵٦ لامپ LED دارد و ما در میکرو کنترلر حداکثر ۳۲ خروجی داریم ؟ جواب این سوال این است که در هیچ زمانی همه ی این LED ها با هم روشن نیستند . همانطور که می دانیم چشم ما مقداری خطا دارد به این ترتیب که چشم تغییرات در زمان کمتر از ۱۰۰ میلی ثانیه را تقریبا نمی تواند تشخیص دهد یعنی اگر یک لامپ در ثانیه ۲۰ مرتبه خاموش و روشن شود چشم ما اصلا نمی تواند آنرا تشخیص دهد و آن را به صورت روشن دایم می بیند و به همین ترتیب بود که فیلم و سینما نیز اختراع شد به گونه ای که می دانید هر فیلم در هر ثانیه معمولا ۲٤ فریم مختلف دارد که به سرعت و در کنار هم حرکت می کنند و ما آنرا به صورت فیلم متحرک میبینیم . تابلو LED ها نیز از همین خاصیت استفاده می کنند یعنی در هر لحظه فقط یک ردیف مثلا ۸ تایی از LED ها روشن است و در لحظه ی بعدی ردیف دیگری از LED ها روشن است به این ترتیب ردیف های مختلف LED با سرعت زیادی روشن و خاموش می شوند و نوشته ای را پدید می آورند تا ما آنرا ببینیم به این عمل که هر بار در روی تابلو LED صورت می گیرد Refresh می گویند یعنی تازه سازی ، این نام به این خاطر است که LED ها هر بار روشن شده و در چند لحظه بعد دوباره احیا و روشن می شوند.

در این مثال که یک تابلوی کوچک ۸×۸ می باشد را شرح می دهیم سخت افزار این مدار متشکل از یک تابلوی ۸×۸ از LED هاست (این تابلو به عنوان یک قطعه در نرم افزار PROTEUS موجود می باشد) در این مدار از ۲ بافر ۷٤۲٤٤ برای تقویت جریان پورت ها استفاده شده است پورت یک به سطر های تابلو و پورت ۲ به ستون های تابلو متصل است.



همانگونه که در شکل مشاهده می شود در این مدار از ۲ آی سی ۷٤۲٤٤ برای تقویت جریان استفاده شده است و سطر ها به پورت یک ستون ها به پورت دو متصل است توجه شود که پر ارزش ترین بیت بالاترین سطر و کم ارزش ترین بیت پایین ترین بیت است ، حال اگر بخواهیم اطلاعات روی یک ستون (مثلا ستون متصل به **P2.3** یا ستون چهارماز سمت چپ) بریزیم باید اطلاعات را روی پورت یک ریخته و **P2.3** را صفر کنیم (در حالی که بقیه پورت ۲ یک میباشد) و اگر فرض کنیم در هر ردیف به طور متوسط ٤ **LED** در هر مرحله روشن باشد نیاز به حدود ۱۰۰ اهم مقاومت داریم که به صورت کلی در روی پایه ی هر ستون قرار می دهیم. حال به سراغ برنامه این تابلو می رویم. می خواهیم حرف **A** را روی آن نمایش دهیم.



همانگونه که در شکل بالا می بینید برای نوشتن هر حرف مانند حرف **A** نیاز به ۸ بایت اطلاعات داریم که آنها را در آدرسی از حافظه می نویسیم و بعدا به کار می بریم مثلا برای حرف **A** ستون اول در آدرس ۳۲ هگز نوشتیم ، ستون دوم در ۳۳ ، ستون سوم در ۳٤ و.... ستون هشتم در **39H** . برای هر حرف و کاراکتر نیاز به همین ۸ بیت اطلاعات داریم

```
.ORG 0000H
        MOV 32H,#0
        MOV 33H,#0
        MOV 34H,#00011110B
        MOV 35H,#00101000B
        MOV 36H,#01001000B
        MOV 37H,#00101000B
        MOV 38H,#00011110B
        MOV 39H,#0
```

در این قسمت **A** را طوری مقدار می دهیم که در هر بار یک ستون روشن باشد و با دستور **RL A** هر بار آنرا به به سمت چپ شیفت می دهیم و **R0** را اضافه می کنیم تا ستون بعدی نمایش داده شود در اینجا ما نیاز به تاخیری داریم تا **LED** ها زمانی روشن بمانند ، برای فهم بهتر فرض کنیم ما می خواهیم این پروسه ۱۰۰ بار در ثانیه تکرار شود تا چشم ما اصلا نتواند آنرا تشخیص دهد برای این کار هر پروسه باید ۱۰ میلی ثانیه طول بکشد یعنی ۱۰ تقسیم بر ۸ برای هر ستون . پس هر ستون باید ۱,۲۵ میلی ثانیه روشن باشد و این تابع **DEL1** حدود ۱,۲۵ میلی ثانیه تاخیر روی هر ستون می گذارد . در این برنامه دقت کنیم که ما فقط یک کراکتر را نشان دادیم اگر بخواهیم تعداد بیشتری کاراکتر را نیز نمایش دهیم باید به همین ترتیب عمل کنیم و بدانیم که هر کاراکتر در کجای حافظه است مثلا فرض کنیم برای کاراکتر **B** در آدرس **40H** تا **47H** نوشته ایم برای نمایش باید به **R0** در خط **D1** (در برنامه) به جای مقدار ۳۲ مقدار ٤۰ را بدهیم به این ترتیب هر حرف با یک مکان مشخص می شود و می توان آنرا نمایش داد .

```
D1:     MOV R0,#32H
        MOV A,#11111110B
BACK:   MOV P1,@R0
        MOV P2,A
        RL A
        INC R0
        LCALL DEL1
        CJNE  R0,#40H,BACK
        LJMP D1

DEL1:   MOV R7,#5
L1:     MOV R6,#250
L2:     DJNZ R6,L2
        DJNZ R7,L1
        RET
.END
```

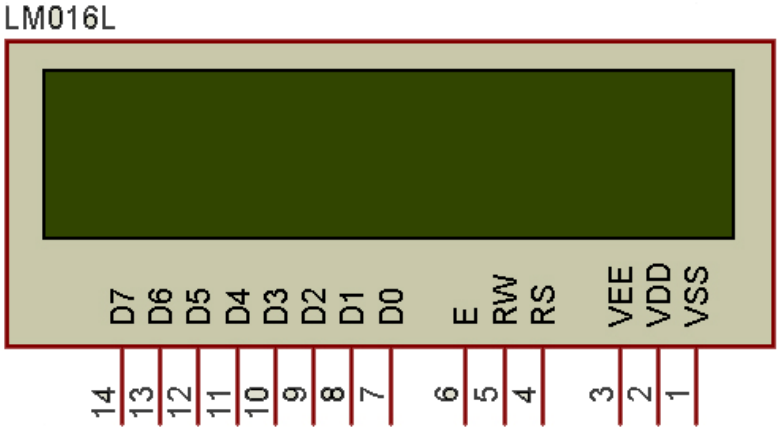
لازم به ذکر است برای نمایش متحرک حروف نیز با تغییر کوچکی در همین برنامه می توان این عمل را انجام داد. به مثال روبرو توجه کنید در این زیر برنامه اگر بدانیم که هر پروسه ی کامل ۱۰ میلی ثانیه طول می کشد ما می خواهیم هر ۱۰۰ میلی ثانیه یک خانه به سمت چپ برویم برای این عمل باید مقداری که در **A** هست را یک خانه به سمت راست شیفت دهیم به همین منظور زیر برنامه ی **SHIFT** با توجه به مقدار **R4** که باعث می شود هر پروسه ۱۰ مرتبه تکرار شود (یعنی ۱۰۰ میلی ثانیه) و سپس یک خانه مقدار **R5** که همان مقدار **A** می باشد را به سمت راست شیفت می دهد پس با اجرای این برنامه ما روی تابلو **LED** خود حرف **A** را می بینیم که مکررا به سمت چپ رفته و دوباره از سمت راست وارد می شود. به همین ترتیب ما میتوانیم انواع **effect** و **motion** (حرکت) را روی حروف داشته باشیم.

