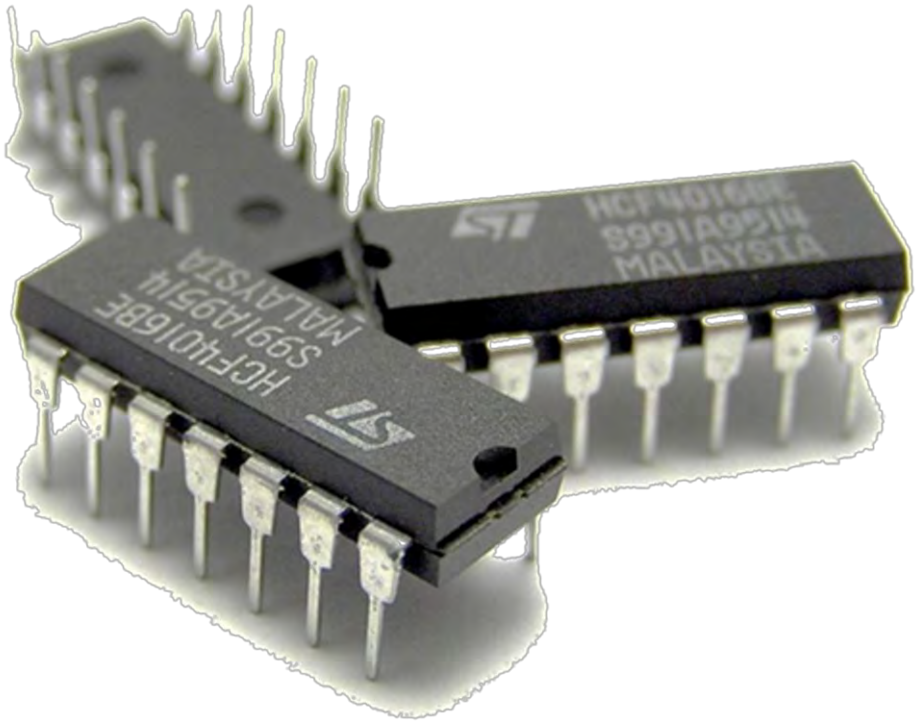
اصول میکروکامپیوترها

دکتر سید حسن نبوی کریزی



☑ آشنایی با اصول کار میکروکامپیوترها (با تأکید بر پروسسور ۸۰۸۸)

☑ مختصری راجع به میکروکنترلرهای AVR

در **بخش اول** این کتاب:

☑ با اصول کار میکروکامپیوترها از پایه و مبنا آشنا می‌شوید.

☑ طرزکار مبدل‌های آنالوگ‌به‌دیجیتال و بلعکس را آموخته و انواع آن‌ها را فرامی‌گیرید.

☑ انواع حافظه‌ها و اصول کارکرد آن‌ها را می‌آموزید.

☑ با تاریخچه ریزپردازنده‌ها و ساختار و نحوه کار پردازنده Intel® 8088 آشنا می‌شوید.

☑ و نحوه برنامه‌نویسی به زبان اسمبلی را خواهید آموخت.

همچنین در **بخش دوم** این کتاب:

☑ به‌طور خلاصه با میکروکنترلرها و انواع آن‌ها آشنا می شوید و معایب و مزایای هرکدام از آن‌ها را خواهید آموخت.

☑ با مفاهیم اصطلاحات مربوط به میکروکنترلرها آشنا می‌شوید.

☑ در مورد برخی از مدل های موجود آن‌ها و ویژگی هایشان مطالبی خواهید خواند.

☑ و در پایان با نحوه برنامه نویسی به زبان C در محیط Code Vision و زبان Basic در محیط Bascom آشنا می‌شوید.

به نام خدا

اصول میکروکامپیوترها

گردآوری، تالیف و تدوین:

دکتر سید حسن نبوی کریزی

بخش اول: آشنایی با اصول کار میکروکامپیوترها با تأکید بر پروسسور ۸۰۸۸

بخش دوم: مختصری راجع به میکروکنترلرهای AVR

اصول میکروکامپیوترها

بخش اول: آشنایی با اصول کار میکروکامپیوترها با تأکید بر پروسسور ۸۰۸۸

فصل اول : مقدمه

فصل دوم : مبدلهای A/D و D/A

فصل سوم : حافظه‌ها

فصل چهارم : ریزپردازنده ها

فصل پنجم : برنامه‌نویسی اسمبلی

برخی مراجع:

1. IBM PC Assembly Language and Programming By: Abel, Peter

2. Microprocessors and Digital Systems. By: Hall (Chapters 5, 6, 7)

3. Microprocessors and Programmed Logic. By: Short (Chapters 3, 4, 5)

۱-۱- تعریف کامپیوتر

کامپیوتر ماشینی است قابل برنامه‌ریزی که از ترکیب اجزای الکترونیکی و الکترومکانیکی تشکیل شده است و می‌تواند پس از دریافت ورودیها، بر اساس دنباله ای از دستورالعملهای مشخص، پردازشهای خاصی را انجام داده و سپس نتیجه را ذخیره نموده و یا به خروجی بفرستد. بسیاری از امور روزمره ما نیز بر اساس روال «ورودی-پردازش-خروجی» صورت می‌گیرد؛ مثلاً وقتی از شخصی سؤالی را می‌پرسید، سؤال شما در حکم ورودی، فکرکردن او در حکم پردازش و بیان پاسخ در حکم خروجی است.

داده (Data): به مجموعه‌ی اطلاعات خام که پیش از پردازش و به عنوان ورودی در اختیار داریم «داده» گفته می‌شود. مثلاً نمره‌های موجود در کارنامه‌یک دانشجو، داده هستند.

اطلاعات(Information): پس از هر پردازش خاص، داده‌ها به اطلاعات تبدیل می‌شوند. مثلاً معدل و رتبه دانشجو اطلاعاتی هستند که با پردازش روی داده‌های کارنامه بدست آمده‌اند.

پردازش(Process): به مجموعه‌ی عملیاتی که بر روی داده‌ها صورت می‌گیرد پردازش گفته می‌شود. مثل محاسبات انجام شده بر روی نمره‌های کارنامه‌یک دانشجو.

۱-۲- طبقه‌بندی کامپیوترها

بر اساس توانایی و قدرت پردازش، کامپیوترها به چهارگروه اصلی تقسیم می‌شوند که عبارتند از:

ابرکامپیوترها(Super Computers)، *کامپیوترهای بزرگ* (Mainframe Computers)، *کامپیوترهای کوچک* (Mini Computers) و *ریزکامپیوترها*(Micro Computers).

ابرکامپیوترها، کامپیوترهایی هستند که قدرت پردازش، سرعت و توانایی فوق العاده‌ای دارند و اندازه‌ی آنها بسیار بزرگ است مثلاً در حد یک ساختمان. در پروژه‌هایی مانند پیش بینی اوضاع جوی و امور نظامی و فضایی-که نیاز به محاسبات پیچیده وپیشرفته دارند- استفاده می‌شوند.

کامپیوترهای بزرگ برای محاسبات بسیار پیچیده و سنگین طراحی شده‌اند و در مؤسساتی بکارگرفته می‌شوند که حجم اطلاعاتی که در آنها پردازش می‌شود بسیار زیاد است مثلاً مؤسسه‌هایی که باید اطلاعات مربوط به آب و برق و تلفن شهروندان را پردازش کنند. حجم این کامپیوترها زیاد است و قسمتهای تشکیل دهنده آنها مجزا از هم هستند. کاربران این نوع کامپیوترها معمولاً از طریق شبکه به آن دسترسی دارند و بصورت مشترک از امکانات آن بهره می‌برند.

کامپیوترهای کوچک کامپیوترهایی در حد متوسط هستند که حجم داده‌ها و تنوع کارهای آنها نسبتاً زیاد است و می‌توان از آنها برای پردازش کارهای کاربران شبکه استفاده کرد.

ریزکامپیوترها بخاطر حجم کمتر و قیمت پایین‌تر از سایر رده‌ها، کاربرد بسیار زیادی در همه‌ی زمینه‌ها دارند و بر اساس یک ریزپردازنده ساخته می‌شوند. کامپیوترهای شخصی (PC) از این نوع هستند. که در سه شکل رومیزی (Desktop)، کیفی (Laptop) و دستیار دیجیتالی (Personal Digital Assistant) عرضه می‌شوند. کامپیوترهای رومیزی معمولاً شامل صفحه نمایش، صفحه کلید و یک واحد سیستم هستند که بندرت جابجا می‌شوند. اما کامپیوترهای کیفی که تکنولوژی ساخت آنها ظریف است اغلب برای مواردی که نیاز به جابجایی است بکار برده می‌شوند. کامپیوترهای دستیار بعنوان کامپیوترهای جیبی شناخته می‌شوند و دارای امکاناتی نظیر دفترچه‌یادداشت، ماشین حساب، تقویم و همچنین ارتباط با شبکه‌ها هستند.

۱-۳- سخت افزار کامپیوتر

به تجهیزات فیزیکی یک کامپیوتر، اعم از قسمتهای الکترونیکی و الکترومکانیکی که قابل لمس باشند، سخت افزار(Hardware) می‌گویند. بطور کلی، کامپیوتر شامل چهار واحد اصلی است: *واحد پردازش مرکزی* (CPU)، *واحد حافظه*(Memory Unit)، *واحد ورودی*(Input Unit) و *واحد خروجی*(Output Unit) .

واحد پردازش مرکزی، به عنوان مغزکامپیوتر، شامل واحدهای محاسبه و منطق، واحد کنترل و رجیسترها است و در تراشه‌ای قرار دارد که ***ریزپردازنده*** نامیده می‌شود. **واحد محاسبه و منطق** وظیفه‌ی تجزیه و تحلیل و اجرای دستورات را در CPU برعهده دارد. عملیات محاسباتی (اعمال ریاضی) شامل جمع، تفریق، ضرب و تقسیم و عملیات منطقی شامل اعمال مقایسه‌ای هستند . **واحد کنترل** از مدارهای الکترونیکی پیچیده‌ای تشکیل شده است و مشخص می‌کند که هر قسمت چه وظیفه‌ای دارد و ترتیب اجرای دستورالعملها را هم مشخص می‌کند. به عبارت دیگر با نظارتی که بر عملکرد سایر واحدهای کامپیوتر دارد، عمل هماهنگی و هدایت واحدهای اصلی کامپیوتر را انجام می‌دهد. واکشی (Fetch) و رمزگشایی(Decoding) دو عمل اصلی واحد کنترل هستند. منظور از واکشی، خارج کردن دستورالعملها از حافظه به واحد پردازش و منظور از رمزگشایی، تفسیر دستورالعملها برای اجراست.

ثباتها یا رجیسترهای داخلی حافظه‌هایی موقتی هستند که داده‌ها و آدرسهای در حال پردازش CPU بطور موقت در آن قرار می‌گیرند. سرعت دسترسی CPU به این نوع حافظه‌ها در مقایسه با حافظه‌های اصلی بسیار بیشتر است.

واحدهای ورودی و خروجی عامل ارتباط بین انسان و کامپیوتر هستند. چنانکه میدانیم زبان قابل فهم برای انسان از حروف، اعداد و علائم تشکیل شده است؛ در حالیکه زبان کامپیوتر فقط از صفر و یک تشکیل شده است. وظیفه دستگاه ورودی، تبدیل داده‌های قابل فهم انسان به داده‌های قابل پردازش برای کامپیوتر است. اغلب دستگاههای خروجی هم داده‌هایی را که کامپیوتر آنها را پردازش کرده به داده‌های مناسب و قابل فهم برای انسان تبدیل می‌کنند. صفحه کلید، ماوس، اسکنر، دیجیتایزر، قلم نوری، دوربین دیجیتال و میکروفن و Webcam از جمله دستگاههای ورودی و مانیتور، چاپگر، پلاتر، بلندگو و هدفن از جمله دستگاههای خروجی کامپیوتر هستند. برخی دستگاهها هم ورودی و هم خروجی هستند مثل دیسک گردان (Disk Drive)، کارت صدا و کارت مودم. (دیسک گردان، داده‌ها را از روی دیسک می‌خواند و به کامپیوتر منتقل می‌کند و داده‌های کامپیوتر را جهت ذخیره‌سازی روی دیسک ذخیره می‌کند. مودم برای تبدیل داده‌های دیجیتالی کامپیوتر به داده‌های آنالوگ مخابراتی و بالعکس بکار می‌رود. کارت صدا ابزاری برای ورود و خروج داده‌های صوتی است که از طریق ورودی ای که میکروفن به آن وصل است صدا را می‌گیرد و از طریق خروجی ای که به بلندگو وصل است صدا را پخش می‌کند).

واحد حافظه، داده‌هاو دستورالعملهای مورد نیاز پردازنده را نگهداری می‌کند. حافظه با هدف ذخیره سازی اطلاعات (بصورت دائم یا موقت) استفاده می‌شود و انواع متفاوتی دارد. که از جمله می‌توان حافظه فقط خواندنی Read Only) (Memory = ROM، حافظه خواندنی- نوشتنی (Random Access Memory = RAM) و حافظه‌های جانبی نظیر فلاپی و دیسک سخت را نام برد. معمولاً اطلاعاتی که نیاز به تغییر ندارند، مثل برنامه آماده سازی سیستم، تست سیستم و کد برنامه، در حافظه ROM ذخیره می‌شوند. استفاده از حافظه RAM، برای نگهداری موقت اطلاعات تا زمان پردازش یا انتقال نتایج به بیرون از کامپیوتر و یا ذخیره در حافظه جانبی است. مدت زمانی را که نیاز است تا اطلاعات در حافظه نوشته شود

و یا از آن خوانده شود زمان دستیابی حافظه[1] گویند.

۱-۴- انواع داده

بیت[2] : کوچکترین واحد حافظه که فقط گنجایش نگهداری صفر یا یک را دارد، بیت گویند.

بایت: هر بایت شامل ۸ بیت است (که با احتساب بیت توازن برخی کتابها آنرا ۹ بیت در نظر می‌گیرند).

کلمه[3]: داده‌ی دو بایتی (۱۶ بیتی)

کلمه مضاعف[4]: داده‌ی ۴ بایتی (۳۲ بیتی)

چهارکلمه: داده‌ی ۸ بایتی (۶۴ بیتی)

پاراگراف: داده‌ی ۱۶ بایتی (۱۲۸ بیتی)

محدوده	نوع داده	اندازه داده (بیت)
۰ تا ۲۵۵	بی علامت	۸
۱۲۸- تا ۱۲۸+	علامتدار	۸
۰ تا ۶۵۵۳۵ = $۲^{۱۶}$	بی علامت	۱۶
۳۲۷۶۸- تا ۳۲۷۶۸+	علامتدار	۱۶
۰ تا ۴۲۹۴۹۶۷۲۷۵ = $۲^{۳۲}$	بی علامت	۳۲
۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۸+	علامتدار	۳۲

- داده‌های کاراکتری بصورت کد اسکی(۷ بیتی) یا کد اسکی توسعه یافته (۸ بیتی) و در قالب یک بایت ذخیره می‌شوند.
- معمولاً هر موقعیت حافظه قابلیت نگهداری ۸ بیت (یک بایت) را دارد. بنابراین اگر بعنوان مثال، یک عملیات به داده ۱۶ بیتی نیاز داشته باشد باید این داده را از دو موقعیت پشت سرهم حافظه بردارد.

گرچه بایت واحد اندازه گیری ظرفیت حافظه است ولی چون در عمل بایت واحد کوچکی است از واحدهای بزرگتری چون کیلوبایت، مگابایت و.. استفاده می‌شود. چدول زیر این واحدها را بر حسب بایت نشان می‌دهد.

واحد	معادل بایت	توان ۲
کیلوبایت	۱۰۲۴ بایت	$۲^{۱۰}$ بایت
مگابایت	۱۰۲۴ کیلوبایت = ۱۰۲۴ × ۱۰۲۴ بایت	$۲^{۲۰}$ بایت
گیگابایت	۱۰۲۴ مگابایت = ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ بایت	$۲^{۳۰}$ بایت
ترابایت	۱۰۲۴ گیگابایت = ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ بایت	$۲^{۴۰}$ بایت
پتابایت	۱۰۲۴ ترابایت = ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ بایت	$۲^{۵۰}$ بایت
اگزابایت	۱۰۲۴ پتابایت = ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ × ۱۰۲۴ بایت	$۲^{۶۰}$ بایت

1 - Access Time
2 - Binary Digit = Bit
3 - Word
4 - Double word

۱-۵- سیستم عامل

سیستم عامل مجموعه‌ای از برنامه‌های کامپیوتری به زبان ماشین است که وظیفه‌ی نظارت و هدایت عملیاتهای کامپیوتر را دارد. در حقیقت سیستم عامل در نقش یک رابط میان ماشین و انسان است. یعنی رابطی بین کاربر و سخت افزار. به این ترتیب سیستم عامل نرم افزاری است که عملیات کامپیوتر، انتقال اطلاعات بین دستگاه ورودی، خروجی و.. را کنترل و نظارت می‌نماید. علاوه بر این وظیفه‌ی ترجمه‌ی برنامه‌های کاربر، فراهم ساختن تسهیلات برای ذخیره و بازیابی اطلاعات از دیسک و فراهم آوردن وسائل ارتباط انسان با سخت افزار را دارد.

۱-۶- سیستمهای نمایش اعداد و کدگذاری داده‌ها

آشناترین سیستم عدد نویسی برای انسان، سیستم عدد نویسی دهدهی است که نشانه‌های آن ارقام ۰ تا ۹ هستند. چون در ساختمان کامپیوترها وجود یا عدم وجود جریان الکتریکی مبنا محسوب می‌شود لذا مبنای دو(باینری) و برخی توانهای آن مانند مبنای هشت (اکتال) و مبنای شانزده (هگزادسیمال) در سیستمهای کامپیوتری از کارایی بالایی برخوردار هستند و یادگیری آنها باعث می‌شود که فهم بیشتری نسبت به عملکرد کامپیوترها پیدا کرده و با الفبای زبان کامپیوتر که همان منطق صفر و یک است آشنایی بیشتری پیدا کنیم.

در محاسبات و امور روزمره از سیستم عدد نویسی دسیمال (دهدهی) استفاده می‌شود. در این سیستم با ده رقم ۰ تا ۹ و ارزشگذاری متفاوت ارقام در محلهای مختلف (یکان، دهگان، صدگان و..) می‌توانیم همه اعداد را بخوانیم و بنویسیم.
در این سیستم عدد نویسی، ارزش هر رقم بستگی به محلی دارد که رقم در آن محل قرار گرفته است. هر مکان ارزشی معادل ۱۰ برابر ارزش مکانی رقم سمت راست خودش را دارد.
مثلاً در رقم ۵۴۲۳، ازش رقمها بصورت زیر است.

$$(5\ 4\ 2\ 3)_{10} = 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$$

یکان $\longrightarrow 3 \times 10^0 = 3$

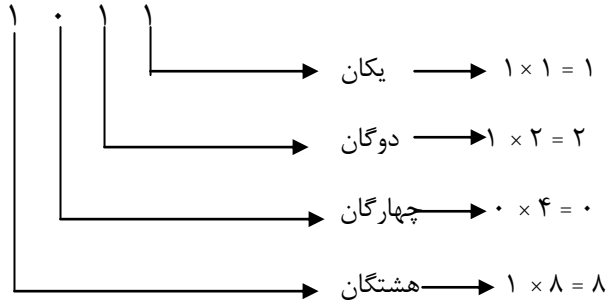
دهگان $\longrightarrow 2 \times 10^1 = 20$

صدگان $\longrightarrow 4 \times 10^2 = 400$

هزارگان $\longrightarrow 5 \times 10^3 = 5000$

۱-۶-۱- سیستم باینری (دودویی)

اگر بخواهیم برای نمایش داده‌ها در کامپیوتر، از سیستم دهدهی استفاده کنیم، در پیاده سازی سخت افزار کامپیوتر، برای ده رقم مختلف، به ۱۰ سیگنال الکتریکی با سطوح متفاوت نیاز داریم. این وضعیت پیچیدگیهای فراوانی را در عمل پدید می‌آورد که هم هزینه طراحی سخت افزار را افزایش می‌دهد و هم احتمال خطا را زیاد می‌کند. به همین دلیل باید بدنبال روشی برای عدد نویسی بگردیم که پیاده سازی آن به کمترین تعداد سیگنالها نیاز داشته باشد. چون ساده ترین وضع سیگنالها وجود یا عدم وجود آنها است باید روشی را پیدا کنیم که فقط دو نماد برای نوشتن اعداد در آن بکار رود. یعنی از سیستم عددنویسی دودویی استفاده کنیم که در آن فقط ارقام ۰ و ۱ بکار می‌روند. در این سیستم عددنویسی، رقم ۰ نشانه عدم حضور سیگنال الکتریکی و رقم ۱ نشانه وجود سیگنال الکتریکی است و ارزش مکانی هر رقم دو برابر ارزش مکانی رقم سمت راست آن است. در اینصورت، به عنوان مثال عددی که به شکل ۱۰۱۱ نوشته شده است، در روش معمولی عددنویسی بصورت زیر محاسبه می‌شود.



$$\Rightarrow (1011)_2 = 1+2+0+8 = 11$$

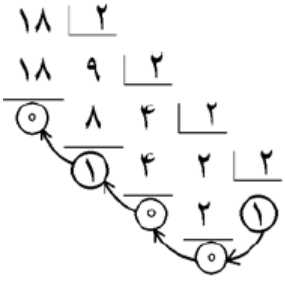
بنابراین عدد ۱۰۱۱ در سیستم دودویی معادل عدد ۱۱ در سیستم دهدهی (دسیمال) است.

برای تعمیم روش نمایش اعداد، عددی را که بُعد ارزش مکانی را مشخص می‌کند، مبنا (پایه) می‌نامیم و هنگام نوشتن عدد در این پایه، مبنا را بصورت زیرنویس در کنار آن قرار می‌دهیم. در اینصورت به عنوان مثال $۱۰۱۱_۲$ خوانده می‌شود: یک، صفر، یک، یک در مبنای دو. به این روش عددنویسی در مبنای دو، سیستم دودویی یا سیستم باینری گفته می‌شود.

در حالت کلی، داده‌هایی که وارد کامپیوتر می‌شوند، ابتدا به کد دودویی تبدیل می‌شوند و عملیات محاسبه و پردازش در مبنای دو صورت می‌گیرد و سرانجام در هنگام نمایش در خروجی، نتایج به کد دهدهی تبدیل می‌شوند.

برای تبدیل یک عدد مبنای ده به مبنای دو از روش تقسیم متوالی بر عدد ۲ استفاده می‌شود. عمل تقسیم را تا زمانی ادامه می‌دهیم که خارج قسمت از مبنا (در اینجا ۲) بزرگتر باشد. سپس آخرین خارج قسمت را می‌نویسیم و باقیمانده‌ها را از انتها به ابتدا می‌نویسیم.

مثال : عدد $۱۸_{۱۰}$ را به مبنای دو تبدیل کنید.



$$(۱۸)_{۱۰} = (۱۰۰۱۰)_۲$$

عملیات پردازش در سیستم دودویی، مثل قوانین کلی محاسبه معمولی است با این تفاوت که رقم نقلی و قرضی در محاسبات، بجای عدد ده، عدد دو است. بنابراین جمع دو عدد باینری $۱۰۱۱_۲$ و $۱۰۰۱_۲$ در مبنای دو بصورت مقابل است.

```
  ۱   ۱ ۱
    ۱ ۰ ۱ ۱
+   ۱ ۰ ۰ ۱
-------------
  ۱ ۰ ۱ ۰ ۰
```

به همین صورت تفریق عدد $۱۰۱_۲$ از $۱۰۰۱۱_۲$ بصورت مقابل است.
چنانکه نتیجه تفریق نشان می‌دهد، اگر در طبقه‌ای رقم بالایی از رقم پایینی کمتر باشد، یک واحد از طبقه‌ی سمت چپ - که معادل دو واحد در طبقه‌ی فعلی است - به این طبقه منتقل می‌شود.

```
    ۱
  ۰ ۲ ۲
  ۱ ۰ ۰ ۱ ۱
- ۰ ۰ ۱ ۰ ۱
-------------
  ۰ ۱ ۱ ۱ ۰
```

برای تبدیل عدد از مبنای دو به مبنای ۱۰ از فرمول زیر استفاده می‌کنیم که اندیسها شماره‌ی مکان هر رقم است.

$$(a_n a_{n-۱} \dots a_۱ a_۰)_۲ = a_n \times ۲^n + a_{n-۱} \times ۲^{n-۱} + \dots + a_۱ \times ۲^۱ + a_۰ \times ۲^۰$$

به عنوان مثال برای عدد $۱۰۱۰۰۱۱_۲$ داریم:

$$(۱۰۱۰۰۱۱)_۲ = ۱\times۲^۶ + ۰\times۲^۵ + ۱\times۲^۴ + ۰\times۲^۳ + ۰\times۲^۲ + ۱\times۲^۱ + ۱\times۲^۰$$
$$= ۶۴ + ۰ + ۱۶ + ۰ + ۰ + ۲ + ۱ = ۸۳$$
$$(۱۰۱۰۰۱۱)_۲ = (۸۳)_{۱۰}$$

۱-۶-۱-۱- اعداد دودویی منفی: اعداد دودویی که آخرین بیت سمت چپ آنها مقدار صفر دارد، مثبت در نظر گرفته می‌شوند. یک عدد منفی یک بیت با ارزش ۱ در آخرین بیت سمت چپ خود دارد. اما تبدیل یک عدد مثبت به‌یک عدد منفی به سادگی تبدیل آخرین بیت به مقدار ۱ نیست. به عنوان مثال نمی‌توان با تبدیل (۶۵+) 01000001 به 11000001 آنرا منفی کرد. بلکه یک عدد منفی بصورت متمم دو نمایش داده می‌شود. قانون منفی کردن یک عدد به اینصورت است که همه مقادیر بیتها را معکوس کرده و با یک جمع کنیم.

مثال: می خواهیم با استفاده از 01000001 که معادل ۶۵+ است عدد ۶۵- را بدست آوریم.

عدد ۶۵+	01000001
معکوس بیتها	10111110
جمع با یک	1
عدد ۶۵-	10111111

برای محاسبه قدرمطلق یک عدد دودویی منفی مجدداً باید متمم دوی آنرا محاسبه کرد.

عدد ۶۵-	10111111
معکوس بیتها	01000000
جمع با یک	1
عدد ۶۵+	01000001

برای بررسی صحت عمل، باید حاصلجمع دو عدد 01000001 و 01000001 برابر صفر شود.

عدد ۶۵+	01000001 +	
عدد ۶۵-	10111111	
حاصلجمع	00000000	1

در حاصلجمع همه‌ی بیتها صفر شده‌اند و تنها یک رقم نقلی داریم که چون رقم نقلی محاسبه نمی‌شود حاصل درست است.

۱-۶-۱-۲ - جمع و تفریق داده‌های علامتدار و بدون علامت:

برخی فیلدهای عددی بدون علامت[5] و برخی دیگر علامتدار هستند. به عنوان مثال، تعداد مشترکین یک شبکه، تعداد روزهای ماه و... اعدادی مثبت هستند و لذا بدون علامت خوانده می‌شوند ولی بدهکاری تراز مشتریان، اعداد جبری، فاصله طی شده نسبت به یک مبدأ و... ممکن است منفی یا مثبت باشند به همین خاطر آنها را علامتدار گویند. در داده‌های بدون علامت همه‌ی بیتها نشان دهنده مقدار (بیت داده) هستند اما در اعداد علامتدار، آخرین بیت سمت چپ (مثلاً در سیستم ۸ بیتی $b_3b_2b_1b_0$ $b_7b_6b_5b_4$ بیت b_7) بیت علامت است بطوریکه اگر این بیت ۰ باشد عدد مثبت و اگر ۱ باشد عدد منفی در نظر گرفته می‌شود.

به این ترتیب به عنوان مثال در سیستم ۱۶ بیتی عدد بدون علامت می‌تواند از ۰ تا ۶۵۵۳۵=2^{16} باشد در حالی که برای اعداد علامتدار می‌تواند بیانگر یک عدد در محدوده ۳۲۷۶۷- تا ۳۲۷۶۷+ باشد. عدد 11111001 در سیستم اعداد بدون علامت معادل عدد ۲۴۹ دسیمال است در حالی که در سیستم اعداد علامتدار معادل عدد ۷- است.

در عملیات جمع و تفریق اعداد علامتدار و بدون علامت باید به دو پرچم رقم نقلی و سرریز محاسباتی دقت کرد.

- وقتی یک رقم نقلی در داده‌های بدون علامت رخ دهد نتیجه نامعتبر است.
- اگر یک سرریز محاسباتی بر روی داده‌های علامتدار رخ دهد نتیجه نامعتبر است.

یک عملیات محاسباتی پرچم سرریز را وقتی یک می‌کند که رقم نقلی به بیت علامت، رقم نقلی خروجی نداشته باشد یا در صورت عدم وجود رقم نقلی ورودی، یک رقم نقلی خروجی رخ دهد.

به مثالهای زیر دقت کنید.

مثال۱:

دهدهی علامتدار	دهدهی بدون علامت	دودویی
+ -۷	+ ۲۴۹	+ 11111001
+۲	۲	00000010
-۵	۲۵۱	11111011

در این مثال رقم نقلی و سرریز محاسباتی وجود ندارد بنابراین، هم برای سیستم اعداد بدون علامت و هم سیستم اعداد علامتدار نتیجه محاسبات معتبر است.

مثال۲:

دهدهی علامتدار	دهدهی بدون علامت	دودویی
+ -۴	+ ۲۵۲	+ 11111100
+۵	+۵	00000101
+۱	۱	1 00000001

5 - Unsigned

در این مثال چون رقم نقلی ایجاد می‌شود بنابراین برای حالت بدون علامت نتیجه محاسبات نامعتبر است ولی چون سرریز محاسباتی وجود ندارد بنابراین نتیجه محاسبات برای اعداد علامتدار معتبر است.

مثال۳:

دهدهی علامتدار		دهدهی بدون علامت		دودویی
+۱۲۱	+	۱۲۱	+	01111001 +
+۱۱		+۱۱		00001011
-۱۲۴		۱۳۲		10000100

در این مثال چون رقم نقلی نداریم بنابراین برای حالت بدون علامت نتیجه محاسبات معتبر است ولی چون سرریز محاسباتی ایجاد می‌شود بنابراین نتیجه محاسبات برای اعداد علامتدار نامعتبر است.

مثال۴:

دهدهی علامتدار		دهدهی بدون علامت		دودویی
-۷	+	۲۴۹	+	11111001 +
-۱۲۲		۱۳۴		10000110
+۱۲۷		۱۲۷		1 01111111

در این مثال هم رقم نقلی داریم و هم سرریز محاسباتی. بنابراین نتیجه محاسبات هم برای سیستم اعداد علامتدار و هم برای بدون علامت نامعتبر است.

به همین خاطر است که در تمام پروسسورها عملیات جمع و تفریق هر دو فلگ رقم نقلی و سرریز محاسباتی را متأثر می‌کنند.

در عملیات محاسباتی مراقب سرریز باشید، مخصوصاً برای داده‌های علامتدار. از آنجا که یک بایت برای فقط یک بیت علامت و ۷ بیت داده (از ۱۲۸- تا ۱۲۸+) فراهم شده است، یک عملیات محاسباتی براحتی از ظرفیت یک رجیستر یک بایتی تجاوز می‌کند و ممکن است نتایج حاصله خارج از انتظار داشته باشد. به عنوان مثال اگر رجیستر ۸ بیتی AL شامل 80 H باشد و دستور ADD AL,20 H اجرا شود 80 H در AL تولید می‌شود که معادل دودویی 10000000 دارد که در سیستم اعداد علامتدار عدد منفی ۱۲۸- است که به جای عدد ۱۲۸+ حاصل شده است.

تمرین: حاصل جمع دو عدد باینری 0011 1001 و 1101 1011 را بدست آورده و معادل این دو عدد و حاصل جمع آنها را در سیستم اعداد بدون علامت و سیستم اعداد علامتدار بدست آورده و با توجه به آنها در مورد صحت و یا عدم صحت نتیجه حاصلجمع فوق بحث کنید.

برای انجام عمل تفریق دو عدد باینری عدد تفریق شونده را متمم دو کرده و با عدد اول جمع می‌کنیم. به عنوان مثال برای تفریق عدد ۴۲ (00101010) از عدد ۶۵+ (01000001) داریم.

عدد ۶۵+	01000001 +
عدد ۴۲-	11010110
حاصل تفریق	1 00010111

عدد ۴۲+	00101010
معکوس بیتها	11010101
جمع با یک	1
عدد ۴۲-	11010110

چنانکه ملاحظه شد حاصل تفریق عدد باینری 00010111 شد که معادل عدد ۲۳ دسیمال است و یک رقم نقلی خروجی نیز دارد ولی سرریز محاسباتی ندارد.

۱-۶-۲- سیستم هشت‌تایی (اکتال)

در سیستم عددنویسی اکتال، برای نمایش اعداد از ارقام ۰ تا ۷ استفاده می‌شود و مثل سیسستم دودویی، برای تبدیل مبنا از ۸ به ۱۰ از عمل ضرب و برای تبدیل مبنا از ۱۰ به ۸ از عمل تقسیم متوالی استفاده می‌کنیم. به عنوان مثال داریم:

$$(۲۳)_۸ = ۲ \times ۸^۱ + ۳ \times ۸^۰ = ۱۹$$

۱۹ | ۸
۱۶ (۲)
(۳)

$$(۱۹)_{۱۰} = (۲۳)_۸$$

در این سیستم، رقم نقلی و رقم قرضی ۸ است، یعنی اگر در جمع دو عدد در مبنای ۸، مجموع از ۷ بیشتر شد، ۸ واحد از آن کم می‌کنیم و یک واحد را به سمت چپ انتقال می‌دهیم. به مثالهای زیر توجه کنید.

۱ ۱
۶ ۰ ۵
+ ۳ ۷ ۶
۱ ۲ ۰ ۳

۷
۳ ۸ ۱۴
۴ ۰ ۶
− ۱ ۵ ۷
۲ ۲ ۷

در مثال فوق مجموع دو عدد ۵ و ۶ برابر ۱۱ است که از بزرگترین نماد در مبنای هشت (یعنی ۷) بزرگتر شده است؛ پس ۸ واحد از آن کم می‌کنیم و یک واحد به طبقه‌ی بالاتر اضافه می‌کنیم و عدد ۳ یعنی حاصل تفریق را در پایین می‌نویسیم و این کار را برای سایر سطوح نیز انجام می‌دهیم. همچنین در تفریق عدد ۷ از ۶ ابتدا ۸ واحد بصورت قرضی به عدد ۶ داده می‌شود تا عدد ۱۴ حاصل شود (بنابراین باید یک واحد از طبقه‌ی سمت چپ کم شود اما چون رقم سمت چپ صفر است و کم کردن از آن امکانپذیر نیست ابتدا یک واحد از عدد ۴ کم می‌کنیم، سپس ۸ واحد به عدد صفر اضافه کرده و در نهایت یک واحد از آن کم می‌کنیم.

۱-۶-۳- سیستم شانزده‌تایی (هگزادسیمال)

در سیستم عددنویسی هگزادسیمال، برای نمایش اعداد از ارقام ۰ تا ۹ و شش نماد A, B, C, D, E, F استفاده می‌شود که نمادهای A تا F معادل اعداد ۱۰ تا ۱۵ هستند. در این سیسستم ارزش هر طبقه، ۱۶ برابر ارزش طبقه‌ی سمت راست آنست. زبان اسمبلی بطور قابل توجهی از سیستم هگزادسیمال (مبنای ۱۶) استفاده می‌کند. یک برنامه اسمبلی شده، تمام آدرسهای کد ماشین و محتویات رجیسترها را بصورت مبنای ۱۶ نشان می‌دهد. در این سیستم، عدد بعد از F بصورت 10 است که مقدار دهدهی آن ۱۶ است. به مثالهای زیر توجه کنید.

$$(۴EB)_{۱۶} = ۴ \times ۱۶^۲ + ۱۴ \times ۱۶ + ۱۱ = ۱۲۵۹$$

۱۲۵۹ | ۱۶
۱۲۴۸ ۷۸ | ۱۶
(۱۱)(= B) ۶۴ (۴)
(۱۴)(= E)

38 H +
18 H
50 H

10 H +
30 H
40 H

FF H +
1 H
100 H

۱-۷-۴- تبدیل مستقیم توانهای دو

با توجه به آنکه اعداد ۸ و ۱۶ توانهایی از دو هستند، می‌توان آنها را به روش ساده‌ای به هم تبدیل کرد. در این روش ابتدا عدد را در مبنای دو و سپس به مبنای مورد نظر تبدیل می‌کنیم. مثلاً در تبدیل از مبنای ۲ به مبنای ۱۶، هر چهار رقم در مبنای ۲ معادل یک رقم در مبنای ۱۶ است. به همین ترتیب، هر ۳ رقم در مبنای ۲ معادل یک رقم در مبنای ۸ است. به مثالهای زیر توجه کنید.

$$\begin{array}{ccc}(۷ & ۲ & ۵)_۸ \\ \downarrow & \downarrow & \downarrow \\ (۱۱۱ & ۰۱۰ & ۱۰۱)_۲\end{array} \rightarrow (۷۲۵)_۸ = (۱۱۱۰۱۰۱۰۱)_۲$$

$$(۴FB۸)_{۱۶} = (۰۱۰۰ \quad ۱۱۱۱ \quad ۱۰۱۱ \quad ۱۰۰۰)_۲$$

$$(۱۰۱۱ \quad ۰۱۱۱ \quad ۱۰۱۰)_۲ = (B۷A)_{۱۶}$$

$$(۰۱۰ \quad ۱۰۱ \quad ۰۰۱)_۲ = (۲۵۱)_۸$$

$$(۴AF)_{۱۶} = (۰۱۰۰ \quad ۱۰۱۰ \quad ۱۱۱۱)_۲ = (۰۱۰ \quad ۰۱۰ \quad ۱۰۱ \quad ۱۱۱)_۲ = (۲۲۵۷)_۸$$

۱-۸- کد اسکی و Unicode

چون کامپیوترها چیزی به غیر از کدهای مبنای دو رقمی را نمی فهمند لذا به منظور استانداردسازی بیان اطلاعات جهت نمایش حروف، ارقام و علامتها باید یک کد در نظر گرفت. یک روش متداول برای کدگذاری حروف، ارقام و علامتها در کامپیوتر، استفاده از کد اسکی[۶] است که توسط آن انتقال اطلاعات بین دستگاههای کامپیوتری مختلف ممکن می‌شود. با ۸ بیت کد اسکی (کد اسکی توسعه یافته) که کامپیوترهای شخصی استفاده می‌کنند می‌توان ۲۵۶= $۲^۸$ نشانه را کدگذاری کرد. مثلاً کد حرف A عدد ۶۵ (41 H) است که بصورت 01000001 نمایش داده می‌شود. کد ارقام ۰ تا ۹ اعداد 30 H تا 39 H است. کدهای ۱۲۸ تا ۲۵۵ از سوی سازندگان کامپیوتر و برنامه نویسان به منظورهای خاصی استفاده می‌شوند؛ مثلاً برای اینکه کامپیوتر حروف فارسی را هم بشناسد و بتوانیم در برنامه‌ها از این حروف استفاده کنیم، از این کدها استفاده می‌کنیم. بر خلاف کدهای ۰ تا ۱۲۷ که ثابت هستند، می‌توان نشانه‌های مربوط به کدهای ۱۲۸ تا ۲۵۵ را تغییر داد. در مجموعه کدهای اسکی (کد اسکی معمولی یعنی ۷ بیتی)، ۳۲ کد اولیه برای کاراکترهای ارتباطی و کنترلی مانند کنترل چاپگر و غیره بکار می‌روند و ۹۶ کد دیگر، برای حروف کوچک و بزرگ انگلیسی، ارقام ۰ تا ۹ و سایر علایم موجود روی صفحه کلید بکار می‌روند. در سیستم عاملهایی مانند ویندوز، برای استفاده از نویسه‌های زبانهای دیگر، کدگذاریهایی با تعداد بیت بیشتر استفاده می‌شود. Unicode یکی از این نوع کدگذاریها است که به استانداردی بین المللی برای برآوردن نیازهای مربوط به تبادل اطلاعات چند زبانه تبدیل شده است. در این استاندارد تلاش بر این است که همه‌ی حروف و

6 - American Standard Code for Information Interchange

نمادهای مورد استفاده کشورهای مختلف آورده شود و نمادهای آن در همه جا قابل نمایش است و نیاز به امکانات خاصی ندارد. این کد ۱۶ بیتی بوده و قادر است ۶۵۵۳۶ حالت مختلف از اعداد دودویی را ایجاد کند که هر حالت می‌تواند معادل یک حرف یا یک نشانه زبانهای مختلف باشد.

آشنایی بیشتر با کدهای اسکی:

کاراکتر		مبنای ۱۰		مبنای ۱۶
A	⟶	65	⟶	41
a	⟶	97	⟶	61
Tab	⟶	9	⟶	9
0	⟶	48	⟶	30
2	⟶	50	⟶	32

کدهای اسکی را در دو فرم معمولی و توسعه یافته بصورت کامل می توانید در صفحات بعد مشاهده کنید.

در ادامه بحث، ابتدا مبدلهای دیجیتال به آنالوگ [۷] و آنالوگ به دیجیتال [۸] را به عنوان بخشی از دستگاههای خروجی و ورودی در فصل دوم بررسی خواهیم کرد. در فصل سوم انواع حافظه‌ها و چگونگی دیکود نمودن آدرس آنها را بررسی می‌کنیم. فصل چهارم به آشنایی با ساختمان داخلی پروسسور 8088 اختصاص دارد که بر اساس آن یک سیستم کامپیوتری ارائه خواهیم کرد که با حداقل امکانات قادر به پیاده سازی اهداف ما برای کاربرد مورد نظرمان باشد. به چنین سیستمی، سیستم مینیموم گفته می‌شود. در فصل پنجم، به بررسی دستورالعملها و برنامه نویسی سیستم به زبان اسمبلی می‌پردازیم و به عنوان نتیجه کار برنامه‌هایی ساده به زبان اسمبلی خواهیم نوشت. در فصل آخر مروری بر پروسسور Z80 خواهیم داشت.