همانگونه که گفتیم به راحتی می توان کلمات مختلف را به این وسیله نشان داد برای مثال فرض کنیم می خواهیم کلمه ی **DOG** را نمایش دهیم و کاراکتر **D** را در آدرس ۵۰ تا ۵۷ هگز و **O** را در **58H** تا **5FH** و **G** را در **5FH** تا **67H** . برای نمایش این سه در آدرسی دلخواه از حافظه که با حروف ما تداخل نداشته باشد (مثلا **80H**) می نویسیم **50H** و در خانه ی بعد از آن یعنی در **81H** می نویسیم **58H** و در **82H** می نویسیم **5FH** . حال به **R1** مقدار **80H** را می دهیم و آنرا طوری در برنامه می گذاریم که هر بار یک کاراکتر از صفحه بیرون می رود کاراکتر بعدی به دنبال آن حرکت کند و به داخل صفحه بیاید.

```
.ORG 0000H
        MOV 32H,#0
        MOV 33H,#0
        MOV 34H,#00011110B
        MOV 35H,#00101000B
        MOV 36H,#01001000B
        MOV 37H,#00101000B
        MOV 38H,#00011110B
        MOV 39H,#0

        MOV R4,#5
        MOV R5,#11111110B
D1:     MOV R0,#32H
        MOV A,R5
        LCALL SHIFT
BACK:   MOV P1,@R0
        MOV P2,A
        RL A
        INC R0
        LCALL DEL1
        CJNE  R0,#40H,BACK
        LJMP D1

DEL1:   MOV R7,#5
L1:     MOV R6,#250
L2:     DJNZ R6,L2
        DJNZ R7,L1
        RET

SHIFT:  DJNZ R4,D2
         MOV R4,#10
         MOV A,R5
         RR A
         MOV R5,A
D2:     RET

.END
```

پروژه ی قفل الکترونیکی :

در این پروژه سعی داریم یک قفل الکترونیکی با ورودی صفحه کلید و خروجی LCD بسازیم همانطور که میدانید LCD یا نمایشگر های کریستال مایع (Liquid Crystal Display) امروزه در انواع وسایل الکترونیکی کاربرد بسیار زیادی دارند مثلا در انواع و اقسام تلفن های خانگی و همراه ، ماشین های حساب ، اسباب بازی ها و انواع قفل ها وسیستم های امنیتی و ... . در این پروژه فرض کنیم قفلی برای در یک اتاق داریم که به وسیله یک صفحه کلید تلفنی به آن یک پسورد ۵ رقمی می دهیم و در صورتی که درست بود یک رله که به قفل یک در متصل است را روشن می کند و در باز می شود و منتظر می شود تا در بسته شود و دوباره به همان مراحل اول بر می گردد. در این برنامه از LCD استفاده شده است پس بهتر است ابتدا ساختمان داخلی و طرز کار LCD را توضیح دهیم.در این برنامه ما از یک LCD ۲×۱۶ استفاده می کنیم یعنی این LCD دو سطر ۱۶ خانه ای دارد و می تواند در هر سطر تا ۱۶ کاراکتر را نمایش دهد . مزیت LCD نسبت به 7SEG این است که این قطعه کلیه کاراکتر های ASCII را در خود دارد و در صورت نیاز می توان کاراکتر های جدیدی نیز برای آن تعریف کرد (مانند کاراکتر های زبان فارسی) و کار با آن نیز بسیار راحت و آسان می باشد در زیر شکل یک LCD را می بینید که پایه های آن به شرح زیر است.



VSS : زمین

VDD : منبع تغذیه

VEE : برای تنظیم میزان روشنایی LCD

RS : برای انتخاب داده یا دستور العمل . اگر RS=0 باشد هر چیزی که به پایه های داده ی LCD بدهیم دستور العمل محسوب می شود و اگر RS=1 باشد هر چیزی که به پایه های داده بدهیم داده به حساب می آیند.

RW : این پایه مشخص می کند که از LCD داریم اطلاعات می خوانیم یا بر روی آن می نویسیم اگر RW=1 باشد داده را از LCD می خوانیم واگر RW=0 باشد بر روی LCD داده یا دستور العمل می نویسیم .

E : این پایه کلاک ورودی LCD است و زمانی که می خواهیم داده یا اطلاعات بر روی LCD بنویسیم یا بخوانیم باید یک لبه ی بالا به پایین بر روی این پایه بفرستیم .

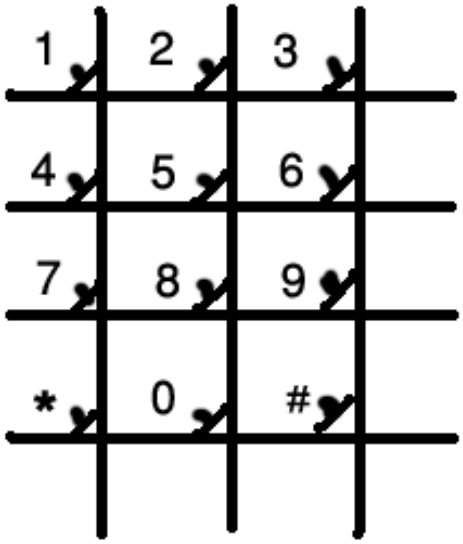
D0 تا D7 : این پایه ها برای دادن داده یا دستور بر روی LCD می باشد به این ترتیب ما می توانیم ۲ به توان ۸ یعنی ۲۵۶ داده یا دستور به LCD بدهیم.

هر کاراکتر در LCD از یک ماتریس تشکیل شده است به این ترتیب که مثلا در این LCD ما باید LCD خود را برای ماتریس ۷×۵ ساماندهی کنیم ، منظور از ماتریس این است که مانند تابلو LED در اینجا نیز هر

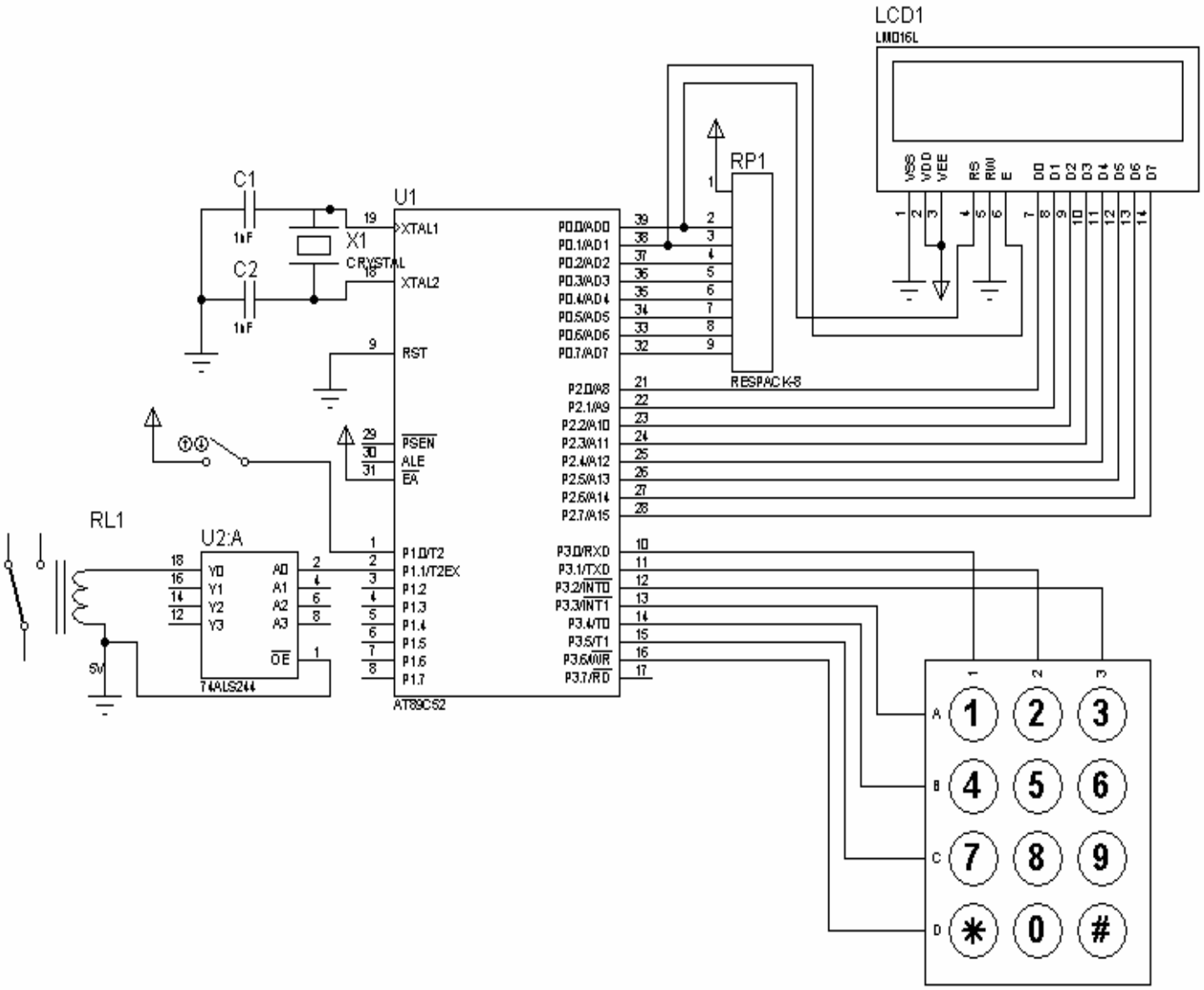
کاراکتر از بخش های بسیار ریزی تشکیل شده است تا بتوانند این کاراکتر را بسازند این نقطه های بسیار ریز **Pixel** های سازنده ی کاراکتر ها هستند.

برای کار با **LCD** که در این مثال به کار رفته باید چند عمل را انجام دهیم ابتدا آنرا به ۲ سطر و ماتریس ۵×۷ ساماندهی کنیم سپس **LCD** را روشن کنیم و در صورت نیاز مکان نما را روشن یا خاموش کنیم و بقیه کارها را با استفاده از جدول دستور العمل ها انجام دهیم . توجه شود که اگر ما کاراکتری را بر روی **LCD** نمایش دادیم و کاراکتر بعدی را فرستادیم این کاراکتر بلافاصله بعد از قبلی قرار میگیرد و در صورت نیاز باید از دستور العمل برای دادن افکت های مختلف بر روی **LCD** استفاده کرد.

کد HEX	دستور العمل به LCD
1	پاک کردن صفحه نمایش
2	بازگشت مکان نما به اول (شروع)
4	کاهش مکان نما (جابجایی مکان نما به چپ)
6	افزایش مکان نما (جابجایی مکان نما به راست)
5	جابجایی نمایش به راست
7	جابجایه نمایش به چپ
8	نمایش خاموش مکان نما خاموش
A	نمایش خاموش مکان نما روشن
C	نمایش روشن مکان نما خاموش
E	نمایش روشن مکان نما روشن
F	نمایش روشن مکان نما چشمک زن
10	جابجایی محل مکان نما به چپ
14	جابجایی محل مکان نما به راست
18	کل صفحه نمایش به چپ
1C	کل نمایش یه راست
C0	مکان نما به آغاز خط دوم برود
38	ساماندهی ۲ خط و ماتریس ۵×۷



در این پروژه قطعه ی دیگری نیز داریم که صفحه کلید می باشد صفحه کلید تلفنی ٤ سطر و ۳ ستون دارد و ساختمان داخلی آن به شکل روبرو می باشد با فشرده شدن هر یک از این کلید ها یک سطر و یک ستون به هم متصل می شوند مثلا وقتی عدد ۸ را فشار دهیم ستون وسط به سطر سوم متصل می شود . ما نیز از این خاصیت استفاده می کنیم و می توانیم تشخیص دهیم که کدام کلید فشرده شده است به این صورت که روی یک سطر صفر و روی بقیه یک می گذاریم در صورتی که روی ستونی صفر ببینیم کلید مورد نظر را می توانیم تشخیص دهیم . حال سخت افزار مدار را به طور کامل می بینیم.



در این شکل همانطور که می بینید یک صفحه کلید به پورت ۳ متصل شده است (ستون های آن به **P3.0** تا **P3.2** و سطر های آن به **P3.3** تا **P3.7**) پایه های داده ی **LCD** به پورت ۲ متصل شده اند و پایه ی **RW** نیز زمین شده است ، پایه ی **RS** به **P0.0** و پایه ی **E** به **P0.1** که همانگونه که قبلا گفتیم پورت صفر باید **Pull up** شود که این کار توسط مقاومت ۸ تایی انجام شده است ، پایه ی **P1.1** از میکرو به رله در متصل است و هنگامی که این پایه فعال شود رله نیز فعال شده و در را باز می کند ، توجه داشته باشید که به آی سی ۷٤۲٤٤ برای تقویت جریان نیاز داریم ، پایه ی **P1.0** نیز به سنسور در متصل است و برای ما مشخص می کند که در باز یا بسته است در صورتی که در بسته باشد این پایه یک و در غیر اینصورت صفر می شود (البته سخت افزار این مدار یک نقص کوچک دارد آن را بیابید).

```
ORG 0000H
        CLR P1.1
        MOV P2,#38H
        LCALL CWRITE
        MOV P2,#0EH
        LCALL CWRITE
        LCALL NEW
F1:     LCALL WAIT
        LJMP F1

CWRITE: LCALL DELAY
        CLR P0.0
        SETB P0.1
        CLR P0.1
        RET

DWRITE: LCALL DELAY
        SETB P0.0
        SETB P0.1
        CLR P0.1
        RET

DELAY:  MOV R7,#255
L1:     MOV R6,#255
L2:     DJNZ R6,L2
        DJNZ R7,L1
        RET

CHANGE: MOV P2,#01H
        LCALL CWRITE
        MOV P2,#02H
        LCALL CWRITE
        MOV P2,#"O"
        LCALL DWRITE
        MOV P2,#"L"
        LCALL DWRITE
        MOV P2,#"D"
        LCALL DWRITE
        MOV P2,#" "
        LCALL DWRITE
        MOV P2,#"P"
        LCALL DWRITE
        MOV P2,#"A"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"W"
        LCALL DWRITE
        MOV P2,#"O"
        LCALL DWRITE
        MOV P2,#"R"
        LCALL DWRITE
        MOV P2,#"D"
        LCALL DWRITE
        MOV P2,#"?"
        LCALL DWRITE
        MOV P2,#0C0H
        LCALL CWRITE
        LCALL GETKEY
        MOV 50H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
```

رله را خاموش می کنیم

روی پورت ۲ عدد **38H** را می ریزیم تا **LCD** را به ۲ سطر ۷×۵ ساماندهی کنیم

CWRITE برای نوشتن دستور العمل بر روی **LCD** می باشد

با ریختن **0EH** بر روی پورت ۲ **LCD** و مکان نما را روشن می کنیم

حال با زیر برنامه ی **NEW** پسوردی را می گیریم

زیر برنامه ی **WAIT** منتظر می ماند تا کلیدی زده شده و سپس پسورد را چک می کند و به عبارت دیگر زیر برنامه ی اصلی ما می باشد

این زیر برنامه دستور العمل را بر روی **LCD** می ریزد و این عمل با صفر کردن پایه ی **RS** و زدن کلاک بالا به پایین بر روی پایه ی **E** صورت میگیرد

این زیر برنامه داده را برای نمایش روی **LCD** می ریزد و این عمل با یک کردن پایه ی **RS** و زدن کلاک بالا به پایین بر روی پایه ی **E** صورت می گیرد

تابع تاخیر در این زیر برنامه ها به ۲ جهت استفاده شده است یکی اینکه وقتی ما داده یا دستوری به **LCD** می دهیم زمانی طول می کشد تا آنرا نمایش یا اجرا کند و در این مدت اگر چیزی به **LCD** بدهیم کار نخواهد کرد به همین خاطر تاخیری در مدت می گذاریم و دوم اینکه برای نمایش این تاخیر حالت زیبایی را ایجاد می کند.

این زیر برنامه برای عوض کردن پسورد است و در صورتی که * را فشار دهیم به این زیر برنامه می آییم همانگونه که می بینید خط اول زیر برنامه برای پاک کردن **LCD** و خط سوم برای آوردن مکان نما به اول می باشد و سپس **"OLD PASSWORD?"** را نمایش می دهد

در اینجا به ابتدای خط دوم می رود و منتظر می شود تا پسورد قبلی را بزنیم و آنرا در آدرس **50H** تا **54H** ذخیره می کند و به ازای هر کلیدی که زده می شود بر روی یک * نشان می دهد به این ترتیب ۵ رقم را باید وارد کنیم و سپس کلید # را بزنیمتا به محل اخذ پسورد جدید برویم

زیر برنامه ی **GETKEY** منتظر می ماند تا کلیدی زده شود سپس مقدار آنرا در ثبات **A** می ریزد با توجه به اینکه **10= *** و **11= #**

```
        MOV 51H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 52H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 53H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 54H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        CJNE A,#11,NO1
        LCALL CHECK
        CJNE R0,#1,NO1
        LJMP NEW
NO1:    LJMP NO
NEW:    MOV P2,#01H
        LCALL CWRITE
        MOV P2,#02H
        LCALL CWRITE
        MOV P2,#"I"
        LCALL DWRITE
        MOV P2,#"N"
        LCALL DWRITE
        MOV P2,#"T"
        LCALL DWRITE
        MOV P2,#"E"
        LCALL DWRITE
        MOV P2,#"R"
        LCALL DWRITE
        MOV P2,#" "
        LCALL DWRITE
        MOV P2,#"N"
        LCALL DWRITE
        MOV P2,#"E"
        LCALL DWRITE
        MOV P2,#"W"
        LCALL DWRITE
        MOV P2,#" "
        LCALL DWRITE
        MOV P2,#"P"
        LCALL DWRITE
        MOV P2,#"A"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"."
        LCALL DWRITE
        MOV P2,#0C0H
        LCALL CWRITE
        LCALL GETKEY
        MOV 60H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 61H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 62H,A
```