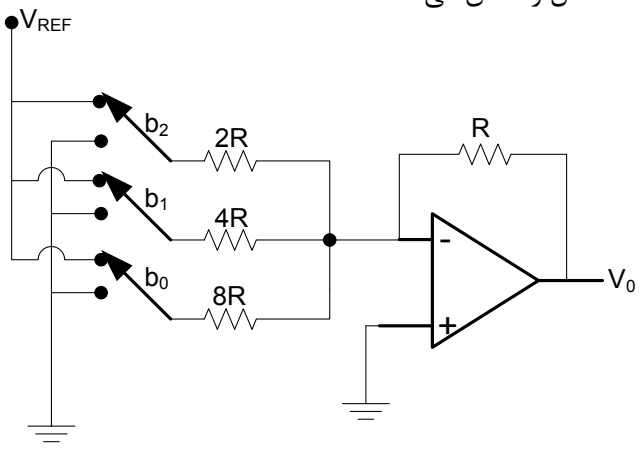
7 - Digital to Analog = D/A
8 -Analog to Digital = A/D

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

۲-۱- مبدلهای دیجیتال به آنالوگ

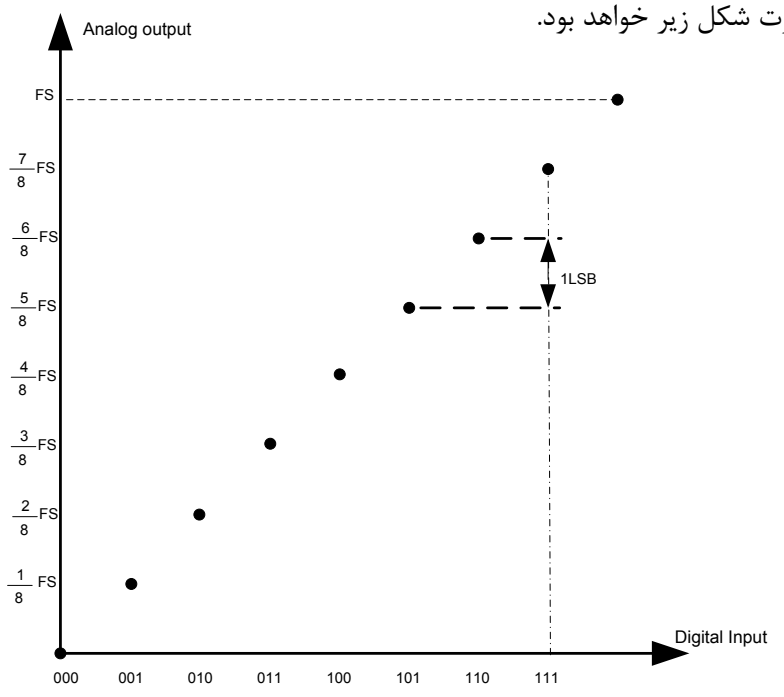
این مبدلها عمل تبدیل یک کد دیجیتال به ولتاژ یا جریان آنالوگ معادل آنرا انجام می‌دهند. شکل زیر یک مدار ساده جهت تبدیل عدد باینری سه بیتی $b_2b_1b_0$ به ولتاژ آنالوگ معادل را نشان می‌دهد.



ولتاژ آنالوگ خروجی، Vo، برابر است با :

$$Vo = -V_{REF}\left(\frac{R}{2R}b_2 + \frac{R}{4R}b_1 + \frac{R}{8R}b_0\right) = -\frac{V_{REF}}{8}\left(2^2 b_2 + 2^1 b_1 + 2^0 b_0\right)$$

در فرمول فوق b_n برای $(n = 0,1,2)$ می‌تواندصفر یا یک باشد. در صورت یک بودن b_n سویچ مربوطه جریان را به پایه منفی جمع کننده هدایت می‌کند و در صورتی که b_n صفر باشد، سویچ متناظر به آن باز است (در شکل فوق هر سه بیت یک فرض شده اند و در نتیجه هر سه سویچ بسته اند). اگر عناصر مدار فوق ایده آل فرض شوند، مشخصه ورودی و خروجیD/A سه بیتی فوق به صورت شکل زیر خواهد بود.

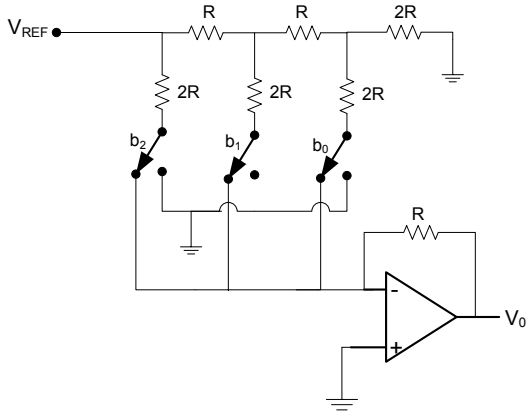


FS = FULL Scale

حداکثر ولتاژ خروجی مبدل دیجیتال به آنالوگ به اندازه یک پله کمتر از ولتاژ تمام رنج (FULL Scale) است. به عنوان مثال، اگر ولتاژ تمام رنج (FULL Scale) برای D/A برابر ۱۰ ولت و D/A از نوع ۱۲ بیتی فرض شود یک LSB معادل $\frac{1}{2^{12}} \times 10 = 0.00244V = 2.44mV$ خواهد بود و حداکثر ولتاژ خروجی مبدل D/A برابر 9.99756 = 0.00244 – 10 خواهد بود که به اندازه یک پله کمتر از ۱۰ ولت است.

$1LSB = 0.00244V = 0.000244FS = 244\,ppm$
PPm = parts per million

از آنجا که ساخت مبدل D/A با مدار ساده صفحه قبل مستلزم استفاده از رنج وسیعی از مقاومتها است، مدل شکل زیر که تنها به دو نوع مقاومت نیاز دارد بیشتر در مدارات مجتمع معمول است.



براحتی می‌توان نشان داد که رابطه ولتاژ خروجی این مبدل نیز مشابه مبدل شکل صفحه قبل از رابطه زیر بدست می‌آید.

$$Vo = -V_{REF}\left(\frac{R}{2R}b_2 + \frac{R}{4R}b_1 + \frac{R}{8R}b_0\right) = -\frac{V_{REF}}{8}\left(2^2 b_2 + 2^1 b_1 + 2^0 b_0\right)$$

مبدلهای D/A که در فوق بررسی شد از نوع تک قطبی (unipolar) هستند. به عنوان نمونه اگر ولتاژ مرجع (V_{REF}) منفی باشد، به ازای هر عدد دیجیتال ورودی، ولتاژ آنالوگ خروجی یک مقدار مثبت (بین صفر ولت تا یک پله کمتر از ولتاژ تمام رنج) خواهد بود. در صوتیکه هدف تبدیل اعداد دیجیتال به یک ولتاژ دوقطبی (ولتاژ خروجی دارای مقدار مثبت ومنفی) باشد باید از مبدل دو قطبی (Bipolar) استفاده کرد. یکی از راههای ساخت مبدل دو قطبی، بکارگیری مبدل تک قطبی است بطوریکه خروجی آن به اندازه 1MSB شیفت داده شود. معمولا برای بهترین نتیجه این افست توسط ولتاژ مرجع ایجاد می‌شود.

۲-۲- معرفی چند آی‌سی D/A :

۲-۲-۱- آی‌سی MC 1408 : این آی‌سی شامل یک منبع جریان مرجع، شبکه نردبانی R-2R و هشت سویچ جریان است. جریان خروجی این مبدل توسط هشت بیت باینری ورودی و جریان مرجع کنترل می‌شود و رابطه آن به صورت زیر است :

$$I_{out} = \frac{I_{REF}}{256}(2^7 b_7 + 2^6 b_6 + \ldots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0) \quad ,b_n = \begin{cases} 1 & High \\ 0 & Low \end{cases}$$

در این آی‌سی با جریان مرجع $I_{REF} = \frac{V_{REF}}{R}$ دردمای $25\ ^0C$ حداکثر خطای $\pm\frac{1}{2}LSB$ ضمانت شده است و زمان پاسخ مبدل حدود 70ns است. شکل زیر بلوک دیاگرام این آی‌سی را نشان می‌دهد.

BLOCK DIAGRAM

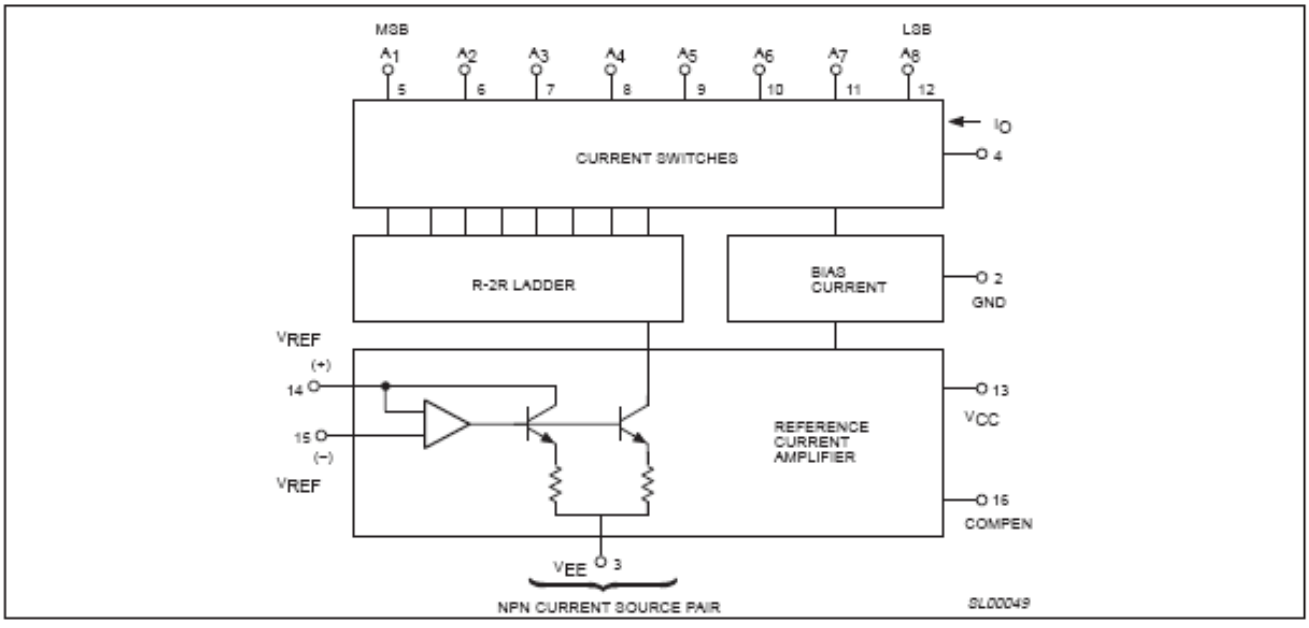
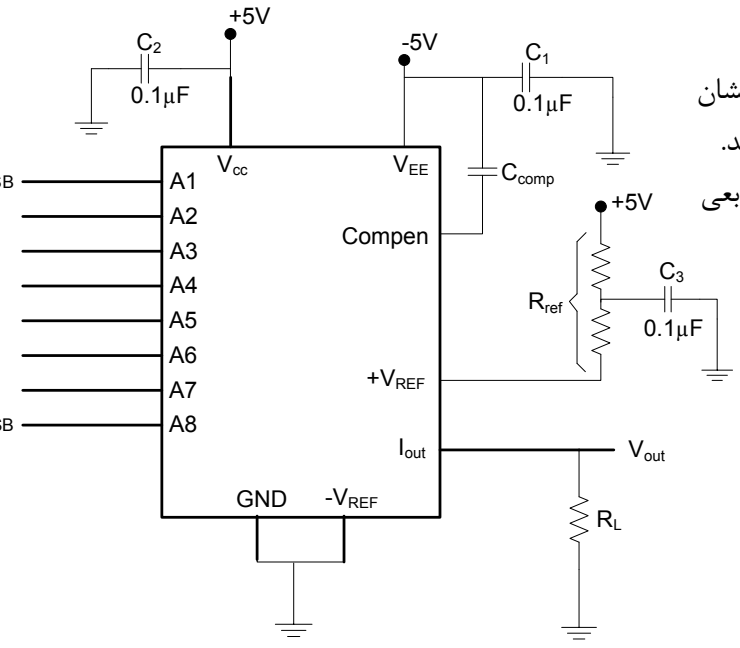


Figure 2. Block Diagram



شکل زیر مدار ساده‌ای از مبدل D/A با این آی‌سی را نشان می‌دهد. V_{EE} باید حداقل ۳ ولت منفی تر از V_{ref}– باشد.

خازن C_{comp} برای پایداری سیستم است و مقدار آن تابعی از R_{ref} است.

جدول زیر چند مقدار پیشنهادی برای R_{ref} و C_{comp} را نشان می‌دهد.

$R_{ref}(k\Omega)$	$C_{comp}(pf)$
1	15
2.5	37
5	75

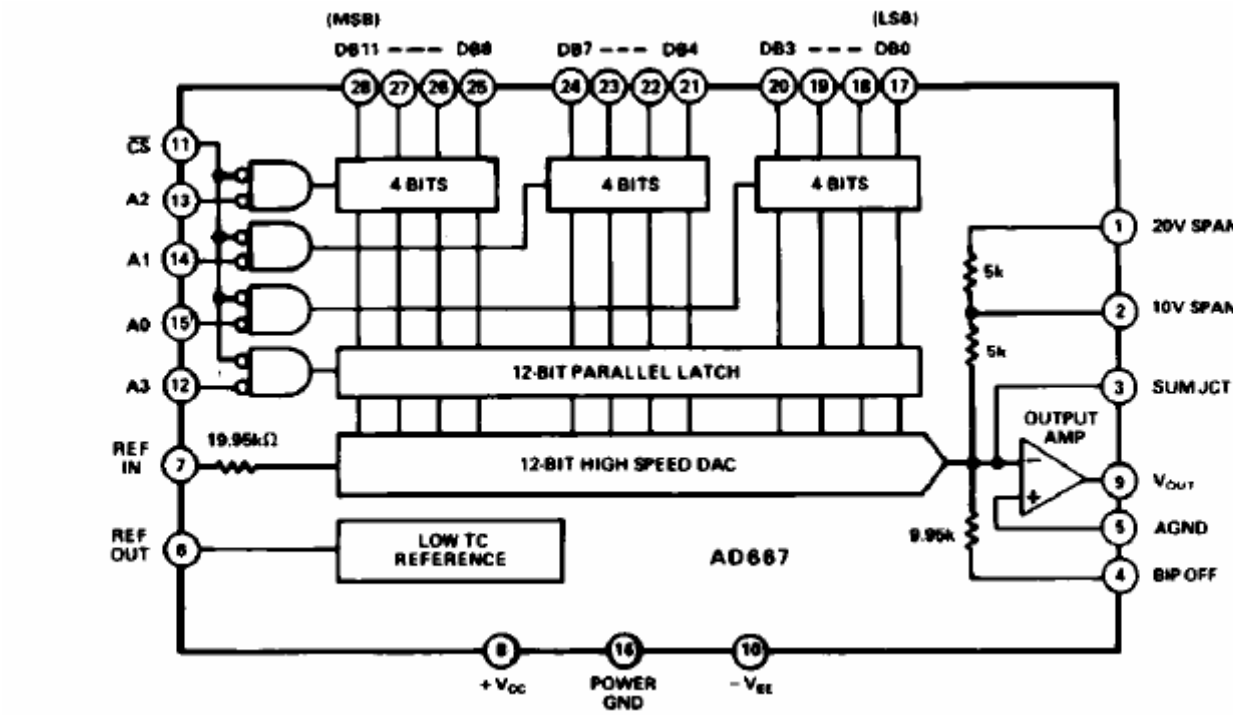
جریان مرجع $I_{REF} = \frac{V_{REF}}{R_{ref}}$ کنترل شیب را در خروجی بر عهده دارد. چون درستی سیستم تابعی از ولتاژ مرجع است لذا باید این ولتاژ را مستقل از تغذیه مدار(و مقدار کاملا رگوله شده‌ای) در نظر گرفت. در کاربردهای معمولی می‌توان مطابق شکل فوق توسط تقسیم R_{ref} به دو مقاومت سری و قراردادن یک خازن بین سر وسط آنها و زمین تا حدی نوسانات V_{REF} را فیلتر کرد.
جهت داشتن سرعت پاسخ مناسب، کارخانه سازنده آی‌سی توصیه کرده است که مقاومت R_L کمتر از 500Ω و خازن نشتی آن نیز کمتر از ۲۵PF باشد. اگر جریان مرجع I_{ref}=2mA وR_L=500Ω باشد، ولتاژ حداکثر خروجی برابر است با :

$$V_{out} = 500 \times \frac{2mA}{256} \times 255 = 0.996V$$

با بکارگیری یک تقویت کنده عملیاتی و اتصال مستقیم پایه I_{out} به پایه منفی تقویت‌کننده و تنظیم مقاومت فیدبک بین خروجی و پایه منفی (تقویت‌کننده عملیاتی وارونگر) می‌توان به هر مقدار خروجی دست یافت. ولتاژ خروجی در این حالت منفی است و در صورتیکه ولتاژ مثبت مد نظر باشد باید جهت جریان خروجی را عوض کرد. این کار را می‌توان با زمین کردن $+V_{REF}$ و قرار دادن مقاومت R_{ref} بین تغذیه منفی و پایه $-V_{REF}$ عملی کرد.

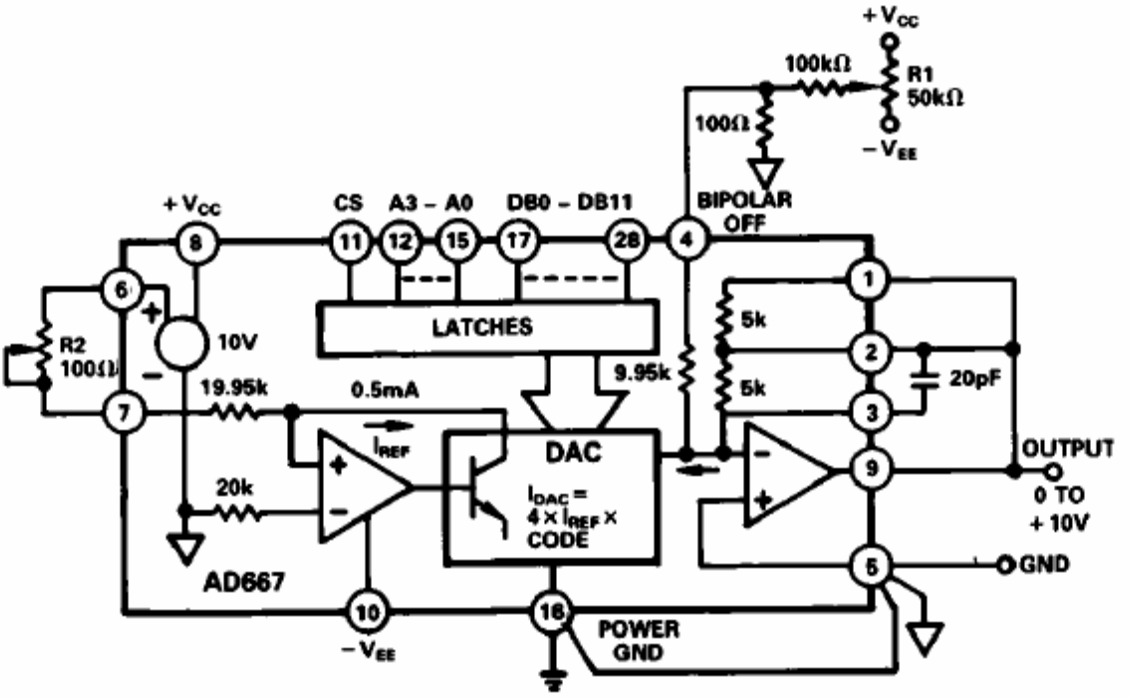
۲-۲-۲- آی‌سی AD667 :

آی‌سیMC1408 نیاز به قطعات اضافی نظیر منبع ولتاژ تثبیت شده، تقویت‌کننده عملیاتی و تعدادی مدار Latch جهت دریافت و نگهداری اطلاعات دریافتی از میکروپروسسور دارد. آی‌سی AD667 محصول شرکت Analog Device کلیه مدارات جانبی مورد نیاز فوق را در خود دارد. این آی‌سی یک D/A دوازده بیتی است که بلوک دیاگرام داخلی آن بصورت زیر است (www.alldatasheet.com).

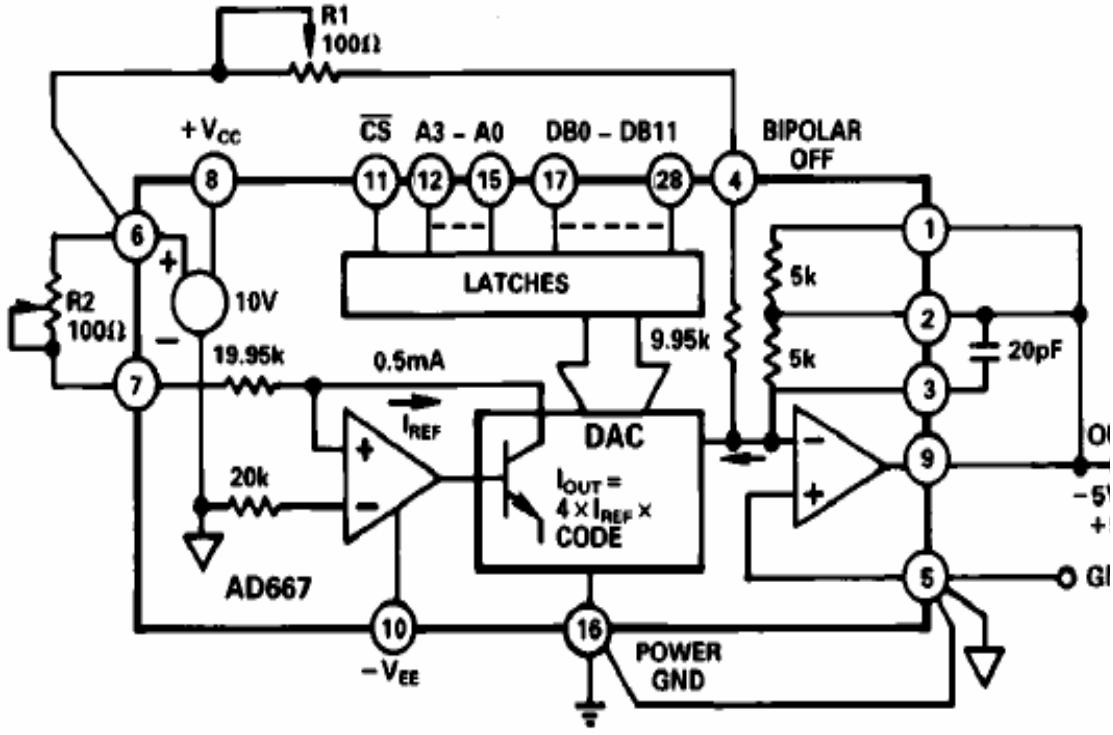


بلوک دیاگرام داخلی مبدل دیجیتال به آنالوگ AD667

چنانکه بلوک دیاگرام آی‌سی AD667 نشان می‌دهد اطلاعات ورودی می‌تواند توسط پروسسورهای ۴، ۸ یا ۱۲ بیتی وارد مبدل شوند. این عمل با بکارگیری دو طبقه Latch و کنترل آنها توسط A_0 تا A_3 امکانپذیر است. شکلهای زیر اتصالات لازم جهت ایجاد ولتاژ خروجی صفر تا ده ولت (تک قطبی) و ایجاد ولتاژ خروجی ۵- تا ۵+ ولت (دو قطبی) را نشان می‌دهند.



مبدلD/A با ولتاژ خروجی صفر الی ۱۰ ولت توسط آی سی AD667

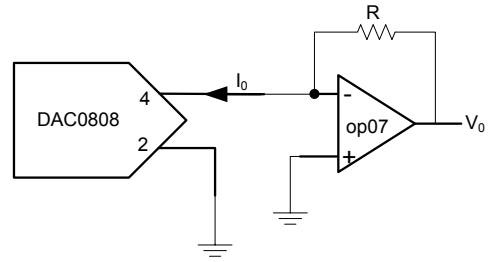


مبدلD/A با ولتاژ خروجی ۵- الی ۵+ ولت توسط آی سی AD667

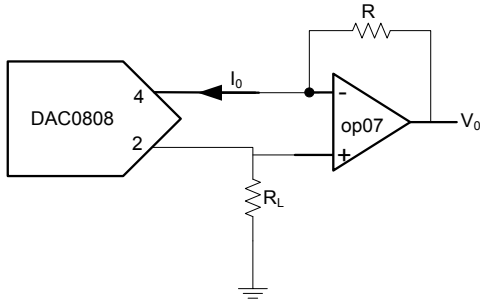
۲-۲-۳- آی‌سی DAC0808 :

این آی‌سی محصول کارخانه National semiconductor بوده و یک مبدل ۸ بیتی است. ورودی دیجیتالی اعمال شده به آی‌سی، باعث تغییر وضعیت سویچها شده و باعث می‌شود که جریان در پایه های ۲ و ۴ (شکل زیر) بطور متناسب تغییر کند. مجموع جریان این دو پایه همواره ثابت و برابر $\frac{255}{256} I_{REF}$ است که I_{REF} جریان وارد شده به پایه $+V_{REF}$ (پایه شماره ۱۴) بوده و $I_{REF} = \frac{V_{REF}}{R_{ref}}$ است. یکی از مزایای این آی‌سی، رنج وسیع تغذیه آن است که می‌تواند از $\pm 4.5V$ ولت تا $\pm 18V$ ولت باشد. جهت آشنایی با ساختمان داخلی این آی سی می‌توانید به دیتاشیت این آی سی مراجعه کنید.

در اغلب اوقات نیاز به ولتاژ خروجی است تا جریان لذا باید عمل تبدیل جریان به ولتاژ را انجام دهیم که به این منظور می توان از مدار های زیر ولتاژ صفر تا FS یا ولتاژ FS– تا FS+ را از این آی‌سی دریافت کرد.

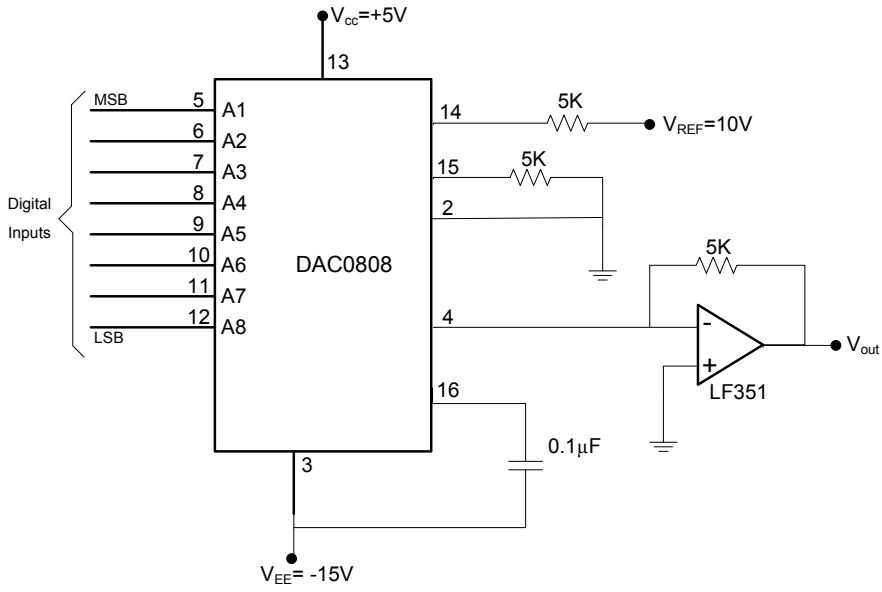


خروجی صفر تا FS+ ولت



خروجی FS- تا FS+ ولت

شکل زیر یک مدار نمونه مبدل DAC0808 را نشان می‌دهد.



مبدل دیجیتال به آنالوگ با ولتاژ خروجی ۱۰ ولت با استفاده از DAC0808

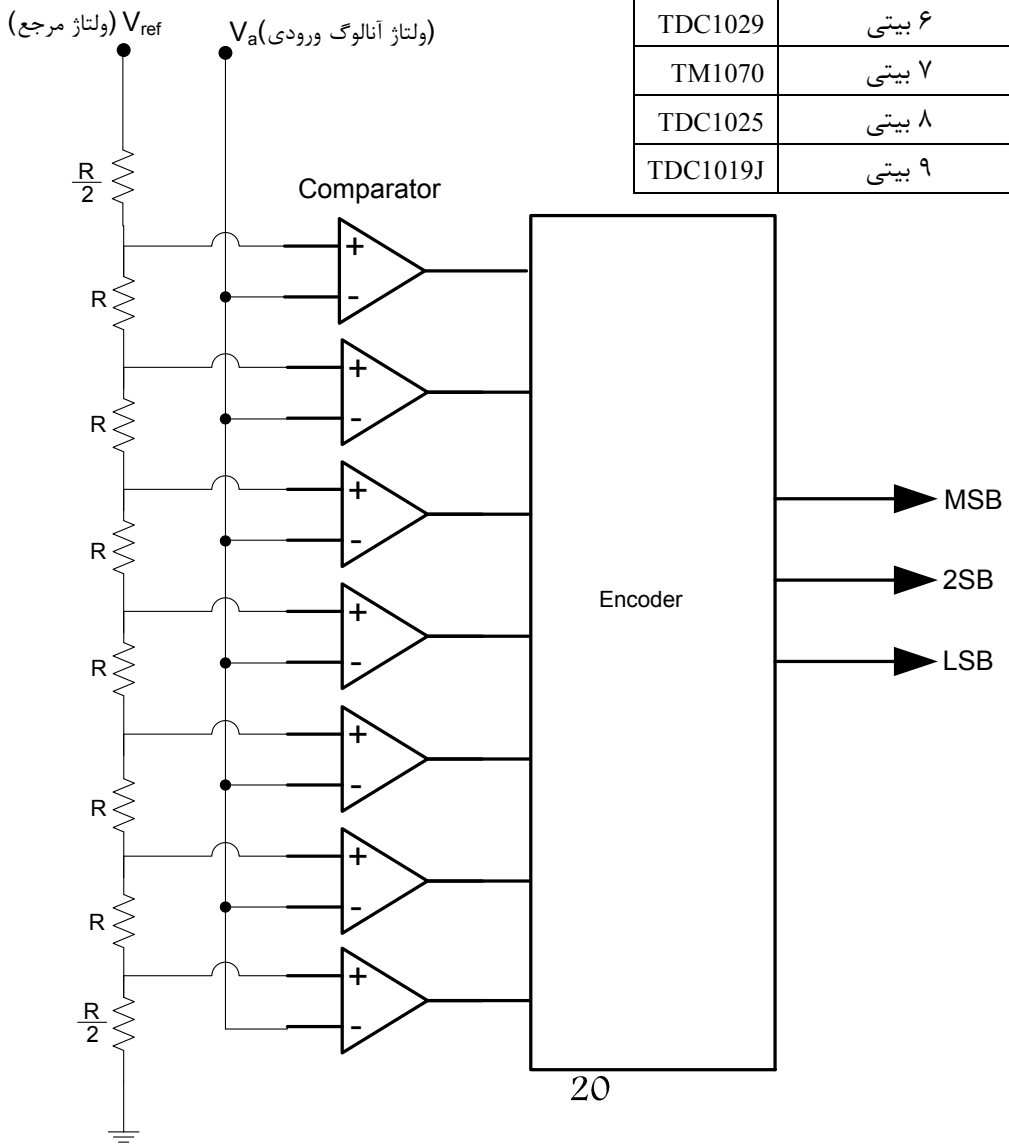
۲-۳- مبدلهای آنالوگ به دیجیتال

روشهای مختلفی جهت انجام تبدیل ولتاژ آنالوگ به یک کد دیجیتال وجود دارد. انتخاب روش می‌تواند در عواملی چون سرعت تبدیل، مصونیت نسبت به نویز و بالاخره حساسیت نسبت به تغییر پارامترهای مبدل نظیر فرکانس و . . . مؤثر باشد. در ادامه بحث به معرفی تعدادی از روشهای موجود و بررسی نقاط ضعف و قوت هر روش می‌پردازیم.

۲-۳-۱ - مبدلهای موازی (همزمان) :

شکل زیر شمای ساده‌ای از یک مبدل موازی، که به کمک ۷ عدد مقایسه کننده، یک شبکه مقاومتی و یک مدار انکودر (Encoder) ساخته شده است، را نشان می‌دهد. این مبدل از نوع سه بیتی بوده و در حالت کلی در صورت نیاز به قدرت تفکیک N بیت، احتیاج به $2^N - 1$ مقایسه کننده می‌باشد. گر چه این نوع مبدل از سریعترین مبدلهای موجود است ولی کاربرد آن به صورت مدار مجتمع به علت بالا رفتن تصاعدی مقایسه کننده ها، به حداکثر ۸ یا ۹ بیت محدود گشته است. برخی آی‌سی های با این روش تبدیل عبارتند از :

نام آی سی	رزولوشن (تعداد بیت)	نرخ تبدیل
CA3300D	۶ بیتی	15MHZ
TDC1029	۶ بیتی	100MHZ
TM1070	۷ بیتی	15MHZ
TDC1025	۸ بیتی	75MHZ
TDC1019J	۹ بیتی	25MHZ

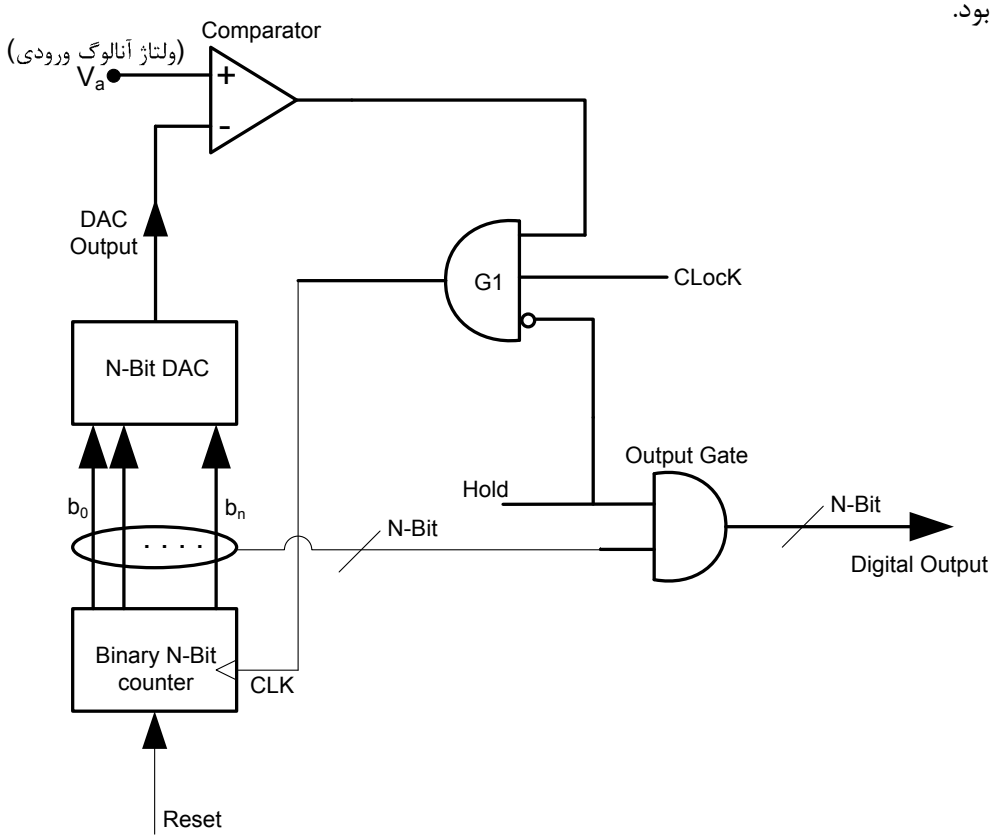


۲-۳-۲- مبدل A/D از نوع پله‌ای :

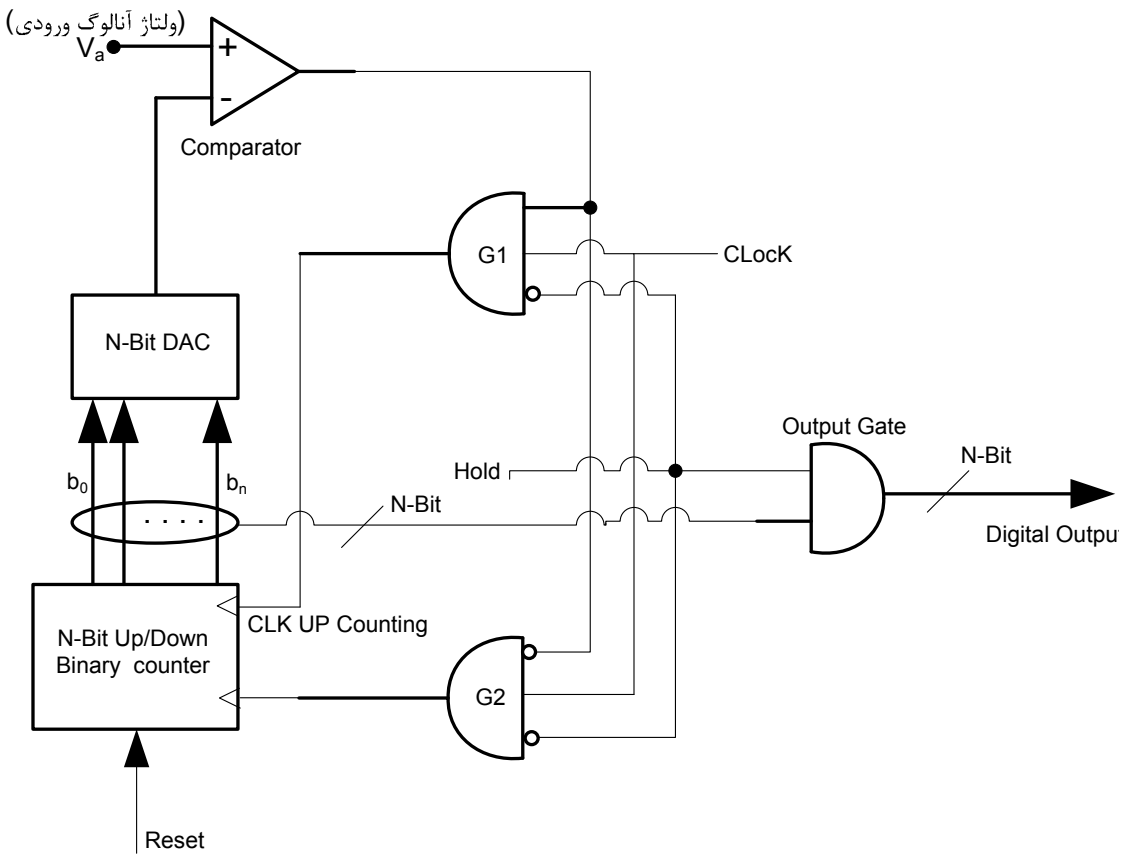
شکل زیر شمای ساده‌ای از مبدل آنالوگ به دیجیتال از نوع پله‌ای را نشان می‌دهد. چنانکه شکل نشان می‌دهد، خروجی یک شمارنده باینری N بیتی به یک مبدل دیجیتال به آنالوگ متصل شده است و ایجاد یک موج پله‌ای در خروجی مبدل دیجیتال به آنالوگ می‌کند. ولتاژ پله‌ای خروجی D/A ، با اعمال هر پالس که به شمارنده اعمال می‌گردد به اندازه یک پله افزایش یافته و به پایه منفی یک مقایسه کننده اعمال می‌گردد. پایه مثبت مقایسه کننده به ولتاژ آنالوگ ورودی V_a (که هدف تبدیل این ولتاژ به مقدار دیجیتال است) متصل شده است. در ابتدای تبدیل محتوای شمارنده (و در نتیجه خروجی مبدل D/A) صفر بوده و خروجی مقایسه کننده یک است. لذا پالسهای کلاک از طریق گیت G_1 به شمارنده منتقل شده و با شروع شمارش، موج پله‌ای شروع به افزایش می‌کند. به محض اینکه پایه منفی به اندازه 1LSB از V_a بیشتر شود خروجی مقایسه کننده صفر شده و شمارش متوقف می‌شود. در این لحظه محتوای شمارنده را می‌توان با ولتاژ V_a به کمک رابطه تعادلی زیر مربوط کرد.

$$V_a = V_{D/A} = KV_{ref}.N \Rightarrow N = \frac{V_a}{KV_{ref}}$$

با انتخاب $KV_{ref} = 1000$ می‌توان ولتاژ ورودی را با دقت هزارم ولت (میلی ولت) نمایش داد. زمان تبدیل تابعی از ولتاژ آنالوگ ورودی بوده که حداکثر آن $2^N T_{Clock}$ است که بازای حداکثر ولتاژ ورودی اتفاق می‌افتد. به عنوان مثال برای یک مبدل ۱۴ بیتی با فرض $f_{Clock} = 10MHZ$ زمان فوق $2^N T_{Clock} = 2^{14} \times \frac{1}{10 \times 10^6} = 1.64m\sec$ و فرکانس تبدیل ۶۱۰ تبدیل بر ثانیه خواهد بود.



با تعویض شمارنده قبلی با یک شمارنده باینری از نوع up/down مطابق شکل زیر می‌توان زمان متوسط تبدیل را تا حد قابل توجهی کاهش داد.

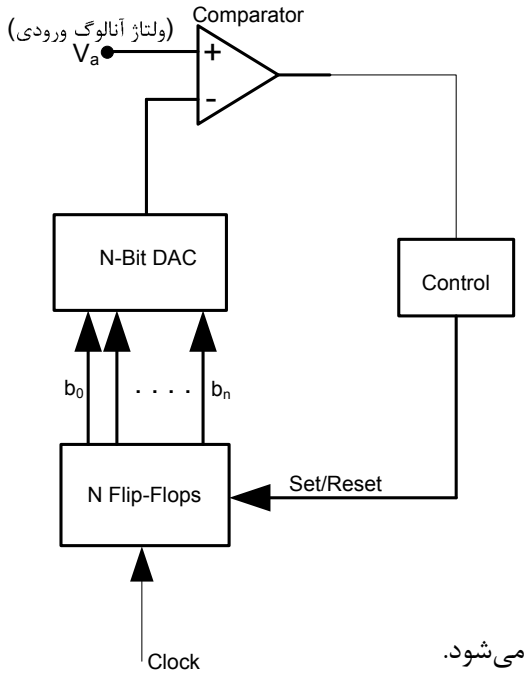


این نوع مبدل را Servo A/D یا Tracking می‌گویند. در این حالت بر خلاف حالت قبلی بعد از پایان هر تبدیل، شمارنده صفر نشده و مقدار خود را حفظ می‌کند. بر حسب اینکه ولتاژ آنالوگ اعمال شده در نمونه بعدی، بیشتر یا کمتر باشد، محتوای شمارنده زیاد و یا کم می‌گردد. در این حالت اگر ولتاژ ورودی دارای تغییرات کندی باشد مبدل خیلی سریع خود را با ورودی هماهنگ می‌کند. ولی اگر ولتاژ وردی تغییرات سریع مثلاً از حداکثر به حداقل داشته باشد، زمان تبدیل تفاوتی با حالت ساده قبلی نخواهد داشت. از مزایای جالب این روش، عدم وابستگی به فرکانس کلاک می‌باشد. سیگنال HOLD در هردو مبدل جهت نگهداری مقدار تبدیل شده در شمارنده است. در ساختمان مبدل نوع پله‌ای از حلقه فیدبک استفاده شده است. نوع دیگری که در ساختمان آن از حلقه فیدبک استفاده شده است، روش تقریبات متوالی است که دقت آن همانند نوع پله‌ای است ولی زمان تبدیل آن به مراتب کمتر از نوع پله‌ای ساده است.

۲-۳-۳- مبدل A/D با روش تقریبات متوالی :

شمای ساده‌ای از مبدل آنالوگ به دیجیتال با روش تقریبات متوالی را در شکل صفحه بعد مشاهده می‌کنید. چنانکه شکل نشان می‌دهد، این مبدل N بیتی شامل N فلیپ فلاپ بوده که توسط مدار کنترل، مقادیر صفر یا یک به خود می‌گیرند. این فلیپ فلاپها در ورودی یک مبدل D/A قرار گرفته و خروجی مبدل D/A توسط مقایسه کننده، با ولتاژ آنالوگ مورد نظر(V_a) مقایسه می‌گردد.