در اینجا پس از ۵ کلید زده شده باید # رابزنیم تا به زیر برنامه ی **CHECK** برویم و ببینیم که رمز ما درست بوده است یا خیر در غیر این صورت به زیر برنامه ی **NO** می رویم که اعلام خطا می کند. در اینجا می بینیم که ما برای پرش به زیر برنامه **NO** ابتدا به **NO1** که در نزدیکی **CJNE R0,#1,NO1** قرار دارد پرش می کنیم و سپس به **NO** زیرا دستور **CJNE** تا به ۱۶ خط بالاتر و پایین تر نمی تواند پرش کند و برای جبران باید به جایی نزدیک پرش کنیم و سپس پرش بلند انجام دهیم همچنین در اینجا **R0** نشان دهنده ی درست یا غلط بودن پسورد است اگر درست بود **R0=1** که به زیر برنامه ی **NEW** برای گرفتن پسورد جدید می رویم و در غیر اینصورت **R0=0** خواهد بود که باید به زیر برنامه ی **NO** پرش کنیم

زیر برنامه ی **NEW** پسورد اصلی ما را می گیرد و در آدرس **60H** تا **64H** می ریزد در اینجا نیز همانند قبل ابتدا **LCD** را پاک کرده و به ابتدا می رویم و می نویسیم **INTER NEW PASS.** و سپس به خط دوم رفته و به ازای هر کلیدی که زده شد یک * چاپ می کنیم و پس از زده شدن ۵ کلید باید کلید # رافشار دهیم تا رمز تغییر یابد و گرنه دوباره باید رمز جدید را وارد کنیم

```
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 63H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 64H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        CJNE A,#11,NEW1
        RET
NEW1:   LJMP NEW

NO:     MOV P2,#01H
        LCALL CWRITE
        MOV P2,#02H
        LCALL CWRITE
        MOV P2,#"W"
        LCALL DWRITE
        MOV P2,#"R"
        LCALL DWRITE
        MOV P2,#"O"
        LCALL DWRITE
        MOV P2,#"N"
        LCALL DWRITE
        MOV P2,#"G"
        LCALL DWRITE
        DEC SP
        DEC SP
        LJMP F1

WAIT:   MOV P2,#01H
        LCALL CWRITE
        MOV P2,#02H
        LCALL CWRITE
        MOV P2,#"I"
        LCALL DWRITE
        MOV P2,#"N"
        LCALL DWRITE
        MOV P2,#"T"
        LCALL DWRITE
        MOV P2,#"E"
        LCALL DWRITE
        MOV P2,#"R"
        LCALL DWRITE
        MOV P2,#" "
        LCALL DWRITE
        MOV P2,#"T"
        LCALL DWRITE
        MOV P2,#"H"
        LCALL DWRITE
        MOV P2,#"E"
        LCALL DWRITE
        MOV P2,#" "
        LCALL DWRITE
        MOV P2,#"P"
        LCALL DWRITE
        MOV P2,#"A"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"S"
        LCALL DWRITE
        MOV P2,#"."
        LCALL DWRITE
```

در اینجا اگر کلید ششم زده شده # نبود دوباره به زیر برنامه ی **NEW** بر میگردیم که باز هم به دلیل استفاده از **CJNE** و زیاد بودن فاصله باید ابتدا به **NEW1** پرش کنیم

این زیر برنامه ابتدا **LCD** را پاک کرده و سپس بر روی آن می نویسد **WRONG** که در صورت اشتباه بودن عمل در هر زیر برنامه ای که باشیم به اینجا پرش می کنیم در این زیر برنامه باید توجه کنیم که ما همواره به این زیر برنامه پرش کرده ایم پس در اینصورت ما نمی توانیم در انتهای این زیر برنامه **RET** بگذاریم پس برای بازگشت باید چه کاری انجام دهیم ؟ برای اجرای دستور **LCALL** همانطور که قبلا گفتیم ما آدرس فعلی را در جایی ذخیره می کنیم و با دستور **RET** آنرا دوباره باز می گردانیم . ما در میکرو ثباتی داریم به نام **SP** که در در هنگام اجرای دستور **LCALL** به آنجاییِ که ما آدرس محل را ریخته ایم اشاره می کند یعنی زمانی که دستور **RET** اجرا می شود به این ثبات نگاه می کند و به هر آدرسی که در این ثبات بود می رود و در آنجا آدرس جایی که قبل از دستور **LCALL** بوده ایم را می یابد اما در اینجا ما چون همواره از یک زیر برنامه به اینجا پرش کرده ایم و در زیر برنامه نیز از دستور **RET** به علت پرش به این زیر برنامه استفاده نکرده ایم باید **SP** را دو واحد (به علت ١٦ بیتی بودن آدرس باس و ٨ بیتی بودن **SP**) کاهش می دهیم و به ابتدای حلقه برنامه پرش می کنیم .

در زیر برنامه ی **WAIT** ما ابتدا **LCD** را پاک کرده و روی آن می نویسیم **.INTER THE PASS** و با استفاده از زیر برنامه ی **GETKEY** منتظر می مانیم تا کلیدی زده شود و آنگاه به ازای هر کلید یک * روی **LCD** چاپ می کنیم

در اینجا ما به ابتدای خط دوم رفته ایم و منتظر زدن کلیدی شده ایم وآنرا رد آدرس ۵۰ تا ٦۰ هگز به صورت موقت ریخته ایم

```
        MOV P2,#0C0H
        LCALL CWRITE
        LCALL GETKEY
        MOV 50H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 51H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 52H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 53H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        MOV 54H,A
        MOV P2,#"*"
        LCALL DWRITE
        LCALL GETKEY
        CJNE A,#11,NO2
        LCALL CHECK
        CJNE R0,#1,NO2
        SETB P1.1
        MOV P2,#01H
        LCALL CWRITE
        MOV P2,#02H
        LCALL CWRITE
        MOV P2,#"O"
        LCALL DWRITE
        MOV P2,#"K"
        LCALL DWRITE
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        CLR P1.1
        LJMP FIN2
        NO2:LJMP NO
FIN2:   RET
```

در اینجا نیز اگر کلید ششم # بود به مر حله بعد می رویم در غیر اینصورت به زیر برنامه ی **NO** پرش می کنیم

بعد از اجرای زیر برنامه ی **CHECK** اگر **R0=1** بـود در ایــن صــورت پــسورد ورودی درست بوده ، **OK** را روی **LCD** نوشته و رله ی در را فعال می کنیم برای مدت چدین تاخیر تا در کاملا باز شده سپس دوباره به مراحل اصلی باز می گردیم

زیر برنامه **CHECK** نیز به بررسی این می پردازد که آیا رمز ورودی (آدرس ۵۰ تا ١٤ هگز) با رمز اصلی همخوانی دارد یا خیر .
اگر درست بود **R0** را یک و در غیر اینصورت **R0** را صفر می کند

```
CHECK:  MOV A,50H
        CJNE A,60H,WRONG
        MOV A,51H
        CJNE A,61H,WRONG
        MOV A,52H
        CJNE A,62H,WRONG
        MOV A,53H
        CJNE A,63H,WRONG
        MOV A,54H
        CJNE A,64H,WRONG
        MOV R0,#1
        LJMP FIN1
WRONG:  MOV R0,#0
```

```
FIN1:     RET

GETKEY:   MOV P3,#0FFH
          CLR P3.3
          CLR P3.4
          CLR P3.5
          CLR P3.6
          JNB P3.0,HIT
          JNB P3.1,HIT
          JNB P3.2,HIT
          LJMP GETKEY
HIT:      MOV P3,#0FFH
          CLR P3.3
          JNB P3.0,G1
          JNB P3.1,G2
          JNB P3.2,G3
          MOV P3,#0FFH
          CLR P3.4
          JNB P3.0,G4
          JNB P3.1,G5
          JNB P3.2,G6
          MOV P3,#0FFH
          CLR P3.5
          JNB P3.0,G7
          JNB P3.1,G8
          JNB P3.2,G9
          MOV P3,#0FFH
          CLR P3.6
          JNB P3.0,G10
          JNB P3.1,G0
          JNB P3.2,G11
G1:       MOV A,#1
          LJMP GO
G2:       MOV A,#2
          LJMP GO
G3:       MOV A,#3
          LJMP GO
G4:       MOV A,#4
          LJMP GO
G5:       MOV A,#5
          LJMP GO
G6:       MOV A,#6
          LJMP GO
G7:       MOV A,#7
          LJMP GO
G8:       MOV A,#8
          LJMP GO
G9:       MOV A,#9
          LJMP GO
G10:      MOV A,#10
          LJMP GO
G0:       MOV A,#0
          LJMP GO
G11:      MOV A,#11
          LJMP GO

GO:       LCALL DELAY
          CJNE A,#10,FIN
          DEC SP
          DEC SP
          LCALL CHANGE
FIN:      RET

END
```

زیر برنامه ی **GET KEY** ابتدا با صفر کردن همه ی سطرها منتظر می ماند تا ستونی صفر شودکه در اینصورت می توان فهمید کلیدی فشرده شده اگر کلیدی فشرده شود به زیر برنامه ی **HIT** که برای تشخیص شماره ی کلید است می رویم و اگر کلیدی زده نشود ما در این حلقه می مانیم تا کلیدی زده شود