برای درک بهتر عملکرد مدار فرض کنید ولتاژ آنالوگ ورودی ۱۰/۷ ولت بوده ، N=۴ و حداکثر ولتاژ خروجی مبدل D/A برابر ۱۶ ولت (البته به اندازه یک پله کمتر) باشد مراحل کار مبدل بصورت زیر است :



در ابتدا همه بیت ها بجز MSB صفر هستند و ولتاژ خروجی D/A معادل N (عدد ورودی D/A) است یعنی معادل ۸ ولت است.

MSB			LSB
1	0	0	0

کلاک اول : چونV_a <۸ است لذا توسط مدار کنترل در کلاک دوم مقدار MSB همان یک باقی مانده و بیت بعدی یک می‌شود. در این حالت خروجی D/A برابر ۱۲ ولت می‌شود.

MSB			LSB
1	1	0	0

کلاک دوم چون V_a >۱۲ است لذا مدار کنترل بیت دوم را صفر کرده و بیت سوم را یک می‌کند که در این حالت خروجی D/A برابر ۱۰ ولت می‌شود.

MSB			LSB
1	0	1	0

کلاک سوم : چون V_a <۱۰ است لذا مدار کنترل ضمن حفظ تغییر قبلی، بیت آخر (LSB) را یک کرده و عمل تبدیل پایان می پذیرد. در این حالت عدد ۱۱ روی نمایشگر ظاهر خواهد شد.

MSB			LSB
1	0	1	1

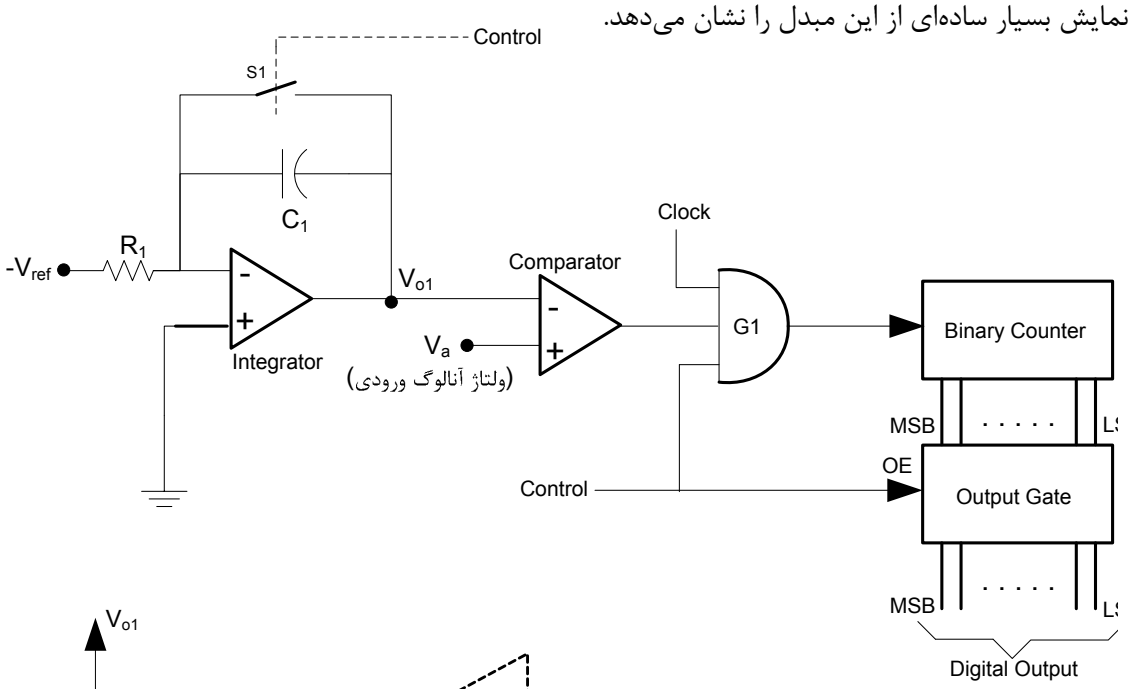
کلاک چهارم : چونV_a >۱۱ است لذا مقدارLSB مجددا به صفر برگشته و تبدیل پایان می پذیرد.

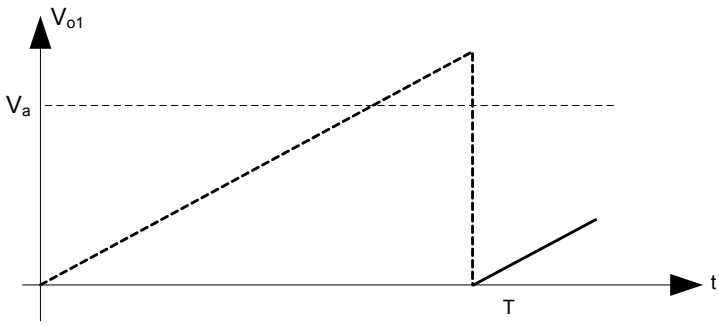
MSB			LSB
1	0	1	0

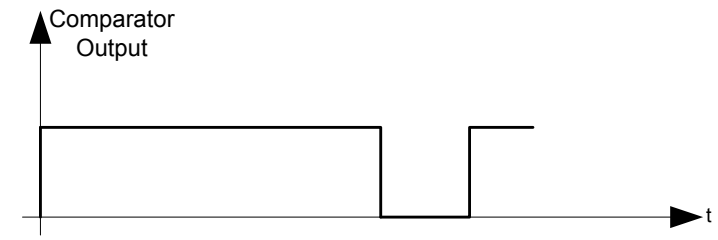
به این ترتیب ملاحظه می‌شود که زمان تبدیل مستقل از ولتاژ ورودی بوده و برای یک مبدل N بیتی برابر $N \times T_{Clock}$ است. دقت روش تقریبات متوالی همانند نوع پله‌ای است ولی زمان تبدیل آن به مراتب کمتر از نوع پله‌ای ساده است و در مقایسه با زمان تبدیل مبدل پله‌ای از نوع Tracking ، سیگنال ورودی را محدود به تغییرات کند نمی نماید.

۲-۳-۴- مبدل A/D با روش تبدیل ولتاژ به زمان یک شیبی:

شکل زیر نمایش بسیار ساده‌ای از این مبدل را نشان می‌دهد.







در این روش از ولتاژ مرجع (که مقدار ثابتی است) انتگرال گرفته شده و نتیجه (V_{01}) که به صورت شیب است توسط یک مقایسه کننده با ولتاژ مورد نظر V_a مقایسه می‌گردد. خروجی مقایسه کننده تا زمانی که V_{01} به V_a نرسیده است دارای منطق یک بوده و لذا به گیت AND اجازه ورود کلاک به شمارنده را می‌دهد. شمارش تا لحظه رسیدن V_{01} به V_a ادامه خواهد داشت و سپس متوقف شده و داریم :

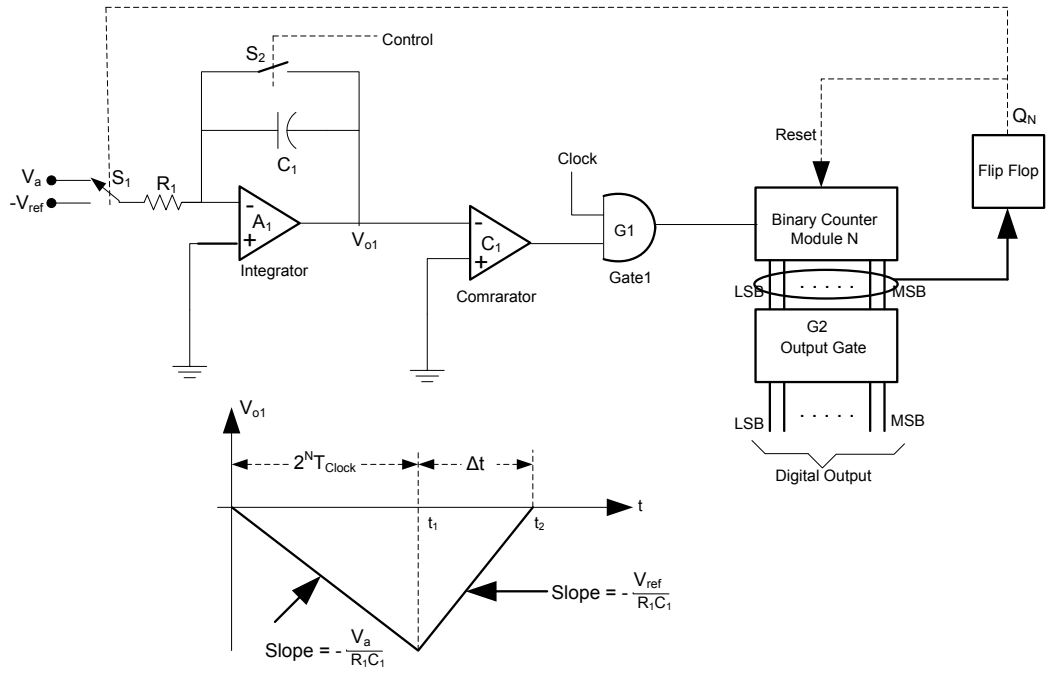
$$V_a = \frac{V_{ref}}{R_1C_1}T = \frac{V_{ref}}{R_1C_1}\frac{n}{f_{Clock}} \Rightarrow n = \frac{R_1C_1}{V_{ref}}.V_a \cdot f_{Clock}$$

رابطه فوق نشان می‌دهد که مراحل تبدیل عبارتند از تبدیل ولتاژ ورودی به زمان و سپس اندازه‌گیری زمان توسط شمارش پالس. چنانکه رابطه نشان می‌دهد مقدار دیجیتال خروجی به فرکانس، ظرفیت خازن و مقاومت انتگراتور بستگی دارد.

تذکر: مبدلهایی که تا کنون بحث شدند همگی نسبت به نویز و یا تداخل که با سیگنال ورودی جمع شده باشد حساس هستند. علت این مساله، مقایسه سیگنال ورودی و سیگنال تولید شده توسط مبدل D/A در یک لحظه خاص از زمان است. دو روش باقیمانده در ادامه بحث، به علت انتگرالگیری از سیگنال ورودی دارای مصونیت ذاتی نسبت به نویز بوده و در صورت انتخاب مناسب زمان انتگرالگیری می‌توانند کاملا ایمن در برابر تداخل فرکانسهای مزاحمی نظیر ۵۰ هرتز برق شهر باشند.

۲-۳-۵- مبدل A/D دو شیبی :

شکل زیر مدار بسیار ساده‌ای از مبدل دو شیبی را نشان می‌دهد.



در ابتدا t_1=۰ و کلید S_1 به ولتاژ آنالوگ ورودی V_a وصل بوده و کلید S_2 باز است و انتگرالگیری از ولتاژ ورودی V_a آغاز می‌شود. ولتاژ خروجی انتگراتور برابر است با :

$$V_{o1} = \frac{-1}{R_1 C_1}\int_0^t V_a .dt = \frac{-V_a}{R_1 C_1}t = \frac{-V_a}{\tau_1}t$$

بطوریکه $\tau_1 = R_1 C_1$ ثابت زمانی انتگراتور بوده و V_a در طول زمان انتگرالگیری ثابت فرض می‌شود. شمارنده باینری N بیتی نیز همزمان با شروع انتگرالگیری به علت باز بودن گیت AND با فرکانس کلاک شروع به شمارش می‌کند و پس از پایان 2^N کلاک (یعنی در لحظه $t = 2^N T_{Clock}$) خروجی فلیپ فلاپ تغییر وضعیت داده ($Q_N = 1$ شده) که خود منجر به تغییر وضعیت کلید S_1 از V_a به $-V_{ref}$ می‌شود. درست در همین لحظه شمارنده باینری که به انتهای شمارش خود رسیده به موقعیت صفر بر می‌گردد و به علت منفی بودن ولتاژ مرجع، خازن انتگراتور در جهت معکوس شارژ شده و ولتاژ انتگراتور از منفی به سمت صفر و مثبت شدن تغییر می‌کند بطوریکه ولتاژ خروجی انتگراتور برابر خواهد بود با :

$$V_{o1} = -V_a \times \frac{2^N}{\tau} T_{Cl;ock} + V_{ref} \times \frac{\Delta t}{\tau}$$

شمارنده تا زمانی که V_{01} مثبت است به شمارش خود ادامه می‌دهد. در زمان $t = t_2$ مقدار V_{01} صفر شده و مقایسه کننده، گیت را بسته و شمارش متوقف و داریم :

$$V_{o1} = 0 \Rightarrow V_a \times \frac{2^N}{\tau} T_{Cl;ock} = \frac{V_{ref}}{\tau}\Delta t$$

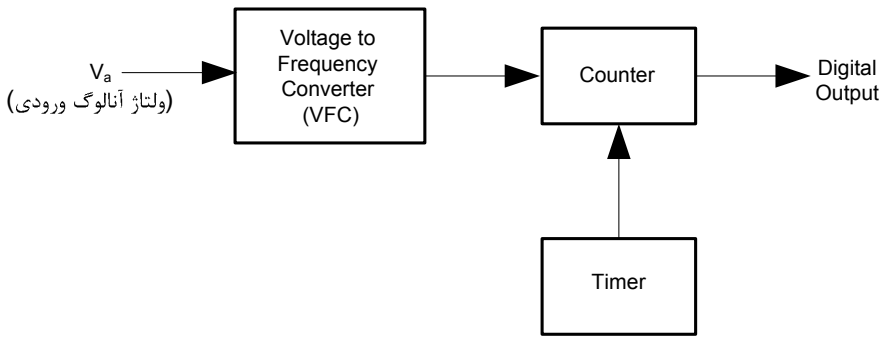
با فرض اینکه در فاصله زمانی t_1 تا t_2 (Δt) شمارنده تا عدد n شمارش کرده باشد می‌توان نوشت $\Delta t = n \times T_{Cl;ock}$ و لذا خواهیم داشت :

$$n = \frac{2^N}{V_{ref}} V_a$$

به علت دو انتگرالگیری متوالی از V_a و V_{ref} عدد حاصل (n) مستقل از ثابت زمانی و فرکانس کلاک بوده و تنها عامل مؤثر در دقت تبدیل، ضریب حرارتی ولتاژ مرجع است. چون طبقه انتگراتور ورودی عملاً یک فیلتر پایین گذر است لذا اگر نویز با فرکانس زیاد روی ولتاژ ورودی سوار شده باشد با عبور از این طبقه تا حد زیادی تضعیف شده و بر خلاف روشهای قبلی تأثیر خیلی کمی در مقدار تبدیل شده خواهد داشت. با انتخاب زمان انتگرالگیری موج ورودی به صورت مضربی از دوره تناوب فرکانس مزاحم می‌توان اثر فرکانس مزاحم (مثلاً برق شهر) را بطور کلی حذف کرد (چرا که انتگرال یک موج سینوسی در یک دوره تناوب کامل صفر است.)

۲-۳-۶- مبدل A/D با روش تبدیل ولتاژ به فرکانس :

در این روش، ولتاژ آنالوگ ورودی توسط یک مبدل ولتاژ به فرکانس خیلی دقیق (VFC) به پالسهایی که فرکانس آنها متناسب با ولتاژ ورودی است تبدیل می‌شود. پالسهای حاصل توسط یک شمارنده در یک فاصله زمانی مشخص شمرده می‌شوند. بدین ترتیب عملاً از ولتاژ ورودی در مدت زمان شمارش انتگرال گرفته می‌شود و در نتیجه مانند مبدل دو شیبی قابلیت حذف نویز را دارد. البته سرعت آن از مبدل دو شیبی کمتر است.

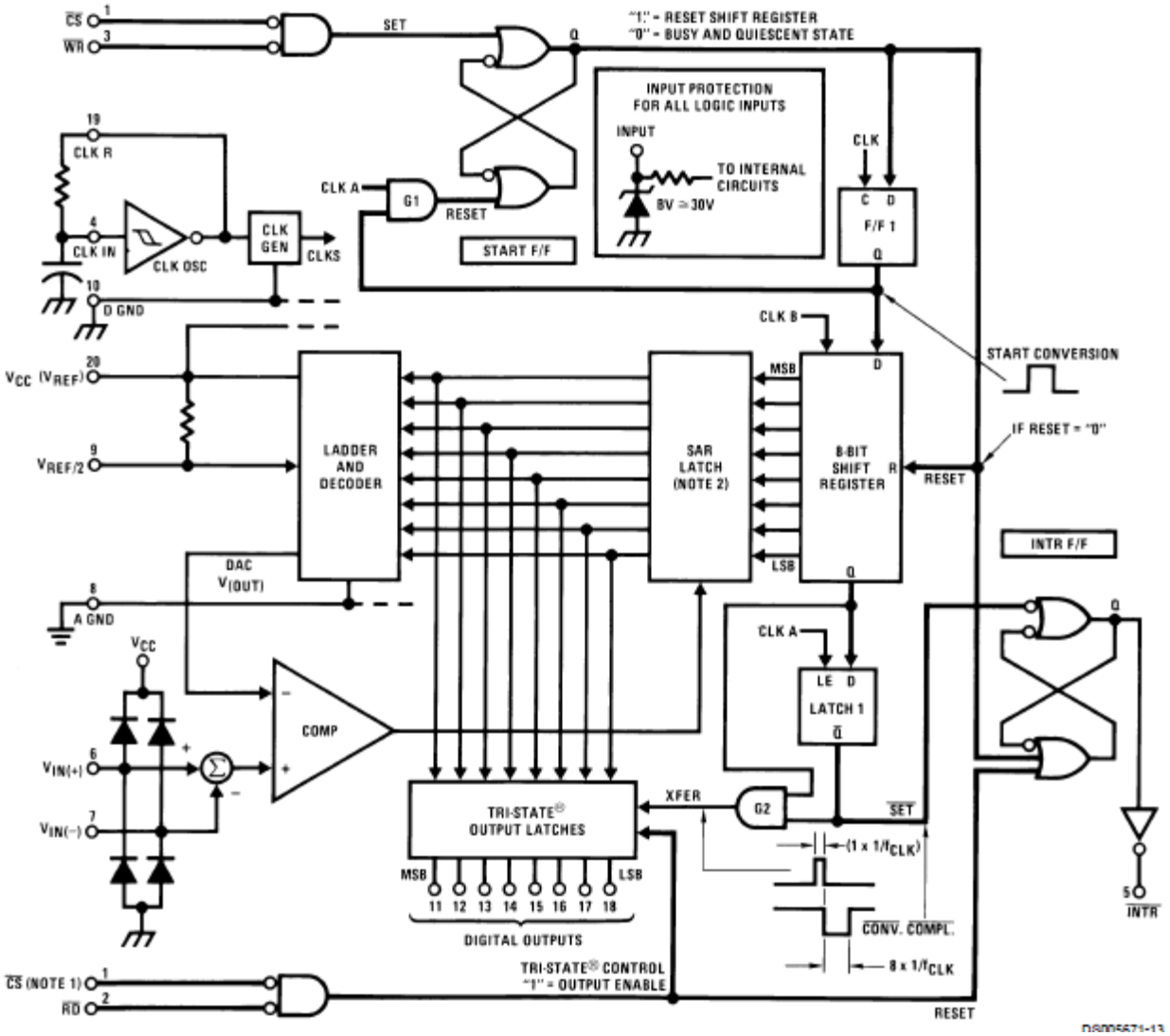


درستی و دقت مبدل بستگی به میزان خطی بودن و پایداری VFC دارد.
برخی کارخانجات سازنده مدارات مجتمع، مدارهایی جهت تبدیل ولتاژ به فرکانس ساخته اند که از آن جمله می‌توان آی‌سی LM331 را نام برد که میزان غیر خطی بودن این مبدل در محدوده 1HZ تا 100 KHZ فقط ۰/۰۱ درصد است. یا آی‌سی PLL با شماره CD4046 که دارای یک VCO است که میزان غیر خطی بودن آن در محدوده 1HZ تا 400 KHZ فقط یک درصد است.

۲-۴- معرفی برخی آی‌سی های A/D:

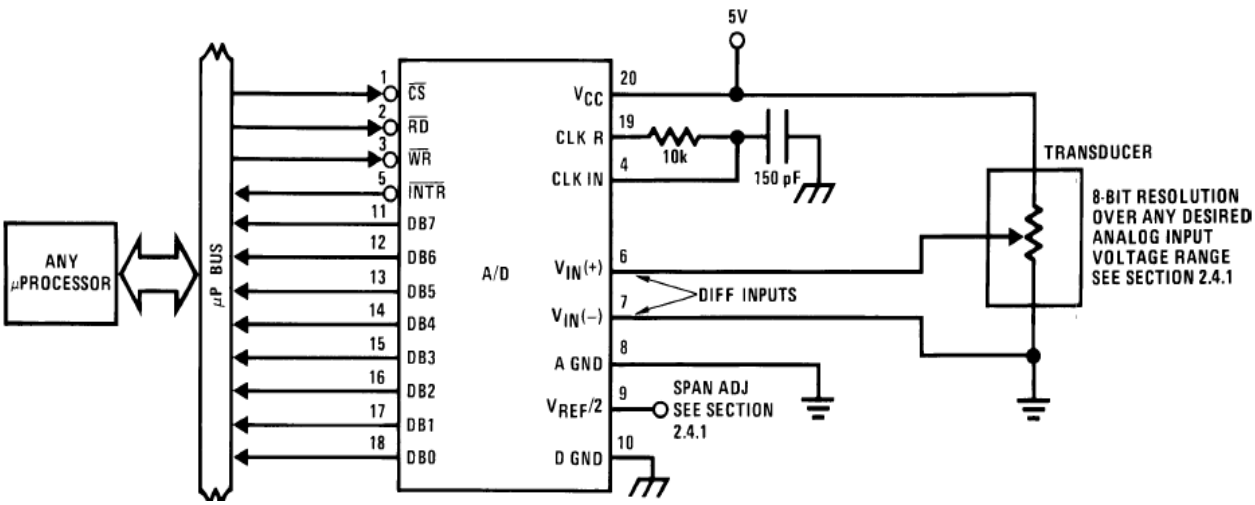
۲-۴-۱- آی سی ADC0801 :

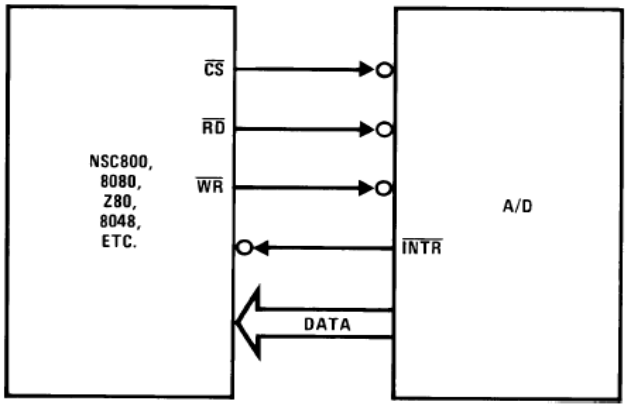
این مبدل یکی از آی‌سی های پر استفاده آنالوگ به دیجیتال ۸ بیتی قابل انطباق با میکروپروسسور است. از مزایای این آی‌سی نیاز به تنها یک منبع تغذیه ۵ ولت و داشتن نوسانساز داخلی است. همچنین خروجی سه حالته آن امکان اتصال مستقیم آن به دیتا باس میکروپروسسور را فراهم می‌کند. شکل زیر ساختمان داخلی آی‌سی را نشان می‌دهد.



ورودی آنالوگ به صورت تفاضلی بین پایه های (+) V_{in} و (-) V_{in} متصل می‌شود. پس از فعال شدن CS و پس از آن WR عملیات تبدیل در داخل آی‌سی شروع می‌شود و پس از زمان تقریبی 120µs (این زمان به فرکانس اسیلاتور داخلی بستگی دارد که در این آی سی بصورت $f = 1/1.1R_C$ است) مقدار دیجیتال معادل در لچ خروجی قرار می‌گیرد و همزمان پایه INTR از آی‌سی به عنوان پایان عملیات تبدیل فعال می‌شود. برای خواندن محتوای لچ باید CS و RD فعال شوند. در

ضمن خواندن پایه INTR نیز غیر فعال خواهد شد. V_{cc} به عنوان ولتاژ مرجع در این آ ی سی استفاده می‌شود. در صورتی که پایه $\frac{V_{ref}}{2}$ آ ی سی آزاد باشد، مقدار ولتاژ روی آن $\frac{V_{CC}}{2}$ است. از پایه $\frac{V_{ref}}{2}$ می‌توان به عنوان تنظیم انحراف حداکثر (FS) نیز استفاده کرد. از(-) V_{in} برای تنظیم صفر و از $\frac{V_{ref}}{2}$ برای تنظیم FS استفاده می‌شود. به عنوان مثال اگر بخواهیم ورودی ۰/۵ تا ۳/۵ ولت را توسط این آی‌سی به دیجیتال تبدیل کنیم باید حداقل ورودی یعنی ۰/۵ ولت را به (-) Vin وصل کنیم ((-) Vin به ولتاژ ۰/۵ ولت به عنوان حداقل ولتاژ ورودی وصل شود) و از آنجایی که رنج تغییرات ورودی ۳ ولت است باید پایه $\frac{V_{ref}}{2}$ را به $\frac{3}{2}=1.5V$ ولت وصل کنیم. تحت این شرایط ورودی 0.5 ولت مثل صفر و 3.5 ولت به عنوان FS تلقی می‌شود. شکل زیر یک مدار نمونه از کاربرد این آی‌سی و ارتباط آن با یک میکرو پروسسورمثل Z80 یا 8080 را نشان میدهد.





۲-۴-۲- آی سی AD7569 :

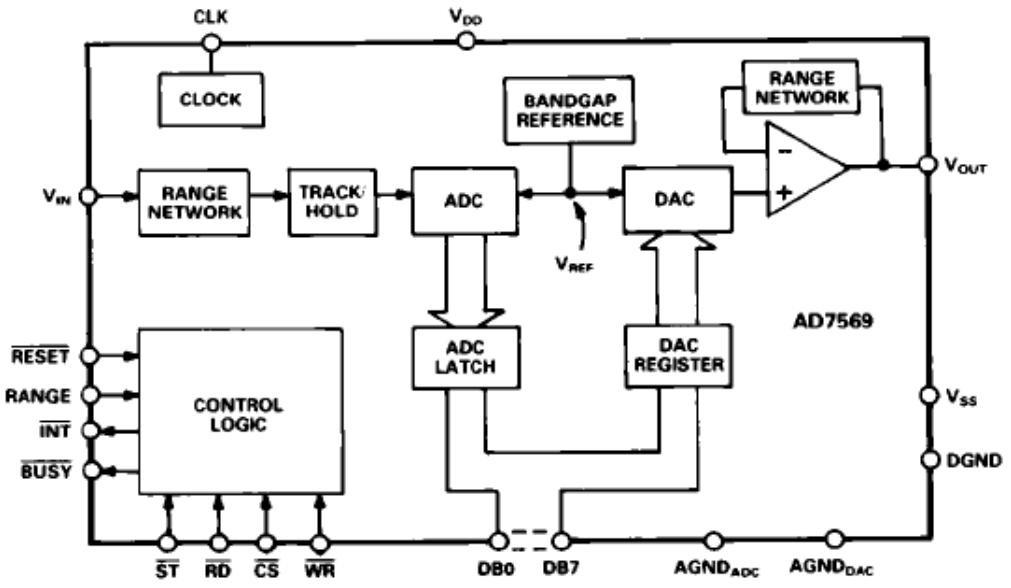
این آی‌سی شامل D/A و A/D هشت بیتی است و به همین خاطر یک مدار کامل I/O است. با استفاده از دو پایه Range و V_{ss} می‌توان چهار رنج متفاوت ورودی و خروجی را مطابق جدول زیر انتخاب کرد.

Table I. Input/Output Ranges

Range	V_{SS}	Input/Output Voltage Range	DB0–DB7 Data Format
0	0 V	0 V to +1.25 V	Binary
1	0 V	0 V to +2.5 V	Binary
0	–5 V	±1.25 V	2s Complement
1	–5 V	±2.5 V	2s Complement

بلوک دیاگرام این آی‌سی در شکل زیر آمده است. پس از فعال شدن CS، توسط لبه بالا رونده WR محتوای بالس در لچ داخلی ذخیره شده و پس از مدت زمان کوتاهی ولتاژ آنالوگ در خروجی D/A ظاهر می‌شود.

AD7569 FUNCTIONAL BLOCK DIAGRAM



در این آی‌سی به دو روش می‌توان A/D را فعال کرد که در برگه اطلاعات آی‌سی با عنوان مد ۱ و مد ۲ از آن یاد شده است. در مد یک، از ST (start conversion) برای شروع عمل تبدیل آنالوگ به دیجیتال استفاده می‌شود و پس از انجام عمل تبدیل، پایه INTR فعال شده که توسط فعال کردن RD می‌توان به محتوای A/D دسترسی پیدا کرد. در مد دو، پایهST همواره غیر فعال بوده و شروع عمل تبدیل توسط RD, CS آغاز می‌شود. در این مرحله با فعال شدن BUSY عمل

تبدیل شروع شده و پس از پایان عمل تبدیل (غیر فعال شدن BUSY) پایه INTR فعال شده و می‌توان داده متناظر با مقدار آنالوگ ورودی را خواند. زمانبندی مربوط به مدهای کاری این آی سی را در زیر مشاهده می کنید.

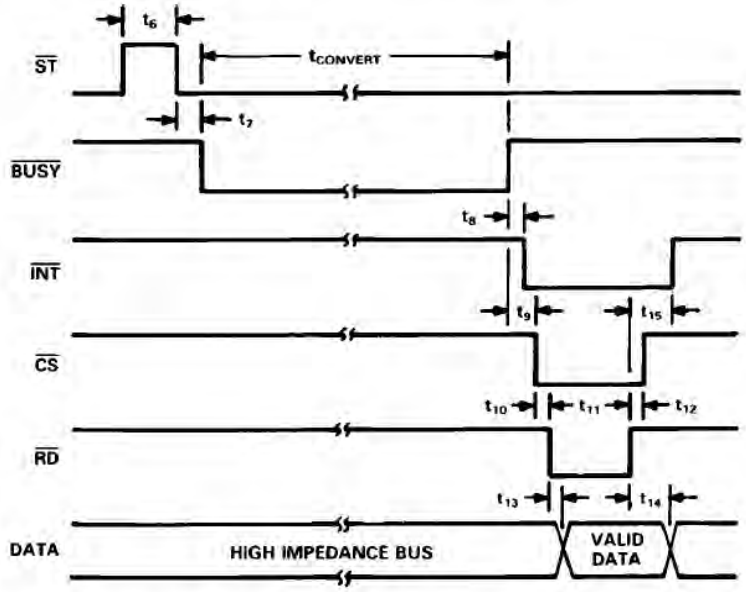


Figure 10. ADC Mode 1 Interface Timing

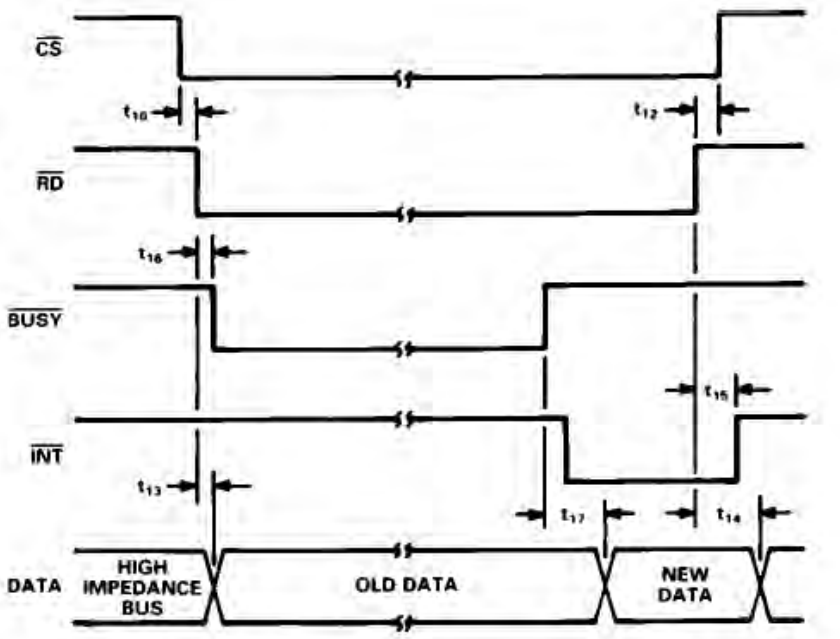


Figure 12. ADC Mode 2 Interface Timing

جزئیات مربوط یه مد های کاری A/D رامی‌توان در برگه اطلاعات آی‌سی مطالعه کرد.

۳-۱- انواع حافظه‌ها

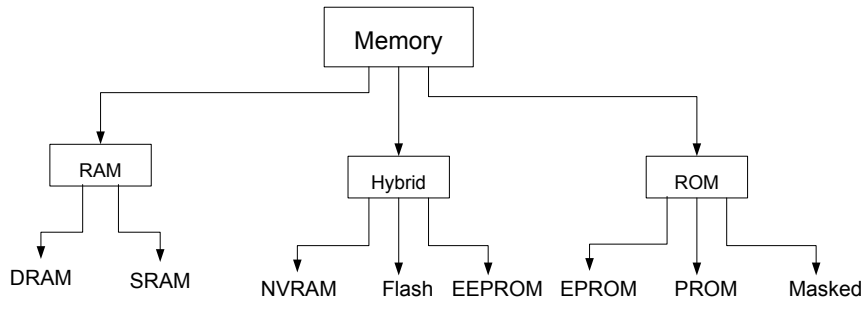
حافظه با هدف ذخیره سازی اطلاعات (بصورت دائم یا موقت) استفاده می‌شود. استفاده از حافظه صرفاً محدود به کامپیوترهای شخصی نبوده و در دستگاههای متفاوتی نظیر تلفنهآی‌سلار، دوربینهای دیجیتالی، تلویزیون و.. نیز در ابعاد وسیعی از آنها استفاده می‌شود. هر یک از دستگاههای فوق مد لهای متفاوتی از حافظه را استفاده می‌کنند.

تقسیم بندیهای متفاوتی برای حافظه‌های موجود در یک سیستم کامپیوتری ارائه شده است.

در یک تقسیم بندی، حافظه‌ها به دو گروه پاک شدنی [۹] و غیر پاک شدنی [۱۰] تقسیم می‌شوند. حافظه‌های پاک شدنی، بلافاصله پس از خاموش شدن سیستم اطلاعات خود را از دست می‌دهند و همواره برای نگهداری اطلاعات خود به منبع تأمین انرژی نیاز دارند. حافظه‌های RAM در این گروه قرار دارند. حافظه‌های غیر پاک شدنی، داده‌های خود را همچنان پس از خاموش شدن سیستم نیز حفظ می‌کنند. حافظه ROM نمونه‌ای از این نوع حافظه‌ها است.

در یک تقسیم بندی دیگر، حافظه‌ها از دیدگاه نوع کاربردشان در یک سیستم کامپیوتری به سه گروه تقسیم می‌شوند. *حافظه داخلی پروسسور* که شامل یکسری رجیسترهای داخلی پروسسور بوده که دارای سرعت زیادی (زمان دسترسی کوتاهی) بوده و برای ذخیره موقت دستورالعملها و داده‌ها بکار می روند. *حافظه اصلی* که نسبتاً سریع بوده و برای ذخیره برنامه و داده‌ها در طی کار کامپیوتر بکار می‌رود و حافظه‌های جانبی کامپیوتر که معمولاً دارای حجم زیاد و زمان دسترسی به آن (نسبت به حافظه اصلی) زیاد است و برای ذخیره برنامه سیستم و داده‌های زیادی که بطور مستمر مورد نیاز CPU نیستند بکار می‌رود.

در یک دسته بندی دیگر حافظه‌هایی که در سیستمهای الکترونیکی استفاده می‌شوند به دو نوع حافظه‌های مغناطیسی (مثل فلاپی دیسکها و دیسکهای سخت) و نیمه‌هادی تقسیم می‌شوند. در اینجا هدف ما بررسی حافظه‌های نیمه‌هادی است. حافظه‌های نیمه‌هادی که بر خلاف حافظه‌های مغناطیسی فاقد اجزای متحرک و مکانیکی هستند از آرایه‌هایی از سلولهای حافظه تشکیل شده‌اند که این آرایه‌ها بسته به نوع حافظه از تعدادی عنصر الکترونیکی مثل ترانزیستور و خازن تشکیل شده‌اند. این نوع حافظه‌ها به سه دسته‌ی کلی به نام های ROM, RAM و Hybrid تقسیم می‌شوند که نوع Hybrid ترکیبی از دو نوع اول می‌باشد. این تقسیم بندی حافظه‌ها در شکل ۳-۱ نشان داده شده است.



شکل ۳-۱- تقسیم بندی حافظه‌ها

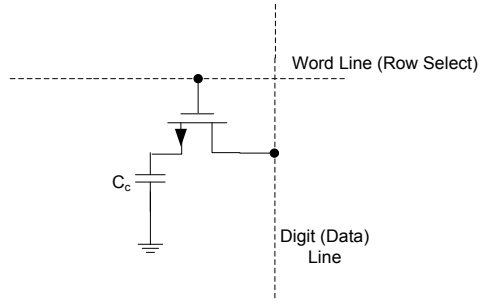
9 -Volatile
10 -Nonvolatile

۳-۱-۱- حافظه‌های RAM

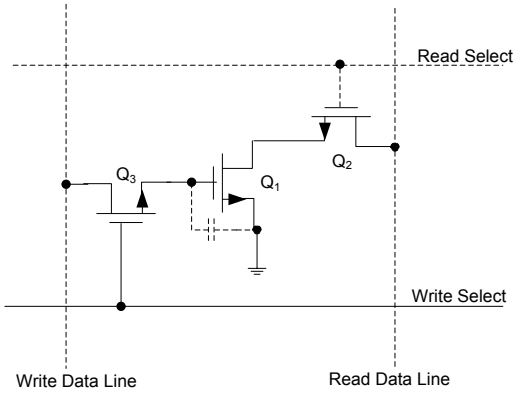
RAM (Random Access Memory) از تعدادی خانه‌یا سلول تشکیل شده است و هر خانه قابلیت نگهداری یک داده را دارد و با آدرسی منحصر به فرد مشخص می‌شود. استفاده از حافظه RAM، برای نگهداری موقت اطلاعات تا زمان پردازش یا انتقال نتایج به بیرون از کامپیوتر و یا ذخیره در حافظه جانبی است. داده‌های موجود در RAM قابل پاک شدن و جایگزینی با داده‌های دیگر هستند. مهمترین ویژگی RAM ها ناپایدار بودن اطلاعات موجود در آنهاست یعنی تا زمانی که تغذیه به آنها وصل باشد اطلاعات نگهداری می‌شوند و قطع تغذیه موجب از بین رفتن داده‌های موجود در RAM می‌شود. از آنجا که داده‌ها می‌توانند در هر قسمت از حافظه RAM ذخیره شده و از آن قسمت بازیابی شوند و سرعت انجام این کار به محل داده‌ها بستگی ندارد به این نوع حافظه‌ها، حافظه با دسترسی تصادفی می گویند. به RAM حافظه‌ی خواندنی و نوشتنی (Read Write Memory=RWM) نیز گفته می‌شود. *برنامه سیستم عامل، برنامه‌های کاربردی و داده‌های مورد نیاز CPU (برای انجام عملیات جاری) اطلاعاتی هستند که در حافظه RAM نگهداری می‌شوند.*

RAM ها به دو نوع DRAM (RAM پویا) و SRAM (RAM ایستا) تقسیم می‌شوند که از لحاظ الکترونیکی ، تفاوت آنها در اجزای سازنده‌ی آنهاست (شکلهای ۳-۲ را ببینید).

DRAM مخفف Dynamic RAM می‌باشد که دلیل این نام استفاده از خازن در ساختمان این نوع حافظه است. در این نوع حافظه‌ها برای سلول‌های حافظه از یک زوج ترانزیستورو خازن استفاده می‌گردد . به علت وجود خازن، جهت حفظ اطلاعات در DRAM، باید اطلاعات موجود در سلولهای حافظه ، نوسازی (Refresh) شوند تا خازنها شارژ شوند (شکل ۳-۲- الف و ب).



شکل ۳-۲ - الف- یک سلول DRAM یک ترانزیستوری

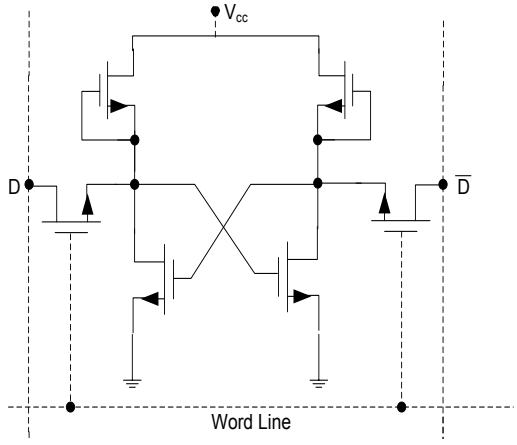


شکل ۳-۲ - ب- یک سلول DRAM سه ترانزیستوری

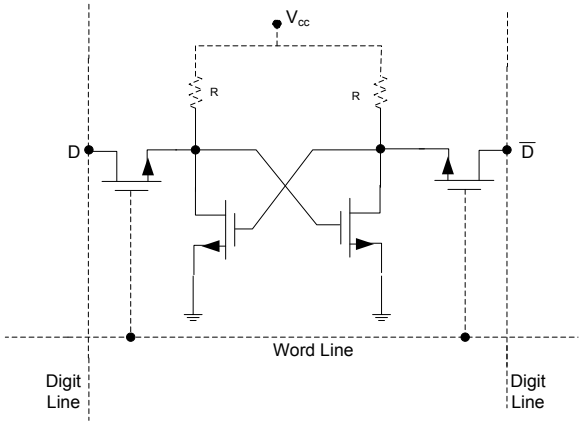
در شکل ۳-۲-ب ظرفیت خازنی گیت ترانزیستور Q_1 به عنوان خازن ذخیره کننده بار استفاده شده است و با این کار نیاز به خازن اضافی نیست که این به نوبه خود باعث افزایش چگالی سطحی حافظه می‌شود. در شکل ۳-۲-پ نیز ساختار یک آرایه DRAM با ظرفیت m*n سلول (یک ترانزیستوری) را مشاهده می‌کنید.

عمل نوسازی حافظه DRAM مشابه عمل خواندن است (یک خواندن غیرضروری) با این تفاوت که در این حالت دیکودرهای ستون غیر فعال هستند. ردیفهای حافظه (Row Select) را به ترتیب فعال کرده، توسط حسگرهایی اطلاعات آنها را خوانده و دوباره (تقویت شده) همان اطلاعات را را به آن ردیف اعمال می‌کنیم.

حافظه‌های SRAM یا Static RAM از اجزایی به نام فلیپ فلاپ تشکیل شده‌اند و برای حفظ اطلاعات فقط نیاز به تغذیه دارند. حافظه‌های SRAM از چندین ترانزیستور (چهار تا شش) برای هر سلول حافظه استفاده می‌کنند (شکل ۳-۲-ت و ث).



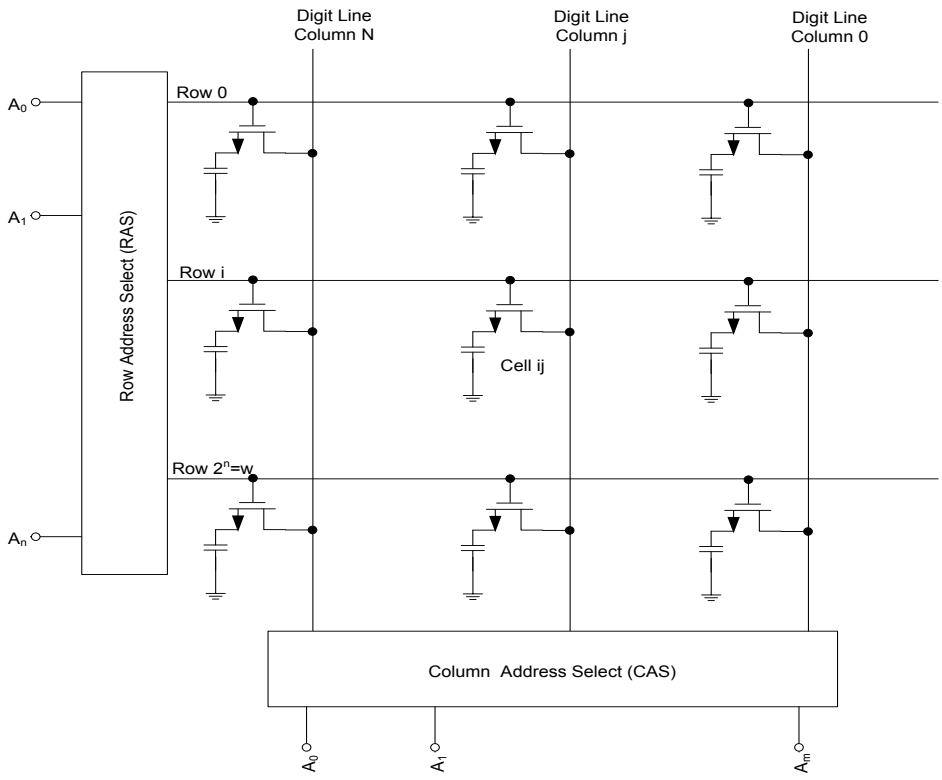
شکل ۳-۲- ت- یک سلول RAM استاتیک با بار فعال



شکل ۳-۲ - ث- یک سلول RAM استاتیک با بار مقاومتی

واضح است که از لحاظ مداری حافظه‌های DRAM از پیچیدگی کمتری در مقایسه با SRAM ها برخوردارند زیرا هر فلیپ فلاپ خود از چندین ترانزیستور تشکیل شده است. از طرف دیگر حافظه‌ی DRAM به دلیل وجود خازن، بر خلاف نوع دیگر توانایی نگهداری اطلاعات را در غیاب تغذیه در حدود چند میلی ثانیه دارا می‌باشد. اما مهترین ویژگی SRAM در مقایسه با DRAM سرعت بالاتر (حدود ۴ برابر) آن است زیرا DRAM در مدت نوسازی قادر به نوشتن یا خواندن اطلاعات نیست. به دلیل ساده‌تر بودن ساختمان DRAM ها، قیمت آنها نسبت به SRAMها پایین‌تر بوده و به همین خاطر تمام حافظه‌های RAM موجود در کامپیوتر از نوع DRAM هستند. از SRAM ها به خاطر سرعت بالاترشان معمولاً در Cache (حافظه پنهان) پردازنده‌ها استفاده می‌شود. محل حافظه پنهان در بعضی کامپیوترها درون CPU و در بعضی دیگر روی برد اصلی است. داده‌ها ابتدا از RAM وارد حافظه پنهان شده و سپس در اختیار CPU قرار می‌گیرند. دلیل این کار این است که در اکثر مواقع، داده‌های مورد نیاز CPU تکراری هستند و اگر این داده‌ها درون حافظه پنهان باشند، با سرعت بیشتری نسبت به RAM در اختیار CPU قرار می‌گیرند. بنابراین هرگاه CPU به داده‌هایی نیاز داشته باشد، ابتدا حافظه‌ی پنهان مورد بررسی قرار می‌گیرد و اگر داده‌ها در آنجا وجود نداشتند، از RAM وارد Cashe شده و در اختیار CPU قرار می‌گیرند. در مواقعی که CPU با دستگاههای کندتر- مثل اغلب دستگاههای ورودی و خروجی- کار می‌کند، از بافرهایی استفاده می‌کند تا داده‌های خود را در این دستگاهها بریزد و معطل این دستگاهها نشود. مثلاً وقتی که دستور چاپ به چاپگر ارسال می‌شود، داده‌ها در بافر چاپگر ریخته می‌شود تا در فرصت مناسب چاپ شود و CPU وقت خود را برای عمل چاپ از دست ندهد. آی‌سی2114 با ظرفیت 1k*4 bit، آی‌سی6116 با ظرفیت 2k*8 bit، آی‌سی6264 با ظرفیت 8k*8 bit نمونه هایی از آی‌سی های RAM هستند.

شکل ۳-۲ - پ- یک آرایه DRAM با m*n سلول حافظه را نشان می دهد.



شکل ۳-۲ - پ- یک آرایه DRAM با m*n سلول

به چه میزان حافظه RAM نیاز است؟

حافظه RAM یکی از مهمترین فاکتورهای موجود در زمینه ارتقاء کارآیی یک کامپیوتر است. افزایش حافظه بر روی یک کامپیوتر با توجه به نوع استفاده می‌تواند در مقاطع زمانی متفاوتی انجام گیرد. برای سیستم عامل لینوکس صرفا " به ۴ مگابایت حافظه ، سیستم عامل ویندوز 3.1 حداقل به ۱۲ مگابایت، ویندوز ۹۵ و یا ۹۸ حداقل به ۳۲ مگابایت، سیستم عامل ویندوز ۲۰۰۰ حداقل به ۶۴ مگابایت حافظه نیاز است. که در همه‌ی موارد فوق استفاده از دوبرابر حداقل حافظه مورد نیاز توصیه می‌شود. میزان حافظه اشاره شده برای هر یک از سیستم های فوق بر اساس کاربردهای معمولی ارائه شده است. دستیابی به اینترنت، استفاده از برنامه‌های کاربردی خاص و سرگرم کننده، نرم‌افزارهای خاص طراحی، انیمیشن سه بعدی و... مستلزم استفاده از حافظه بمراتب بیشتری خواهد بود.

۳-۱-۲- حافظه‌های ROM

ROM (Read Only Memory) حافظه‌ی فقط خواندنی است. ROM ها براساس روش نوشتن اطلاعات جدید و تعداد بازنویسی، تقسیم بندی می‌شوند. اصولا ROM ها از آرایه‌ای از ترانزیستورها تشکیل شده‌اند که هر کدام از سلولها دارای یک فیوز ذوب شدنی است که در زمان پروگرام شدن در صورتی که نیاز به وجود صفر منطقی

باشد فیوز آن سلول ذوب می‌شود و در غیر اینصورت آن خانه حاوی یک منطقی می‌باشد. بنابراین اطلاعات موجود در ROM ها غیر فرار بوده و در غیاب تغذیه نیز حفظ می‌شوند و معمولاً برای نگهداری کد نرم‌افزارها در سیستمهای میکروپروسسوری استفاده می‌شوند.

ROM ها به سه دسته تقسیم می‌شوند که‌یک نوع آن ROM پوششی یا Masked ROM است که معمولاً توسط کارخانه سازنده برنامه‌ریزی می‌شود. این نوع ROM ها پس از نوشتن قابل پاک شدن نیستند و معمولاً در تیراژ تولیدی بالا بسیار ارزان قیمت هستند. یک مرحله بالاتر از ROM های پوششی، PROM ها یا ROM های قابل برنامه‌ریزی می‌باشند که بوسیله دستگاهی به نام پروگرامر اطلاعات مورد نیاز درون آنها قرار می‌گیرد. PROM ها فقط یک بار برنامه‌ریزی می‌شوند. پس از PROM ها، EPROM ها قرار دارند که همانند PROM ها قابل برنامه‌ریزی هستند اما اطلاعات موجود در آنها قابل پاک شدن است. پاک کردن اطلاعات یا Reset کردن EPROM بوسیله‌ی اشعه‌ی فرابنفش انجام می‌شود بدین صورت که تراشه سیلیکونی بوسیله‌ی پنجره‌ای که روی Package آن قرار دارد در معرض اشعه ماورای بنفش قرار داده می‌شود و اطلاعات موجود در آن پاک می‌شود. البته در تعداد دفعات پاک شدن این حافظه‌ها محدودیت وجود دارد که برای اطلاع از این تعداد باید به برگه اطلاعاتی[11] آن مراجعه کرد. برای پاک کردن باید اشعه فرا بنفش را در طول موج ۲۵۳۷ انگستروم و با شدت یکنواخت 12000uv/cm2 و در مدت ۵ تا ۱۵ دقیقه باشد .

یک دستگاه پاک کننده EPROM، یک تولید کننده امواج فرا بنفش است که دارای یک فضای بسته است که دارای یک کشو می‌باشد و در بالای کشو، تولید کننده امواج فرا بنفش می باشد و چیپها درون کشو قرار می‌گیرند.

هر کامپیوتر معمولاً دارای مقدار کمی حافظه ROM است که برنامه BIOS سیستم در آن نگهداری می‌شود. با روشن کردن کامپیوتر، این برنامه از ROM به RAM منتقل می‌شود.

برخی از آی سی های EPROM عبارتند از:

2708 (1kbyte*8bit), 2732 (4kbyte*8 bit), 2764 (8kbyte*8 bit), 27128 (16kbyte*8 bit), 27256 (32kbyte*8 bit), 27512 (64kbyte*8 bit)

۳-۱-۳- حافظه‌های ترکیبی یا Hybrid

با پیشرفت تکنولوژی حافظه‌ها ، در سالهای اخیر، مرز بین ROM و RAM محو شده است. بدین صورت که حافظه‌هایی ساخته شده‌اند که از یک سو اطلاعات موجود در آنها در غیاب تغذیه حفظ می‌شود و از سوی دیگر بوسیله‌ی سیگنالهای الکتریکی قابل بازنویسی هستند. بنابراین از این حافظه‌ها به نام ترکیبی یا Hybrid یاد می‌شود. حافظه‌های ترکیبی به سه نوع EEPROM[12], Flash و NVRAM تقسیم می‌شوند که دوتای اولی از نسل ROM ها هستند و NVRAM نوع تغییر یافته‌ای از RAM است.

EEPROM همانند EPROM قابل برنامه‌ریزی مجدد است اما برای پاک شدن نیازی به اشعه‌ی ماورا بنفش ندارد و بصورت الکتریکی قابل پاک شدن است. در حافظه‌های EEPROM اولاً برای بازنویسی تراشه نیازی به

11 - Data Sheet

12 - Electrically Erasable and Programmable ROM

جدا نمودن تراشه از محل نصب شده نیست .ثانیاً :برای تغییر بخشی از تراشه نیاز به پاک نمودن تمام محتویات آن نمی‌باشد و ثالثاً اعمال تغییرات در این نوع تراشه‌ها مستلزم بکارگیری یک دستگاه اختصاصی نیست .

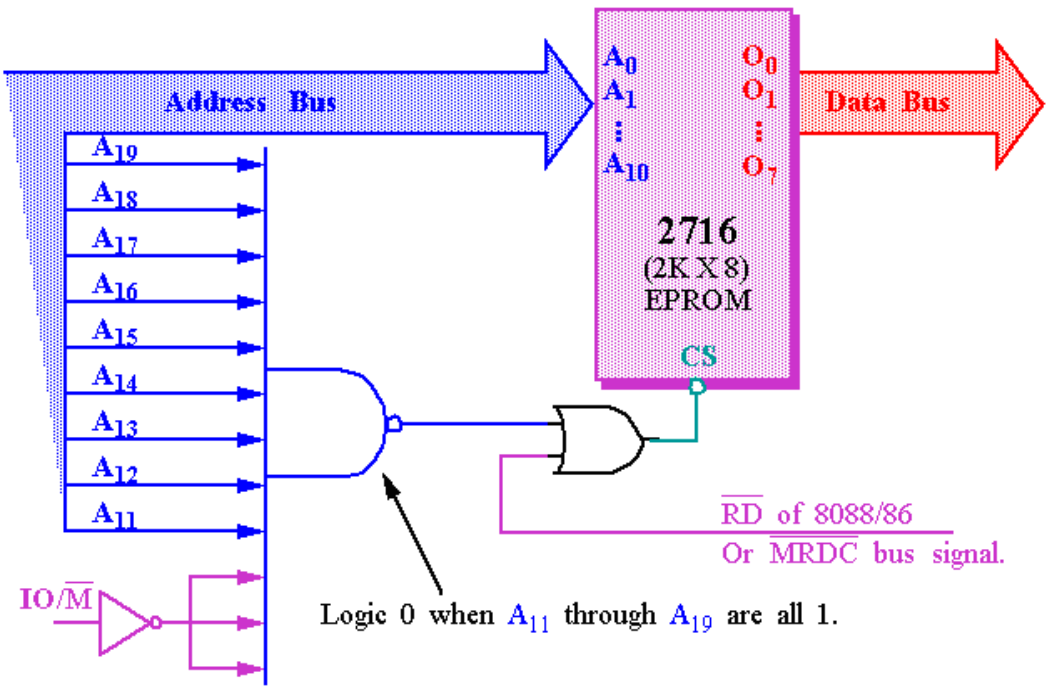
حافظه‌های Flash حاوی ترکیبی از بهترین مشخصات حافظه‌هایی که تاکنون بررسی شد هستند. این حافظه‌ها دارای چگالی بالا، قیمت پایین، غیر فرار، سرعت بالا (در خوندن) و نوشتن الکتریکی هستند از این رو این حافظه‌ها در بسیاری از موارد جایگزین حافظه‌های EEPROM شده‌اند. از دیدگاه نرم‌افزاری حافظه‌های Flash بسیار شبیه EEPROM ها هستند اما تفاوت آنها در این است که در حافظه‌های Flash در هر لحظه امکان پاک شدن یک سکتور وجود دارد نه‌یک بایت. در حالی که EEPROM ها امکان پاک شدن بایت به بایت را دارند. معمولاً سکتورها اندازه‌ای بین ۲۵۶ بایت تا ۱۶ کیلو بایت را دارند. با وجود این مزیت EEPROM ها نسبت به Flash، حافظه‌های Flash از محبوبیت بیشتری برخوردارند و همان طور که اشاره شد جایگزین بسیاری از ROM ها شده‌اند. شکل ۲ یک حافظه‌ی Flash را نشان می‌دهد که به عنوان Bios کامپیوتر استفاده شده است.

شکل ۲- حافظه‌ی Flash به عنوان Bios کامپیوتر

سومین نوع از حافظه‌های ترکیبی NVRAM ها هستند. این نوع حافظه مثل ROM ها غیر فرار هستند اما از لحاظ ساختمانی تفاوت زیادی با ROM ها دارند. در واقع این نوع حافظه‌یک نوع SRAM است که دارای یک باتری پشتیبان است. وظیفه این باتری این است که در زمان قطع تغذیه جریان لازم را برای حفظ اطلاعات در حافظه تأمین کند. این نوع حافظه‌ها بدلیل وجود باتری ، گران قیمت هستند و استفاده از آنها محدود است و برای حفظ مقدار کمی اطلاعات استفاده می‌شوند.

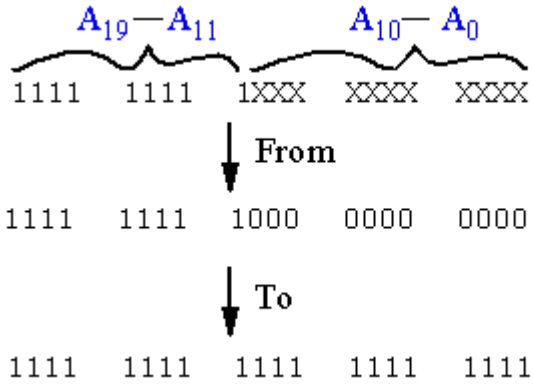
۳-۲- دیکود کردن آدرس حافظه (Memory Address Decoding)

اگر پروسسور دارای n خط آدرس باشد می‌تواند 2^n موقعیت حافظه را آدرس‌دهی کند. به عنوان مثال پروسسور ۸۰۸۸ با داشتن ۲۰ خط آدرس می‌تواند 2^{20} =۱۰۴۸۵۷۶ موقعیت حافظه (یک مگابایت حافظه) را آدرس‌دهی کند. ولی ممکن است یک حافظه به تنهایی، دارای این مقدار سلول نباشد. به عنوان مثال حافظه ۲۷۱۶، مثلاً به عنوان BIOS سیستم، یک حافظه EPROM دو کیلوبایتی است که فقط به ۱۱ خط آدرس نیاز دارد. در چنین مواردی به دیکود کردن آدرس حافظه نیاز است تا با استفاده از آن بتوان فضای حافظه مورد نیاز پروسسور (مثلاً یک مگابایت در مورد پروسسور ۸۰۸۸) را تأمین کرد و حافظه دو کیلوبایتی ۲۷۱۶ را بعنوان بخشی از فضای حافظه ۸۰۸۸ مورد استفاده قرار داد. به عبارت دیگر، یک پروسسور می‌تواند معمولاً فضایی بیش از فضای اشغال شده توسط یک حافظه را آدرس‌دهی کند و به همین خاطر نیاز به دیکود کردن آدرس حافظه داریم. به عنوان مثال شکل ۳ یک نوع دیکودینگ برای این حافظه را نشان می‌دهد. پروسسور (۸۰۸۸) زمانی به این حافظه دسترسی پیدا می‌کند که سیگنال خواندن از حافظه (RD) فعال بوده و خطوط آدرس A11 تا A19 همگی در منطق یک باشند.



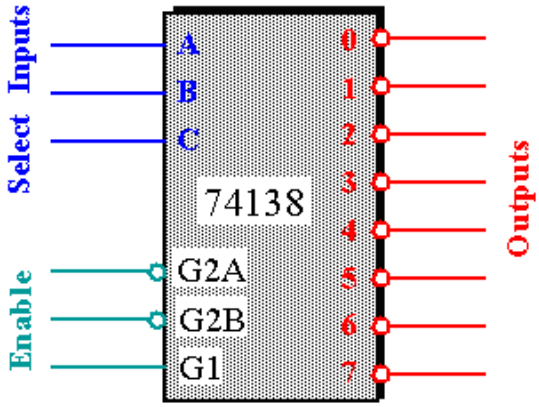
شکل ۳- یک نوع دیکودینگ حافظه با استفاده از گیتهای منطقی

برای آنکه رنج مربوط به فضایی را که این حافظه در سیستم اشغال کرده است (این حافظه به آن فضای آدرس نگاشت شده است) را بدست آوریم به صورت زیر عمل می‌کنیم.



چنانکه ملاحظه می‌شود این قطعه دو کیلوبایتی حافظه، به محدوده آدرس FF800H تا FFFFFH نگاشته می‌شود که این فضا متناظر به موقعیت Reset پروسسور۸۰۸۸ و ۸۰۸۶ است.

معمولاً برای دیکودکردن حافظه از گیتهای NAND استفاده نمی‌شود بلکه استفاده از دیکودرهایی نظیر 74138 معمولتر است (74138 یک دیکودر ۳ به ۸ است). شکل ۴، آی سی74138 و جدول عملکرد آنرا نشان می‌دهد.

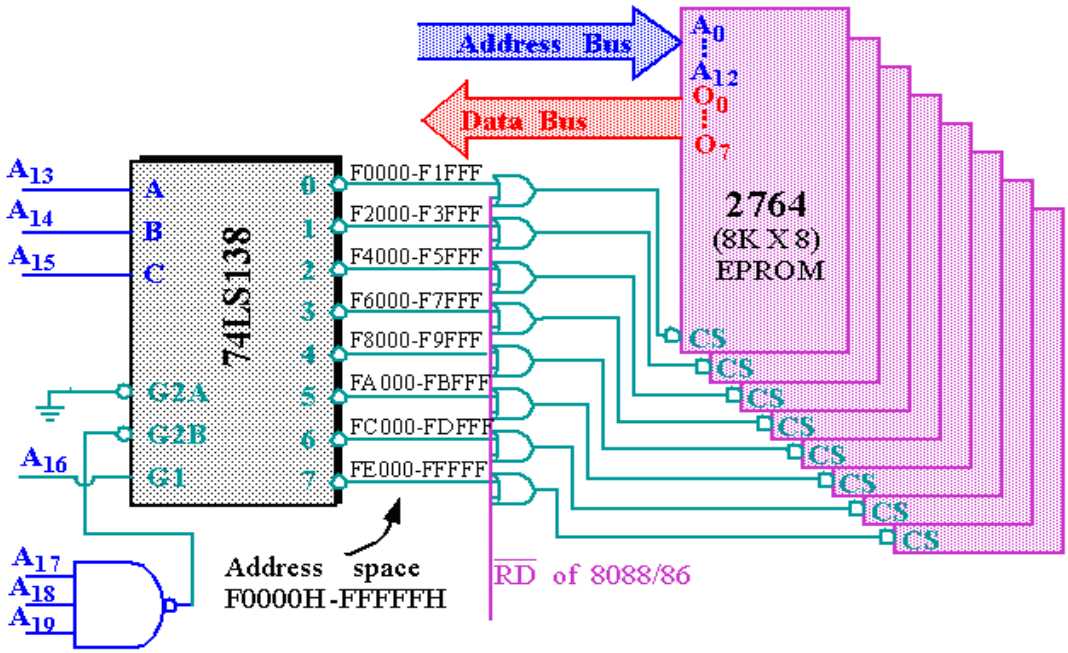


Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

شکل ۴- شکل و جدول عملکرد 74138

شکل ۵ استفاده از آی‌سی 74138، برای دیکودکردن ۶۴ کیلوبایت از فضای حافظه (۸ قطعه حافظه ۸ کیلوبایتی از نوع EPROM 2764) را نشان می‌دهد. هر کدام از خروجیهای آی‌سی دسترسی به‌یکی از قطعات ۸ کیلوبایتی حافظه را امکان پذیر می‌سازد. چنانکه شکل نشان می‌دهد برای دسترسی به هر کدام از این قطعات حافظه نیاز است که بیتهای آدرس A16 تا A19 همگی در سطح منطقی یک باشند. بسته به حالتهای مختلف بیتهای آدرس A13 تا A15، دسترسی به یکی از قطعات ۶۴ کیلوبایتی امکان پذیر خواهد شد. به عنوان مثال اگر بیتهای آدرس A13 تا A15 هر سه در منطق صفر باشند دسترسی به قطعه اول، که فضای آدرس F0000H

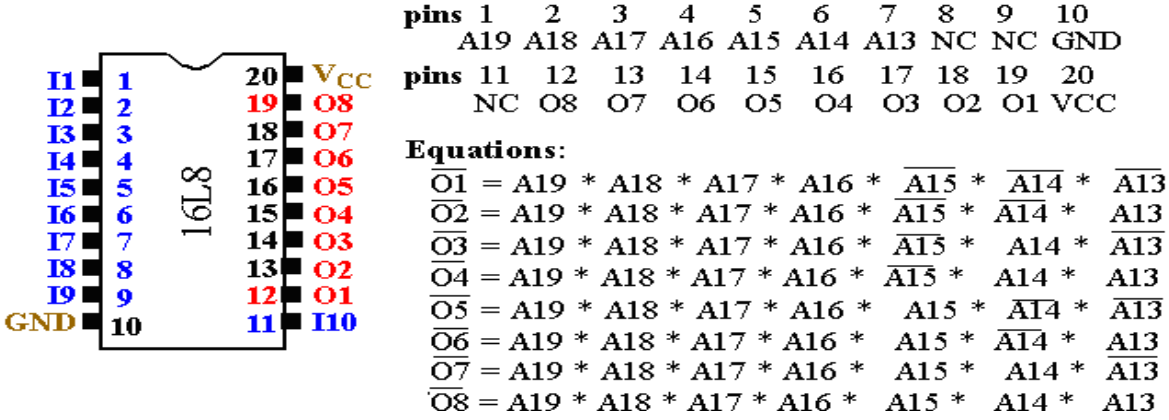
تا F1FFFH حافظه است، و اگر هر سه بیت در منطق یک باشند دسترسی به قطعه آخر، که فضای آدرس FE000H تا FFFFFH را تشکیل می‌دهد، امکانپذیر می‌شود.



شکل ۵- دیکودینگ قطعات ۸ کیلوبایتی با استفاده از 74138

در پروسسورهای پیشرفته امروزی، برای دیکود کردن فضای حافظه از آی‌سی‌های PLD[۱۳] استفاده می‌شود. سه گونه متفاوت از این آی‌سی‌ها شامل PLA[۱۴]، PAL[۱۵] و GAL[۱۶] است. PLA و PAL همانند PROMها از نوع Fuse-Programmed بوده و فقط یکبار قابل برنامه‌ریزی هستند ولی GAL ها همانند EPROMها قابل پاک شدن و برنامه‌ریزی مجدد می‌باشند. شکل ۶ استفاده از PAL به شماره 16L8 (که برای دیکودکردن آدرس ۳۲ بیتی پروسسور 80386DX و بالاتر از آن معمول است) برای دیکودکردن آدرس را نشان می‌دهد.

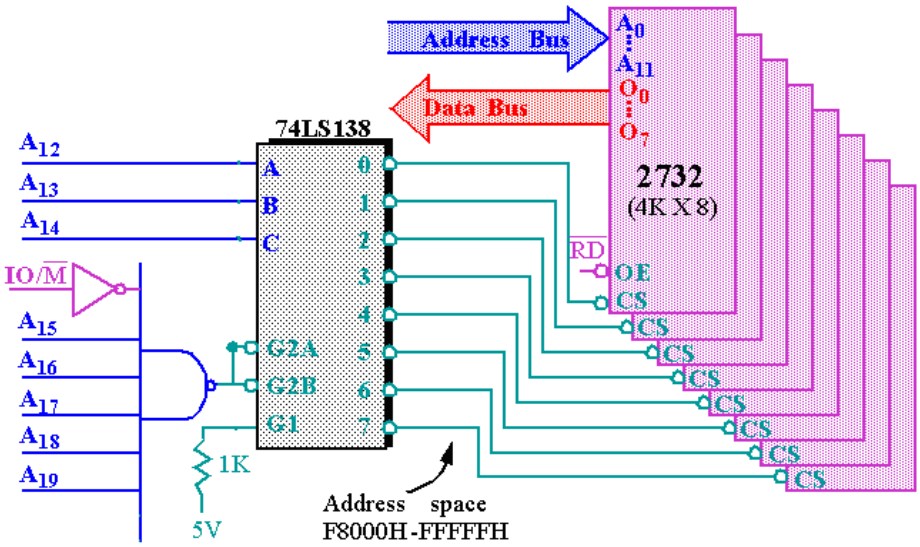
[13] - Programmable Logic Device
[14] - Programmable Logic Array
[15] - Programmable Array Logic
[16] - Gated Array Logic



شکل۶- استفاده از PAL برای دیکود کردن حافظه‌های سیستم

این آی‌سی دارای ۱۰ ورودی ثابت (پینهای ۱ تا ۹ و پین ۱۱)، دو خروجی ثابت (پینهای ۱۲ و ۱۹) و ۶ پین به عنوان ورودی یا خروجی (پینهای ۱۳ تا ۱۸) است. این آی‌سی طوری برنامه‌ریزی شده است که خطوط آدرس A13 تا A19 را به ۸ خروجی (متناظر به ۸ قطعه حافظه) نگاشت دهد.

پروسسور ۸۰۸۸ دارای ۲۰ خط آدرس(A19~A0)، هشت خط داده(AD7~AD0) و ۳ سیگنال کنترلی IO/M, RD و WR است. در شکل ۷، اتصال پروسسور ۸۰۸۸ با ۳۲ کیلوبایت حافظه EPROM (که فضای آدرس F8000H تا FFFFFH را تشکیل می‌دهند) با استفاده از آی‌سی 74138 و ۸ قطعه EPROM چهارکیلوبایتی 2732 شان داده شده است.



شکل ۷- دیکودینگ قطعات ۴ کیلوبایتی در فضای آدرس F8000H تا FFFFFH

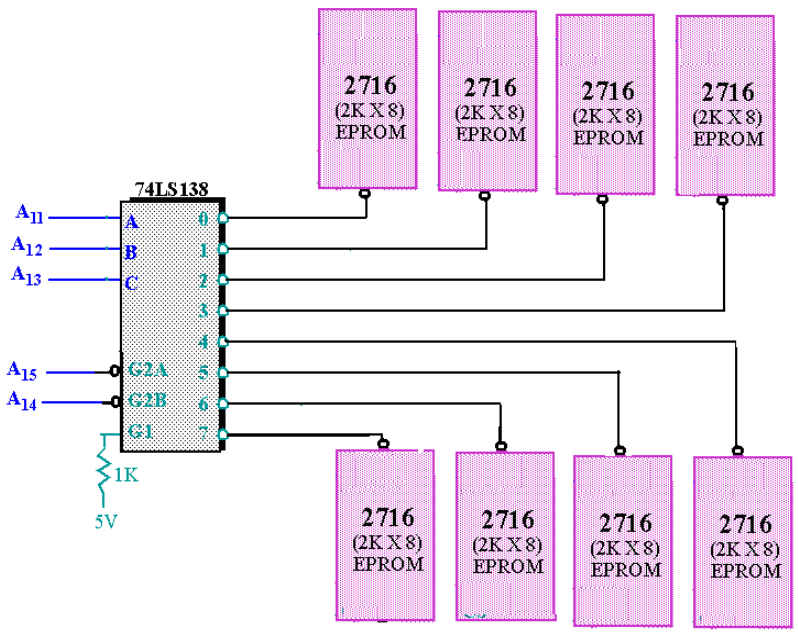
به عنوان مثالی دیگر، مدار شکل ۸ دیکودینگ حافظه‌های 2716 را نشان می‌دهد. به عنوان یک مثال می‌خواهیم محدوده‌ای را که این حافظه‌ها اشغال می‌کنند بدست آوریم. چنانکه شکل ۸ نشان می‌دهد، برای

آنکه دیکودر فعال شود باید بیتهای A14 و A15 هر دو صفر باشند. بیتهای آدرس A0 تا A10 می‌توانند از 000H تا 7FFH تغییر کنند. بسته به وضعیت بیتهای A11 تا A13 یکی از قطعات حافظه آدرس‌دهی خواهند شد. به عنوان مثال برای اولین حافظه (متصل به خروجی صفر آی‌سی 74138) داریم.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH

یعنی اولین حافظه محدوده آدرس 0000 H تا 07FF H را اشغال کرده است. به همین ترتیب برای پنجمین حافظه داریم:

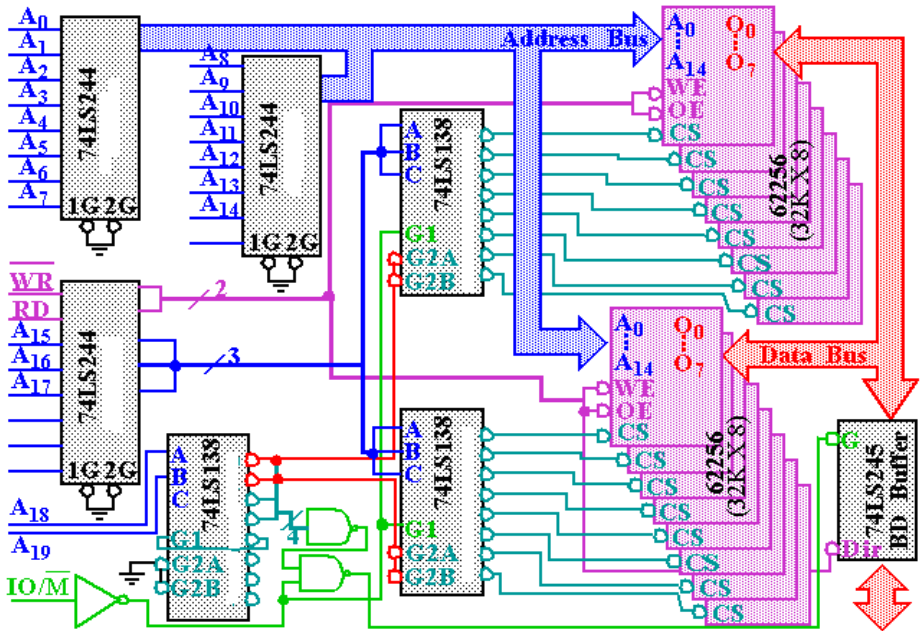
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	27FFH



شکل ۸- دیکودینگ قطعات ۸ کیلوبایتی توسط 74138

یعنی پنجمین حافظه محدوده آدرس 2000 H تا 27FF H را اشغال کرده است. برای هشتمین حافظه نیز می‌توان نشان داد که محدوده اشغال شده 3800 H تا 3FFF H است. به عبارت دیگر، مدار فوق فاصله ۱۶ کیلوبایتی از آدرس 0000 H تا 3FFF H حافظه را تشکیل می‌دهد.

شکل ۹ نیز عمل دیکودینگ ۵۱۲ کیلوبایت حافظه SRAM (که فضای آدرس 00000H تا7FFFFH را تشکیل می‌دهند) در پروسسور ۸۰۸۸ را نشان می‌دهد. در این شکل از ۱۶ عدد حافظه RAM با ظرفیت ۳۲ کیلوبایتی، دیکودر 74138، بافر یکطرفه 74244 و بافر دوطرفه 74245 استفاده شده است.

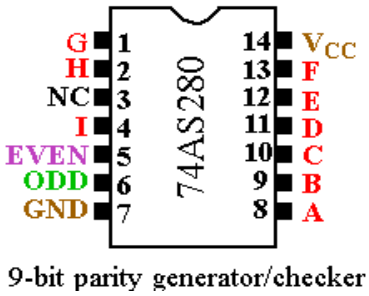


شکل ۹- دیکودینگ قطعات ۳۲ کیلوبایتی در فضای آدرس 00000H تا 7FFFFH

۳-۳- آشکارسازی خطا:

اطمینان از عدم رخ دادن خطا در سیستمهای دیجیتال بسیار مهم است به عنوان مثال داده هشت بیتی 00000000 را در نظر بگیرید. فرض کنید باارزشترین بیت آن در اثر خطا (مثلا نویز) به یک تغییر کند و داده به صورت 10000000 در آید. در این صورت با تغییر فقط یک بیت مقدار داده به طور شگفت انگیزی تغییر خواهد کرد (از مقدار صفر به مقدار ۱۲۸). به همین خاطر باید مطمئن شویم که در هنگام خواندن داده های حافظه یا در هنگام انتقال اطلاعات دیجیتال اگر خطایی رخ دهد بتوانیم آنرا حداقل آشکارسازی کنیم و در شرایط بهتر آنرا تصحیح کنیم. به منظور تشخیص خطا از روشهای متفاوتی استفاده می‌شود که تعدادی آز آنها را در اینجا بررسی می‌کنیم.

۳-۳-۱- توازن(Parity) : در این روش یک بیت به داده مورد نظر طوری اضافه می‌شود که تعداد یک های داده (با احتساب بیت اضافه شده) را فرد یا زوج کند که بر این اساس به ترتیب توازن فرد یا توازن زوج را خواهیم داشت. فرستنده می‌تواند با جمع معمولی بیتهای پیام یا با Exclusive OR کردن تمام بیتها، بیت پریتی را بسازد. اما روش پریتی، فقط تغییرات فرد در بیتها را تشخیص می دهد یعنی اگر ۱، ۳، ۵،... تغییر در بیتها صورت گیرد، پریتی می‌تواند خطاها را تشخیص دهد و این اشکال عمده روش پریتی است. توازن داده دریافتی محاسبه شده و خطا تشخیص داده می‌شود. اشکال این روش این است که فقط قادر است خطاهای فرد (یک بیت خطا، سه بیت خطا، ۵ بیت خطا و . . .) را تشخیص دهد. به عنوان مثال اگر به طور همزمان در دو بیت، خطا رخ دهد این روش قادر به تشخیص آن نیست. یکی از سخت افزارهایی که برای تولید بیت توازن و نیز چک کردن توازن داده های هشت بیتی استفاده می‌شود آی سی 74280 است که در شکل ۱۰ آی‌سی و عملکرد آن نشان داده شده است.



Number of inputs A thru I that are HIGH	Outputs EVEN	ODD
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

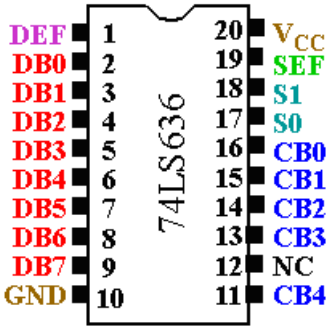
شکل ۱۰- آی سی مولد/ تشخیص پریتی (توازن)

این آی سی دارای ۹ ورودی A تا I است و می‌تواند در دو مد کار کند.

در مد مولد بیت توازن(Parity Generator)، هشت بیت داده به ورودیهای A تا H داده شده و ورودی I آن زمین می‌شود. اگر توازن در ۸ بیت داده وارده از نوع زوج باشد خروجی EVEN آن یک شده و در صورت توازن فرد، خروجی ODD آن یک می‌شود. از خروجی EVEN یا ODD آن به همراه ۸ بیت داده ورودی به صورت یک مجموعه ۹ بیتی برای نوشتن در حافظه استفاده می‌شود. اگر از خروجی EVEN به عنوان بیت نهم استفاده شود توازن داده های نوشته شده در حافظه از نوع توازن فرد است اما اگر از خروجی ODD به عنوان بیت نهم استفاده شود توازن داده های نوشته شده در حافظه از نوع توازن فرد خواهد بود.

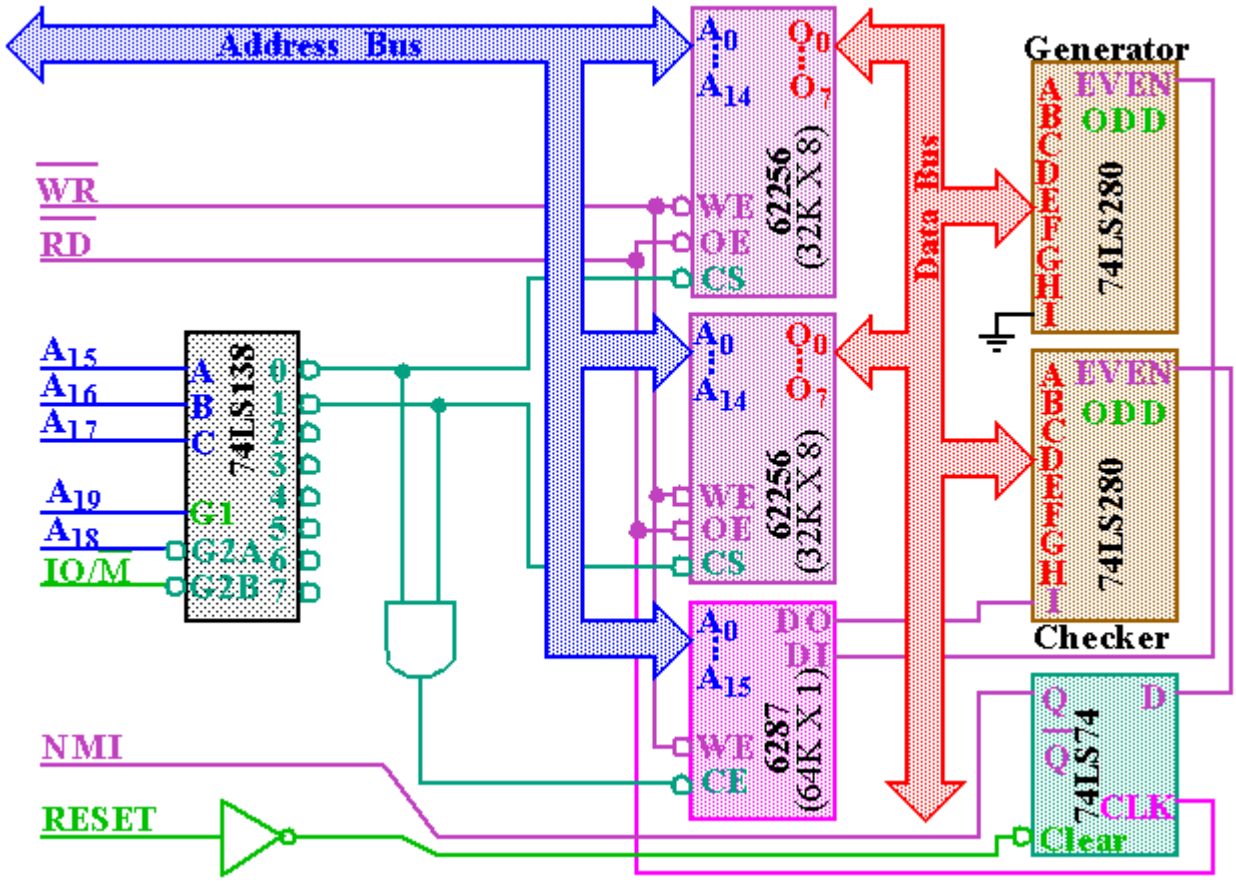
در مد مولد چک کننده بیت توازن(Parity Checker)، نه بیت خوانده شده از حافظه به ورودیهای A تا I داده شده و خروجیهای EVEN یا ODD بررسی می‌شوند. چنانچه توازن از نوع توازن فرد باشد باید خروجی ODD یک باشد و چنانچه خروجیEVEN یک باشد به منزله وجود خطا در داده خوانده شده است لذا خروجی EVEN آی سی می‌تواند به پایه اینتراپت پروسسور متصل شده و پروسسور را از وجود خطا در داده خوانده شده آگاه کند.

یکی دیگر از آی سی هایی که تعدادی بیت توازن در بین داد ه ها قرار می دهد آی سی 74636 است که اساس کار آن بر عهده خواننده واگذار می شود.



The 74LS636 corrects errors by storing 5 parity bits with each byte of data

شکل ۱۱ استفاده از آی‌سی 74280 به عنوان آشکارساز خطای حافظه را نشان می‌دهد.



شکل ۱۱- آشکارسازی خطای حافظه

۳-۳-۲- M-of-N Codes : در این روش بطور مثال در حالت ۲ از ۵، هر بلوک ۵ بیتی داده که ارسال می‌شود، دقیقا دارای دوتا ۱ می باشد. بنابراین خروجی می‌تواند یک خطا را تشخیص دهد(در حالتی که یک بیت معکوس شده باشد) اما این توانایی را برای ۲ بیت خطا ندارد. (مثلا حالتی که یک بیت از صفر به یک و دیگری از صفر به یک تغییر کرده باشد)

۳-۳-۳- کد همینگ: یکی از کدهای بلوکی (Block Codes) است که در تشخیص و تصحیح خطا به روش تصحیح خطای مستقیم (Forward Error Correction = FEC) به کار برده می‌شود. **همینگ** در دهه۱۹۴۰روشهای مختلفی را برای تشخیص و تصحیح خطا ارائه کرد که هر یک پیشرفت بزرگی محسوب می‌شدند و همگی آنها برمبنای طراحی پریتی بیت هایی بود که هریک تعدادی از بیت های داده را چک می‌کنند. اما در نقاطی که هریک از آنها را چک می‌کنند همپوشانی (Overlap) دارند و می توانند صحت خود و داده ها را چک کنند.کد همینگ قابلیت تصحیح یک خطای منفرد (Single Error Correction = SEC) و تشخیص یک زوج خطا(Double Error Detection = DED) را دارد.

قبل از معرفی الگوریتم کد همینگ، معرفی دو مفهوم زیر ضروری به نظر می‌رسد.