حال اگر کلیدی زده شد ما با صفر کردن یک سطر و یک کردن بقیه سطر ها می بینیم که آیا کلیدی در آن سطر زده شده یا خیر اگر زده شده بود عدد آن کلید را در ثبات **A** می ریزیم و از زیر برنامه خارج می شویم که البته ممکن است نویزی باعث صفر شدن یک پایه در یک لحظه شود که برای گرفتن نویز نیز می توانیم با تاخیر کمی چند مرتبه آن پایه را چک کنیم اگر همچنان صفر باشد می توان فه مید که این نویز نبوده و فشرده شدن کلید است که ما در اینجا این عمل را انجام نداده ایم

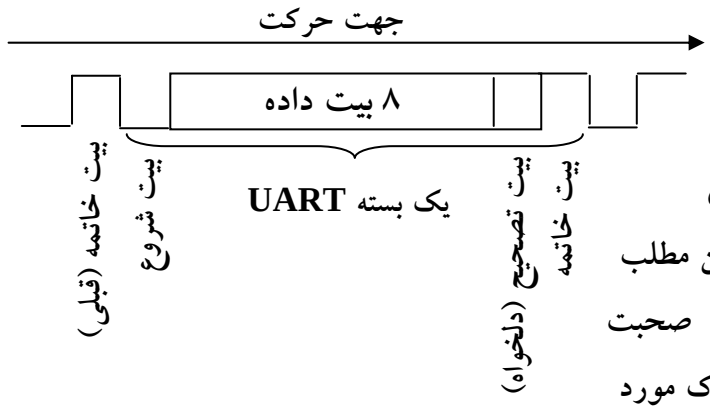
این برچسب ها که با زدن هر عددی به همان برچسب وارد می شویم عدد مورد نظر (یا کد کلید) را در ثبات **A** می ریزد و به انتهای برنامه می رود

در اینجا می بینید که از تاخیر استفاده شده علت آن هم این است که اگر یک کلیدی را فشار دهیم و تاخیر در اینجا نباشد میکرو آن کلید را چندین مرتبه می خواند اما در اینجا اگر بیش از زمن تاخیر دست ما روی کلیدی باشد آنرا ۲ یا چند مرتبه می خواند همچنین اگر کلید * را زدیم چون در زمان اجرای زیر برنامه هستیم **SP** را دو واحد کاهش می دهیم و به زیر برنامه ی عوض کردن رمز پرش می کنیم

باید دقت کرد که این برنامه از نظر امنیتی اشکالاتی دارد که باید آنرا بر طرف کرد . آیا می توانید این اشکالات را بیابید؟

پروژه ارتباط سریال بین دو میکروکنترلر

در این مثال می خواهیم با استفاده از ارتباط سریال بین دو میکروکنترلر ، به یک میکرو دو عـدد داده (ورودی به یک پورت بدهیم) و همان دوعدد را به عنوان خروجی در میکروی دیگر ببینیم ایـــن ارتبـــاط بـــین ۲ میکروکنترلر را می توانیم به صورت بسیار ساده با استفاده از یک پورت انجام دهیم اما در اینصورت نیاز به ۸ سیم داریم پس هزینه ی بیشتری می خواهیم ، ۸ بیت از ۲ میکروی ما مشغول شده پـس پایـه هـای بیـشتری مشغولند ، بحث مهم تری که در اینجا پیش می آید این است که چون در اینجا ۸ سیم داریم احتما اینکه نویز روی ۸ بیت بیفتد بسیار بیشتر از یک یا دو سیم است ، پس در کل به این نتیجه می رسیم که در جا هایی کـه به سرعت خیلی بالا نیاز نداریم ارتباط سریال بسیار مفید تر و کار آمد تر می باشد . در ارتباط سـریال مـا از یک سری قرارداد استفاده می کنیم . مهم ترین قراردادی که در ارتباط سریال بین میکرو کنترلر ها با کـامپیوتر و میکرو های دیگر استفاده می شود UART می باشد این پروتکل از ۱۰ یا ۱۱ بیت تشکیل شده اسـت کـه بیت اول آن با نام بیت شروع حتما صفر می باشد و پس از آن ۸ بیت داده ی مـا مـی آینـد و پـس از آن در صورت نیاز بیت پریتی یا بیت تصحیح خطا می آید و بیت بعدی که بیت ۱۰ یـا ۱۱ مـی باشـد بیـت خاتمـه نامیده می شود و حتما باید ۱ باشد.



حال به سراغ میکرو و ارتباط سریال به وسیله آن می رویم : میکروکنترلر ثباتی به نام SCON دارد که این ثبات برای ارتباط سریال می باشد. قبل از شرح این مطلب ابتدا کمی در مورد سرعت انتقال داده صحبت می کنیم . در میکرو کنترلر ۸۰۵۱ کلاک مورد برای ارتباط سریال توسط تایمر یک تولید می شود (البته در میکرو هایی که تایمر ۲ دارند تـایمر ۲ نیـز مـی تواند تولید کننده ی این کلاک باشد) کلاک تولید شده توسط تایمر یک در مد ارتباط سریل تقسیم بر ۳۲ می شود به این ترتیب اگر بدانیم که کلاک ورودی میکرو بعد از این که تقـسیم بـر ۱۲ شـد در هنگـام ورود بـه بلوک ارتباط سریال تقسیم بر ۳۲ می شود و کریستال ما از نوع 11.0592 MHz باشد پس از تقسیم کـلاک بر ۱۲ فرکانس 921.6 KHz را داریم و پس از تقسیم بر ۳۲ فرکانس 28800 Hz را داریـم پـس مـاکزیمم سرعتی که ما می توانیم با آن ارتباط سریال برقرار کنیم همین 28800 بیت بـر ثانیـه (bps) مـی باشـد بـا استفاده از تایمر یک ما باید سرعت را به صورت دلخواه تنظیم کنیم . در اینجا بهتر است بدانیم یـک بیـت در چه صورتی ارسال یا دریافت می شود ؟ پاسخ این سوال این است که هر زمانی که TF1 یک شـد مـا یـک بیت را می فرستیم با این توصیف اگر ما فرضا در مد ۲ تایمر ۱ باشیم و عدد ۲۵۳=۳-۲۵۶ را در تـایمر یـک بریزیم ۳ کلاک اسیلاتور سریال طول می کشد تا TF1 یک شود پس سرعت انتقـال اطلاعـات مـا برابـر بـا ۲۸۸۰۰ تقسیم بر ۳ که می شود 9600 bps به این ترتیب ما سرعت های مختلفی را می توانیم ایجاد کنیم.

حال برای توضیح ارتباط سریال ثبات **SCON** را توضیح می دهیم .

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

RI : این بیت نشان دهنده ی این است که دریافت کامل شده است و این بیت در نیمه های راه بیت ختم یک می شود و باید برای دریافت بعدی آنرا پاک کرد.

TI : این بیت نشان دهنده این است که ارسال کامل شده است و در هنگامی که بیت ختم را ارسال کردیم این بیت یک می شود و باید به صورت نرم افزاری آنرا صفر کرد ، البته توجه شود که این بیت و بیت قبلی پرچم های وقفه ی سریال نیز هستند و در صورتی که وقفهی سریال فعال باشد با فعال شدن این بیتها به زیر برنامه ی وقف ی سریال خواهیم رفت.

RB8 و **TB8** : این دو بیت به ترتیب بیت نهم دریافت و بیت نهم ارسال می باشد و به عنوان همان بیت تصحیح خطا که قبلا گفتیم در موارد بسیار محدود به کار می روند.

REN : این بیت فعال ساز دریافت می باشد در صورتی که این بیت صفر باشد دریافت صورت نمی گیرد.