الف- وزن همینگ (Hamming Weight): وزن همینگ یک رشته ، تعداد سمبل هایی است که در الفبای مورد استفاده برای رشته، مخالف صفر هستند. در یک رشته باینری، وزن همینگ برابر تعداد ۱ ها در رشته است. بطور مثال:

Alphabet	string	hamming weight
1،0	11101	4
1،0	11101000	4
1،0	00000000	0
a-z،‘ ’	hello world	10

ب- فاصله همینگ (Hammig Distance): فاصله همینگ بین دو رشته **با طول مساوی** برابر تعداد تفاوتهایی است که بین سمبل های متناظر دو رشته وجود دارد. به بیان دیگر، این فاصله حداقل تعویض مکان های لازم برای تبدیل یک رشته به دیگری را بیان می‌کند.

مثال: فاصله همینگ بین دو رشته 1011101 و1001001 برابر ۲ است. فاصله همینگ بین دو رشته 2143896 و2233796 برابر ۳ است. فاصله همینگ بین دو رشته "toned" و"roses" برابر ۳ است.

برای دو رشته باینری A و B این فاصله برابر است با وزن همینگ عدد حاصل ازعمل A xor B

مثال : اگر A = 10011011 و B = 10001101 باشد A xor B = 00010110 و لذا فاصله همینگ برابر ۳ خواهد بود.

توضیح تئوری کار

فرض کنید می‌خواهیم این امکان را داشته باشیم که تعداد k خطا را در داده های ارسالی با طول n بیت تشخیص دهیم. اگر بتوانیم این رشته های n بیتی را به رشته هایی از پیش تعیین شده با طول n+k+1 بیت گسترش دهیم به شرط آنکه :

الف- بین هر کد n بیتی و کد n+k+1 بیتی، یک تناظر یک به یک برقرار باشد و

ب- فاصله همینگ بین هر دو کد n+k+1 بیتی تولید شده برابر با k+1 باشد،

آنگاه به مقصود خود، دست یافته ایم.

اکنون اگر یک کد n+k+1 بیت ارسال شود و تعداد x خطا (که x<k+1است) در مسیر انتقال روی بیتهای آن رخ دهد، کدی که گیرنده دریافت می‌کند، با هیچ یک از کد های از پیش تعیین شده تطابق نخواهد داشت و تعداد این خطاها برای تبدیل یک کد به کد تعریف شده دیگر هم کافی نخواهد بود. زیرا طبق شرط ۲ حداقل تعداد تغییرات برای تبدیل یک کد به کد دیگر k+1 است. بنابراین گیرنده متوجه بروز خطا می‌شود و این حالت برای از ۱ تا k خطا برقرار است.

حال فرض کنیم این کدهای n+k+1 بیتی، به گونه ای طراحی شده باشند که در صورت بروز یک خطا، یعنی تغییر یک بیت ، اگر این تغییر در بیت متفاوتی صورت گیرد، آنگاه خروجی نادرست متمایزی هم ایجاد شود، آنگاه خواهیم توانست محل این یک خطا را تشخیص داده و آن را اصلاح کنیم.

همینگ در طراحی کدهای خود به طور همزمان بر روی دو مسئله تمرکز داشت :

۱- افزایش فاصله همینگ بین رشته های ارسالی برای سادگی تشخیص خطا و کاهش احتمال تبدیل یک کد معتبر به دیگر.

۲ -افزایش نرخ کد(Code rate) یعنی افزایش تعداد بیتهای داده اصلی به ازاء یک تعداد مشخص بیت ارسالی.

الگوریتمی که همینگ برای این منظور به کار برد،برای داده n= 15 و K =3 بیتی به کار برد به صورت زیر است:

فرض کنیم داده ای با طول ۱۵ بیت قرار است در بلوکی با طول ۲۰ بیت قرار گیرد. به این ترتیب که :

۱- بیتهای پریتی، به ترتیب در خانه هایی که با توانهای ۲ مشخص می‌شوند، یعنی (۱،۲،۴،۸ و۱۶) قرارگیرند.

۲- تمام مابقی خانه ها، مخصوص داده ای است که قرار است تبدیل به کد شود یعنی خانه های (۳،۵،۶،۷،۹و....۲۰)

۳ – هر بیت پریتی، توازن را برای تعدادی از خانه ها در کد نهایی چک کند. مکان یا توالی هر بیت پریتی تعیین می‌کند که کدام خانه ها را، چک و کدام خانه ها را رد کند. بااین منظور باید روند زیر را طی کنیم که در آن هر موقعیت بیتی برخی بیتها را رد (skip) و برخی بیتها را چک (check) می‌کند.

Position 1 (n=1): skip 0 bit (0=n−1)، check 1 bit (n)، skip 1 bit (n)، check 1 bit (n)، skip 1 bit (n)، etc. (1،3،5،7،9،11،13،15،...)

Position 2 (n=2): skip 1 bit (1=n−1)، check 2 bits (n)، skip 2 bits (n)، check 2 bits (n)، skip 2 bits (n)، etc. (2،3،6،7،10،11،14،15،...)

Position 4 (n=4): skip 3 bits (3=n−1)، check 4 bits (n)، skip 4 bits (n)، check 4 bits (n)، skip 4 bits (n)، etc. (4،5،6،7،12،13،14،15،20،21،22،23،...)

Position 8 (n=8): skip 7 bits (7=n−1)، check 8 bits (n)، skip 8 bits (n)، check 8 bits (n)، skip 8 bits (n)، etc. (8-15،24-31،40-47،...)

Position 16 (n=16): skip 15 bits (15=n−1)، check 16 bits (n)، skip 16 bits (n)، check 16 bits (n)، skip 16 bits (n)، etc. (16-31،48-63،80-95،...)

Position 32 (n=32): skip 31 bits (31=n−1)، check 32 bits (n)، skip 32 bits (n)، check 32 bits (n)، skip 32 bits (n)، etc. (32-63،96-127،160-191،...)

قانون کلی برای مکان n این است که n-1 خانه را رد میکند/n خانه راچک می‌کند /n خانه را رد می‌کند /n خانه را چک می‌کند/ و...

این قانون کلی می‌تواند به صورت تصویری زیر نمایش داده شود:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
مکان بیت																					
بیتهای داده کد شده		p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7	d8	d9	d10	d11	p5	d12	d13	d14	d15
	p1	X		X		X		X		X		X		X		X		X		X	
پوشش	p2		X	X			X	X			X	X			X	X			X	X	
بیتهای	p3				X	X	X	X					X	X	X	X					X
پریتی	p4								X	X	X	X	X	X	X	X					
	p5																X	X	X	X	X

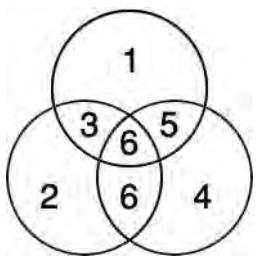
در جدول فوق، تنها یک رشته با طول ۲۰ نمایش داده شده است اما این الگو می‌تواند بصورت نامتناهی ادامه یابد.

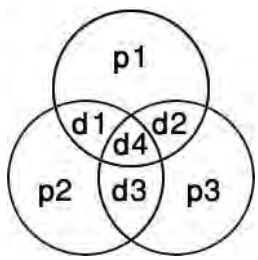
نکته اصلی در باره کد همینگ که به راحتی از جدول هم قابل تشخیص است این است که هر بیت، اعم از پریتی یا داده، پوشش پریتی منحصر به فردی دارد. به طور مثال، تنها بیتی که فقط به وسیله P_3 ، P_4 پوشش داده شده، بیت ۱۲ یا d8 است. همین پوشش منحصر به فرد است که به کد همینگ اجازه می دهد، خطاهای فرد را اصلاح کند. به این صورت که در صورت معکوس شدن هریک از بیت های داده یا پریتی، در مسیر انتقال، بیتهای پریتی متناظر با آن بیت در خروجی با مقدار بیت داده تناسب نخواهند داشت و با بررسی این نکته که، کدام پریتی بیت ها غلط است، می‌توان به داده معکوس شده، پی برد.

کد همینگ در اندازه های مختلف وجود دارد که متداولترین نوع آن (4,7) است که در سال ۱۹۵۰ معرفی شد. امروزه بیشتر از این کد به عنوان کد همینگ یاد می‌شود. نمایش (4,7) به این مفهوم است که این کد دارای سه بیت چک به ازاء هر ۴ بیت داده است که باهم ۷ بیت را تشکیل میدهند. این کد قابلیت تشخیص و تصحیح، هر خطای منفرد (SEC) و نیز تشخیص جفت خطا (DED) را دارد. در صورتی که عملکرد تصحیح مورد نظر باشد، حداکثر فاصله همینگ بین داده ارسال شده با داده دریافت شده، نباید بیش از یک باشد. و همچنین برای تشخیص خطا حداکثر، این فاصله نباید بیش از ۲ باشد.

نمایش تصویری کد همینگ (7،4) به صورت زیر است.

شماره بیت	1	2	3	4	5	6	7
بیت ارسالی	p1	p2	d1	p3	d2	d3	d4
p1	yes	no	yes	no	yes	no	yes
p2	no	yes	yes	no	no	yes	yes
p3	no	no	no	yes	yes	yes	yes





در جدول سمت چپ می بینیم که هر بیت پریتی، کدام بیتها را پوشش می دهد. از روی این جدول، و طراحی ۷ بیت ارسال شده به فرم شکل میانی، میتوان به نمایش شکل راست رسید که (Overlap) بیتها را تداعی می‌کند.

همانگونه که در شکل هم مشخص است، بیتهای پریتی P1 (توازن را برای بیتهای d1)،d2،d4 و P2 (توازن را برای بیتهای d1)،d3،d4 و P3 (توازن را برای بیتهای d2،d3،d4) چک می‌کنند.

بنابراین روال کار برای دیکد کردن۴ بیت داده به ۷ بیت و ارسال آن به این صورت است که،

۱- ابتدا داده ها در مکان های متناظر خود قرار می گیرند.

۲- هر بیت پریتی، توازن را برای خانه هایی که برایش تعریف شده چک می‌کند و بر این اساس مقدار می گیرد. اکنون می توان داده را ارسال کرد. در طرف گیرنده، تناسب هر بیت پریتی با خانه هایی که آنها را پوشش می دهد، بررسی می‌شود و از روی بیت های پریتی که این تناسب را ندارند (با توجه به یکتا بودن پوشش پریتی برای هر بیت خاص) می توان بیت معیوب را کشف و اصلاح نمود.

مثال: فرض کنید داده ای که قرار است ارسال شود 1011 باشد

1	2	3	4	5	6	7
P1	P2	D1	P3	D2	D3	D4
0	1	1	0	0	1	1

1	2	3	4	5	6	7
P1	P2	D1	P3	D2	D3	D4
		1		0	1	1

فرض کنیم یک خطای تک بیتی در اطلاعات ارسال شده رخ دهد. مثلا بیت ششم از یک به صفرتغییر کند و در گیرنده کد 0110001 دریافت شود.گیرنده با توجه به شماره بیتها شروع به چک کردن توازن برای هر بیت پریتی می‌کند. با توجه به قرارداد فوق باید XOR بیتهایی که هر پریتی کنترل میکند با احتساب خودش، برابر صفر شود زیرا هر پریتی خودش را نیز چک می‌کند.

b1	b2	b3	b4	b5	b6	b7
P1	P2	D1	P3	D2	D3	D4
0	1	1	0	0	0	1

C_1=XOR(b1، b3، b5، b7)= XOR(0، 1، 0، 0)=0 صحیح

C_2=XOR(b2، b3، b6، b7)= XOR(1، 1، 0، 0)=1 غلط

C_3=XOR(b4، b5، b6، b7)= XOR(0، 0، 1، 0)=1 غلط

ترکیب C_3،C_2،C_1 برای نشان دادن بیت خطا به صورت $C_3C_2C_1=(110)_B = 6$ بنابراین بیت ششم دچار خطا شده که اگر آنرا معکوس کنیم کد درست به شکل 0110011 در می‌آید که با حذف بیتهای پریتی داده به صورت 1011 خواهد بود.

همانگونه که در این مثال دیده شد، هر بیت با کد منحصر به فردی که از ترکیب کدهای چک حاصل می‌شود، مشخص شده و اگر در این بیت خطایی حاصل شود، پریتی بیت هایی که آن را در هنگام ارسال پوشش داده اند نیز در خروجی نامتناسب تشخیص داده شده و از ترکیب آنها شماره بیت خطا حاصل می‌شود.

۴-۳-۳- CRC (Cyclic Redundancy Check)

CRC تکنیکی برای پیداکردن خطاها در اطلاعات دیجیتالی است گرچه این تکنیک قادر به آشکارسازی خطا است اما برای تصحیح خطاهای کشف شده کارایی ندارد. در این روش بیتهایی مشخص که معمولاً check bits

یا check sum نامیده می‌شوند به پیغامی که قرار است انتقال یابد پیوست می‌شود و توسط آن گیرنده پیام می‌تواند مشخص کند که آیا check bits با داده دریافتی مطابقت دارد یا خیر. اگر خطایی اتفاق افتاده باشد، دریافت کننده یک پیغام تحت عنوان Negative Acknowledgement (NAK) به فرستنده بر می گرداند که به موجب آن به گیرنده می گوید که پیغام دوبار فرستاده شود. این روش گاهی اوقات برای وسایل ذخیره کننده داده مانند DISK DRIVE ها نیز بکار می‌رود. در این حالت هر بلوک روی دیسک، باید شامل check bits باشد و اگر خطایی کشف شود، سخت افزار ممکن است بطور اتوماتیک یک بازخوانی را بر روی بلوک آغاز کند و یا اینکه وجود خطا را به نرم افزار گزارش کند. این روش پیام اولیه را به عنوان رشته‌ای از داده‌های سریال با طول n بیت در نظر می‌گیرد که این بیتها به عنوان ضرایب یک چند جمله‌ای مشخصه M(x) در نظر گرفته می‌شوند.

تئوری CRC : تئوری CRC بر اساس علم حساب چندجمله ایها و بخصوص بر اساس محاسبه باقیمانده یک چندجمله‌ای بر چندجمله‌ای دیگر بنا شده است. اما در این روش بر اساس مقادیر صفر و یکی که بیتها بخود می گیرند ضرایب چندجمله‌ای حاصله دارای دومقدار صفر یا یک است. به هر رشته داده یک چندجمله‌ای متناظر می‌شود که داده های رشته متناظر به ضرایب چندجمله ای هستند. به عنوان مثال برای داده ۰۱۱۰۱۱۰۱ چندجمله‌ای $M(x) = x^6 + x^5 + x^3 + x^2 + 1$ متناظر می‌شود. در حالت کلی اگر پیام ارسالی دارای m بیت باشد و بخواهیم تعداد r بیت check sum به آن اضافه کنیم باید از یک چندجمله‌ای مولد $G(x)$ از درجه r (که دارای r+1 جمله است) استفاده کنیم. در چنین شرایطی داریم:

$$CRC = \frac{M(x).x^r}{G(x)}$$

که در آن $M(x)$ چندجمله ای مربوط به پیام ارسالی، $G(x)$ چندجمله‌ای مولد و r تعداد بیت check sum است که قرار است به پیام اضافه شود. $G(x)$ چندجمله‌ای مولد بوده و دارای خواص ویژه‌ای است که از حوصله این بحث خارج است. $G(x)$های استانداردی که استفاده می‌شوند در جدول زیر آمده اند.

نام متداول	r	چندجمله ای	هگزادسیمال
CRC-12	12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	180F
CRC-16	16	$x^{16} + x^{15} + x^2 + 1$	18005
CRC-CCITT-16	16	$x^{16} + x^{12} + x^5 + 1$	11021
CRC-32	32	$x^{32} + x^{28} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	110C11DB7

جدول مربوط به G(x) های استانداردی که بین گیرنده و فرستنده مشترک است

از آنجا که $G(x)$ از درجه r است لذا باقیمانده تقسیم فوق که آنرا $R(x)$ می نامیم حداکثر از درجه (r-1) خواهد بود و داریم:

$$M(x).x^r = Q(x).G(x) + R(x)$$

نکته بسیارمهم در تقسیم چندجمله ایها در روش CRC این است که این تقسیم با تقسیم چندجمله ایهای معمولی اندکی متفاوت است به این معنا که در هر مرحله از تقسیم برای بدست آوردن باقیمانده بجای تفریق از عمل منطقی Exclusive OR استفاده می کنیم. به عنوان مثال اگر $M(x) = x^7 + x^6 + x^5 + x^2 + x$ را بر $x^3 + x + 1$ تقسیم کنیم داریم:

$$
\begin{array}{l|l}
x^7+x^6+x^5+x^2+x & x^3+x+1 \\
\oplus\ x^7+x^5+x^4 & \hline
\quad x^6+x^4+x^2+x & x^4+x^3+1 \\
\quad \oplus\ x^6+x^4+x^3 & \\
\quad\quad x^3+x^2+x & \\
\quad\quad \oplus\ x^3+x+1 & \\
\quad\quad\quad x^2+1 &
\end{array}
$$

که در آن علامت $\oplus$ به معنای Exclusive OR دو چندجمله‌ای است که به موجب آن جملات مشترک بین دو چندجمله ای حذف شده و جملات غیریکسان باقی می مانند. از درس دیجیتال به خاطر دارید که اگر A و B دو مجموعه یا دو چندجمله ای مثلاً بر حسب x باشند عمل Exclusive OR بصورت زیر تعریف می‌شود.

$$A \oplus B = AB' + A'B$$

معنی عبارت بالا این است که اگر A و B را Exclusive OR کنیم، جواب برابر است با :

(جملاتی که در A هست و در B نیست) + (جملاتی که در B هست و در A نیست)

جهت آشنایی با روش تولید CRC یک متال می زنیم:
مثال: فرض کنید بخواهیم یک رشته بیتی بصورت 0000111101100100 = 0F64 H را به عنوان پیام برای گیرنده ارسال کنیم (پیام ارسالی بصورت 0F62 H باشد). و بخواهیم r = 16 بیت به این پیام به عنوان check sum اضافه کنیم. در اینصورت داریم:

$$M(x) = x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^2$$

اگر از چندجمله ای مولد $G(x) = x^{16} + x^{15} + x^2 + 1$ استفاده کنیم (چون $G(x)$ از درجه 16 است لذا باقیمانده تقسیم فوق یعنی $R(x)$ حداکثر از درجه 15 خواهد بود و داریم:

$$\frac{M(x).x^r}{G(x)} = \frac{M(x).x^{16}}{G(x)} = \frac{(x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^2).x^{16}}{x^{16} + x^{15} + x^2 + 1} = \frac{x^{27} + x^{26} + x^{25} + x^{24} + x^{22} + x^{21} + x^{18}}{x^{16} + x^{15} + x^2 + 1}$$

و اگر تقسیم فوق را انجام دهیم داریم:

$$
\begin{array}{l|l}
x^{27}+x^{26}+x^{25}+x^{24}+x^{22}+x^{21}+x^{18} & x^{16}+x^{15}+x^{2}+1 \\
\oplus\ x^{27}+x^{26}+x^{13}+x^{11} & \hline
x^{11}+x^{9}+x^{6}+x^{2}+x+1 \\
\hline
x^{25}+x^{24}+x^{22}+x^{21}+x^{18}+x^{13}+x^{11} & \\
\oplus\ x^{25}+x^{24}+x^{11}+x^{9} & \\
\hline
x^{22}+x^{21}+x^{18}+x^{13}+x^{9} & \\
\oplus\ x^{22}+x^{21}+x^{8}+x^{6} & \\
\hline
x^{18}+x^{13}+x^{9}+x^{8}+x^{6} & \\
\oplus\ x^{18}+x^{17}+x^{4}+x^{2} & \\
\hline
x^{17}+x^{13}+x^{9}+x^{8}+x^{6}+x^{4}+x^{2} & \\
\oplus\ x^{17}+x^{16}+x^{3}+x & \\
\hline
x^{16}+x^{13}+x^{9}+x^{8}+x^{6}+x^{4}+x^{3}+x^{2}+x & \\
\oplus\ x^{16}+x^{15}+x^{2}+x & \\
\hline
x^{15}+x^{13}+x^{9}+x^{8}+x^{6}+x^{4}+x^{3}+x+1 &
\end{array}
$$

در این حالت باقیمانده یا R(x) عبارتست از $R(x) = x^{15} + x^{13} + x^9 + x^8 + x^6 + x^4 + x^3 + x + 1$ و خارج قسمت Q(x) عبارتست از $Q(x) = x^{11} + x^9 + x^6 + x^2 + x + 1$.

دقت شود که R(x) محاسبه شده همان check sum است که به صورت ۱۶ بیت (متناظر با چندجمله ای R(x)) به پیام اولیه پیوست می‌شود. ۱۶ بیت متناظر با R(x) بصورت 1010001101011011= 5CAD H است که به پیام 0F64 اضافه شده و سپس حاصل 0F645CAD H برای گیرنده ارسال می‌شود.

نکته: باید حاصل $Q(x).G(x) + R(x)$ برابر $M(x).x^r$ شود. به منظور صحت عمل، عبارت فوق را بصورت عادی ضرب و جمع کرده و جملاتی را که دو بار تکرار شده اند حذف می کنیم. جملاتی که باقی مانده اند باید همان $M(x).x^r$ باشد.

گیرنده چگونه از صحت داده دریافتی اطلاع پیدا می‌کند؟ آنچه را که گیرنده دریافت می‌کند شامل $M(x)$ و R(x) است. گیرنده $M(x).x^r$ را منهای R(x) (check sum) کرده و این مقدار را بر $G(x)$ تقسیم می‌کند. اگر باقیمانده حاصل این تقسیم صفر شد داده دریافتی درست است و در غیر اینصورت نادرست است و گیرنده دوباره تقاضای داده درست از فرستنده را خواهد کرد.

۴-۱- سیر تکاملی ریزپردازنده‌های خانواده 80x86

واژه ریزپردازنده[17] در صنعت نیمه‌هادی توسط شرکت Intel ابداع شد. آنها این واژه را برای توصیف یک مدار مجتمع ماشین حساب گونه‌ی چهار بیتی که تازه طرح کرده بودند به کار بردند. اولین ریزپردازنده در سال ۱۹۷۱ توسط شرکت اینتل ساخته شد. این شرکت در ابتدا ریزپردازنده‌های خانواده 80x86 را ارائه کرد که عبارت بودند از: 8086، 8088، 80286، 80386، 80486 و از سال ۱۹۹۳ خانواده پنتیوم را معرفی کرد که تا کنون این خانواده به چهار گروه Pentium،Pentium II، Pentium III و Pentium IV تقسیم شده‌اند. پس از سری80386، رجیسترهای ۳۲ بیتی جایگزین رجیسترهای ۱۶ بیتی قدیمی‌شدند. پس از خانواده 80486 به علت افزایش چگالی ترانزیستورهای بکار رفته در ساختمان ریزپردازنده، خنک کننده‌ای هم بر روی ریزپردازنده تعبیه شد. جدول ۴-۱ سیر تکامل ریزپردازنده‌ها را نشان می‌دهد.

جدول۴- ۱ سیر تکاملی ریزپردازنده‌ها

ردیف	شماره ریزپردازنده	سال تولید	سرعت(مگاهرتز)	تعداد تقریبی ترانزیستور
۱	8086	۱۹۷۸	۵-۱۰	۲۹۰۰۰
۲	8088	۱۹۷۹	۴/۷۷	۲۹۰۰۰
۳	80286	۱۹۸۲	۸-۱۲	۱۳۴۰۰۰
۴	80386	۱۹۸۵	۱۶-۳۳	۲۷۵۰۰۰
۵	80486	۱۹۸۹	۲۵-۱۰۰	۲ میلیون
۶	Pentium	۱۹۹۳	۶۰-۱۶۶	۳/۳ میلیون
۷	Pentium Pro	۱۹۹۵	۱۵۰-۲۰۰	۵/۵ میلیون
۸	Pentium II	۱۹۹۷	۲۳۳-۵۰۰	۷/۵ میلیون
۹	Pentium III	۲۰۰۰	۴۵۰-۱۰۰۰	۱۰ میلیون
۱۰	Pentium IV	۲۰۰۲	۱۷۰۰-۳۲۰۰	۱۲/۵ میلیون

امروزه ریزپردازنده به‌ای‌سی‌هایی گفته می‌شود که اساس یک میکرو کامپیوتر را تشکیل می‌دهند. مثلاً کامپیوتر شخصی IBMبر اساس میکروپروسسور Intel 8088 و اپل مکینتاش(Apple Macintosh) بر اساس موتورولا Motorola 68000 ساخته شده‌اند.

در پردازنده‌های 80x86 با $x \leq 3$ برای انجام محاسبات ممیز شناور [18] از کوپروسسور 80x87 استفاده می‌شود ولی از 80486 به بعد هم حافظه Cache و هم واحد محاسبات ممیز شناور در داخل CPU کار گذاشته شده است.

نحوه عملکرد ریزپردازنده

17 - Microprocessor
18 - Floating Point

هر پردازنده برای پردازش اطلاعات، به این صورت کار می‌کند که ابتدا واحد کنترل، دستورها را از حافظه واکشی می‌کند و در مرحله بعد دستورالعمل رمزگشایی می‌شود. اگر این دستور به داده‌هایی احتیاج داشته باشد، داده‌ها پس از بازیابی، در یکی از رجیسترهای واحد محاسبه و منطق قرار می‌گیرند و سپس دستور در واحد محاسبه و منطق اجرا می‌شود.

۴-۲- گذرگاههای یک سیستم کامپیوتری

در هر سیستم کامپیوتری یکسری خطوط ارتباطی وجود دارند که اجزای مختلف آنرا به هم مرتبط می‌سازند که به آن گذرگاه(BUS) می‌گویند. به عبارت دیگر، گذرگاه به خطوط ارتباطی میان اجزای داخلی کامپیوتر گفته می‌شود که از طریق آن داده‌ها و آدرسها به پردازنده می‌رسند. این خطوط ارتباطی، مسیرهایی سخت افزاری هستند که از سیمهای ظریفی ساخته شده و ارتباط میان پردازنده و اجزای دیگر مانند حافظه و دستگاههای جانبی را برقرار می‌کنند. گذرگاه ممکن است یکطرفه یا دوطرفه باشد مثلاً دستگاه ورودی فقط داده‌ها را ارسال و دستگاه خروجی داده‌ها را فقط دریافت می‌کند اما گذرگاه داده برای حافظه دوطرفه است یعنی هم داده‌ها از این گذرگاه به حافظه وارد و هم از آن خارج می‌شوند. از آنجا که از یک گذرگاه مشترک برای انتقال اطلاعات استفاده می‌شود، نمی‌توان همزمان روی آن دو نوع اطلاعات قرار داد. در لحظه‌ای که بین دو جزء ارتباط از طریق گذرگاه برقرار است بقیه اجزاء توسط عناصر سه حالته (مانند بافرها و...) از گذرگاه جدا هستند. با توجه به نوع اطلاعاتی که در گذرگاهها رد و بدل می‌شود، سه نوع گذرگاه وجود دارد که عبارتند از: گذرگاه داده[۱۹]، گذرگاه آدرس[۲۰] و گذرگاه کنترل[۲۱].

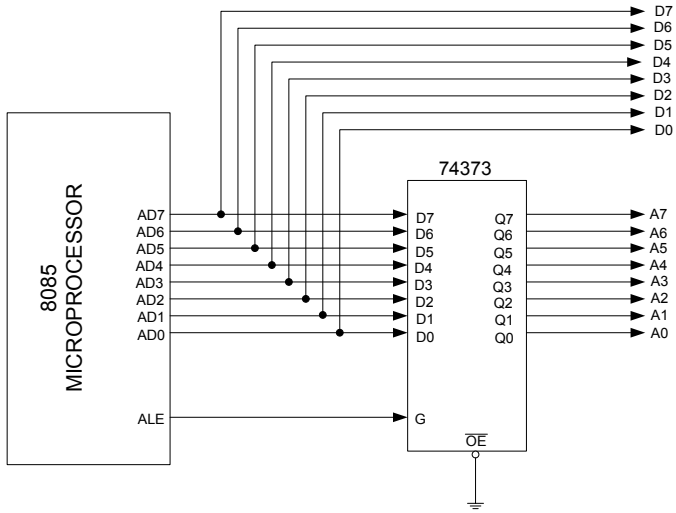
الف- گذرگاه داده: گذرگاهی که داده‌های بازیابی شده از حافظه (یا داده‌های دریافتی از دستگاه ورودی) را در اختیار CPU قرار می‌دهد و داده‌های پردازش شده (توسط CPU) را به حافظه (یا دستگاه خروجی) منتقل می‌کند، گذرگاه داده نامیده می‌شود. هرچه ظرفیت این گذرگاه بیشتر باشد، داده‌های بیشتری بصورت همزمان منتقل می‌شوند و سرعت انجام عملیات بیشتر می‌شود. در یک کامپیوتر ۱۶ بیتی، گذرگاه داده ۱۶ بیتی است. جدول ۴-۲ ظرفیت گذرگاه داده و آدرس را در پردازنده‌های مختلف نشان می‌دهد.

جدول۴-۲- ظرفیت گذرگاه داده و آدرس در برخی ریزپردازنده‌ها

ردیف	شماره ریزپردازنده	تعداد خطوط گذرگاه داده	تعداد خطوط گذرگاه آدرس
۱	8086	۱۶	۲۰
۲	8088	۸	۲۰
۳	80286	۱۶	۲۴
۴	80386	۳۲	۳۲
۵	80486	۳۲	۳۲
۶	Pentium	۳۲	۳۲
۷	Pentium Pro	۶۴	۳۲
۸	Pentium II	۶۴	۶۴
۹	Pentium III	۶۴	۶۴
۱۰	Pentium IV	۶۴	۶۴

19 - Data Bus
20 - Address Bus
21 - Control Bus

ب- گذرگاه آدرس: برای آنکه CPU به اطلاعات درون خانه‌های حافظه دسترسی پیدا کند، باید آدرس آن خانه‌ها را مشخص کند. برای این منظور CPU آدرس را روی گذرگاه آدرس قرار می‌دهد. تعداد خطوط گذرگاه آدرس به ظرفیت حافظه‌های سیستم و تعداد دستگاههای ورودی و خروجی بستگی دارد. به عنوان مثال گذرگاه آدرس با ۱۰ خط آدرس می‌تواند $۲^{۱۰}$=۱۰۲۴ خانه حافظه (یک کیلو بایت حافظه) را آدرس‌دهی کند. ظرفیت گذرگاه آدرس(تعداد خطوط آدرس) در پروسسورهای مختلف متفاوت است. جدول ۲، ظرفیت گذرگاه آدرس در برخی پروسسورها را نشان می‌دهد. بیشتر باسهای آدرس از نوع سه حالته هستند و در مدت عملیات عادی پروسسور در حالت امپدانس زیاد [۲۲] قرار دارند. در برخی میکروپروسسورها برای صرفه‌جویی در تعداد پایه‌ها، خطوط گذرگاه آدرس با خطوط گذرگاه داده یا برخی سیگنالهای کنترلی مالتی پلکس شده‌اند. در چنین پروسسورهایی معمولاً یک سیگنال کنترلی، جهت جداسازی خطوط آدرس از خطوط مالتی پلکس شده با آنها نیز پیش‌بینی شده است. به عنوان مثال در میکروپروسسور 8085 خطوط آدرس A0 تا A7 با خطوط داده D0 تا D7 مالتی پلکس شده‌اند و سیگنال کنترلی ALE خطوط آدرس را از خطوط داده جدا می‌کند. این سیگنال در سیکل T1 هر ماشین سیکل (به نشانه وجود آدرس روی خطوط AD0 تا AD7) فعال می‌شود. یک مدار نمونه برای جداسازی خطوط آدرس از خطوط داده‌ی مالتی پلکس شده با آنها در شکل ۴-۳ آمده است.



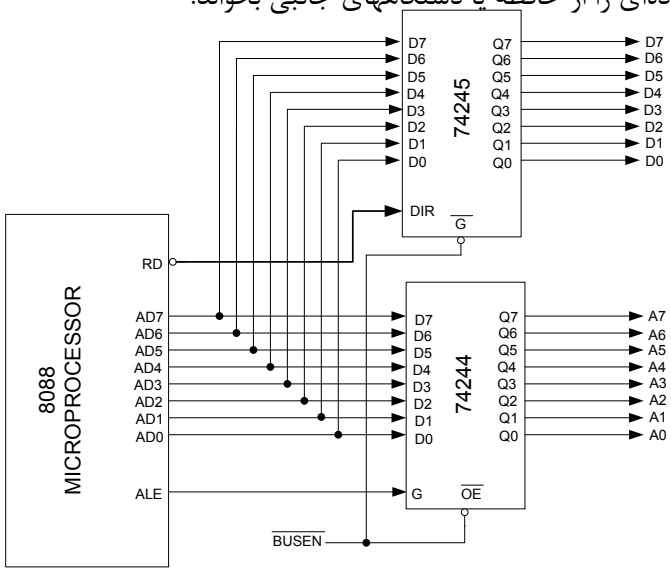
شکل ۴-۳- دی مالتی پلکس کردن آدرس در پروسسور 8085

در شکل ۴-۳، چنانچه سیگنال کنترلی ALE فعال (در سطح منطقی یک) باشد لچ 74373 فعال شده و ورودیهای آن (ADiها) به خروجی آن (Aiها) منتقل شده و لچ می‌شوند وتا زمانی که دوباره پایه‌ی کنترلی G فعال نشود این اطلاعات در خروجی لچ ثابت خواهند بود.

معمولاً خطوط آدرس گذرگاه آدرس قادر به راه‌اندازی بیش از یک گیت TTL استاندارد نیستند. برای آنکه قابلیت جریان دهی خطوط آدرس و داده افزایش یابد معمولاً آنها را بافر می‌کنند. خطوط آدرس را که یکطرفه هستند، بافر یکطرفه و خطوط داده را بافر دوطرفه می‌کنند. به عنوان مثال، نحوه بافر کردن خطوط آدرس و داده در میکروپروسسور 8088 در شکل ۴-۴ نشان داده شده است. با وجود این بافرها، خطوط آدرس و داده می‌توانند تا حدود ۲۰ گیت TTL استاندارد را راه‌اندازی کنند (گرچه وجود این بافرها باعث ایجاد تأخیر در مدار میکروپروسسور می‌شود).

22 - High Impedance

سیگنال RD از پروسسور 8088 یک سیگنال فعال سطح پایین است که جهت باس داده را برای بافر دوطرفه 74LS245 تغییر می‌دهد. وقتی سیگنال RD فعال می‌شود داده از بیرون به سمت پروسسور حرکت می‌کند. این سیگنال زمانی فعال می‌شود که پروسسور بخواهد داده‌ای را از حافظه یا دستگاههای جانبی بخواند.



شکل۴-۴- بافر کردن خطوط آدرس و داده در میکروپروسسور 8088

در طی نوشتن در حافظه یا دستگاه جانبی پین RD غیر فعال بوده و اجازه می‌دهد که داده از سمت پروسسور به خارج جریان پیدا کند.

ج- گذرگاه کنترل: گذرگاه کنترل، مسیری است که سیگنالهای کنترلی برای نظارت بر کلیه‌ی عملیات کامپیوتر از طریق آن ارسال می‌شود. واحد کنترل، برای اجرای نظارت بر بخشهای مختلف، از این خطوط استفاده می‌کند. عمده‌ترین سیگنالهای کنترلی گذرگاه کنترل، مربوط به سیگنالهای خواندن از حافظه یا دستگاههای جانبی و نوشتن در حافظه یا دستگاههای جانبی است. نام این سیگنالها در پروسسورهای مختلف متفاوت است. مثلاً در پروسسور 8080 این سیگنالها با عنوان MEMWR،MEMR ، IOWR، IOR و در پروسسور 8088 با عنوان RD،WR، IO/M هستند.

برای مواجهه با دستگاههای ورودی/خروجی پروسسورهای مختلف، یکی از دو تکنیک زیر را استفاده می‌کنند

الف- Memory Mapped I/O : در این تکنیک، با دستگاههای ورودی و خروجی بعنوان حافظه رفتار می‌شود. هنگام خواندن از یک دستگاه جانبی مثل این است که بخواهیم از یک آدرس مشخص حافظه اطلاعاتی را برداریم. یا هنگام نوشتن در یک دستگاه جانبی مثل این است که بخواهیم در یک موقعیت حافظه اطلاعاتی را بنویسیم. هر دستگاه ورودی یا خروجی به عنوان یک موقعیت حافظه در نظر گرفته می‌شود و نوشتن یا خواندن در یک دستگاه جانبی همانند نوشتن یا خواندن از حافظه توسط سیگنالهای کنترلی Read و Write صورت می‌گیرد.

ب- Isolated I/O (I/O Mapped I/O) : در این تکنیک، دستگاههای ورودی و خروجی هر کدام بعنوان یک دستگاه I/O جداگانه در نظر گرفته شده و آدرس یکتایی دارند. پروسسورهایی که از این تکنیک استفاده می‌کنند دارای دستورالعملهای خاصی نظیر IN و OUT نیز هستند تا داده را از دستگاههای جانبی بخوانند یا برای آنها بفرستند.

چنانکه ملاحظه شد در در تکنیک اول پروسسور فقط به دو سیگنال کنترلی MRD و MWR نیاز دارد در حالیکه در تکنیک دوم نیازمند چهار سیگنال کنترلی MRD، MWR، IORD و IOWR است. در عوض تکنیک دوم گرچه به چهار سیگنال کنترلی نیاز دارد ولی برای عملکرد خود نیازی به فضای حافظه ندارد.

اینکه کدامیک از دو تکنیک فوق بهتر است بستگی به نوع کاربرد و میکروپروسسور انتخاب شده برای کاربرد مورد نظر دارد.

گذرگاه کنترل علاوه بر سیگنالهای کنترلی مربوط به خواندن و نوشتن، سیگنالهای کنترلی دیگری نظیر INTR، DMA، RDY وReset نیز دارد.

پروسسور با سیگنال **ورودی اینتراپت** در حقیقت دستگاههای ورودی/ خروجی کند (مثل صفحه کلید) را پشتیبانی می‌کند تا این دستگاهها موجب کاهش کارایی سیستم نشوند. در پاسخ به هر اینتراپت، پروسسور سابروتین متناظر به آنرا اجرا می‌کند.

ورودی Reset باعث می‌شود که پروسسور کارش را از ابتدا شروع کند. بیشتر پروسسورها پس از فشرده شدن کلید Reset به موقعیت خاصی از حافظه پرش کرده و شروع به اجرای دستوراتی می‌کنند که این دستورات، عملیات مربوط به تنظیمات اولیه سیستم[۲۳] را انجام می‌دهند.

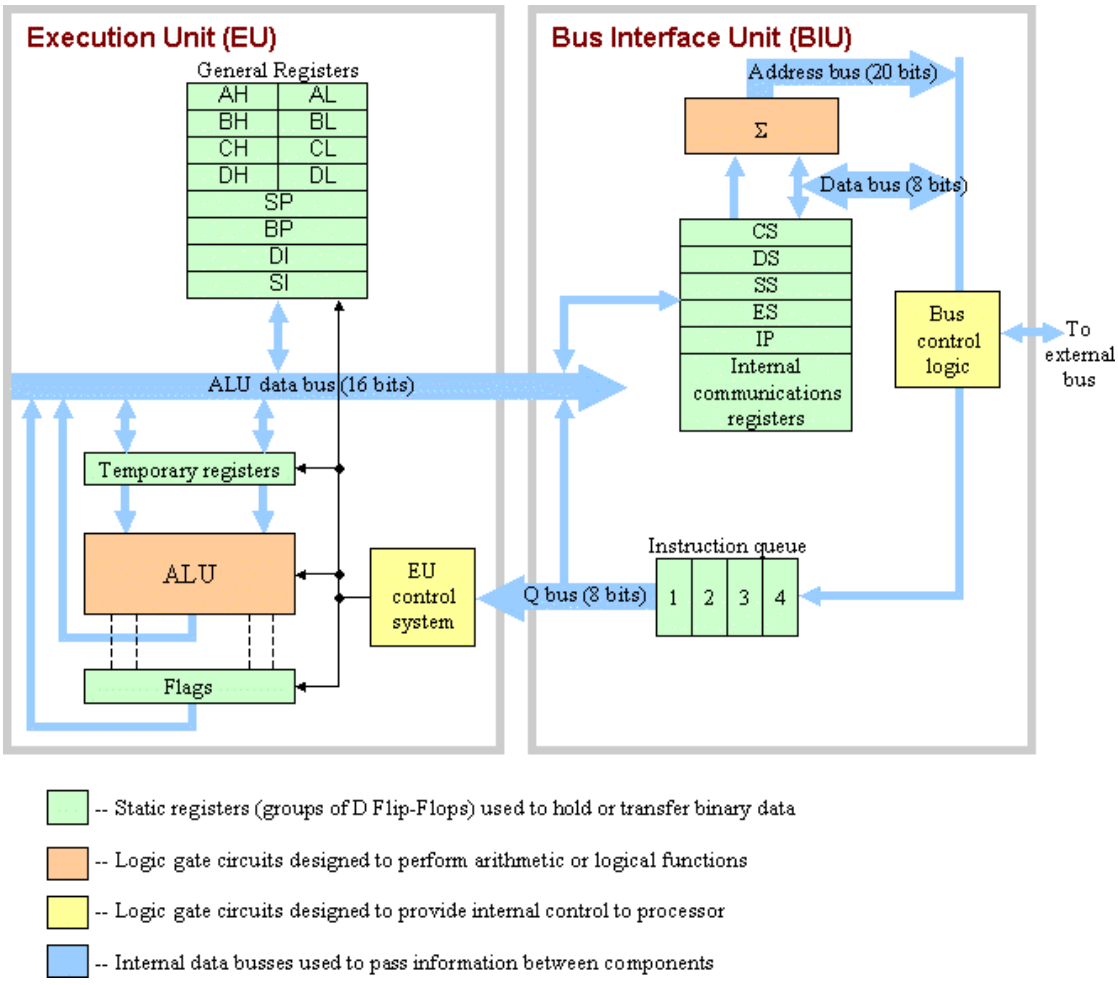
۴-۳- پروسسور 8088

شکل ۴-۵ بلوک دیاگرام پروسسور ۸۰۸۸ را نشان می‌دهد. همانطور که شکل نشان می‌دهد، این پروسسور از نظر منطقی به دو واحد تقسیم می‌شود. واحد اجرایی (EU) و واحد رابط باس(BIU) . وظیفه EU اجرای دستورات دریافتی از BIU است. واحد اجرایی شامل قسمتهای محاسباتی و واحد منطقی (ALU)، واحد کنترل و تعدادی رجیستر است که این قسمتها به منظور اجرای دستورات، عملیات حسابی و منطقی فراهم شده‌اند. مهمترین عمل BIU، مدیریت کنترل باس، رجیسترهای سگمنت و صف دستورالعملها است. مدیریت کنترل باس شامل کنترل عبور اطلاعات بین EU و دستگاههای خارج از پروسسور نظیر حافظه و I/Oها است.