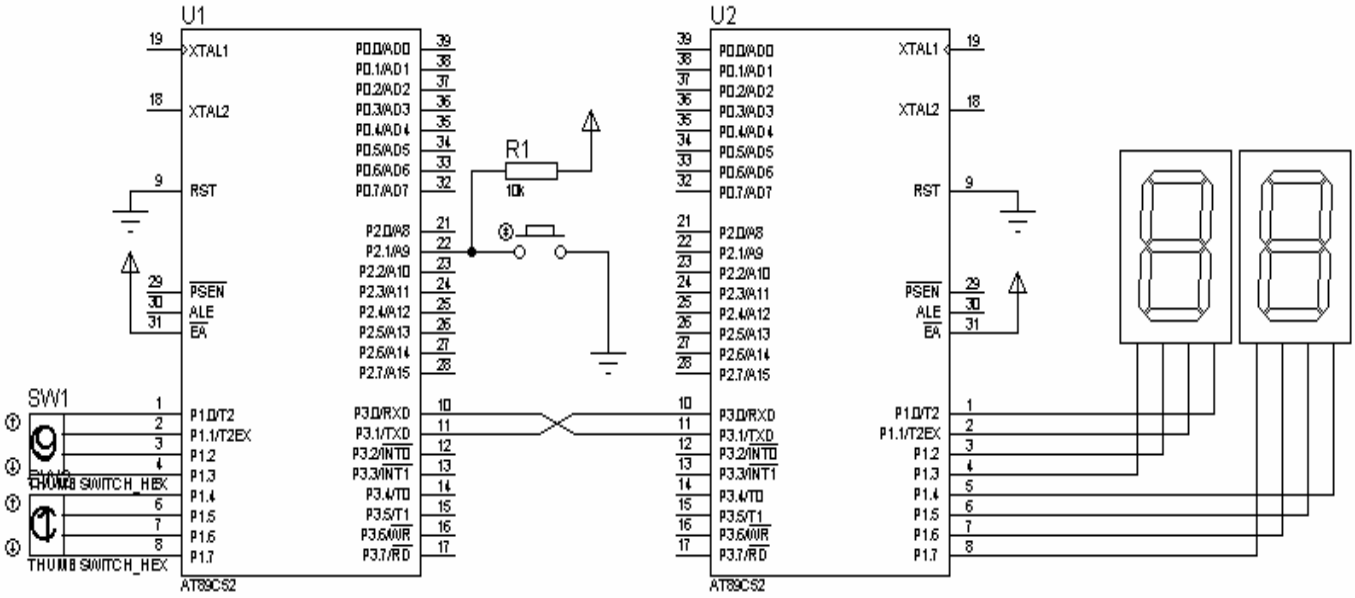
SM2 و **SM1** و **SM0** : این سه بیت برای تعیین مد ارتباط سریال می باشند.

SM0	SM1	مد ارتباط سریال
0	0	مد صفر ارتباط سریال
0	1	مد یک ارتباط سریال
1	0	مد دو ارتباط سریال
1	1	مد سه ارتباط سریال

SM2 نیز برای مد چند پردازنده ای می باشد که در اینجا به توضیح آن نمی پردازیم.

برای ارتباط سریال ما ثباتی به نام **SBUF** داریم که هر چیزی که قرار است ارسال یا دریافت شود در این ثبات ریخته می شود و با این ثبات نیز همانند یک ثبات معمولی با سدتور **MOV** کار می کنیم . از میان مد های مختلف مد صفر مد شیفت رجیستر می باشد به این ترتیب که برای ارسال هر چه در **SBUF** بود را تک تک به روی پایه ی **TXD** می ریزد و برای دریافت نیز هرچه روی پایه ی **RXD** بود را به ترتیب در **SBUF** می ریزد و تک تک به سمت چپ شیفت می دهد البته این مد با فرکانس اسیلاتور تقسیم بر ۱۲ کار میکند و هیچ سرعت دیگری ندارد . مد یک ارتباط سریال که مهمترین مد ارتباط می باشد و ما از آن استفاده می کنیم مد ارسال و دریافت **UART** بدون پریتی (بیت تصحیح) میباشد ، که به همان صورت یک بیت شروع وخاتمه و ۸ بیت داده می باشد که در کل ۱۰ بیت را در این مد برای هر فرستادن یا گرفتن هر بسته خواهیم داشت ، مد ۲ و ۳ نیز زیاد کاربرد ندارند و به عنوان مثال مد ۲ همانند مد یک با سرعت ثابت و مد ۳

نیز همانند مد ۱ با سرعت متغیر می تواند ارسال یا دریافت کند . حال به سراغ مدار پروژه خود می رویم . در این مدار ما ۲ میکرو کنترلر داریم که به یکی با استفاده از کلید های فرضی یک ورودی بین **00H** تـا **FFH** می دهیم و در خروجی میکروی دیگر آنرا مشاهده می کنیم که این ۲ میکرو تنها با استفاده از ۲ سیم بـه هـم متصلند که پایه های **RXD** و **TXD** از پورت ۳ می باشد که **RXD** برای دریافـت و **TXD** بـرای ارسـال اطلاعات می باشد البته در اینجا می دانیم که یک میکرو تنها فرستنده و یکی تنها گیرنده اسـت کـه بـه ایـن ترتیب فقط باید یک سیم بین این دو میکرو می کشیدیم که از فرستنده ی اولی **TXD** بـه گیرنـده ی دومـی متصل شده باشد **RXD** . حال سخت افزار مدار را می بینیم.



در این مدار که با استفاده از نرم افزار **PROTEUS** رسم شده است از کلید های ٤ بیتی استفاده کرده ایم که به پورت یک از میکروی فرستنده (سمت چپ) متصل شده اند همچنین در این مـدار از **7SEG** هـای مخصوصی استفاده کرده ایم که در خود آی سی ٧٤٤٧ را نیز دارند و خود دیکدر نیز هـستند پـس کـافی است که ٤ بیت عدد را به آنها بدهیم تا آنرا نمایش بدهند که این **7SEG** ها بـه پـورت یـک از میکـروی گیرنده متصل شده اند در این مدار اگر پایه ی **P2.1** صفر شود اصلاعات انتقال مـی یابـد و روی **7SEG** ها نمایش داده می شوند .

برنامه ی میکروی اول (فرستنده) :

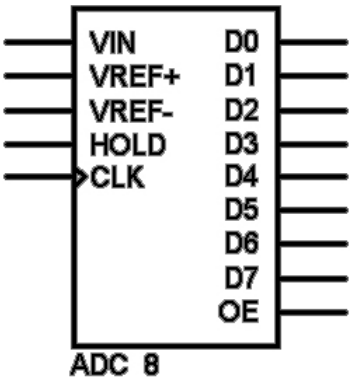
در اینجا یک میکروی فرستنده و یک گیرنده داریم
همانطور که می بینید اسمبلر به حرف کوچک و بزرگ حساس نیست

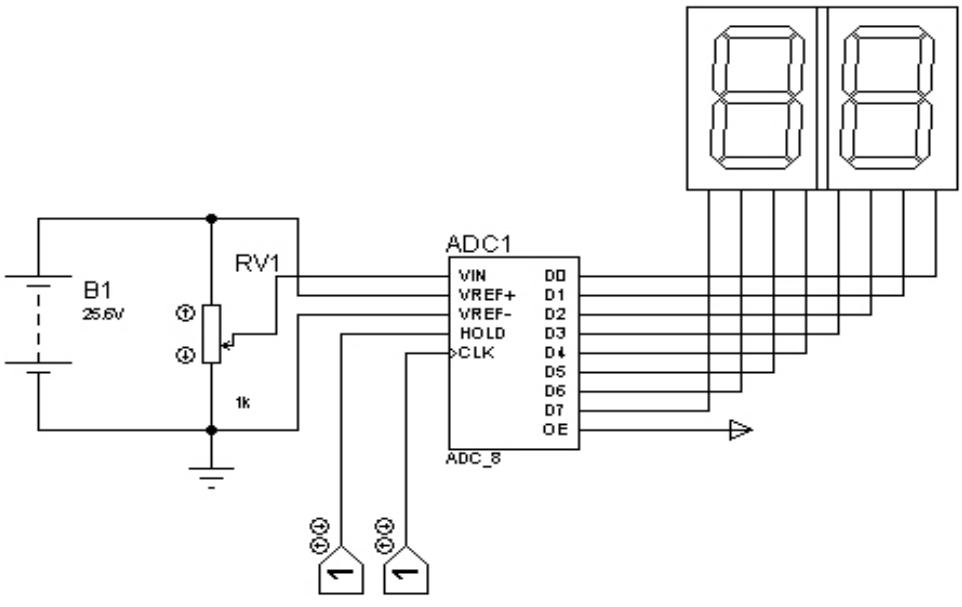
```
org 0000h
        mov tmod,#20h      ; تایمر یک را در مد ۲ می گذاریم
        mov tl1,#-250      ; سرعت ارتباط 28800/6=4800 bps می باشد زیرا 256-250=6
        mov th1,#-250
        clr sm0            ; مد یک ارتباط سریال را به کار میگیریم
        setb sm1           ; در یافت را فعال می کنیم
        setb scon.4
l1:     jb p2.1,l1         ; منتظر می مانیم تا P2.1 صفر شود یعنی کلید زده شود
        lcall send         ; به زیر برنامه ی ارسال می رویم
        sjmp l1            ; همین چرخه را تکرار میکنیم

send:   mov a,p1           ; هرچه در پورت یک بود را در A می ریزیم
        mov sbuf,a         ; A را در SBUF می ریزیم
        setb tr1           ; تایمر یک را فعال می کنیم
l2:     jnb ti,l2          ; منتظر می مانیم تا ارسال تمام شود (TI=1)
        clr tr1            ; تایمر را خاموش و TI را پاک می کنیم
        clr ti
        ret
end
```

برنامه ی میکروی دوم (گیرنده) :

```
org 0000h
        mov tmod,#20h      ; کلیه کارهایی که برای تنظیم و مقدار دهی به ثبات ها را در بالا کردیم
        mov tl1,#-250      ; در اینجا نیز انجام می دهیم
        mov th1,#-250
        clr sm0
        setb sm1
        setb scon.4
l3:     setb tr1           ; در اینجا تایمر ۱ را روشن می کنیم ومنتظر می مانیم تا RI یا پرچم
l2:     jnb ri,l2          ; دریافت بالا برود
        clr tr1            ; تایمر راخاموش کرده و هرچه در SBUF بود را در پورت یک برای
        lcall reci         ; نمایش می ریزیم
        sjmp l3

reci:   mov a,sbuf
        mov p1,a
        clr ri             ; حال RI را صفر می کنیم
        ret
end
```

در پایان این فصل جهت تکمیل مثال ها مثالی نیز از A/D می زنیم در این مثال به علت سادگی کار با A/D و D/A از میکرو کنترلر استفاده نکرده و تنها با استفاده از یک A/D مقداری آنالوگ را گرفته و آنرا به صورت دیجیتال نمایش می دهیم . شکل کلی یک A/D هشت بیتی به صورت زیر است این IC ها با نام های مختلفی به بازار می آیند معروفترین آنها ADC804 و ADC808 می باشد که ADC804 یک آی سی A/D هشت بیتی یک کاناله به مانند شکل روبرو می باشد ولی ADC808 یک آی سی با ۸ کانال A/D می باشد که با استفاده از ۳ خط آدرس یکی از آنها را انتخاب می کنیم. همانطور که در این شکل می بینید ما پایه ای به نام VIN داریم که ورودی آنالوگ ما می باشد ، پایه های +VREF و -VREF نیز ولتاژ مرجع ما را معین می کنند و اگر فرضا بخواهیم یک ولتاژ سینوسی با ماکزیمم ۵+ و مینیمم ۵- ولت را به دیجیتال تبدیل کنیم باید به +VREF مقدار ۵+ و به -VREF مقدار ۵- را بدهیم ، پایه های D0 تا D7 نیز همانگونه که می دانیم خروجی دیجیتال ما هستند و پایه OE نیز فعال ساز خروجی می باشد که در صورتی که این پایه یک باشد می توانیم خروجی را بر روی پایه های D0 تا D7 داشته باشیم ، HOLD نیز یک کلاک برای ورودی می باشد برای اینکه از اطلاعات نمونه برداری کنیم باید یک لبه ی پایین به بالا به این پایه بدهیم در این صورت از ورودی نمونه برداری میکند ، CLK نیز کلاک لازم برای نمایش است که پس از اینکه نمونه برداری کردیم با دادن یک لبه ی بالا رونده به این پایه تبدیل انجام شده و مقدار آن را در خروجی نمایش می دهد. شکل این مدار را در زیر می بینید . این تست با استفاده از نرم افزار PROTEUS انجام شده است و از قطعه ی مجازی به نام LOGIC STATE برای کلاک دادن دستی به پایه ها استفاده شده این قطعه صفر یا یک منطقی را برای ما ایجاد می کند و هر زمان که خواستیم می توانیم آنرا یک یا صفر کنیم . برای کار با مدار ابتدا HOLD را کلاک زده و سپس CLK را کلاک می زنیم و بر روی 7SEG ها مقدار ورودی را که قسمی از ۲۵,٦ ولت است را می بینیم در اینجا ما ۲۵,٦ را به ۲۵٦ قسمت تقسیم می کنیم یعنی به ازای هر ۰,۱ ولت یکی به خروجی هگز اضافه می شود.





فصل پنجم :

آموزش مقدماتی نرم افزار های PROTEUS و FRANKLIN:

در این فصل به آموزش سطحی نرم افزار های PROTEUS و FRANKLIN می پردازیم . نرم افزار PROTEUS یک برنامه برای شبیه سازی مدارات آنالوگ و دیجیتال می باشد و شاید به جرات بتوان گفت این برنامه قویترین برنامه در زمینه ی الکترونیک دیجیتال می باشد زیرا به شبیه سازی بسیار دقیق پرداخته و این برنامه علاوه بر آی سی های دیجیتال قادر به شبیه سازی میکرو کنترلر های 8051 و AVR و PIC و ... نیز هست که در کمتر برنامه ای می توان چنین قابلیتی را دید. به هر حال تجربه نشان داده است برای بستن مداری با میکرو کنترلر ابتدا باید با این برنامه آنرا امتحان نمود و در صورتی که جواب کار صحیح بود به سراغ مدار عملی برویم ، نرم افزار FRANKLIN نیز یک کامپایلر و اسمبلر بسیار خوب برای سری ۸۰۵۱ می باشد با استفاده از HELP بسیار قوی قابلیت کامپایل و اسمبل زبان های اسمبلی و C را برای سری ۸۰۵۱ دارد . این نرم افزار با دقت نسبتا خوبی در هنگام برنامه نویسی به گرفتن ایرادات برنامه و شبیه سازی خط به خط و کلی برنامه کمک می کند . این دو برنامه را با جستجو در اینترنت و دانلود کردن آن می توانید استفاده نمایید. برنامه ی PROTEUS را از سایت http://www.labcenter.co.uk می توانید دریافت کنید و برنامه ی FRANKLIN را نیز از سایت http://www.fsinc.com دانلود نمایید که البته برنامه ی پروتوس نیاز به رجیستر و خرید دارد که یافتن کرک آن نیز کار ساده ای نیست اما نسخه ی دانش آموزی فرانکلین در دسترس می باشد. ما در اینجا فقط به توضیح بسیار اجمالی در مورد این دو نرم افزار می پردازیم و کار بیشر و یاد گیری کامل آن بر عهده ی خواننده می باشد. پس از نصب نرم افزار FRANKLIN که معمولا با سریال EVAL رجیستر می شود به منوی START رفته و در ALL PROGRAMS به دنبال FRANKLIN SOFTWARE می گردیم سپس برنامه ی PROVIEW32 را باز نموده و به منوی OPTION در بالای نرم افزار می رویم سپس از گزینه ی PROJECT وارد درخت L51 می شویم و روی گزینه ی LINKER کلیک می کنیم در سمت راست گزینه ی INTEL HEX را تیک می زنیم و از این قسمت خارج می شویم .حال به گزینه ی FILE رفته و NEW را می زنیم از چهار گزینه ی آمده ASSEMBELER FILES را می زنیم تا پنجره مربوط به نوشتن برنامه ی اسمبلی باز شود حال برنامه ی خود را با ORG شروع می کنیم و می نویسیم پس از نوشتن برنامه به منوی FILE رفته و آنرا SAVE می کنیم بهتر است برای هر پروژه پوشه ای جدا تعریف کنیم حال به منوی DEBUG رفته و START را می زنیم در این قسمت VIRYUAL MACHINE با گزینه ی 80C52 را انتخاب می کنیم و OK را می زنیم در این قسمت اگر مشکلی در برنامه نداشته باشیم به مرحله ی شبیه سازی می رویم در صورتی که در اینجا GO را در بالای صفحه بزنیم برنامه به طور کامل شبیه سازی می شود و می توانیم با استفاده از جدولی که برای ثبات ها و پورت ها داده شده مقدار دهی و شبیه سازی کنیم و در صورتی که نیاز به چک کردن خط به خط برنامه دارید با استفاده از کلید F7 به این کار بپردازید پس از این که این برنامه به طور