BIU دستورالعملها را از حافظه واکشی نموده و آنها را در یک صف (آماده برای اجرا) قرار می‌دهد. اندازه این صف بسته به نوع پروسسور متفاوت بوده و در ۸۰۸۸ برابر ۴ است. اگرچه EU وBIU با هم بطور موازی عمل می‌کنند ولی همواره BIU یک قدم جلوتر از EU است. اگر BIU نیازمند دستیابی به داده‌ها در حافظه‌یا یک دستگاه جانبی باشد EU آنرا آگاه می‌سازد. همچنین EU دستورالعملها را از صف دستورالعمل BIU مطالبه می‌کند. بالاترین دستورالعمل صف، دستورالعمل قابل اجرای جاری می‌باشد. زمانیکه EU در حال اجرای یک دستورالعمل است، BIU دستورالعمل بعدی را از حافظه واکشی می‌کند. به خاطر همزمانی این واکشی با اجرا، سرعت پردازش افزایش می یابد.

23 - Setup

منطق کنترل باس (Bus Control Logic)، مجموعه‌ای از گیتها است که دستیابی به باس خارجی پروسسور ۸۰۸۸ را کنترل می‌کند. این موضوع شامل حافظه‌های خارجی، دستگاههای ورودی/خروجی و کلیه‌ی منابعی است که از طریق باس با پروسسور تبادل اطلاعات دارند.



شکل۴-۵- بلوک دیاگرام میکروپروسسور 8088

ALU بخش محاسباتی واحد EU است. هر وقت نیاز به انجام عملیات ریاضی یا منطقی بر روی اعداد باشد، این اعداد از طریق رجیسترهای همه منظوره به سمت ALU فرستاده می‌شوند. ALU بر روی آنها عمل ریاضی یا منطقی مورد نظر را انجام داده و سپس نتیجه عملیات را به رجیسترها بر می‌گرداند.

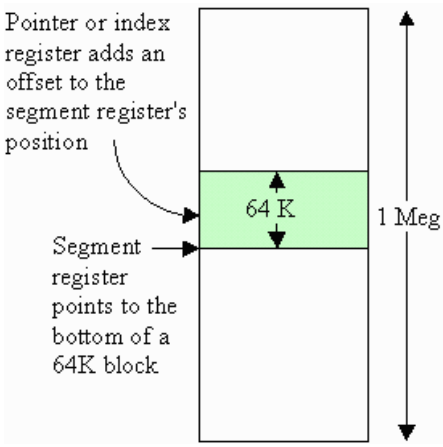
سیستم کنترل واحد اجرا (EU Control System)، مجموعه‌ای از گیتهاست که زمانبندی و عبور دیتا در واحد اجرا را کنترل می‌کند.

اجرای هر دستورالعمل کد ماشین، شامل سه مرحله است

۱- واکشی دستورالعمل که طی آن دستورالعمل بعدی که باید اجرا شود از حافظه بازیابی می‌شود. در ۸۰۸۸ این کار توسط منطق کنترل باس صورت می‌گیرد.

۲- رمزگشایی دستورالعمل که مشخص می‌کند کدام مدارها جهت اجرای دستور واکشی شده مورد نیاز هستند. در ۸۰۸۸ این کار توسط سیستم کنترل واحد اجرا انجام می‌شود.
۳- اجرای دستورالعمل که این کار توسط ALU انجام می‌شود.

۴-۳-۱- مفهوم سگمنت در ۸۰۸۸ : باس داده در ۸۰۸۸ شانزده بیتی و باس آدرس ۲۰ بیتی است. حال این سؤال پیش می‌آید که چگونه می‌شود از داده ۱۶ بیتی، آدرس ۲۰ بیتی درست کرد. به این منظور، پروسسور ۸۰۸۸ آدرس را از ترکیب دو رجیستر ۱۶ بیتی بدست می‌آورد. گرچه حافظه قابل آدرس‌دهی در ۸۰۸۸ یک مگابایت است (2^{20}=1MB) اما این فضا به قطعات ۶۴ کیلوبایتی تقسیم شده است که هر قطعه را یک سگمنت گویند.



شروع قطعه (که آدرس آن بر ۱۶ قابل قسمت است)، همراه با افست (فاصله شروع قطعه تا محل مورد نظر)، مجموعاً آدرس را تشکیل می‌دهند.
رجیسترهای سگمنت به شروع (ابتدای) هر سگمنت اشاره می‌کنند و پویش فضای ۶۴ کیلوبایتی داخل هر سگمنت با رجیسترهای افست (شامل رجیستر اشاره‌یا رجیستر ایندکس) صورت می‌گیرد.
به این نوع آدرس، آدرس نسبی، سگمنتی یا افستی نیز می‌گویند مثلاً 23B8:12A6 .

آدرس فیزیکی یک موقعیت حافظه، از ترکیب یک رجیستر سگمنت با یک رجیستر اشاره یا یک رجیستر ایندکس بدست می‌آید. رجیسترهای سگمنت شامل DS،ES، CS، SS ، رجیسترهای اشاره شامل IP،SP و رجیسترهای ایندکس شامل SI، DI است. به صورت زیر:

- CS:IP -- **code segment:instruction pointer** points to the physical address of the next instruction in memory to execute.
- SS:SP -- **stack segment:stack pointer** points to the stack in memory، a temporary storage place for data.
- DS:DI -- **data segment:destination index** points to the physical address in memory where data is to be **stored** using a pointer.
- DS:SI -- **data segment:source index** points to the physical address in memory where data is to be **retrieved** using a pointer.

برای بدست آوردن آدرس فیزیکی یک موقعیت حافظه (که با این روش آدرس‌دهی می‌شود) به این صورت عمل می‌شود که ابتدای محتوای رجیستر سگمنت، ۴ بیت به سمت چپ شیفت داده می‌شود (در ۱۶ ضرب می‌شود) سپس مقدار آن با مقدار رجیستر اشاره یا رجیستر ایندکس جمع می‌شود. به عنوان مثال اگر مقدار رجیستر CS برابر 3241 H و مقدار رجیستر IP برابر A34E H باشد، آدرس فیزیکی دستورالعمل بعدی در حافظه (که برای اجرا باید فراخوانی شود) به صورت زیر بدست می‌آید.

```
  0011 0010 0100 0001 0000   (hexadecimal 32410)
+     1010 0011 0100 1110    (hexadecimal A34E)
  0011 1100 0111 0101 1110   (hexadecimal 3C75E)
```

این محاسبات در داخل بلوک Address Summing Block (Σ) صورت می‌گیرد که مستقیماً در بالای رجیسترهای سگمنت (در واحد BIU) قرار گرفته است. بنابراین پروسه دستیابی به‌یک موقعیت حافظه در پروسسور ۸۰۸۸ شامل ۳ مرحله زیر است.

- a 16-bit **segment address** contained in one of the segment registers;
- a 16-bit **offset address** contained in a pointer or index register; and
- a 20-bit physical address which is the output from the address summing block.

این گونه برخورد با روش تهیه آدرس دو مزیت دارد.

۱- تهیه آدرس ۲۰ بیتی با استفاده از داده‌های ۱۶ بیتی

۲- امکان بار کردن (Load) کردن چندین برنامه در یک زمان [۲۴] که با عنوان relocatable code گفته می‌شود. در این حالت، هر برنامه خودش کنترل رجیسترهای اشاره‌گر و ایندکس را بر عهده دارد و سیستم عامل، رجیسترهای سگمنت را کنترل می‌کند. سیستم عامل، هر برنامه را مجبور می‌کند که در سگمنت مشخصی مقیم شود.

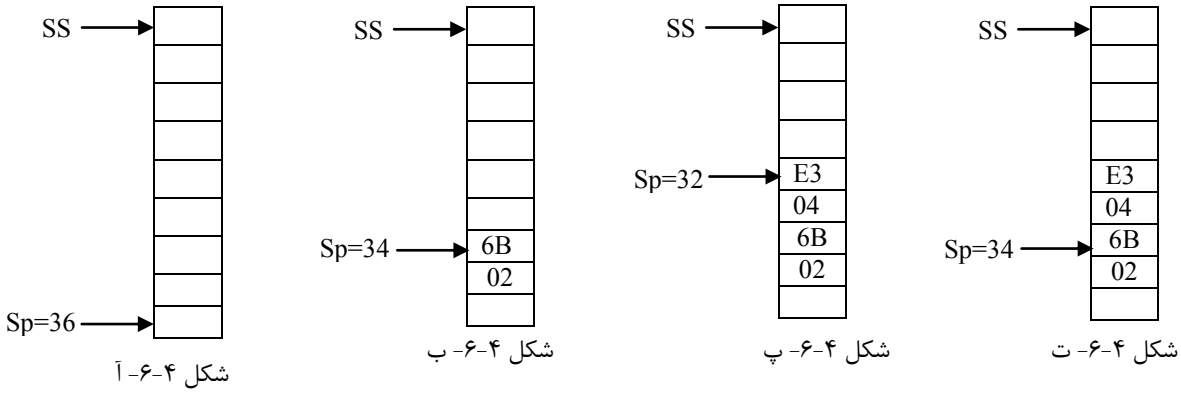
در یک برنامه اسمبلی با قالب EXE، سه نوع سگمنت تعریف می‌شود سگمنت پشته[۲۵]، سگمنت داده[۲۶] و سگمنت کد[۲۷] که به ترتیب با رجیسترهای SS، DS و CS به ابتدای این سگمنتها اشاره می‌شود. در برخی برنامه‌ها ممکن است به سگمنت دیگری که با عنوان سگمنت داده‌های اضافی [۲۸] خوانده می‌شود نیز نیاز باشد که با رجیستر ES به ابتدای آن اشاره می‌شود. بیشترین مقدار هر یک از سگمنتها ۶۴ کیلوبایت است.

سگمنت پشته محل ذخیره‌سازی موقتی اقلام داده و آدرس است. در برنامه‌های اسمبلی با فرمت COM، بارگذار برنامه، به طور اتوماتیک پشته‌ی برنامه را تعریف می‌کند ولی در برنامه‌های EXE. برنامه نویس باید صریحاً پشته را تعریف کند. هر عنصر داده در پشته یک کلمه (دو بایت) است. بارگذار برنامه رجیستر SS را با آدرس شروع پشته مقداردهی می‌کند. در ابتدا، مقدار رجیستر SP برابر اندازه‌ی پشته است یعنی به آخرین بایت انتهای پشته اشاره می‌کند. پشته با بقیه‌ی سگمنتها در روش ذخیره‌سازی داده تفاوت دارد. پشته، داده‌ها را از بالاترین موقعیت سگمنت آغاز کرده و به سمت پایین حافظه ادامه می‌دهد. دستورات PUSH و POP دو دستوری هستند که محتویات رجیستر SP را تغییر می‌دهند و برای ذخیره‌سازی داده و بازیابی آن بکار می‌روند. دستور PUSH باعث ذخیره کردن یک کلمه در پشته و کاهش عدد ۲ از SP می‌شود. دستور POP با بازیابی یک مقدار دوبایتی از پشته و اضافه کردن عدد ۲ به SP (اشاره به موقعیت کلمه‌ی ذخیره شده بعدی) انجام می‌شود. به عنوان مثال فرض کنید رجیسترهای AX و BX به ترتیب حاوی مقادیر 026B H و 04E3 H باشند و SP حاوی مقدار ۳۶ باشد (یعنی پشته ای با عمق ۳۶ بایت تعریف شده باشد) ابتدا پشته خالی است و مانند شکل ۴-۶-آ به نظر می رسد. می خواهیم نتیجه اجرای دستورات زیر را بر روی حافظه پشته بررسی کنیم.

```
PUSH AX
PUSH BX
POP BX
POP AX
```

24 - load multiple programs at one time
25 - Stack Segment
26 - Data Segment
27 - Code Segment
28 - Extra Segment

با اجرای اولین دستور، ۲ واحد از SP کم شده (SP=34 شده) و محتوای AX بصورت بایت معکوس در موقعیتهای ۳۴ و ۳۵ حافظه پشته ذخیره می‌شود (شکل ۴-۶-ب). اجرای دستور دوم باعث کاهش مجدد SP به اندازه ۲ واحد شده (SP=32 شده) و محتوای BX نیز بصورت بایت معکوس در موقعیتهای ۳۲ و ۳۳ حافظه پشته ذخیره می‌گردد (شکل ۴-۶-پ). اگر دستور سوم یعنی POP BX اجرا شود، ۲ واحد به SP اضافه شده(SP=34 شده) و محتوای SP-1 و SP-2 (در اینجا خانه های ۳۳ و ۳۲) به عنوان بایتهای بالا و پایین در رجیستر BX ذخیره می‌شود



شکل ۴-۶- آ شکل ۴-۶- ب شکل ۴-۶- پ شکل ۴-۶- ت

که با این کار همان داده اولیه BX به آن برگشت داده می‌شود (شکل ۴-۶-ت). بالاخره اجرای دستور چهارم باعث SP=36 شده و داده اولیه رجیستر AX به آن برگشت داده می‌شود.

چنانکه ملاحظه شد، دستور POP به ترتیب معکوس از دستور PUSH کد می‌شود بطوری که در مثال فوق رجیسترهای AX و BX بر روی پشته قرار گرفتند در حالیکه BX و AX از پشته برداشته شدند. همچنین مقادیری که بر روی پشته قرار گرفتند هنوز در آنجا هستند اگرچه SP دیگر به آنجا اشاره نمی‌کند. دستورات PUSHF و POPF که عمل ذخیره و بازیابی محتویات رجیستر پرچم را انجام می‌دهند، از جمله دستورهای دیگری هستند که مقادیر بر روی پشته قرار می‌دهند یا از آن بر می‌دارند. در مدلهای ۸۰۲۸۶ و بالاتر دستورهای عمومی PUSHA و POPA عمل ذخیره و بازیابی کلیه‌ی رجیسترهای عمومی را انجام می‌دهند.

مثال: دستورات زیر را در محیط DEBUG وارد کرده و نتیجه کار را ببینید تا بیشتر با دستور PUSH آشنا شوید.

```
-A100
0AFA:0100 MOV AX,1234
0AFA:0103 MOV DI,2B36
0AFA:0106 MOV SP,1236
0AFA:0109 PUSH AX
0AFA:010A PUSH DI
0AFA:010B NOP
```

برای مشاهده محتوای محدوده حرکت SP از دستور D 1230 123F استفاده کنید.

```
-D 1230 123F
0AFA:1230  5E 05 36 2B 34 12 26 8A-0D 47 C6 06 B2 91 00 C3
```

تمرین: دستورات زیر را در محیط DEBUG تایپ کرده و پس از اجرای هر دستور مقدار SP و محتوای خانه‌ای را که به آن اشاره می‌کند ببینید تا با مفهوم PUSH و POP بیشتر آشنا شوید.

```
-A100
0AFA:0100 MOV AX,1234
0AFA:0103 MOV BX,1235
0AFA:0106 MOV SP,1236
0AFA:0109 PUSH AX
0AFA:010A PUSH BX
0AFA:010B POP AX
0AFA:010C POP BX
0AFA:010D
```

سگمنت داده محل ذخیره داده‌هایی است که دستورات به آن مراجعه می‌کنند. رجیستر DS آدرس شروع سگمنت داده را مشخص می‌کند. عملوند یک دستور بر افست سگمنت داده‌ای که به آن مراجعه می‌شود دلالت دارد.

سگمنت کد، شامل دستوراتی است که باید اجرا شود. آدرس شروع سگمنت کد در رجیستر CS قرار دارد و رجیستر IP بر افست دستور جاری در سگمنت کد دلالت دارد که باید اجرا شود.

جدول زیر رجیسترهای سگمنت و رجیسترهای افست متناظر به آنها را نشان می‌دهد.

Segment Registers	CS	DS	ES	SS
Offset Register	IP	SI,DI,BX	SI,DI,BX	SP,BP

۴-۳-۲- رجیسترهای داخلی 8088 و کاربرد آنها

پروسسور 8088 همانند بقیه پروسسورها دارای تعدادی رجیستر داخلی است که هر کدام کاربرد ویژه‌ای دارد. برای دستیابی به آدرس حافظه و انجام دستورالعملها و عملیات مربوط به آنها از رجیسترهای داخلی استفاده می‌شود. این رجیسترها را می‌توان به دسته‌های زیر تقسیم کرد.

الف- رجیسترهای سگمنت[۲۹]: هر رجیستر سگمنت برای آدرس‌دهی یک ناحیه از حافظه بکار می‌رود که به عنوان سگمنت جاری شناخته می‌شود. چون یک سگمنت از مرز پاراگراف شروع می‌شود، همیشه ۴ بیت صفر در سمت راست آن در نظر گرفته می‌شود. رجیسترهای سگمنت شامل رجیستر سگمنت کد (CS)، رجیستر سگمنت داده (DS)، رجیستر سگمنت پشته(SS) و رجیستر سگمنت اضافی(ES) هستند.

رجیستر CS دارای آدرس شروع سگمنت کد است. این آدرس بعلاوه مقدار افست در اشاره‌گر دستورالعمل (IP) مشخص کننده آدرس دستورالعملی است که جهت اجرا واکشی می‌شود.

رجیستر DS دارای آدرس شروع سگمنت داده است. این آدرس بعلاوه یک افست (در رجیسترهای DI،SI یا BX) سبب ارجاع به یک مکان مشخص از سگمنت داه‌ها می‌شود.

با رجیستر SS می‌توان یک پشته در حافظه مهیا کرد که یک برنامه برای ذخیره موقت آدرس و داده از آن استفاده کند. سیستم، آدرس شروع پشته را در رجیستر SS می‌نویسد. این آدرس بعلاوه یک افست در رجیستر اشاره‌گر پشته (SP)، کلمه جاری در پشته را آدرس‌دهی می‌کند.

29 - Segment Registers

برخی اعمال رشته‌ای، **رجیستر ES** را برای دستکاری آدرس‌دهی حافظه استفاده می‌کنند. اگر این رجیستر مورد نیاز باشد باید توسط برنامه اسمبلی ارزشدهی شود.

نکته: در پروسسورهای 80386 به بالا رجیسترهای سگمنت اضافی دیگری نظیر FS و GS نیز وجود دارند.

ب- رجیسترهای اشاره‌گر[30]: شامل سه رجیستر ۱۶ بیتی BP،SP،IP هستند (در پروسسورهای 80386 به بالا این رجیسترها ۳۲ بیتی بوده و با نامهای EBP،ESP،EIP هستند). رجیستر IP حاوی افست آدرس دستور بعدی جهت اجرا است(در بعضی پروسسورها این رجیستر با عنوان PC خوانده می‌شود). به عنوان مثال اگر رجیستر CS حاوی 39B4 H و IP حاوی 514H باشد، برای یافتن دستور بعدی جهت اجرا، پردازشگر، آدرس CS و افست IP را بصورت زیر ترکیب می‌کند.

آدرس سگمنت در CS	39B40 H +
افست IP	514 H
آدرس دستور بعدی	3A054 H

یعنی دستور بعدی که بایستی اجرا شود، در خانه 3A054 H حافظه قرار دارد.

رجیستر اشاره‌گر پشته (SP) و اشاره‌گر پایه (BP) با رجیستر SS در ارتباطند و به سیستم اجازه می‌دهند تا داده‌ها را در سگمنت پشته دستیابی کند. به عنوان مثال اگر رجیستر SS حاوی 4BB3 H و رجیستر SP حاوی افست 412H باشد، برای یافتن آدرس کلمه جاری در پشته، پردازشگر، آدرس SS و افست SP را بصورت زیر ترکیب می‌کند.

آدرس سگمنت در SS	4BB30 H +
افست SP	412 H
آدرس پشته	4BF42 H

از رجیستر اشاره‌گر پایه (BP) برای آدرس دهی داده در پشته استفاده می‌شود. معمولاً در دستورات Call و Return، شامل شروع پشته‌ی سابروتین جاری است. پردازشگر، آدرس SS و افست BP را مشابه دو مثال قبل ترکیب می‌کند.

پ- رجیسترهای عمومی: شامل چهار رجیستر ۱۶ بیتی DX[31] و AX[32]، BX[33]، CX[34] هستند که برای انجام پردازشهای محاسباتی استفاده می‌شوند (در پروسسورهای 80386 به بالا علاوه بر این رجیسترها، یک نسخه توسعه‌یافته آنها که ۳۲ بیتی بوده و با نامهای EDX EAX،EBX،ECX، است نیز وجود دارد). هر کدام از این رجیسترها از دو بخش ۸ بیتی تشکیل شده‌اند که با آدرس‌دهی مناسب، دسترسی به هر کدام از بخشهای پایین و بالای آنها امکانپذیر است. به عنوان مثال، رجیستر AX شامل رجیسترهای AL و AH است که AL هشت بیت پایین و AH هشت بیت بالای آنرا تشکیل می‌دهند. به همین ترتیب رجیسترهای BX، CX،DX به ترتیب شامل رجیسترهای هشت بیتی BL و BH ، CL و CH ، DL و DH هستند. به عنوان مثال دستور اسمبلی MOV AX،00 مقدار ۱۶ بیت صفر در رجیستر AX و دستور MOV BH،00 مقدار ۸ بیت صفر در رجیستر BH قرار می‌دهند.

رجیستر AX رجیستر آکومولاتور است که برای اعمال ورودی، خروجی، بعضی اعمال رشته‌ای و اعمال حسابی استفاده می‌شود.

30 - Pointer Registers
31 - Data Register
32 - Accumulator Register
33 - Base Register
34 - Counter Register

رجیستر BX رجیستر پایه است که می‌تواند به عنوان یک شاخص برای توسعه آدرس‌دهی استفاده شود. از آن به منظور محاسبات نیز استفاده می‌شود.

رجیستر CX رجیستر شمارنده است که برای کنترل کردن تعداد دفعات تکرار یک حلقه استفاده می‌شود. از آن به منظور محاسبات نیز استفاده می‌شود.

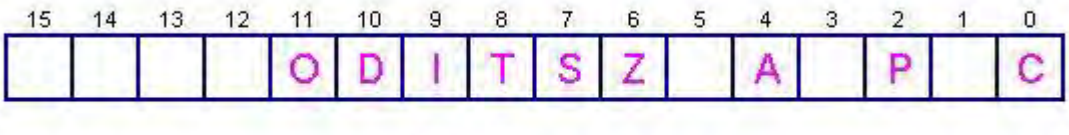
رجیستر DX به رجیستر داده معروف است و برای برخی اعمال ورودی، خروجی استفاده می‌شود. اعمال ضرب و تقسیمی که نتیجه آنها بزرگتر از ۱۶ بیت است، نیز از جفت رجیستر DX،AX استفاده می‌کنند بطوریکه DX، ۱۶ بیت وزن بالا را می‌سازد.

ت- رجیسترهای شاخص: شامل دو رجیستر ۱۶ بیتی DI وSI است که برای آدرس‌دهی شاخص و جمع و تفریق استفاده می‌شوند.

رجیستر SI به عنوان شاخص مبدأ شناخته شده و برای برخی اعمال رشته‌ای مورد نیاز است و با رجیستر DS مرتبط است.

رجیستر DI به عنوان شاخص مقصد شناخته شده و برای برخی اعمال رشته‌ای مورد نیاز است و با رجیستر ES مرتبط است.

ث- ثبات پرچم ها: ثبات پرچم ها در ریزپردازنده ۸۰۸۸/۸۰۸۶ نیز همانند سایر ثباتهای آن ۱۶ بیتی است. تک تک بیت های آن به صورت مستقل تحت تاثیر عملیات محاسباتی یا منطقی که در ALU انجام می‌شود قرار می گیرند. البته ۳ بیت پرچم کنترلی هم در آن قرار داده شده که می‌توان آنها را صفر و یا یک نمود و دستوراتی در ریزپردازنده وجود دارد که با توجه به مقادیر این سه بیت کارهای متفاوتی انجام می دهند.



ثبات پرچم های 8086/8088

همانطورکه در شکل مشاهده می‌شود فقط ۹ بیت از ۱۶ بیت ثبات پرچم ها استفاده شده و بقیه آنها بلااستفاده است و به صورت عادی موقع خوانده شدن صفر خوانده می‌شود. از نظر ساختمان ثبات پرچم ها، این بیت ها طوری در نظر گرفته شده اند که ۸ بیت کم ارزش آن با بیت های ثبات پرچم ها در ۸۰۸۵ یکسان و سازگار باشد.

در ریزپردازنده ۸۰۸۸/۸۰۸۶ **بیت های پرچم به دو دسته تقسیم می‌شوند: دسته اول بیت های پرچم کنترل و دسته دوم بیت های پرچم وضعیت.** بیت های پرچم کنترل سه بیت D ، I،T هستند که عمل کنترل ریزپردازنده را انجام می دهند. بدین صورت که قبل از اجرای بعضی دستورات (که این بیت ها از آنها استفاده می کنند) باید مقادیر این سه بیت تعیین شده باشد. بیت های پرچم وضعیت تحت تأثیر آخرین عمل محاسباتی یا منطقی انجام یافته در ALU قرار می گیرند و عبارتند از : S، Z ، P ، A، O ، C. بنابراین مقادیر این بیت ها بعد از اجرای دستورات (در اثر اجرای آنها) مشخص می‌شود. حال به بررسی تک تک بیتهای رجیستر پرچم می‌پردازیم:

پرچم C

پرچم نقلی بوده و به اختصار CF نامیده می‌شود. مشخص کننده بیت نقلی (انتقالی) خروجی مربوط به با ارزش ترین بیت در طی عملیات محاسباتی است. بیت شیفت یافته در عملیات جابجایی نیز در این پرچم قرار می گیرد. این بیت همچنین مشخص کننده بیت قرضی در عملیات تفریق نیز هست. عملیات ممکن است به صورت ۸ بیت یا ۱۶ بیت باشد.

اگر پس از عملیات محاسباتی بیت نقلی یا بیت قرضی وجود داشته باشد، این پرچم یک خواهد شد، در غیر این صورت صفر می‌شود. با دو دستور JC و JNC می‌توان این پرچم را تست کرد.

پرچم P

پرچم توازن است و به اختصار آن را با PF نشان می‌دهیم و مشخص کننده وضعیت توازن هشت بیت کم ارزش حاصل عملیات است. توازن زوج بر روی یک ها محاسبه می‌شود یعنی اگر هشت بیت کم ارزش نتیجه عملیات دارای تعداد زوجی از یک ها باشد این پرچم برابر یک خواهد شد و در غیر این صورت صفر می‌شود. از این بیت برای بررسی خطا های انتقال داده ها استفاده می‌گردد و همانطورکه گفته شد فقط برای هشت بیت کم ارزش داده ها قابل استفاده است. با دو دستور JP و JPO می‌توان این فلگ را تست کرد.

پرچم A

بیت پرچم نقلی کمکی نامیده می‌شود و به اختصار آن را با AF نشان می‌دهیم. مشخص کننده بیت نقلی کمکی یا بیت قرضی کمکی مربوط به چهار بیت اول از هشت بیت کم ارزش حاصل عملیات است. از این بیت برای تصحیح و تطبیق عملیات محاسباتی دهدهی (BCD) استفاده می‌شود. در اعمال حسابی، اگر در موقع محاسبه از بیت چهارم به پنجم رقم نقلی داشته باشیم (در داده‌های ۸ بیتی) این بیت یک می‌شود.

پرچم O

ب نام بیت پرچم سرریز نامیده می‌شود و به اختصار آن را با OF نشان می‌دهیم و مشخص کننده سرریز در عملیات محاسباتی هشت یا شانزده بیتی است. اگر حاصل عملیات محاسباتی هشت بیتی در هشت بیت جا نشود و همچنین حاصل عملیات محاسباتی شانزده بیتی در شانزده بیت جا نشود سرریز اتفاق می افتد و این بیت یک خواهد شد و در غیر این صورت صفر است. در CPU ، بیت سرریز از حاصل XOR بیت های نقلی آخر و ماقبل آخر به دست می آید. (XOR بیت های نقلی وارد شده و خارج شده از باارزش ترین بیت)

پرچم Z

پرچم صفر نامیده می‌شود و به اختصار آن را با ZF نشان می‌دهیم و مشخص کننده صفر یا غیر صفر بودن حاصل عملیات است. پس از اتمام عملیات اگر حاصل عملیات برابر صفر باشد این بیت یک خواهد شد، در غیر این صورت این بیت صفر می‌شود.

پرچم S

پرچم علامت نامیده می‌شود و به اختصار آن را با SF نشان می‌دهیم. مشخص کننده علامت نتیجه عملیات است. یعنی بیت علامت حاصل عملیات در این بیت پرچم ذخیره می‌شود. لازم به ذکر است که در ریزپردازنده ۸۰۸۸/۸۰۸۶ همانند بسیاری از CPUهای معمول، اعداد منفی در فرمت متمم ۲ استفاده می‌شود و بنابراین آخرین بیت سمت چپ عدد بیانگر علامت آن خواهد بود. پس می‌توان گفت بیت پرچم S برابر باارزش ترین بیت حاصل عملیات است.

پرچم D

پرچم جهت نامیده می‌شود و با DF آن را نمایش می‌دهیم. تمام بیت های پرچم که تاکنون بررسی شد همه از نوع وضعیت بودند ولی این بیت، بیت پرچم کنترلی است. یعنی قبل از اینکه عمل مورد نظر انجام گیرد، باید مقدار این بیت مشخص شده باشد. این بیت در عملیات مربوط به دستورالعمل های رشته STRING به کار برده می‌شود و جهت افزایش یا کاهش آدرس ها (محتویات ثبات های شاخص) را تعیین می‌کند. در دستورالعمل های رشته، ثبات های شاخص به عنوان ثبات های آدرس های مبدا و مقصد به کار می‌رود و مقدار این بیت تعیین می‌کند که آیا بعد از هر عمل رشته، محتویات ثبات های شاخص یک (یا دو) واحد افزایش یابد، یا از مقدار آنها کم شود؟ اگر بیت پرچم D یک باشد آدرس ها کاهشی و اگر این بیت صفر باشد آدرس ها افزایشی است.

پرچم I

به نام پرچم وقفه نامیده می‌شود و به اختصار با IF نمایش داده می‌شود. این بیت وضعیت توانا یا ناتوان بودن وقفه ی پوشش پذیر را تعیین می‌کند. لازم به ذکر است که ریزپردازنده ۸۰۸۸/۸۰۸۶ دارای یک پایه ورودی وقفه (پوشش پذیر) است و با یک قرار دادن بیت IF می‌توان آن را توانا ساخت. اگر این بیت پرچم یک باشد و اگر وقفه ای به ریزپردازنده از طریق پایه وقفه پوشش پذیر وارد شود، ریزپردازنده وقفه را خواهد پذیرفت. اگر بیت پرچم وقفه قبل از وقوع وقفه صفر شده باشد، در صورت وقوع وقفه، به آن ترتیب اثر داده نخواهد شد.

پرچم T

به نام پرچم تله (Trapیا Trace) نامیده می‌شود و به اختصار آن را با TF نشان می‌دهیم. اگر این بیت یک گردد ریزپردازنده را در وضعیت یک مرحله ای قرار می دهد و با اجرای پله به پله (تک تک) دستورالعمل ها می‌توان به اشکال زدایی برنامه پرداخت. در حالت عادی این بیت برابر صفر است و برنامه ها به صورت معمولی اجرا می‌شود. وقتی در حالت یک مرحله ای قرار گرفت (TF برابر یک شد) پس از اجرای هر دستورالعمل، CPU به طور اتوماتیک یک وقفه داخلی تولید می‌کند تا برنامه به صورت دستور به دستور اجرا گردد. در پروسسور 8088، این پرچم بیت D8 رجیستر پرچم را تشکیل داده و برای فعال کردن آن معمولاً از دستورات زیر استفاده می‌شود.

```
PUSH F
POP AX
OR AX, 0000000100000000B
PUSH AX
POP F
```

مثال- اگر دو عدد باینری 10110101_2 و 10010110_2 را با یکدیگر جمع کنیم تأثیر آن بر فلگهای وضعیت را بررسی کنید.

```
  1 0 1 1 0 1 0 1
+ 1 0 0 1 0 1 1 0
1 0 1 0 0 1 0 1 1
```

- OF=1 -- There was an overflow, i.e., adding two negative numbers resulted in a positive number.

- SF=0 -- The result is positive.
- ZF=0 -- The result does not equal zero.
- AF=0 -- For now we won't worry about the auxiliary flag.
- PF=0 -- For now we won't worry about the parity flag.
- CF=1 -- There was a carry.

۴-۳-۳- مدهای آدرس‌دهی در ۸۰۸۸

وقتی ۸۰۸۸ دستورالعملی را انجام می‌دهد در حقیقت عملیاتی را بر روی داده‌ها انجام می‌دهد. این داده‌ها که عملوند [۳۵] خوانده می‌شوند ممکن است بخشی از یک دستور باشند، در رجیسترهای داخلی پروسسور مقیم باشند یا در یک آدرس حافظه ذخیره شده باشند. برای دسترسی به عملوندها ممکن است به شیوه‌های متفاوتی عمل شود که به آنها مدهای آدرس‌دهی می‌گویند. گرچه پروسسورهای ۸۰۲۸۶ و بالاتر دارای دو مد آدرس‌دهی حقیقی و حفاظت شده هستند ولی پروسسور ۸۰۸۸ فقط دارای عملکرد مد حقیقی است. در مد حقیقی به پروسسور اجازه آدرس‌دهی فقط یک مگابایت اول حافظه داده می‌شود(حتی در یک کامپیوتر پنتیوم). در آدرس‌دهی مد حقیقی، آدرس شامل یک رجیسترسگمنت بعلاوه یک رجیستر افست است (در مد حفاظت شده از مقدار رجیستر سگمنت به عنوان یک ایندکس جهت مراجعه به یک جدول تبدیل استفاده می‌شود). پروسسور ۸۰۸۸ در مد حقیقی(تنها مد کاری اش)، برای آدرس‌دهی عملوند از روشهای مندرج در جدول ۴-۸ استفاده می‌کند.

جدول ۴-۸- مدهای آدرس دهی در ۸۰۸۸

Addressing Mode	Operand	Default Segment
Immediate	Data	None
Direct	[offset]	DS
Register	Reg	None
Register Indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX]+disp [BP]+disp	DS SS
Indexed Relative	[DI]+disp [SI]+disp	DS DS
Based Indexed Relative	[BX][SI or DI]+disp [BP][SI or DI]+disp	DS SS

حال به شرح مختصر این مدها می پردازیم.

35 - Operand

۱- آدرس‌دهی بلافصل(Immediate Adressing Mode)

در این مد، عملوند بصورت مستقیم در خود دستور قرار دارد. مانند دستور MOV AX،1234H که به موجب آن مقدار 1234H به رجیستر AX منتقل می‌شود.

```
MOV AX، 1234h          ; Move to AX the value 1234h
MOV AX، X              ;  Move to AX the address or offset of the variable X
MOV AX، CONST          ; Move to AX the constant defined as CONST
```

۲- آدرس‌دهی مستقیم(Direct Adressing Mode)

در این مد، با پیش فرض DS به عنوان رجیستر سگمنت، افست آدرس عملوند در دستور مشخص می‌شود. به عنوان مثال در دستور [MOV AX،1234H] محتوای آدرس DS:1234H برداشته شده و در رجیستر AX قرار می‌گیرد.

مثال:

```
MOV AX، [X]            ; Move to AX the value in memory location DS:X
MOV [X]، AX            ; Move to the memory location pointed to by DS:X the value in AX
```

۳- آدرس‌دهی با رجیستر(Register Adressing Mode)

در این مد، عملوند در یکی از رجیسترهای پردازنده قرار دارد. به عنوان مثال دستورهای MOV AX،BX یا MOV ES، AX یا MOV AH،BL همگی نمونه‌هایی از آدرس‌دهی با رجیستر هستند.

مثال:

```
MOV AX، BX             ; Move to AX the 16-bit value in BX
MOV AX، DI             ; Move to AX the 16-bit value in DI
```

۴- آدرس‌دهی غیرمستقیم رجیستری(Register Indirect Adressing Mode)

در این حالت عملوند با استفاده از رجیسترهای [BP]،[SI]، [DI]، [BX] آدرس‌دهی می‌شود. رجیسترهایSI ،DI، BX با رجیستر DS برای پردازش داده‌های سگمنت داده (به شکل DS:BX، DS:DI و DS:SI) و رجیسترBP با رجیستر SS برای دستکاری داده‌های پشته (به شکل SS:BP) استفاده می‌شوند. به عنوان مثال دستور ADD CL، [BX] محتوای آدرس DS:BX را با محتوای رجیستر CL جمع کرده و در CL قرار می‌دهد.

مثال:

```
MOV AX، [BX]           ; Move to AX the 16-bit value pointed to by DS:BX
MOV [BP]، AX           ; Move to memory address SS:BP the 16-bit value in AX
```

مثال- فرض کنید DS=1120H، SI=2498H و AX=17FE باشند، پس از اجرای دستور MOV [SI]،AX مقدار موقعیت مورد نظر حافظه را مشخص کنید.

حل- در این حالت، آدرس مورد نظر بصورت DS:SI است یعنی آدرس 13698 H. که مقدار رجیستر AX باید در آن قرار گیرد. بنابراین مقدار FE در آدرس 13698 H و مقدار 17 در آدرس 13699 H قرار خواهد گرفت.

۵- آدرس‌دهی نسبی اندیسی(Indexed Relative Adressing Mode)

در این مد آدرس‌دهی، رجیسترهای DI،SI به همراه یک جابجایی ۸ یا ۱۶ بیتی مشخص کننده عملوند هستند. به عنوان مثال دستور MOV AH، 1234 H + [SI] یک نوع آدرس‌دهی نسبی اندیسی است. در این مد ابتدا افست DS:SI را پیدا کرده و با 1234 H جمع می‌کنیم تا آدرس عملوند بدست آید.

مثال:

```
MOV [DI]، AX        ; Move to address DS:DI the 16-bit value in AX
MOV AX، [DI]        ; Move to AX the 16-bit value pointed to by DS:DI
```

مثال: آدرس فیزیکی مربوط به دستور MOV [DI-8]،BL چیست اگر DS=200H و DI=30 H باشد؟

DS:DI = DS: 30 = 02030 H
PA = 02030 – 8 = 2028 H

۶- آدرس‌دهی نسبی مبنایی(Based Relative Adressing Mode)

در این حالت، عملوند با استفاده از رجیسترهای [BP]،[BX] به همراه یک جابجایی ۸ یا ۱۶ بیتی آدرس‌دهی می‌شود. رجیسترBX با رجیستر DS برای پردازش داده‌های سگمنت داده و رجیستر BP با رجیستر SS برای دستکاری داده‌های سگمنت پشته به کار می‌رود و داریم:

آدرس مؤثر = SS:[BP] + Displcement یا آدرس مؤثر = DS:[BX] + Displcement

مثال: آدرس فیزیکی مربوط به دستور MOV AH،1234 H+[BX] چیست اگرDS=1200 H و BX=130 H باشد؟

DS:DI = DS: 130 = 12130 H
PA = 12130 + 1234 = 13364 H

۷- آدرس‌دهی نسبی اندیسی- مبنایی(Based-Indexed Relative Adressing Mode)

در این نوع مد آدرس‌دهی، برای مشخص کردن آدرس عملوند، هم از رجیسترهای مبنای BP و BX و هم از رجیسترهای اندیسی SI وDI به همراه یک جابجایی ۸ یا ۱۶ بیتی استفاده می‌شود. مجموع مقادیر رجیسترهای مبنا و اندیس به عنوان افست آدرس عملوند در نظر گرفته شده و با مقدار جابجایی جمع می‌شود تا آدرس عملوند بدست آید.

$$PA = \begin{matrix}CS\\SS\\DS\\ES\end{matrix} : \begin{matrix}BX\\BP\end{matrix} + \begin{matrix}SI\\DI\end{matrix} + \begin{matrix}\text{8 bit displacement}\\\text{16 bit displacement}\end{matrix}$$

مثال: فرض کنید DS = 100, BX = 30, SI = 70 باشند. در اینصورت مد آدرس دهی 25 + [BX + SI] توسط پروسسور به صورت آدرس فیزیکی زیر محاسبه می شود.

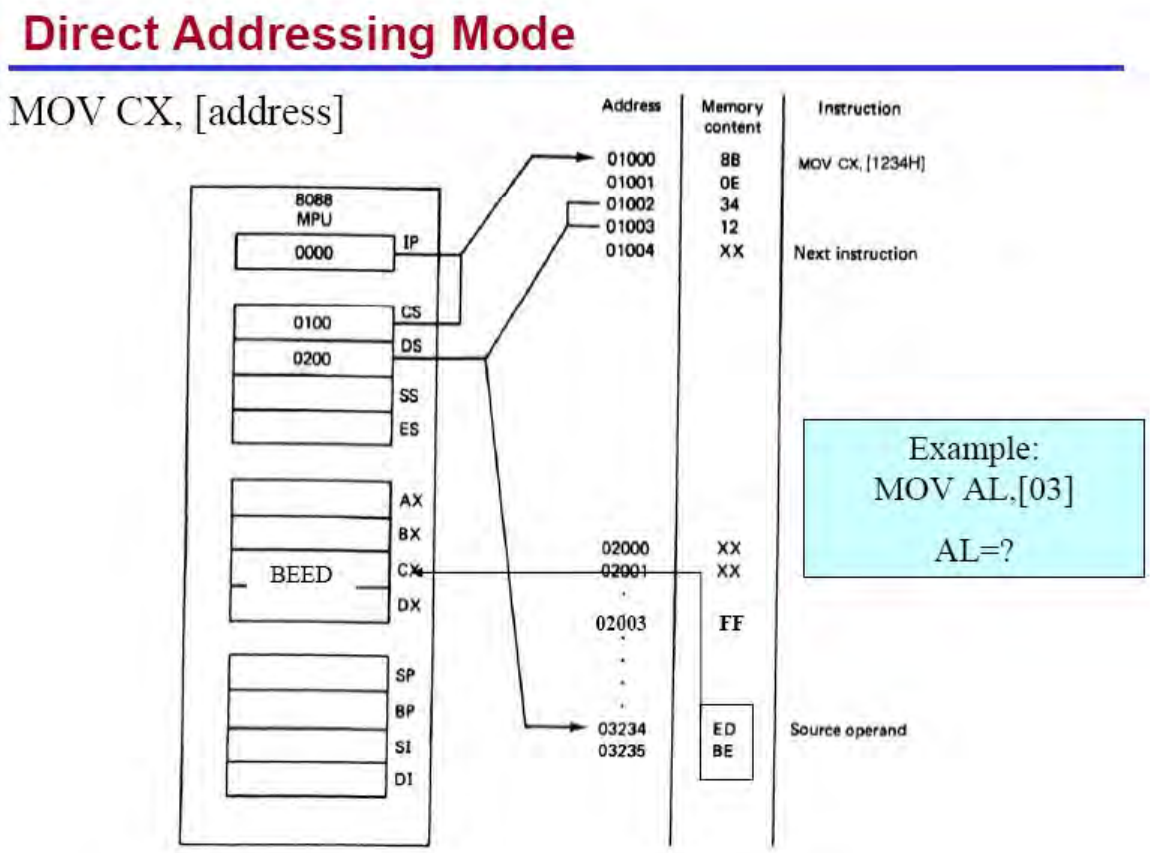
Pysical address = 100 * 16 + 30 + 70 +25 = 1725

در این مد آدرس دهی، به صورت پیش فرض همواره از رجیستر DS برای محاسبه آدرس فیزیکی استفاده می شود مگر برای رجیستر BP که از رجیستر SS برای محاسبه آدرس فیزیکی استفاده می شود.