دلخواه برای مدار شما جواب داد حال آنرا از حالت DEBUG با زدن گزینه ی TERMINATE خارج کنید و وارد نرم افزار PROTEUS برای شبیه سازی نهایی گردید در این قسمت به توضیح مختصری در باره ی برنامه ی PROTEUS می پردازیم و امید واریم کاربر با استفاده از تلاش خود برای یافتن کلیه ی عملکرد های این نرم افزار راه پیشرفت را ادامه دهد . پس از دانلود و نصب این نرم افزار باید ابتدا آنرا کرک کرد در غیر اینصورت تنها می توانید مثال های خود برنامه را مشاهده کنید یا در آنها بنویسید ولی ذخیره نمی توانید بکنید که پیدا کردن کرک آن نیز کار ساده ای نیست . ما با این فرض به آموزش ادامه می دهیم که شما برنامه را به طور کامل نصب و کرک نموده اید . ابتدا به منوی START رفته و در ALL PROGRAMS از گزینه ی PROTEUS X PROFESSIOANL گزینه ی ISIS را انتخاب می کنیم پس از باز کردن برنامه پنجره ای به جلوی شما می آید که از شما می پرسد آیا می خواهید مثال های داخل این برنامه را مشاهده نمایید ؟ در این مرحله اگر می خواهید با طرز کار و نمونه های این برنامه آشنا شوید YES را انتخاب و در غیر اینصورت NO را بزنید تا به برنامه وارد شوید در صورتی که به قسمت مثالها وارد شدید Microprocessor Simulation Samples را انتخاب کنید تا مثال هایی از میکرو های مختلف را ببینید البته در قسمت مثال های ساده نیز برای افراد تازه کار مثال های بسیار مفیدی وجود دارد پس از آن که یکی از مثال ها را انتخاب نمودید یعنی فایل با پسوند DSN آنرا باز کردید دکمه ی PLAY یا ▶ را برای اجرای برنامه فشار دهید و در صورتی که نیاز به فشردن کلید یا سوییچی در برنامه بود آنرا انجام دهید تا نتایج مختلف کار را ببینید حال از کثال ها خارج شده و برنامه را دوباره باز کرده و به سوال برنامه رد مورد دیدن مثال ها پاسخ منفی می دهیم تا به ساختن مثال جدید از خودمان بپردازیم در قسمت سمت راست یا در بعضی از نسخه ها راست قسمتی خالی وجود دارد که در بالای آن قسمت دو دکمه ی کوچک P و L وجود دارند در روی قسمت خالی دو بار کلیک می کنیم در این حالت پنجره ای با عنوان Pick Devices باز می شود در این قسمت ما باید کل قطعاتی که برای ساخت مدار خود نیاز داریم را انتخاب کنیم همانگونه که می بینی» در قسمت بالایی این پنجره نام گروه قطعات وجود دارد مثلا انواع میکروکنترلر ها در قسمت MICRO یا انواع مقاومت در قسمت RESISTORS یا انواع قطعات نمایشکر در DISPLAY ، انواع ترانزیستور BIPOLAR ، انواع FET و انواع آی سی های TTL با عنوان 74XX یا انواع CMOS یا انواع آی سی های RAM و ROM در قسمت MEMORY ووسایلی که در طول مدار ممکن است تغییر کنند مانند منبع متغیر یا کلید و سوییچ و لامپ و فیوز و رله و مقاومت متغیر در قسمت ACTIVE ، وسایلی که بسیار پرکاربردند معمولا در DEVICE پیدا می شوند و وسایل آنالوگ نیز در بخش ANALOG . توصیه می شود قبل از کار با این برنامه به خوبی این پنجره و المان هایی که در هر بخش وجود دارند را مشاهده نمایید در صورتی که نیاز داریم که از قطعه ای در مدار خود استفاده کنیم باید روی آن دوبار کلیک کنیم تا به لیست سمت راست برنامه وارد شود در این قسمت دقت شود که در سمت راست ما برای هر قطعه دو پنجره وجود دارد که یکی مربوط به قسمت شبیه سازی و یکی

مربوط به قسمت طراحی فیبر مدار چاپی می باشد در صورتی که در سمت راست در پنجره ی بالایی با جمله ی **NO SIMULATOR MODEL** مواجه شدید از این قطعه جهت شبیه سازی استفاده نکنید زیرا به علت ناقص بودن نسخه ی برنامه این قطعه کار نمی کند و در صورتی که در پنجره ی پایین سمت راست **NO PCB PACKAGE** را مشاهده کردید تا حد امکان این قطعه را برای مدار چاپی به کار نبرید زیرا باز هم نسخه نقص دارد (در اینجا من از کسانی که توانایی انجام این کار را دارند خواهش می کنم با وارد کردن و تکثیر سی دی ارجینال و کامل این نرم افزار پر ارزش در ایران ، گامی بزرگ در جهت خدمت به جامعه ی مهندسی برق ایران بردارند زیرا این نرم افزار بی شک قویترین نرم افزار های موجود در زمینه ی الکترونیک می باشد) حال که تمامی قطعات مورد نیاز را انتخاب کردیم این پنجره را بسته و در سمت راست بر روی قطعه ی مورد نظر کلیک می کنیم سپس به روی صفحه ی طراحی آمده و در مکانی که می خواهیم قطعه را بگذاریم کلیک می کنیم می بینیم که قطعه در آن محل گذاشته می شود ، به همین ترتیب تمامی قطعات خود را در روی صفحه می آوریم و پس از این کار با نزدیک کردن **MOUSE** به سر پایه های هر قطعه علامت ضربدر را در کنار موس مشاهده می کنیم ، حال کلیک کرده و می بینیم که سیمی به دنبال موس کشیده می شود حال به پایه مقصد رفته و روی آن کلیک می کنیم می بینیم که پایه ها از طریق یک سیم به هم متصل می شوند برای این که مسیر سیم را خودمان انتخاب کنیم باید در هر جایی که می خواهیم رفته و کلیک کنیم تا سیم در هماجا بماند و ادامه ی آن از آن نقطه به دنبال موس می آید حال که همه ی سیمها را متصل کردیم دکمه ی **PLAY** را برای شبیه سازی فشار دهید . در این قسمت نکات اصلی برای کار با برنامه را توضیح می دهیم :

۱ – برای انتخاب یک قطعه و کار با آن یک بار باید روی آن کلیک راست کرد تا رنگ آن به رنگ قرمز در آید در این صورت با یک کلیک بر روی قطعه وارد پنجره ی مشخصات قطعه می شویم که می توانیم پارامتر های هر قطعه مانند مقاومت و ظرفیت خازن و ... را تعیین کنیم ، با گرفتن و حرکت دادن قطعه می توانیم آنرا حرکت دهیم و به محل دلخواه خود ببریم ، با دو بار کلیک راست بر روی هر قطعه آن قطعه پاک می شود .

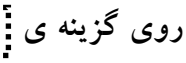
۲– در صورتی که از مدارات دیجیتال استفاده می کنید برای دادن **VCC** و **GND** باید روی گزینه ی  در بالای صفحه کار کلیک کنید و **GND** و **POWER** را برای ولتاژ دهی انتخاب کنید . در صورتی که از مدارات آنالوگ استفاده می کنید بهتر است از باطری برای ولتاژ دهی به مدار استفاده نمایید.

۳– برای انتخاب قطعه باید بر روی قسمت  باشیم تا بتوانیم انتخاب قطعه نماییم و سیم کشی کنیم .

٤– برای وارد کردن قطعات اندازه گیره مانند مولتی متر **DC** و **AC** و اسیلوسکوپ و سیگنال ژنراتور باید بر روی گزینه ای که شبیه یک مولتی متر است کلیک کنیم و قعه مورد نیاز را از سمت راست انتخاب نماییم.

۵ – برای دیدن جریان و ولتاژ هر نقطه از مدار باید از پروبهای ولتاژ و جریان که در قسمت بالای صفحه طراحی با علامت استفاده نماییم .

٦- برای استفاده از باس یعنی قطعه سیم هایی که تعداد زیادی سیم را مـی تـوان در آن قـرار داد تـا فـضای کمتری را اشغال کرده و از شلغی مدار ما بکاهند باید روی گزینه ی  کلیک نماییم و باس های مـورد نیاز را بکشیم.

٧- پس از اینکه این عمل را انجام دادیم (کشیدن باس ها) باید برای هر سیم یک نام مجزا بگذاریم و بـرای محل هایی که به هم متصلند باید نام سیم را یکی بگذاریم تا آنها را به صورت مجازی به هم متـصل کنـد در اینجا به LABLE یا برچسب احتیاج داریم که برای برچسب گذاری باید روی گزینه ی  رفتـه و در هرجا که نیاز به نام گذاری داشتیم کلیک می کنیم . پنجره ای باز می شود که باید نام مورد نضر را بـرای هـر سیم در آن وارد کنیم .

٨- برای ریختن فایل برنامه روی میکرو باید روی آن کلیک راست کرد و سـپس کلیـک کنـیم تـا پنجـره ی مربوط به مشخصات آن باز شود در این این نرم افزار همه ی میکرو ها یک اسیلاتور داخلـی دارنـد کـه مـی توان میزان فرکانس اسیلاتور را در گزینه ی Processor Clock Frequency تعیین کرد همچنـین بـرای ریختن فایل HEX (که قبلا توسط FRANKLIN در پوشه ای که برنامه را ذخیره کرده بودیم ایجاد مـی شود) باید روی پوشه در سمت راست Program File کلیـک کنـیم و آدرس آن فایـل HEX را بـه آن بدهیم تا میکروی ما به صورت مجازی پروگرم شود البته سعی کنید که نام این فایـل HEX بـسیار کوتـاه و سرهم باشد .

امیدواریم با مار بیشتر با این برنامه بتوانید طرز کار کامل برنامه را یاد بگیرید این برنامه قابلیت هـای بـسیاری از جمله دیدن تمامی حافظه ها در هنگام کار ، دیدن کد در حال اجرای برنامه ، تبدیل مدار به NETLIST و بردن آن به برنامه ی ARES برای کشیدن مدار چاپی را دارد که با مار با آن به راحتی می تـوان بـه آنهـا دست یافت.

امید است با خواندن این جزوه بنوانید گامی را در جهت موفقیت بردارید.

امید طالبی – تابستان ٨٤