مثال:

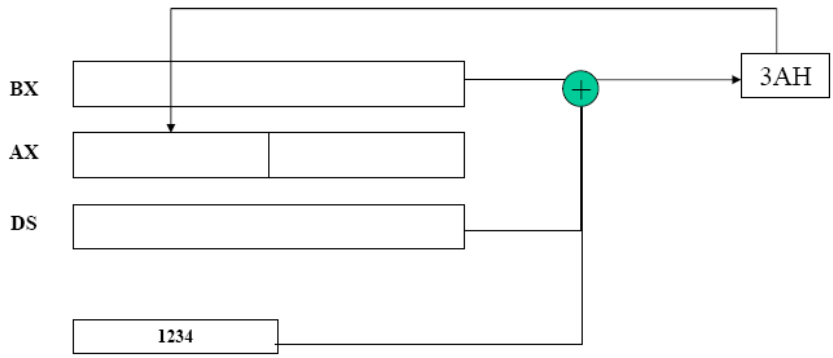
```
MOV AX، [BX + DI]              ; Move to AX the value in memory at DS: BX + DI
MOV [BX + DI]، AX              ; Move to the memory location pointed to by DS:BX + DI the value in AX
MOV AX، [BX + DI] + 1234h      ; Move word in memory location DS:BX + DI + 1234h to AX register
```

برای درک بهتر مدهای آدرس‌دهی به شکلهای زیر دقت کنید.



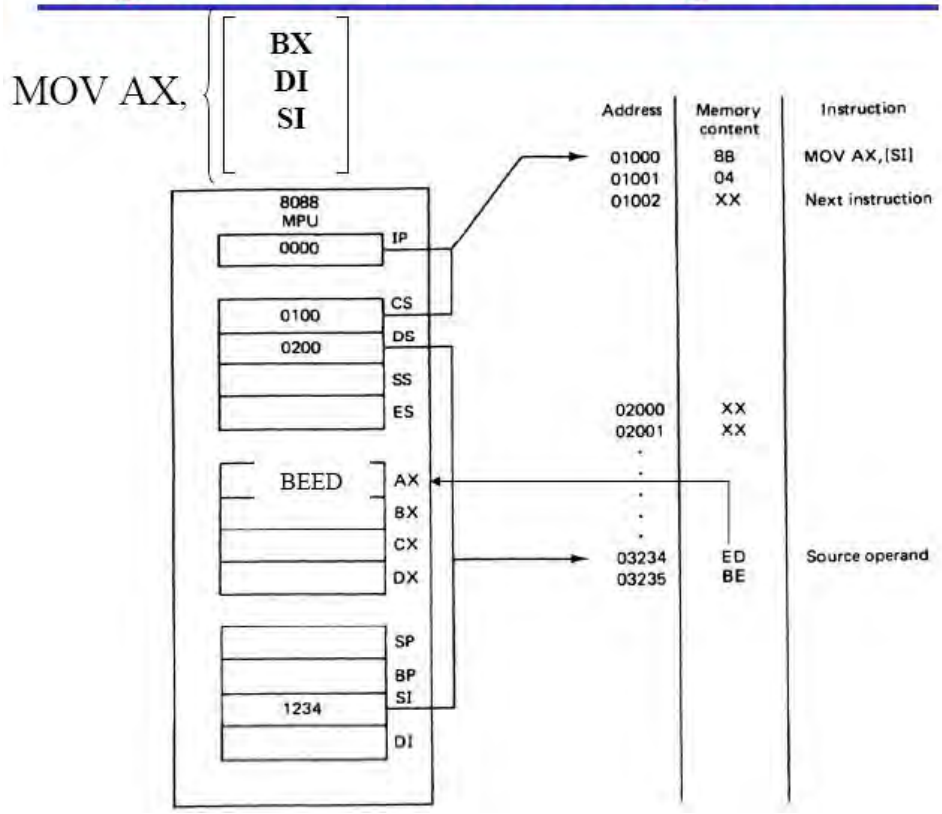
Based-Relative Addressing Mode

MOV AH, [DS:BX / SS:BP] + 1234h



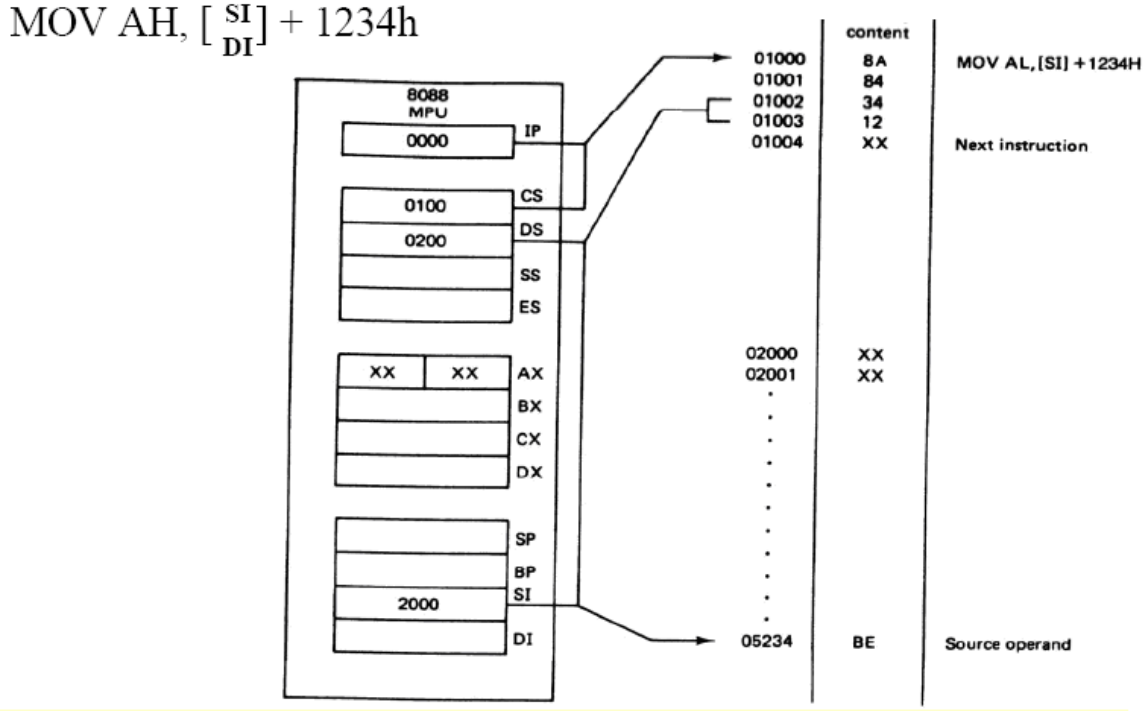
Register Indirect Addressing Mode

MOV AX, { BX / DI / SI }



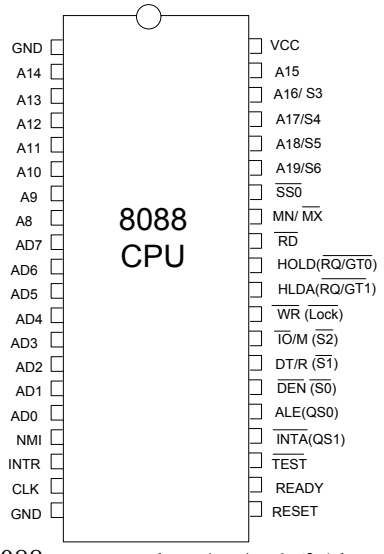
Indexed Relative Addressing Mode

MOV AH, [SI / DI] + 1234h



۴-۳-۴- پایه‌های پروسسور 8088

میکروپروسسورهای 8085، 8086 و 8088 آی‌سی‌هایی ۴۰ پایه هستند که برای عملکرد صحیح خود فقط به‌یک تغذیه 5+ ولت نیاز دارند. 8085 حداکثر جریان 170mA، 8086 حداکثر جریان 360mA و 8088 حداکثر جریان 340mA را استفاده می‌کنند. پایه‌های این میکروپروسسورها، قابلیت جریان دهی 2mA و قابلیت جریان کشی $400\mu A$ را دارند[۳۶]. به همین خاطر در بیشتر سیستمهای میکروپروسسوری پایه‌هایی نظیر خطوط آدرس و خطوط داده را که به بیش از یک گیت متصل می‌شوند بافر می‌کنند. پایه‌های میکروپروسسور 8088 در شکل ۴-۹ نشان داده شده است.



شکل۴-۹- پایه‌های میکروپروسسور 8088

توضیحات مربوط به این پایه‌ها بصورت زیر است.

- **پین ورودی MN/MX** : پروسسور ۸۰۸۸ بسته به وضعیت پین MIN/MAX می‌تواند در دو مد کار کند. اگر این پین حالت یک منطقی داشته باشد، پروسسور در مد مینیموم کار می‌کند. در این حالت پروسسور ۸۰۸۸ خودش به تنهایی کلیه سیگنالهای کنترلی مورد نیاز برای دستگاههای جانبی و حافظه‌ها را فراهم می‌کند. اگر پین MIN/MAX حالت صفر منطقی داشته باشد، پروسسور در مد ماکزیموم عمل می‌کند. این مد با بکارگیری باس کنترلر 8288، برای تولید سیگنالهای زمانبندی باس، امکان استفاده چندین پروسسور از باس سیستم را فراهم می‌کند. به عبارت دیگر، در مد ماکزیموم، چندین پروسسور کمکی دیگر (مثل coprocessor) نیز با پروسسور ۸۰۸۸ در تولید سیگنالهای کنترلی همکاری می‌کنند.

- **پین ورودی Reset** در پروسسور ۸۰۸۸ باعث صفر نمودن رجیسترهای DS،SS، ES و ست کردن مقدار CS به 0FFFF می‌شود. به این ترتیب، نخستین دستورالعملی که پروسسور (پس از ریست کردن) انجام می‌دهد دارای آدرس 0FFFF0 (یعنی ۱۶ خانه مانده به آخر حافظه) است. این پین باید حداقل به اندازه‌ی ۴ کلاک پالس فعال بماند تا عمل ریست صورت گیرد. علاوه بر این باید این سیگنال با پالس ساعت سیستم نیز سنکرون باشد که به این منظور از آی‌سی ۸۲۸۴ استفاده می‌شود.

۳۶- بخش ۱-۳ کتاب Microprocessor/Hardware Interfacing and Applications. Barry B. Brey

- **پین خروجی ALE** : در پروسسور ۸۰۸۸ از ۸ وزن پایین آدرس باس هم به عنوان آدرس و هم به عنوان داده استفاده می‌شود. در سیکل ماشین T1 و قسمتی از T2 سیگنال کنترلی ALE فعال شده و نشان می‌دهدکه خطوط AD0 تاAD7 حاوی بیتهای آدرس A0 تاA7 هستند.

- **پینهای خروجی INTA، WR،RD** سیگنالهای کنترلی هستند که نوع سیکل باس در حال اجرا را مشخص می‌کنند. به عنوان مثال اگر سیکل مربوط به خواندن (از حافظه‌یا از دستگاه جانبی) در حال اجرا باشد سیگنال کنترلی RD فعال خواهد بود. یا اگر سیکل جاری باس، سیکل نوشتن (در حافظه‌یا دستگاه جانبی) باشد سیگنال کنترلی WR فعال خواهد بود.

  در مد ماکزیموم، سیگنالهای کنترلی ALE، IO/M، WR،RD وضعیت CPU را برای باس کنترلر 8288 مشخص می‌کنند و باس کنترلر با استفاده از آنها، سیگنالهای کنترلی دیگری را تولید می‌کند.

- **پین ورودی TEST** : دستورالعملی به نام Wait در مجموعه دستورالعملهای پروسسور ۸۰۸۸ وجود دارد که وقتی پروسسور به این دستورالعمل می‌رسد به پایه‌ی TEST نگاه می‌کند. اگر پایه TEST فعال (صفر منطقی) باشد در حالت انتظار باقی می‌ماند تا این پایه غیر فعال شود. در حالت انتظار اگر INT یا NMI به سیستم وارد شود، پروسسور می‌رود و آنرا اجرا می‌کند و دوباره به حالت انتظار بر می‌گردد و تا زمانی که پایه‌ی TEST غیر فعال نشود در همین وضعیت باقی می‌ماند. از دستورالعمل Wait فقط با غیر فعال کردن ورودی TEST یا Reset کردن پروسسور می‌توان خارج شد. در حالت انتظار، جهت استفاده پروسسورهای دیگر نظیر ۸۰۸۷، پروسسور به وضعیت معلق[37] می‌رود.

- **پین DEN** : یک سیگنال خروجی از پروسسور است که مشخص می‌کند که خطوط باس در حال انتقال داده هستند.

- **پین HOLD** : فعال کردن این پایه توسط یک دستگاه جانبی به معنی تقاضای باس است و به موجب آن، باید پروسسور باس را رها کرده و در اختیار دستگاه جانبی (مثل DMA) قرار دهد. در برخی پروسسورها این پایه، عنوان BUSRQ را دارد.

- **پین HLDA** : یک سیگنال خروجی است که پروسسور با فعال نمودن آن به دستگاه جانبی اطلاع می‌دهد که باس را رها کرده و در اختیار پروسسور دیگری قرار داده است.

- **پین READY** : یک پین ورودی است که توسط بعضی دستگاههای جانبی کند مثل حافظه‌ها و برخی I/O ها استفاده می‌شود. اگر این پین فعال باشد (یک منطقی) به معنی این است که دستگاه جانبی آماده تبادل اطلاعات است.

- **پین LOCK** : این پایه‌ی خروجی کاربردش در حالت Multi Tasking و Multi User است. Task به زیربرنامه‌هایی گفته می‌شود که تا حدی به هم مربوطند ولی زیاد بزرگ نیستند. برای افزایش سرعت کار می‌آیند و یک کار را بصورت موازی توسط چند پروسسور انجام می‌دهند ولی انجام کار بصورت کاملاً موازی امکانپذیر نیست زیرا گاهی اوقات باید نتیجه‌ی مربوط به‌یک قسمت تمام شود تا پردازش روی قسمت دیگر انجام شود. لذا در یک محدوده خاصی از Task که منطقه بحرانی نامیده می‌شود، پروسسور نباید باس را در اختیار پروسسور دیگری قرار دهد که بدین منظور در اول دستورالعملهای مربوط به منطقه بحرانی پیشوند LOCK را قرار می‌دهند. وجود این پیشوند باعث می‌شود که در این

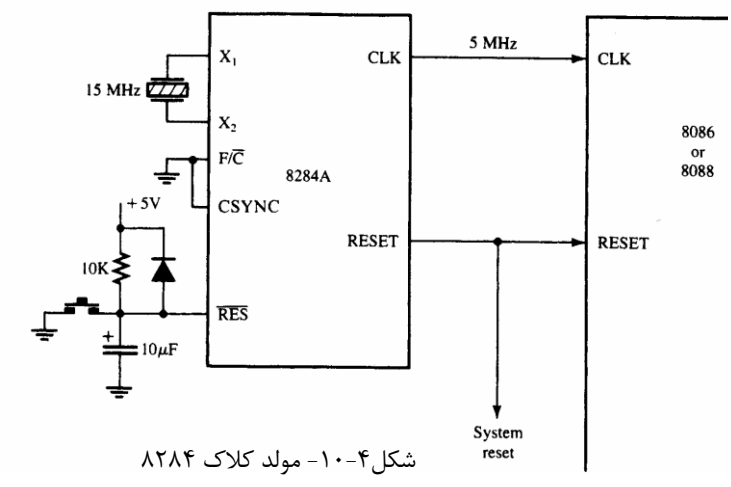
37 - Idle

منطقه کنترل به Task دیگر یا پروسسور دیگری واگذار نشود. با اجرای دستورالعملهای منطقه بحرانی (که پیشوند LOCK دارند) پایه‌ی LOCK فعال (صفرمنطقی) شده و پروسسورهای دیگر را متوجه می‌کند که تقاضای باس نکنند.

```
Task1                    Task2          Task3
....      Critical Region
                         ....           ....
                         ....           ....
LOCK MOV AX, …
LOCK ADD … , …
....                     ....           ....
.....
.....
.....                    ....           ....
```

- **پین INTR** : یک پین ورودی است که دستگاههای جانبی با فعال کردن آن تقاضای اینتراپت (و در نتیجه اجرای یک سابروتین مخصوص به خود را) دارند. پروسسور ۸۰۸۸ دارای ۲۵۶ نوع اینتراپت است که نوع دوم آن از نوع غیرقابل چشم پوشی (NMI) است.
- **پین NMI**: یک پین ورودی است که با فعال کردن آن دستگاه جانبی تقاضای اینتراپت از نوع غیرقابل چشم پوشی را دارد. این اینتراپت، Type 2 اینتراپت بوده و لذا آدرس مربوط به آن در خانه‌های 08 تا 0B H حافظه قرار دارد (بعداً در بخش اینتراپتها بیشتر این موضوع را بررسی خواهیم کرد).
- **INTA** : یک پین خروجی است که فعال شدن آن به معنی پذیرش درخواست اینتراپت است. پس از آنکه‌یک دستگاه جانبی تقاضای اینتراپت می‌کند (اگر شرایط لازم مهیا باشد) و پروسسور ۸۰۸۸ تقاضای او را بپذیرد به نشانه پذیرش، پین INTA را در زمانهای TW،T3،T2 از سیکل پذیرش اینتراپت فعال می‌کند. هدف ریزپردازنده از صدور این سیگنال، بدست آوردن آدرس برنامه ای است که قرار است به اجرای آن بپردازد.
- **پین CLK**: ورودی پالس ساعت سیستم است که باید دارای سیکل وظیفه حدود ۳۳٪ باشد. به همین خاطر معمولاً از مولد کلاک 8284 تأمین می‌شود. 8284 علاوه بر CLK ورودیهای Reset و READY (که با کلاک سیستم سنکرون هستند) را نیز برای پروسسور ۸۰۸۸ تأمین می‌کند.

8284 برای تولید کلاک می‌تواند از یک کریستال یا از یک اسیلاتور دیگر استفاده کند. به این منظور پایه‌ی F/C آنرا باید صفر یا یک منطقی کرد. 8284 فرکانس اسیلاتور را بر ۳ تقسیم کرده و در خروجی پالسی با سیکل وظیفه ۳۳٪ تولید می‌کند. شکل۴-۱۰- مولد کلاک ۸۲۸۴ را نشان می‌دهد که با استفاده از کریستال ۱۵ مگاهرتزی، یک پالس ساعت ۵ مگاهرتزی را برای پروسسور ۸۰۸۸ تأمین می‌کند.



شکل۴-۱۰- مولد کلاک ۸۲۸۴

- **پین DT/R**: یک سیگنال خروجی از پروسسور است که جهت انتقال داده از طریق بافرهای خارجی را نشان می‌دهد. اگر این پین یک منطقی باشد، پروسسور از باس برای ارسال داده (به حافظه‌یا دستگاه جانبی) و اگر صفر منطقی باشد برای دریافت داده (از حافظه‌یا دستگاه جانبی) استفاده می‌کند. ۸۰۸۸ برای ارتباط با دستگاههای جانبی از ۱۶ وزن پایین آدرس باس استفاده می‌کند.

- **SS0** : این پین خروجی در مد مینیموم مشابه پین وضعیت S0 در مد ماکزیموم است. با استفاده از این پین و دو سیگنال کنترلی **DT/R** و **IO/M** پروسسور می‌تواند بطور کامل وضعیت ماشین سیکل جاری را (مطابق جدول ۴-۱۱) مشخص کند.

جدول ۴-۱۱- وضعیت ماشین سیکل

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SSO}$	Characteristics
1(HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O Port
1	1	0	Write I/O Port
1	1	1	Halt
0(LOW)	0	0	Code Access
0	0	1	Read Memory
0	1	0	Write Memory
0	1	1	Passive

- **RQ/GT** : یک پایه دو طرفه است که در مد ماکزیموم استفاده می‌شود و عملکردش مشابه BUSRQ و BUSACK است. یک عملکرد سه مرحله‌ای بین پروسسور و خارج دارد به این معنا که ابتدا پروسسور دیگری که در این سیستم وجود دارد روی این خط پیغامی می فرستد که من باس را لازم دارم سپس ۸۰۸۸ روی همین خط به او پاسخ می‌دهد که باس به تو واگذار شد و او پس از آنکه کارش را تمام کرد روی همین خط اعلام می‌کند که کار من با باس تمام شد. RQ0/GT0 نسبت به RQ1/GT1 از اولویت بیشتری برخوردار است.

- **پینهای وضعیت S3 تا S6** : پینهای وضعیت S3 تا S6 با چهار بیت وزن بالای آدرس مالتی پلکس شده‌اند و وضعیت جاری داخل پروسسور ۸۰۸۸ را نشان می‌دهند. در اعمال مربوط به حافظه، در طی T1 این پایه‌ها ۴ بیت وزن بالای آدرس را تشکیل می‌دهند. در اعمال مربوط به I/Oها این پایه‌ها صفر منطقی هستند. در اعمال مربوط به حافظه‌یا I/O در طی T4،TW،T3،T2 این پینها وضعیت پروسسور ۸۰۸۸ را نشان می‌دهند. S6 همواره صفر است. S5 انعکاسی از فعال بودن بیت پرچم اینتراپت است و سیگنالهای وضعیت S4،S3 طبق جدول ۴-۹ نشان می‌دهند که در حال حاضر کدام رجیستر سگمنت برای دستیابی به داده‌ها استفاده می‌شود.

Table 4-9- Status bits S3 and S4

S4	S3	Function
0	0	Extra Segment
0	1	Stack Segment
1	0	Code or no Segment
1	1	Data Segment

- **پینهای وضعیت S0 تا S2** : پینهای وضعیت S0 تا S2 در مد ماکزیموم و توسط باس کنترلر با مفهوم ارایه شده در جدول ۴-۱۰ استفاده می‌شوند.

جدول ۴-۱۰- وضعیت ماشین سیکل جاری

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Characteristics
0(LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1(HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (No Bus Cycle)

8086

16 bit data bus، 20-bit address bus

16 data، 20 address، 17 control/timing، 3 power = 56

But 8086 has just 40 pins
16 Address/Data lines are multiplexed (AD0-AD15)
Also A16-A19 with S3-S6

Clock and Reset requirements

CLK: 2-12 MHz، 33% duty cycle، low: -.5 to 0.t volts، high 3.9-5.0 volts

RESET: Reset state: CS = 0xFFFF، SS = DS = ES = IP = Flags = 0

RESET must be held high for at least 4 clocks and must be synchronized with system clock and must be activated in synchronization with falling clock of a new T state، Use Intel 8284 Clock Generator to do both

۴-۳-۵- اینتراپت در پروسسور۸۰۸۸

اینتراپت فراهم کننده مکانیسمی برای تغییرات سریع مسیر اجرای برنامه است. به عبارت دیگر، با آمدن اینتراپت به پروسسور اطلاع داده می‌شود که نیاز به اجرای سرویس خاصی است و باید کنترل برنامه به روتین سرویس مربوط به آن اینتراپت واگذار شود. از یک دیدگاه، اینتراپتهای هر پروسسور را می‌توان به دو گروه خارجی و داخلی تقسیم کرد. اینتراپتهای خارجی مربوط به دستگاههای ورودی- خروجی نظیر صفحه کلید، تایمرها، CD-ROM و کارت صدا هستند. به عنوان مثال، اینتراپت برای پرینتر باعث واگذاری برنامه به درایور پرینتر می‌شود. اینتراپتهای داخلی از عملیات داخلی CPU ناشی می‌شوند مثلاً چنانچه در عمل تقسیم توسط پروسسور، تقسیم بر صفر اتفاق بیفتد، یک اینتراپت داخلی صادر می‌شود. **اینتراپتهای ۸۰۸۶ و ۸۰۸۸ به ۵ دسته تقسیم می‌شوند.** اینتراپتهای سخت افزارهای خارجی، اینتراپتهای غیرقابل چشم پوشی، اینتراپتهای نرم افزاری، اینتراپتهای داخلی و Reset.

پروسسورهای ۸۰۸۶ و ۸۰۸۸ در مجموع دارای ۲۵۶ نوع[۳۸] اینتراپت هستند که از INT00 تا INTFF می‌باشد. برای اجرای هر اینتراپت، پروسسور بطور اتوماتیک، رجیستر پرچم (FR)، اشاره‌گر دستورالعمل (IP)، رجیستر سگمنت کد (CS) را در سگمنت پشته ذخیره کرده و به‌یک آدرس مشخص در حافظه (که متناظر با ادرس اینتراپت وارد شده است) مراجعه می‌کند. در پروسسور ۸۰۸۶، موقعیتی از حافظه که به آن مراجعه می‌شود، چهاربرابر مقدار اینتراپت است به عنوان مثال با آمدن اینتراپت ۳ اجرای برنامه به موقعیت 0C H از حافظه منتقل می‌شود.

متناظر به هر اینتراپت یک برنامه اجرایی وجود دارد (سرویس روتین اینتراپت[۳۹]) که پروسسور با آمدن آن اینتراپت، این برنامه را اجرا خواهد کرد. مقدار IP و CS متناظر به این برنامه اجرایی، در جدولی به نام جدول بردار اینتراپت[۴۰] ذخیره شده است. این جدول شامل ۲۵۶ ورودی است که هر کدام شامل ۴ بایت است. دو بایت برای IP و دو بایت برای CS. جدول بردار اینتراپت از آدرس 00000 H شروع و تا آدرس 003FF H ادامه دارد. به عبارت دیگر یک کیلوبایت اول حافظه سیستم به ذخیره آدرس بردار اینتراپت اختصاص دارد. جدول ۴-۱۱، جدول بردار اینتراپت را نشان می‌دهد.

۳۲ بردار اولیه‌اینتراپت به خود پروسسور اختصاص دارد ولی ۲۲۴ بردار بعدی توسط کاربر قابل تعریف هستند. هر چه شماره بردار اینتراپت کوچکتر باشد، اولویت پاسخ دهی به آن بیشتر است. بطوریکه INT0 بیشترین اولویت را دارد.

[38] - Type
[39] - Interrupt Service Routine (ISR) or Interrupt Handler
[40] - Interrupt Vector Table (IVT)

به عنوان مثال برای بردار $INT50_{10}$ آدرس فیزیکی حافظه $200_{10} = C8\ H$ است بطوریکه آدرس 000C8 H شامل اطلاعات مربوط به IP و آدرس 000CA H شامل اطلاعات مربوط به CS اینتراپت ۵۰ است. بطور مشابه، برای INT12 آدرس فیزیکی حافظه $48_{10} = 30\ H$ است بطوریکه آدرسهای 00030 Hو 00031 H شامل اطلاعات مربوط به IP روتین اینتراپت ۱۲ و آدرسهای 00032 Hو 00033 H شامل اطلاعات مربوط به CS اینتراپت ۱۲ هستند.

سرویس دهی به اینتراپتها بر اساس اولویت آنها است بطوریکه ابتدا به اینتراپتهای با اولویت بیشتر (شماره‌اینتراپت کوچکتر) پاسخ داده می‌شود. به عنوان مثال اگر بطور همزمان اینتراپتهای ۳ و ۵ به سیستم وارد شوند ابتدا سیستم به اینتراپت ۳ پاسخ می‌دهد و سپس به اینتراپت ۵.

چنانچه پین NMI فعال (یک منطقی) شود، پروسسور به موقعیت 00008 H پرش نموده و با واکشی محتوای آن مقادیر CS:IP را برای اجرای سابروتین متناظر با NMI بدست می‌آورد.

پروسسور ۸۰۸۸ در سیکل مربوط به آخرین کلاک دستورالعمل پایه INTR را چک می‌کند و در صورت فعال بودن آن (یک منطقی)، چنانچه تصمیم به اجرای آن بگیرد (اولویت اجرا داشته باشد) پایه INTA خود را فعال (صفر منطقی) می‌کند.

آدرس فیزیکی حافظه	ورودی جدول	بردار اینتراپت
3FF	CS 255	
3FC	IP 255	
3FA	CS 254	$Vector\ 255_{10}$
	IP 254	
		$Vector\ 254_{10}$
	…	
	..	
0D	CS 3	
0C	IP 3	
0A	CS 2	
08	IP 2	
06	CS 1	
04	IP 1	
02	(CS Value Vector 0) CS 0	
00	(IP Value Vector 0) IP0	

جدول ۴-۱۱- جدول بردار اینتراپت

عملیات مربوط به اینتراپت مد حقیقی:

۱ - محتوای رجیستر پرچم در داخل حافظه پشته ذخیره قرار می‌گیرد.

۲ - پرچمهای IF و TF (بیت D8 رجیستر پرچم) صفر می‌شوند و با این عمل پایه‌های INTR و مد Single Step غیرفعال می‌شوند.

۳ - محتوای رجیسترهای CS و IP در داخل حافظه پشته ذخیره می‌شود.

۴ - محتوای بردار اینتراپت واکشی شده و به موجب آن مقادیر جدیدی در رجیسترهای CS و IP قرار می‌گیرد بطوریکه دستورالعمل بعدی شامل اجرای روتین مربوط به اینتراپت وارد شده است.

۵ - هنگام برگشت از روتین اینتراپت (با دستورالعمل IRET) پرچمها به وضعیت قبل از آمدن اینتراپت برگشته و اجرای برنامه از موقعیت قبلی آغاز می‌شود.

تفاوت CALL با Interrupt:

۱- با دستور CALL می‌توان به هر موقعیت حافظه در داخل فضای یک مگابایتی پرش کرد اما با آمدن هر اینتراپت به‌یک موقعیت خاص حافظه (که متناظر با آن اینتراپت است) پرش انجام می‌شود.

۲- دستور CALL توسط برنامه نویس و در خلال اجرای دستورات سیستم صورت می‌گیرداما فرمان اینتراپت در هر زمان و توسط دستگاههای جانبی صادر می‌شود.

۳- فرمان CALL نمی‌تواند چشم پوشی شود اما اینتراپت صادر شده توسط تجهیزات جانبی ممکن است بلوکه شود.

۴- با آمدن دستور CALL، قبل از اجرای آن CS:IP ذخیره می‌شود اما با آمدن اینتراپت هم محتوای فلگها و هم CS:IP ذخیره می‌شود.

۵- در انتهای سابروتینی که با دستور CALL به آن پرش شده است از دستور RET استفاده می‌شود اما در انتهای سابروتین اینتراپت از دستور IRET استفاده می‌شود.

دستورالعملهای اینتراپت :این دستورالعملها در جدول ۴-۱۲ آمده‌اند.

جدول ۴-۱۲- دستوراعملهای اینتراپت در پروسسور ۸۰۸۸

Mnemonic	Meaning	Format	Operation	Flag Affected
CLI	Clear Interrupt Flag	CLI	0→(IF)	IF
STI	Set Interrupt Flag	STI	1→(IF)	IF
INT n	Type n software Interrupt	INT n	(Flags)→((SP)-2) (TF),0→(IF) (CS)→((SP)-4) (4*n) → (IP) (4*n+2) → (CS) (IP)→((SP)-6)	TF,IF

IRET	Interrupt Return	IRET	((SP))→(IP) ((SP)+2)→(CS) ((SP)+4) → (Flags) (SP)+6→(SP)	ALL
INTO	Interrupt on overflow	INTO	INT 4 Steps	TF،IF
HLT	Halt	HLT	Wait for an external interrupt or reset to occur	None
WAIT	Wait	WAIT	Wait for TEST input to go active (High)	None

برای آنکه پروسسور اینتراپتهای خارجی را قبول کند باید فلگ مربوط به پذیرش اینتراپت فعال شده باشد((IF)→1).

INT 2 همان اینتراپت غیرقابل چشم پوشی (NMI) است که بردار مربوط به آن از آدرس ۸ حافظه شروع می‌شود.

پس از هر دستورالعمل ریاضی که امکان ایجاد سرریز را دارد باید از دستور INTO استفاده شود. این دستور در حقیقت INT 04 H است که فقط اگر فلگ سرریز صفر باشد پروسسور به آن توجه نموده و با مراجعه به آدرس 00010 H، رجیسترهای CS:IP را مقداردهی نموده و سراغ اجرای روتین اینتراپت ۴ می‌رود. در غیر اینصورت (فلگ سرریز غیر فعال (یک) باشد) با این دستور معادل NOP رفتار می‌شود.

دستور HLT باعث منتظر ماندن پروسسور برای وقوع یک اینتراپت می‌شود.

دستور WAIT باعث منتظر ماندن پروسسور ۸۰۸۸ تا یک شدن پایه تست می‌شود.

۵-۱- مقدمه

زبانی که کامپیوتر با آن کار می‌کند، متشکل از یکسری ۰ و۱ است و زبان ماشین[۴۱] نام دارد. چون کار کردن و فهم زبان ماشین مشکل است، به همین خاطر سازندگان کامپیوتر زبانی به نام زبان اسمبلی را عرضه کردند که بسیار به زبان ماشین نزدیک است و تقریباً همان قدرت را دارد. زبان اسمبلی نیاز به یک مترجم به نام اسمبلر دارد تا دستورات زبان اسمبلی را به معادل صفر و یک آن، که برای کامپیوتر قابل فهم است، ترجمه کند. به عبارت دیگر، اسمبلر یک برنامه است که فایل متنی حاوی دستورات اسمبلی را می خواند و آنها را به کد ماشین (زبان ماشین) تبدیل می‌کند. زبان اسمبلی هر کامپیوتر تابع ساختارهای سخت افزاری آن کامپیوتر بوده و در نتیجه زبان اسمبلی کامپیوترهای مختلف با هم فرق می‌کند.

زبانهای برنامه نویسی کامپیوتر عموما به دو دسته زبانهای سطح بالا[۴۲] و زبانهای سطح پایین[۴۳] تقسیم میشوند. برخی این زبانها را به سه دسته تقسیم بندی کرده‌اند. زبانهای سطح بالا، زبانهای سطح پایین و زبانهای سطح میانی[۴۴]. زبان سطح پایین به زبانی گفته میشود که از لحاظ ساختاری و ترجمه بسیار به زبان ماشین نزدیک است یعنی قابلیت فهم آن برای ماشین بهتر و راحتتر است. اما زبانهای سطح بالا با کاربر رابطه بهتری دارند و کاربر یا برنامه نویس با این زبان راحت تر ارتباط برقرار می‌کند. زبان برنامه نویسی اسمبلی جزو زبانهای سطح پایین، زبانهایی مانند پاسکال، فاکس پرو، بیسیک و. . زبانهای سطح بالا و زبانهایی مانند C و C++ زبانهای سطح میانه اند.

زبانهای سطح بالا و سطح میانه تابع مشخصات سخت افزاری نیستند و اصولاً بر هر کامپیوتری قابل اجرا و یکسان می باشند. ولی مزیت زبان اسمبلی بر این زبانها این است که آنها به صورت مستقیم بر امکانات سخت افزاری دسترسی ندارند و در نتیجه از امکانات سخت افزاری به صورت مؤثر بهره نمی برند و به همین خاطر برنامه‌هایی که به این زبانها نوشته می‌شوند ۱۰۰ ها برابر کندتر از برنامه‌های اسمبلی مشابه هستند. علاوه بر این برای برنامه ریزی ماشین آلات صنعتی مانند ماشینهای بافندگی، ماشین ابزار، ماشینهای برش و.. نیاز به مشخصات و اطلاعات دریافتی از سخت افزار دارد و باید نظارت مستقیم و همه جانبه بر سخت افزار داشته باشیم که این کار با زبان اسمبلی ممکن است. تمامی سیستم عاملها نیز با زبان اسمبلی نوشته شده و می‌شوند. همه‌ی اینها نیلز به برنامه‌نویسی اسمبلی را تأیید می‌کنند.

در زبان اسمبلی به سبب پایین بودن سطح آن ویژگیهایی نهفته است که در هیچ یک از زبانهای دیگر این ویژگیها را نمیتوان یافت. یکی از ویژگیهای مهم این زبان، باز گذاشتن دست کاربر در کنترل سخت افزار بویژه CPU است. در واقع کاربر می‌تواند با همه‌ی اجزای پردازشگر و سخت افزار کامپیوتر ارتباط برقرار کند.

البته‌این زبان دارای مشکلاتی نیز هست. که از جمله مهمترین آنها :

- زیاد بودن تعداد دستوراتی است که کاربر باید برای انجام عملی خاص از آنها استفاده کند.
- برنامه نویس برای برنامه نویسی باید بر معماری ساخت CPU مسلط باشد.
- سورس این برنامه‌ها اصولا دارای تعداد خطوط زیادی است.
- این برنامه‌ها بسته به ماشین عمل می‌کنند. یعنی اگر ساختار اصلی ماشین تغییر کند این برنامه‌ها قابلیت اجرا ندارند.

برای تمامی کامپیوترهای شخصی، به علت داشتن ساختار سخت افزاری مشابه، دستورات زبان اسمبلی یکسان است.

41 - Machine Language
42 - High Level Language
43 - Low Level Language
44 - Middle Level Language

۵-۲- اسمبلر

برای تبدیل زبان اسمبلی به زبان ماشین باید از یک مترجم استفاده کرد که اسمبلر نام دارد. در واقع زبان اسمبلی از طریق اسمبلر به زبان ماشین که صفر و یک است ترجمه می‌شود. هر خط از زبان اسمبلی معادل یک خط در زبان ماشین است. این ویژگی، خاص دستورات اسمبلی است و در زبانها سطح بالا چنین اتفاقی نمی‌افتد. در زبانهای سطح بالا، کمپایلر عمل تبدیل برنامه به زبان ماشین را انجام می دهد. یک اسمبلر خیلی ساده تر از یک کمپایلر است. هر عبارت زبان اسمبلی مستقیماً به یک دستورالعمل زبان ماشین تبدیل می‌شود اما یک عبارت زبان سطح بالا ممکن است شامل چندین دستورالعمل زبان ماشین باشد.

برای اسمبل کردن یک برنامه اسمبلی باید به‌یک اسمبلر دسترسی داشت. عموماً از اسمبلرهایی نظیر TASM[۴۵] ، MASM[۴۶] یا NASM[۴۷] استفاده می‌شود که اولی محصول شرکت turbo و دومی محصول مایکروسافت است. نسخه جدید MASM نرم افزاری است به نام ML که کار کردن با آن نسبت به دو نرم افزار بالایی بسیار ساده تر است. با استفاده از TASM یا MASM سورس برنامه‌ای که در فایلی با پسوند asm نوشته‌اید را به‌یک فایل obj تبدیل می‌کنید. سپس با یک لینکر مثل TLINK می‌توانید فایل را به فایل اجرایی توسط کامپیوتر تبدیل کنید که پسوند exe دارد.

برای نوشتن سورس برنامه کافیست یک ویرایشگر متن داشته باشید که تمامی کامپیوتر ها اصولاً چنین چیزی را دارند. در سیتسم عامل ویندوز می‌توانید از notepad یا word استفاده کنید. اگر از word استفاده می‌کنید یادتان باشد که تغییرات اتوماتیک آنرا برای تصحیح کلمات از کار بیندازید. پس از آنکه سورس برنامه را نوشتید کافیست آنرا با پسوند asm ذخیره کنید. سپس به آدرسی که اسمبلر شما و فایل asm شما قرار دارد بروید.

اگر از نرم افزار MASM (یا TASM) استفاده می‌کنید. کافیست تایپ کنید MASM (یا TASM) و سپس نام فایل سورس با پسوند asm را جلوی آن بنویسید تا فایل obj شما ساخته شود. سپس با استفاده از TLINK و نام فایل با پسوند obj فایل exe را بسازید. به صورت زیر:

```
tasm filename.asm
tlink filename.obj
```

در این فصل ابتدا استفاده از DEBUG برای ورود و اجرای برنامه‌های زبان ماشین را یاد می گیریم سپس روش برنامه‌نویسی اسمبلی را شرح خواهیم داد.

45 - Turbo Assembler
46 - Macro Assembler
47 - Netwide Assembler

۵-۳- استفاده از DEBUG و فرامین آن

در این فصل از یک برنامه DOS به نام DEBUG برای دیدن حافظه، ورود برنامه‌ها در حافظه و پیگیری اجرای آنها استفاده می‌شود. اگرچه برنامه‌های اشکلل زدای پیشرفته تری مثل CODEVIEW و TurboDebugger وجود دارند اما از DEBUG استفاده می‌کنیم زیرا برای استفاده ساده تر است و در دسترس همگان قرار دارد. سیستم عامل DOS با برنامه‌ای به نام DEBUG همراه است که برای تست و اشکال زدایی برنامه‌های اجرایی بکار می‌رود. در این برنامه، نمایش همه کدها و داده‌های برنامه به شکل هگزادسیمال است. بنابراین داده‌های وارد شده به حافظه نیز باید به شکل هگزادسیمال باشند. DEBUG همچنین امکان اجرای تک به تک دستورات را به شما می‌دهد. برای ورود به محیط دیباگ در کامپیوترهای شخصی، کافیست از منوی Start گزینه RUN را انتخاب کرده و در خط فرمان آن تایپ کنید DEBUG. به این ترتیب به محیط DEBUG وارد می‌شوید و می‌توانید شروع به اجرای دستورات به شکل زبان ماشین (مثلاً B8 23 01 به جای دستور MOV AX،0123) یا به شکل دستور سمبولیک زبان اسمبلی(مثلاً MOV AX،0123) کنید.

برخی فرمانهای DEBUG عبارتند از:

فرمانD برای نمایش محتویات یک ناحیه از حافظه

فرمانE برای ورود داده به حافظه با شروع از یک موقعیت خاص

فرمانR برای نمایش محتویات یک یا چند رجیستر

فرمان A(Assemble) که به DEBUG می‌گوید تا دستورات سمبولیک اسمبلی را بپذیرد و آنها را به زبان ماشین تبدیل کند.

فرمان U(Unassemble) که به DEBUG می‌گوید تا کد ماشین دستورهای اسمبلی را نشان دهد.

فرمانT (Trace) برای ورود به حالت اجرای دستور به دستور

فرمانG برای اجرای یک برنامه اجرایی موجود در حافظه

فرمانQ برای خروج از مجموعه DEBUG

فرمانW برای نوشتن یک برنامه روی دیسک

فرمان P برای اجرای روتین وقفه

مثال۱: مشاهده محتویات حافظه

D DS:100

دستور فوق محتویات حافظه (سگمنت داده) را از 100 H (یا ۲۵۶) بایت بعد از شروع این سگمنت نمایش می‌دهد. رجیستر DS شروع ناحیه مورد نظر (سگمنت داده) از حافظه و عدد 100 H افست آن است(آدرس فیزیکی برای شروع سگمنت داده از ضرب کردن مقدار رجیستر DS در عدد ۱۶ بدست می آید).

به عنوان نمونه، نتیجه اجرای فرمان فوق در محیط DEBUG می‌تواند بصورت زیر باشد.

```
0AFA:0100  00 74 1E 3D 05 00 74 19-E9 E3 D8 8B D8 B8 00 44
0AFA:0110  CD 21 B4 3E CD 21 F6 C2-80 75 53 F6 34 00 E9 0A
0AFA:0120  4D 8B 56 05 80 FA 00 74-05 80 FE 3A 74 02 B2 40
0AFA:0130  80 CA 20 80 EA 60 E8 74-E4 73 06 E8 7F DB E9 AD
0AFA:0140  D8 8B D5 83 C2 05 8A 7E-04 80 E7 06 80 FF 06 75
0AFA:0150  18 8B 76 02 B3 3A 38 5C-FE 75 06 C6 46 00 02 EB
0AFA:0160  05 C6 46 00 01 4E E9 83-00 80 FF 02 75 05 C6 46
0AFA:0170  00 00 C3 E8 B6 EB B4 3B-CD 21 72 39 8B FA 33 C0
```

مثال۲: مشاهده وضعیت تجهیزات نصب شده بر روی سیستم: وضعیت تجهیزات نصب شده در قالب یک WORD و در موقعیت 411H – 410H قرار دارد. برای دسترسی به این موقعیت، 40[0]H برای آدرس سگمنت (آخرین صفر مقروض است) و 10 برای افست از آدرس سگمنت در نظر گرفته می‌شود. تفسیر آدرس 40:10 مانند سگمنت 40[0]H بعلاوه افست 10H است. توضیحات مربوط به WORD وضعیت بصورت زیر است.

بیتهای ۱۴ و ۱۵ : تعداد درگاههای چاپگر موازی متصل شده به سیستم

بیتهای ۹ تا ۱۱ : تعداد درگاههای سریال متصل شده به سیستم

بیتهای ۶ و ۷ : تعداد ابزارهای دیسکت

بیتهای ۴ و ۵ : حالت اولیه ویدئو

بیت ۱ : کمک پردازشگر ریاضی وجود دارد یا خیر

بیت صفر: دیسک درایو موجود است یا خیر

اگر دستور D 40:10 را در محیط DEBUG تایپ کنید دو بایت مربوط به WORD وضعیت بصورت مبنای ۱۶ پیدا می‌شوند. به عنوان نمونه، نتیجه اجرای فرمان فوق در محیط DEBUG بصورت زیر است.

```
0040:0010  23 C8 00 80 02 00 00 40-00 00 38 00 38 00 30 0B
```

در نتیجه فوق، WORD وضعیت به صورت C823 است که اگر معادل بیتی آنرا بنویسیم داریم:

C823 = 1100 1000 0010 0011

بیت صفر، وضعیت یک منطقی دارد یعنی اینکه در این سیستم، دیسک درایو موجود است.

بیت یک، وضعیت یک منطقی دارد یعنی اینکه در این سیستم، کمک پردازشگر ریاضی وجود دارد.

بیتهای ۴ و ۵ بصورت صفر و یک منطقی هستند. این به معنی آن است که حالت اولیه ویدئو بصورت 25*40 رنگی است.

بیتهای ۶ و ۷ بصورت صفر و صفر منطقی هستند. این به معنی آن است که تعداد یک ابزار دیسکت در این سیستم وجود دارد.

بیتهای ۹ تا ۱۱ بصورت 100 هستند که نشان می‌دهد ۴ دستگاه سریال به این سیستم متصل است.

بیتهای ۱۴ و ۱۵ بصورت 11 هستند. این به معنی آن است که تعداد سه درگاه چاپگر موازی به سیسنم متصل شده است.

مثال۳: مشاهده عدد سریال و حق Copyright سیستم: عدد سریال کامپیوتر در ROM BIOS در موقعیت FE000 H درج می‌شود. برای دیدن آن، سگمنت FE00[0] H و افست مکان 0 را وارد کنید. یعنی دستور:

D FE00:0

این عملیات باید عدد سریال ۷ رقمی و در ماشینهای قراردادی یک حق کپی رایت محفوظ بعد از آن قرار گیرد. عدد سریال بصورت اعداد مبنای ۱۶ دیده می‌شود در حالیکه حق کپی رایت محفوظ، بیشتر از بقیه کاراکترهای ناحیه ASCII سمت راست قابل تشخیص است. به عنوان نمونه نتیجه اجرای آن می‌تواند بصورت زیر باشد.

```
FE00:0000  41 77 61 72 64 20 53 6F-66 74 77 61 72 65 49 42   Award Softw
FE00:0010  4D 20 43 4F 4D 50 41 54-49 42 4C 45 20 34 38 36   M COMPATIBL
FE00:0020  20 42 49 4F 53 20 43 4F-50 59 52 49 47 48 54 20    BIOS COPYR
FE00:0030  41 77 61 72 64 20 53 6F-66 74 77 61 72 65 20 49   Award Softw
FE00:0040  6E 63 2E 6F 66 74 77 61-72 65 20 49 6E 63 2E 20   nc.oftware
FE00:0050  41 77 03 0C 04 01 01 6F-66 74 77 E9 0E 14 20 43   Aw.....oftw
FE00:0060  18 41 77 61 72 64 20 4D-6F 64 75 6C 61 72 20 42   .Award Modu
FE00:0070  49 4F 53 20 76 36 2E 30-00 A6 32 EC 33 EC 35 EC   IOS v6.0..2
```

مثال۴: مشاهده تاریخ ROM BIOS: تاریخ ساخت ROM BIOS شما بصورت mm/dd/yy در موقعیت FFFF5 H ثبت شده است. برای دیدن آن، سگمنت FFFF[0] H و افست مکان 5 را وارد کنید. دانستن این تاریخ برای دانستن مدل و سن کامپیوتر مفید است.

D FFFF: 5

نمونه‌ای از نتیجه اجرای این دستور بصورت زیر است.

```
FFFF:0000              30 36 2F-31 31 2F 30 33 00 FC 18        06/11/03...
```

چنانکه ملاحظه می‌شود تاریخ ساخت ROM BIOS روز یازدهم از ماه ششم سال ۲۰۰۳ میلادی است.

مثال۵: ورود دستورات برنامه به صورت زبان ماشین: برای این منظور از فرمان E استفاده می‌شود. به عنوان مثال دستور

E CS:100 B8 23 01

کد زبان ماشین دستور MOV AX،0123 است که به موجب آن مقدار 0123 در رجیستر AX کپی می‌شود. بعداً روش مشاهده نتیجه اجرای این دستور را خواهیم گفت.

مثال۶: ورود دستورات برنامه به صورت اسمبلی: آدرس شروع را به صورت زیر مقدار دهی نمایید:

A100 <Enter>

با دستور فوق DEBUG ارزش سگمنت کد و افست را بصورت xxxx:0100 نمایش می‌دهد. هر دستورالعمل را وارد کرده و پس از آن کلید Enter را فشار دهید.

MOV CL ،42

MOV DL ،2A

ADD CL ،DL

NOP

وقتی برنامه را وارد کردید، کلید Enter را دو بار بزنید تا از دستور A خارج شوید یک Enter اضافی به دیباگ می‌گوید که شما دستور سمبولیک دیگری برای ورود ندارید. قبل از اجرای برنامه، اجازه دهید تا از فرمان U برای مشاهده زبان ماشین تولید شده استفاده کنیم. دستور زیر را وارد کنید.

U 100,106 <Enter>

دستور فوق مکان اولین و آخرین دستور را به دیباگ می‌گوید. نتیجه اجرای آن چنین است.

```
-u 100,106
0AFA:0100 B142          MOV     CL,42
0AFA:0102 B22A          MOV     DL,2A
0AFA:0104 00D1          ADD     CL,DL
0AFA:0106 90            NOP
```

حال اجرای برنامه را پیگیری کنید. آنچه که در حقیقت اجرا می‌شود کد ماشین است. برای نمایش محتویات رجیسترها و اولین دستور با R شروع کنید و T را برای پیگیری دستورهای بعدی استفاده کنید. نتیجه کار بصورت زیر است.

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0100   NV UP EI PL NZ NA PO NC
0AFA:0100 B142          MOV     CL,42
-T

AX=0000  BX=0000  CX=0042  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0102   NV UP EI PL NZ NA PO NC
0AFA:0102 B22A          MOV     DL,2A
-T

AX=0000  BX=0000  CX=0042  DX=002A  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0104   NV UP EI PL NZ NA PO NC
0AFA:0104 00D1          ADD     CL,DL
-T

AX=0000  BX=0000  CX=006C  DX=002A  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0106   NV UP EI PL NZ NA PE NC
0AFA:0106 90            NOP
-T

AX=0000  BX=0000  CX=006C  DX=002A  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0107   NV UP EI PL NZ NA PE NC
0AFA:0107 19E9          SBB     CX,BP
```

چنانکه ملاحظه می‌شود وقتی به دستور NOP در موقعیت 106 H می‌رسد، IP حاوی 106 H و CL حاوی 6C H است.

مثال۷: **استفاده از دستورات اینتراپت برای مشاهده تاریخ جاری سیستم**: چندین نوع مختلف اینتراپت در سیستم وجود دارد که برای اجرای برخی از آنها باید در رجیستر AH مقدار مشخصی قرار گیرد. به عنوان مثال برای دیدن تاریخ سیستم (با استفاده از اینتراپت 21H) باید کد 2A در رجیستر AH قرار گیرد. نتیجه اجرای این دستور، روز هفته در رجیستر AL (صفر برای یکشنبه)، سال در رجیستر CX (مثلاً 07D0 H برای سال ۲۰۰۰ میلادی)، ماه در رجیستر DH و روز در رجیستر DL (همگی به صورت اعداد هگزادسیمال) نشان داده می‌شوند. دستورات زیر را تایپ کنید.

```
A 100
MOV AH ،2A
INT 21
NOP
```

دستور R برای دیدن رجیسترها و T برای اجرای MOV تایپ کنید. سپس برای اجرای روتین وقفه P را تایپ کنید. عملیات در دستور NOP متوقف می‌شود. رجیسترها حاوی اطلاعات مربوط به تاریخ جاری سیستم خواهند بود.

```
-a100
0AFA:0100 mov ah,2a
0AFA:0102 int 21
0AFA:0104 nop
0AFA:0105
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0100   NV UP EI PL NZ NA PO
0AFA:0100 B42A          MOV     AH,2A
-t

AX=2A00  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0102   NV UP EI PL NZ NA PO
0AFA:0102 CD21          INT     21
-p

AX=2A02  BX=0000  CX=07D7  DX=0815  SP=FFEE  BP=0000  SI=0000  DI=0
DS=0AFA  ES=0AFA  SS=0AFA  CS=0AFA  IP=0104   NV UP EI PL NZ NA PO
0AFA:0104 90            NOP
```

در نتیجه حاصل شده برای دستورات فوق، مقدار رجیستر AL برابر 02 است (یعنی روز سه شنبه)، مقدار رجیستر CX برابر 07D7 است (یعنی سال ۲۰۰۷ میلادی)، مقدار رجیستر DH برابر 08 است (یعنیماه هشتم) و مقدار رجیستر DL برابر 15 است (یعنی روز 21= H 15).

مثال۸: **استفاده از اینتراپت برای ورودی از صفحه کلید**

```
A 100
MOV AH ،10
INT 16
JMP 100
NOP
```

اولین دستور MOV به INT 16 (با AH=10) می‌گوید که از صفحه کلید داده بپذیرد. عملیات، کد اسکی کاراکتر پذیرفته شده از صفحه کلید را در رجیستر AL بصورت هگزادسیمال تحویل می‌دهد. دستور JMP سبب می‌شود تا پروسسور مقدار IP را با 100 جایگزین سازد، بنابراین دستور بعدی که اجرا می‌شود اولین دستور خواهد بود و به کمک آن می‌توان یک یا چند دستور را مرتباً اجرا نمود. دستور R برای دیدن رجیسترها، T برای اجرای MOV و P برای اجرای روتین وقفه را تایپ کنید.

در اجرای زیر با فشردن کلید ۲ کداسکی آن (32 H) در رجیستر AL ظاهر شده است.

```
-R IP
IP 0107
:100
-R
AX=0924  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0100   NV UP DI PL NZ NA PO NC
00A7:0100 B410          MOV     AH,10
-T

AX=1024  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0102   NV UP DI PL NZ NA PO NC
00A7:0102 CD16          INT     16
-P

AX=0332  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0104   NV UP DI PL NZ NA PO NC
00A7:0104 EBFA          JMP     0100
```

مثال۹: **استفاده از دستورات اینتراپت برای نمایش**

```
A 100
MOV AH ،09
MOV DX ،108
INT 21
NOP
DB „MY NAME IS HASAN“،"$"
```

دو دستور MOV به HNT 21 H (با AH=09) می‌گویند که آنچه که از آدرس 108 آغاز می‌شود (DX=108) را نمایش دهد. توجه دارید که آدرس 108 آغاز پیام نمایشی شما است. علامت "$„ به INT انتهای نمایش را نشان می‌دهد. R را بفشارید تا رجیسترها و اولین دستور را ببینید و فرمان T را برای اجرای دو MOV تایپ کنید. سپس برای اجرای روتین

وقفه P را تایپ کنید و پیام خود را ببینید. عملیات در دستور NOP متوقف می‌شود. نمونه‌ای از اجرای این دستورات را می‌بینید. در این اجرا پیامی به صورت "my name is hasan" بر روی صفحه چاپ شده است.

```
-R IP
IP 0108
:100
-R
AX=0924  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0100   NV UP DI PL NZ NA PO NC
00A7:0100 B409          MOV     AH,09
-T

AX=0924  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0102   NV UP DI PL NZ NA PO NC
00A7:0102 BA0801        MOV     DX,0108
-T

AX=0924  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0105   NV UP DI PL NZ NA PO NC
00A7:0105 CD21          INT     21
-P
my name is hasan
AX=0924  BX=0000  CX=0000  DX=0108  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=00A7  IP=0107   NV UP DI PL NZ NA PO NC
00A7:0107 90            NOP
```

مثال۱۰ : **انتقال داده به موقعیتهای حافظه**: در این مثال می‌خواهیم داده‌ها را به رجیسترها و موقعیتهای حافظه منتقل و آنها را با هم جمع کنیم. از عملگر WORD PTR نیز استفاده شده است. این عملگر به DEBUG می‌گوید که با داده مورد نظر به صورت یک Word (دو بایت) رفتار کند (با 25 H به صورت 0025 H رفتار کند). بطور مشابه، دستور MOV [122] 30،BYTE PTR که عملگر BYTE PTR را استفاده می‌کند باعث انتقال بایت 30 H به موقعیت 122 H حافظه می‌شود. عملگر DWORD PTR نیز برای تعریف داده از نوع دو کلمه‌ای یا ۳۲ بیتی است.

برای وارد کردن برنامه، مطابق شکل زیر ابتدا تایپ کنید A100 <Enter> و سپس دستورات سمبولیک را (مطابق شکل) وارد کنید.

```
-A100
 XXXX:0100 MOV AX,[11A]
XXXX:0103 ADD AX,[11C]
XXXX:0107 ADD AX,25
XXXX:010A MOV [11E],AX
XXXX:010D MOV WORD PTR [120],25
XXXX:0113 MOV BYTE PTR [122],30
XXXX:0118 NOP
XXXX:0119 NOP
XXXX:011A DB 14 23
XXXX:011C DB 05 00
XXXX:011E DB 00 00
XXXX:0120 DB 00 00 00
```

توضیح دستورات فوق بصورت زیر است.

100: انتقال محتویات موقعیتهای 11A H – 11B H به رجیستر AX. کروشه به معنی آدرس حافظه است نه یک مقدار بلافصل.

103: جمع محتویات موقعیت 11C H – 11D H حافظه با رجیستر AX.

107: جمع مقدار بلافصل 25 H با رجیستر AX.

10A: انتقال محتویات AX به موقعیتهای 11E H – 11F H حافظه.

10D: انتقال مقدار بلافصل 25 H به موقعیتهای 121 H–120 H حافظه.

توجه داشته باشید که اگر دستور [120] 25،MOV WORD PTR را بصورت [120]،25 MOV بنویسید DEBUG راهی برای تشخیص طول نخواهد داشت و لذا یک پیغام خطا نشان می‌دهد.

113: انتقال مقدار بلافصل 30 H (بصورت بایت) به موقعیت 122 H حافظه .

11A تا 120 : تعریف مقادیر بایتی 14 H، 23 H، 05 H و 00 H .

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=0100   NV UP EI PL NZ NA PO NC
0AF7:0100 A11A01        MOV     AX,[011A]                       DS:011
-t

AX=2314  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=0103   NV UP EI PL NZ NA PO NC
0AF7:0103 03061C01      ADD     AX,[011C]                       DS:011
-t

AX=2319  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=0107   NV UP EI PL NZ NA PO NC
0AF7:0107 052500        ADD     AX,0025
-t

AX=233E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=010A   NV UP EI PL NZ NA PO NC
0AF7:010A A31E01        MOV     [011E],AX                       DS:011
-t

AX=233E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=010D   NV UP EI PL NZ NA PO NC
0AF7:010D C706200125 00  MOV     WORD PTR [0120],0025            DS:012
-t

AX=233E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=0113   NV UP EI PL NZ NA PO NC
0AF7:0113 C606220130    MOV     BYTE PTR [0122],30              DS:012
-t

AX=233E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0AF7  ES=0AF7  SS=0AF7  CS=0AF7  IP=0118   NV UP EI PL NZ NA PO NC
0AF7:0118 90            NOP
```

برای اجرای برنامه، با وارد کردن R برای دیدن رجیسترها و اولین دستور اقدام کنید، سپس چندین فرمان t را وارد کنید. وقتی به NOP در 118 رسیدید از اجرا خارج شوید. در زیر نمونه‌ای از اجرای دستورات فوق را می‌بینید.

نکته: از آنجا که اسمبلر هر ارجاعی را که با حرف آغاز شود به عنوان یک نام سمبولیک در نظر می‌گیرد، بنابراین همیشه اولین رقم در یک عدد هگزادسیمال باید ۰ تا ۹ باشد. مثلاً 3D H یا 0DE8 H. اگر داده‌ی اخیر را به صورت DE8 H معرفی کنید اسمبلر از آن خطا خواهد گرفت.

۵-۴- برنامه‌نویسی به زبان اسمبلی

برنامه‌نویسی اسمبلی همانند سایر برنامه‌ها بر طبق مجموعه‌ای از قوانین دقیق و با استفاده از یک ویرایشگر در قالب یک فایل نوشته شده و با استفاده از یک برنامه مترجم اسمبلی تبدیل به کد ماشین می‌شود.

در اکثر کاربردها برای برنامه‌نویسی، ترکیبی از زبانهای سطح بالا و سطح پایین استفاده می‌شود. معمولاً کدنویسی قسمت عمده پروژه با زبان سطح بالا و کدنویسی ماژولهای حساس و بحرانی (که باید تأخیر کمتری ایجاد کنند) با زبان اسمبلی صورت می‌گیرد.

علیرغم زبانی که برای برنامه‌نویسی استفاده می‌شود، برنامه‌ای که به این زبان نوشته می‌شود باید برای ماشین به گونه‌ای قابل اجرا ترجمه شود. یک زبان سطح بالا از برنامه *کمپایلر* و یک زبان سطح پایین از *اسمبلر* برای ترجمه کد مبدأ به کد ماشین (کد مقصد) استفاده می‌کند. سپس برنامه الحاق گر [۴۸] برای هر دو سطح پایین و بالا کد مقصد را به زبان ماشین تبدیل می‌کند.

۵-۵- بخشهای مختلف یک برنامه اسمبلی

یک برنامه اسمبلی شامل بخشهای زیر است.

الف- توضیحات برنامه: در زبان اسمبلی، معمولاً منظور از مجموعه‌ای از دستورات اغلب واضح نیست به همین خاطر نیاز است جلوی هر سطر توضیحاتی اضافه کرد تا مفهوم برنامه روشنتر شود. یک توضیح در اسمبلی با (;) آغاز می‌شود و اسمبلر تمام کاراکترهای سمت راست این علامت را به عنوان توضیح در نظر می‌گیرد. توضیح می‌تواند جلوی دستور یا در یک خط جداگانه باشد. به عنوان مثال بصورت

```
AX        ; Store sum in FLDF،MOV FLDF
```

یا به صورت

```
MOV FLDF، AX
; Store sum in FLDF
```

در برنامه نوشته می‌شود.

توضیحات فقط در لیست برنامه مبدأ اسمبلی ظاهر می‌شوند ولی کد ماشین تولید نمی‌کنند. بنابراین هر چقدر توضیحات به برنامه اضافه کنید، تأثیری بر اندازه برنامه اسمبل شده نخواهد داشت.

ب- شناسه‌ها (سمبولها) : شناسه نامی است که برای یک عنصر و جهت ارجاع به آن استفاده می‌شود و ممکن است به دو صورت نام[۴۹] و برچسب[۵۰] باشد.

نام به آدرس یک عنصر داده مانند counter در دستور counter DB 24H اشاره می‌کند.

برچسب به آدرس یک دستور در یک زیر روال اشاره می‌کند. مثل MAIN در عبارت MAIN PROC FAR.

مشابه سایر زبانهای برنامه‌نویسی باید به‌یکسری نکات در مورد نام شناسه دقت کرد. مثلاً از کاراکترهای خاص نظیر علامت سؤال (؟)، خط فاصله (-)، دلار ($)، سمبل @ و.. نمی‌توان استفاده کرد.

پ- دستورات: دستورات اسمبلی دو گروه (نوع) هستند. یک گروه دستوراتی مانند ADD، MOV، INC هستند که اسمبلر آنها را به کد مقصد تبدیل می‌کند (مثل دستور MOV AX،25) این گروه در حقیقت فرامین اجرایی برنامه هستند و

48 - Linker
49 - Name
50 - Lable

گروه دوم دستوراتی که به اسمبلر برای اجرای عملی خاص پیغام می‌دهند و به آنها پیش‌پردازنده می‌گویند. مانند تعریف یک داده نظیر ALI DB 142 .

شکل عمومی یک دستور به صورت زیر است.

[شناسه]	دستورالعمل	[عملوند]	[توضیحات]

بخشهای داخل کروشه اختیاری بوده و ممکن است برخی اوقات موجود نباشند.

مثلاً در دستور COUNTER DB 12 عبارت COUNTER شناسه، عبارت DB دستورالعمل و 12 عملوند است. یا به عنوان مثالی دیگر در دستور MOV AX ،P30: 0 عبارت P30 شناسه، عبارت MOV دستورالعمل و AX ،0 عملوند هستند. یا در دستور INC BX عبارت INC دستورالعمل و BX عملوند است.

۵-۵-۱- پیش‌پردازنده‌ها در یک برنامه اسمبلی: زبان اسمبلی جملاتی دارد که شما را در کنترل روش اسمبل شدن و لیست گیری برنامه کمک می‌کند. این جملات را پیش‌پردازنده می‌گویند. برخی پیش‌پردازنده‌های مهم عبارتند از:

PAGE: پیش‌پردازنده PAGE حداکثر تعداد خط در صفحه و کاراکتر در خط را تعیین می‌کند. به عنوان مثال پیش‌پردازنده PAGE 60 ،132 در هر صفحه ۶۰ خط ۱۳۲ کاراکتری را فراهم می‌کند. شکل کلی آن بصورت PAGE [Length]، [Width] است. حذف این پیش‌پردازنده باعث می‌شود که اسمبلر مقدار PAGE 50 ،80 را در نظر بگیرد.

TITLE: پیش‌پردازنده TITLE دارای شکل کلی [توضیح] TITLE text است که معمولاً در خط دوم برنامه، نام برنامه را مشخص می‌کند مثل:

TITLE ASMSORT Assembly program to sort customer name

ENDS: پیش‌پردازنده ENDS انتهای یک سگمنت را مشخص می‌کند و به صورت زیر تعریف می‌شود.

Segname ENDS

که در آن Segname نام سگمنتی است که پیش‌پردازنده ENDS را برای آن بکار برده‌ایم.

ENDP: پیش‌پردازنده ENDP انتهای یک سابروتین (زیر روال) را مشخص می‌کند و به صورت زیر تعریف می‌شود.

Procname ENDP

که در آن Procname نام سابروتینی است که پیش‌پردازنده ENDP را برای آن بکار برده‌ایم.

END: پیش‌پردازنده END انتهای برنامه را مشخص می‌کندو به عنوان آخرین دستور در برنامه ظاهر می‌شود و به صورت زیر تعریف می‌شود.

END Progname

که در آن Progname نام اولین سابروتینی است که در قسمت PROC به صورت FAR تعریف شده است.

PROC: سگمنت کد با دستور PROC تعریف شده و حاوی کد اجرایی برنامه است. کد اجرایی می‌تواند شامل یک یا چند سابروتین باشد.

ASSUME: یک برنامه اسمبلی از رجیستر SS برای آدرس‌دهی پشته، از رجیستر DS برای آدرس‌دهی سگمنت داده و از رجیستر CS برای آدرس‌دهی سگمنت کد استفاده می‌کند. بدین منظور باید هدف از هر سگمنت را در برنامه برای اسمبلر مشخص کرد. پیش‌پردازنده ASSUME این کار را (با دستور زیر) انجام می‌دهد.

ASSUME SS: stackname، DS: datasegname، CS: codesegname، ES: datasegname

SS: stackname یعنی اینکه اسمبلر نام سگمنت پشته را با SS مرتبط سازد و به همین صورت برای سایر سگمنتها.

EQU: پیش‌پردازنده EQU مقداری را تعریف می‌کند که اسمبلر در دستورات دیگر می‌تواند آنرا جایگزین سازد. به عنوان مثال جمله FACTOR EQU 25 باعث می‌شود که هر وقت FACTOR در یک دستور یا در یک پیش‌پردازنده دیگر ظاهر شود اسمبلر به جای آن مقدار ۲۵ را جایگزین کند. به عنوان مثال جمله TABLE DB FACTOR DUP(?) را اسمبلر به مقدار معادل آن TABLE DB 25 DUP(?) تبدیل می‌کند. مزیت استفاده از EQU این است که خیلی از جملات ممکن است از مقدار تعریف شده با این پیش‌پردازنده استفاده کنند که اگر قرار باشد مقدار آن تغییر کند فقط کافی است جمله EQU را تغییر داد.

Dn: پیش‌پردازنده‌هایی که عناصر داده را تعریف می‌کنند. این پیش‌پردازنده‌ها می‌توانند داده‌ای را که دارای یک مقدار ناشناخته یا یک مقدار ثابت شناخته شده است برای اسمبلر تعریف کنند. Dn می‌تواند پیش‌پردازنده DB (برای تعریف داده‌ی یک بایتی)، DW (برای تعریف داده‌ی یک کلمه‌ای یا دوبایتی)، DD (برای تعریف داده‌ی دو کلمه‌ای یا چهار بایتی)، DF (برای تعریف داده‌ی چهار کلمه‌ای یا هشت بایتی)، DT (برای تعریف داده‌ی ده کلمه‌ای یا بیست بایتی) باشد. مثلاً اگر داده BYTE2 دارای مقدار شناخته شده 48 باشد آنرا به صورت BYTE2 DB 48 و اگر دارای مقدار ناشناخته باشد به صورت BYTE2 DB ? تعریف می‌کنیم. اگر بخواهیم چندین داده‌ی بایتی (با مقادیر شناخته شده) را به اسمبلر معرفی کنیم به صورت مثلاً BYTE2 DB 48، 27، 43، 24، 28 می‌نویسیم که در این شکل، اسمبلر مقادیر ثابت را در بایتهایی مجاور BYTE2 تعریف می‌کند بطوریکه به عنوان مثال رجوع به BYTE2+3 رجوع به چهارمین بایت یعنی 24 (18 H) است.

نکته: از پیش‌پردازنده DB برای تعریف داده‌های کاراکتری (با هر طولی) نیز استفاده می‌شود. مثلاً

BYTE3 DB „COMPUTER PROCESSORS“ ; character string

برای تکرار تعریف یک مقدار ثابت از ساختار زیر استفاده می‌شود.

[name] Dn numrep DUP(statement)

به عنوان مثال دستور BYTE5 DB 12DUP(0) معادل این است که ۱۲ بایت صفر را تعریف کنیم. در اینصورت به عنوان مثال اجرای دستور MOV AL، BYTE5+11 باعث انتقال صفر به رجیستر AL می‌شود. برای تعریف ۱۰ کلمه بدون مقدار از دستور BYTE6 DW 10DUP(?) استفاده می‌شود.

۵-۵-۲- پیش پردازنده سگمنت ساده شده

اسمبلر می‌تواند برخی از تعاریف سگمنت را فراهم سازد برای استفاده از آنها، شما باید مدل حافظه را قبل از تعاریف سگمنت، مقداردهی کنید. قالب عمومی معرفی مدل حافظه چنین است.

.MODEL memory-model

مدل حافظه می‌تواند TINY،SMALL، MEDIUM، COMPACT یا LARGE باشد که جزئیات آن در جدول ۵-۱ آمده است.

جدول ۵-۱- معرفی مدلهای حافظه

مدل حافظه	تعداد سگمنتهای کد	تعداد سگمنتهای داده
TINY	*	*
SMALL	یک	یک
MEDIUM	بیش از یک	یک
COMPACT	یک	بیش از یک
LARGE	بیش از یک	بیش از یک

اسمبلر از مدل TINY برای برنامه‌های اسمبلی با فرمت. COM استفاده می‌کند که داده‌ها، کد و پشته همگی در یک سگمنت ۶۴ کیلویی قرار دارند. مدل SMALL کد را در یک سگمنت ۶۴ کیلویی و داده را در یک سگمنت ۶۴ کیلویی دیگر قرار می‌دهد. پیش پردازنده. MODEL بصورت اتوماتیک جملات ASSUME لازم را تولید می‌کند. قالب عمومی برای پیش‌پردازنده‌های تعریف سگمنت پشته، داده و کد بصورت زیر است.

.STACK [stack size]
.DATA
.CODE

هر یک از این پیش‌پردازنده‌ها سبب می‌شود تا اسمبلر جملات SEGMENT لازم و ENDS متعلق به آنها را تولید کند. اسمبلر نامهای پیش‌فرض برای سگمنتها را بصورت DATA،STACK و TEXT (برای سگمنت کد) در نظر می‌گیرد. اندازه‌ی پیش‌فرض برای سگمنت پشته ۱۰۲۴ بایت در نظر گرفته می‌شود. برای پیش‌پردازنده‌های سگمنت ساده شده باید DS را چنین مقداردهی کنید.

MOV AX،@data
MOV DS،AX

۵-۶- ساختار کلی یک برنامه اسمبلی

با توجه به موارد بخش ۵-۶ برای یک برنامه اسمبلی، ساختار کلی یک برنامه ساده اسمبلی با فرمت EXE. را در شکل ۵-۲ مشاهده می‌کنید.

شکل ۵-۲- ساختار کلی یک برنامه اسمبلی با فرمت EXE.

```
                PAGE 60،132
   TITLE        ASMEXAMPLE skeleton of an assembly program
;-------------------------------------------------------------------------
   STACKSG SEGMENT PARA STACK „stack"
                DW 32DUP(0)
   STACKSG   ENDS                    ; end of segment
;-------------------------------------------------------------------------
   DATASG SEGMENT PARA „data"
              WORD1 DW 175
              WORD2 DW 150
              WORD3 DW  ?
   DATASG   ENDS                     ; end of segment
;-------------------------------------------------------------------------
   CODESG SEGMENT PARA „code"
      MAIN   PROC   FAR
              ASSUME  SS:STACKSG، DS:DATASG، CS:CODESG
              MOV AX،DATASG       ; set address of data
              MOV DS،AX           ; segment in DS

              MOV AX، WORD1       ; mov  0175  to  AX
              ADD  AX، WORD2      ; add  0150  to  AX
              MOV WORD3، AX       ; store  sum  in  WORD3

            MOV AX، 4C00H         ; end processing
              INT   21H
     MAIN   ENDP                  ; end of procedure
   CODESG   ENDS                  ; end of segment
              END   MAIN          ; end of program
```

توجه دارید که برنامه کاربر در داخل کادر مشخص شده نوشته می‌شود و ساختار بقیه برنامه(تقریباً) ثابت است. دستور INT 21H به همراه کد تابعی AH=4C H تقاضا جهت خاتمه اجرای برنامه است.

سگمنتها به این روش تعریف شده‌اند:

- STACKSG حاوی یک ورودی DW برای تعریف کلمه است که ۳۲ کلمه با مقدار صفر را تعریف می‌کندکه فضایی کافی برای فضای پشته در برنامه‌های کوچک است.
- DATASG سه کلمه‌یWORD1 (با مقدار ۱۷۵)، WORD2 (با مقدار ۱۵۰) وWORD3 (بدون مقدار) را تعریف می‌کند.

- CODESG حاوی دستورات اجرایی برنامه است. گرچه اولین جمله‌یعنی ASSUME چنانکه قبلاً گفتیم کد اجرایی تولید نمی‌کندولی عملیات زیر را انجام می‌دهد.

تخصیص STACKSG به رجیستر SS که باعث می‌شود سیستم از آدرس رجیستر SS برای آدرس‌دهی STACKSG استفاده کند.

تخصیص DATASG به رجیستر DS که باعث می‌شود سیستم از آدرس رجیستر DS برای آدرس‌دهی DATASG استفاده کند.

تخصیص CODESG به رجیستر CS که باعث می‌شود سیستم از آدرس رجیستر CS برای آدرس‌دهی CODESG استفاده کند.

وقتی یک برنامه از روی دیسک بر روی حافظه (جهت اجرا) بارگذاری می‌شود، بارگذار برنامه آدرس واقعی را در رجیسترهای CS و SS تنظیم می‌کند و با اولین دو دستور MOV رجیستر DS مقداردهی می‌شود.

شکل ۵-۳ همان برنامه با استفاده از پیش‌پردازنده‌های سگمنت ساده شده را نشان می‌دهد.

شکل ۵-۳- ساختار کلی یک برنامه اسمبلی با فرمت EXE. و سگمنت ساده شده

```
                PAGE 60،132
  TITLE         ASMEXAMPLE skeleton of an assembly program
;-------------------------------------------------------------------------
                .MODEL SMALL
                .STACK 64                ;Define stack
                . DATA                   ;Define data
               WORD1 DW 175
               WORD2 DW 150
               WORD3 DW  ?
;-------------------------------------------------------------------------
                .CODE                    ;Define code
      MAIN    PROC    FAR
                MOV AX،@data      ; set address of data
                MOV DS،AX           ; segment in DS

               ╔═══════════════════════════════════════╗
               ║ MOV AX، WORD1      ; mov  0175  to  AX ║
               ║ ADD  AX، WORD2     ; add   0150  to   AX ║
               ║ MOV WORD3، AX      ; store  sum  in  WORD3 ║
               ╚═══════════════════════════════════════╝

              MOV AX، 4C00H       ; end processing
                INT    21H
      MAIN    ENDP                      ; end of procedure
                END    MAIN              ; end of program
```

در برنامه فوق، مدل حافظه SMALL و پشته ۶۴ بایت (۳۲ کلمه) تعریف شده است.

۵-۷- برخی دستورهای اجرایی زبان اسمبلی

الف- دستور انتقال داده

- **MOV Destination، Source**

(Register to register ، Register to memory، Memory to register، Immediate data to register)

با این دستور می‌توان داده‌ها را بین رجیسترها یا خانه‌های حافظه انتقال داد. به موجب دستور فوق محتوای Source وارد Destination می‌شود. Destination می‌تواندیک رجیستر، نام یک متغیر، یا آدرس یک مکان از حافظه باشد. Source میتواند یک مقدار عددی یا حرفی، نام یک رجیستر و... باشد. دو عملوند دستور MOV باید از نظر اندازه برابر باشند مثل:

```
MOV DL، AL
MOV CL، BH
MOV CX، DX
```

فقط رجیسترهای همه منظوره AX تا DX را می‌توان توسط دستور فوق مقداردهی کرد. در صورتی که بخواهیم رجیسترهایی مثلDS یا ES را مقداردهی کنیم باید از رجیستر AX به عنوان واسطه استفاده کرده و مقدار را از طریق آن انتقال داد. مثلاً نمی‌توانیم بنویسیم MOV ES،20 H ولی می‌توانیم بنویسیم MOV AX،20 H و MOV ES،AX .

مثال:

```
WORDA DW 12          ; define WORDA as a word with value 12
MOV CX، WORDA        ; انتقال داده بین حافظه و رجیستر
MOV CX، 25           ; انتقال داده بلافصل به رجیستر
MOV CX، DX           ; انتقال داده بین دو رجیستر
MOV CX، [DX]         ; انتقال داده از موقعیت DS:DX حافظه به رجیستر
```

ب- دستور تعویض محتویات مبدأ و مقصد

- **XCHG Destination، Source**

با این دستور می‌توان محتوای Destination و Source را با هم عوض کرد یعنی: Destination ↔ Source

Destination و Source می‌توانند رجیستر یا یک موقعیت حافظه باشند با این شرط که‌یک طرف حتماً باید رجیستر باشد.

مثال:

```
WORDB DW 12          ; define WORDA as a word with value 12
....
```

```
XCHG CX، DX          ; تعویض محتویات دو رجیستر
XCHG CX، WORDB       ; تعویض محتویات حافظه و رجیستر
```

پ- دستور مقایسه

- **CMP**

دستور CMP جهت مقایسه دو فیلد داده بکار می‌رود. یک یا هر دو فیلد در رجیستر هستند. نتیجه عملیات مقایسه بر روی فلگهای SF،PF،OF،CF،AF و ZF تأثیر می گذارد به عنوان مثال

```
    CMP DX،0      ; مقایسه رجیستر DX  با مقدار صفر
    JE T20        ; اگر DX=0 است به T20 پرش کن
    MOV CX، 0     ; و گرنه CX را صفر کن
T20:  ADD CX، 10

    XCHHG BX،AX
```

در اولین دستور، اگر DX حاوی صفر باشد، دستور CMP فلگ ZF را یک کرده و در نتیجه، برنامه به T20 پرش خواهد کرد در غیر اینصورت رجیستر CX با مقدار صفر پر می‌شود.

برای مقایسه دو رشته نیز می‌توان از دستور CMP یا دستور CMPS استفاده کرد.

ت- دستور مقداردهی رجیستر با یک افست

- **LEA**

دستور LEA برای مقداردهی رجیستر با یک آدرس افست مفید است. یک مورد استفاده عمومی برای LEA برای مقداردهی یک افست در رجیسترهای DI، SI،BX برای نشانگذاری آدرس در حافظه است. به عنوان مثال

```
DATATBLE DB 23 DUP(?)      ; define A table
BYTEFLD DB ?               ; تعریفBYTEFLD به عنوان یک بایت با مقدار نامعلوم
  …
LEA BX، DATATBLE           ; بارگذاری آدرس افست
MOV BYTEFLD ، [BX]         ; انتقال اولین بایت ازDATATBLE
```

ث- دستور جمع و تفریق:

- **ADD/SUB register،register**

ADD/SUB memory،register
ADD/SUB register،memory
ADD/SUB register،immediate
ADD/SUB memory،immediate

به منظور مراقبت از سرریز محاسباتی، معمولاً از دستورات تبدیل بایت به کلمه (CBW) و کلمه به دو کلمه (CWD) استفاده می‌شود. دستورات اخیر به استفاده از AX محدود شده است. CBW بیت علامت را از AL به AH منتقل می‌کند.

CWD ; توسعه کلمه به DX:AX

CWD بیت علامت را از AH به DH منتقل می‌کند. برای پروسسورهای ۸۰۳۸۶ به بعد دستورات تبدیل کلمه به کلمه ی توسعه یافته (CWDE) و تبدیل دوکلمه به چهارکلمه ی توسعه یافته (CDQ) نیز وجود دارند.

CWDE ; توسعه کلمه به EAX
CDQ ; توسعه دوکلمه به EDX:EAX

مثال: برنامه ای بنویسید که ۱۶ بایت داده را از موقعیتی که با آدرس 20100H شروع می‌شود به موقعیتی که آدرس شروع آن 20120H است منتقل کند.

```
        MOV AX,2000
        MOV DS,AX
        MOV SI, 100
        MOV DI, 120
        MOV CX, 10
NXTPT:  MOV AH, [SI]
        MOV [DI], AH
        INC SI
        INC DI
        DEC CX
        JNZ  NXTPT
```

ج- دستورات پرش:

دستورات پرش به دو گروه شرطی و بدون شرط تقسیم می‌شوند. دستورات پرش بدون شرط آنهایی هستند که بدون شرط عمل پرش از موقعیت فعلی به موقعیت مقصد را انجام می دهند. دستورات پرش شرطی در صورت وقوع شرط مورد نظر، از موقعیت فعلی به موقعیت مقصد پرش انجام می دهند.

Opcode	Meaning	Condition
JA	Jump if above	CF=0 & ZF=0
JAE	Jump if above or equal	CF=0
JB	Jump if below	CF=1
JBE	Jump if below or equal	CF=1 or ZF=1
JC	Jump if carry	CF=1
JCXZ	Jump if CX=0	register CX=0
JE (is the same as JZ)	Jump if equal	ZF=1
JG	Jump if greater (**signed**)	ZF=0 & SF=OF
JGE	Jump if greater or equal (**signed**)	SF=OF
JL	Jump if less (**signed**)	SF != OF
JLE	Jump if less or equal (**signed**)	ZF=1 or SF!=OF
JMP	Unconditional Jump	-
JNA	Jump if not above	CF=1 or ZF=1
JNAE	Jump if not above or equal	CF=1
JNB	Jump if not below	CF=0
JNBE	Jump if not below or equal	CF=1 & ZF=0
JNC	Jump if not carry	CF=0
JNE	Jump if not equal	ZF=0
JNG	Jump if not greater (**signed**)	ZF=1 or SF!=OF
JNGE	Jump if not greater or equal (**signed**)	SF!=OF
JNL	Jump if not less (**signed**)	SF=OF
JNLE	Jump if not less or equal (**signed**)	ZF=0 & SF=OF
JNO	Jump if not overflow (**signed**)	OF=0
JNP	Jump if no parity	PF=0
JNS	Jump if not signed (**signed**)	SF=0
JNZ	Jump if not zero	ZF=0
JO	Jump if overflow (**signed**)	OF=1
JP	Jump if parity	PF=1
JPE	Jump if parity even	PF=1
JPO	Jump if paity odd	PF=0
JS	Jump if signed (**signed**)	SF=1
JZ	Jump if zero	ZF=1

مثال: برنامه ای بنویسید که تا ۱۰ حرف کوچک انگلیسی را که از صفحه کلید وارد می‌شود بخواند و ۱۰ حرف کوچک را به حروف بزرگ تبدیل کرده و آنها را روی صفحه نمایش دهد.

```
; ***              BEGIN OF THE PROGRAM                          ***
;***              DEFINE THE VARIABLES                          ***
CODE_SEG   SEGMENT
BEGIN      PROC FAR
           ASSUME CS:CODE_SEG،DS:DATA_SEG،SS:STACK_SEG
START:     MOV AX،DATA_SEG
           MOV DS،AX
;***              DISPLAY THE MESSAGE1                          ***
           MOV AH،09H
           LEA DX،MESSAGE1
           INT 21H
;***              READ THE CHARACTERS                           ***
READ_CH:   MOV AH،01H
           INT 21H
;***              JUMP IF THE CHARACTERS < ‚a‟                   ***
           CMP AL،61H
           JL READ_CH
;***              JUMP IF THE CHARACTERS < ‚z‟                   ***
           CMP AL،7BH
           JL CORRECT
           JMP READ_CH
;***              CHANGE THE SMALL LETTER TO CAPITAL LETTER ***
CORRECT:   ADD AL،-20H
;***              CHECK THE CHARACTERS READ                     ***
           MOV [MSG+SI]،AL
           ADD SI،1
           CMP SI،10
           JZ WRITE_MSG
;***              ECHO THE STRING READ                          ***
WRITE_MSG: MOV AH،09H
           LEA DX،MESSAGE2
           INT 21H
           LEA DX،MSG
           INT 21H
;***              END OF PROGRAM                                ***
END_SEG:   MOV AH،4CH
           INT 21H
BEGIN      ENDP
```

```
CODE_SEG   ENDS
;***              DATA SEGMENT                                    ***
DATA_SEG   SEGMENT
MESSAG1    DB "ENTER THE 10 CHARACTERS :$"
MESSAG2    DB " ;Corresponding CORRECT characters in capital letter  :$"
MSG        DB     10     DUP(?)
EDD_MSG    DB ,,$"
DATA_SEG   ENDS
;***              STACK SEGMENT                                   ***
STACK_SG   SEGMENT   STACK
           DW     10     DUP(?)
STACK_SEG ENDS
           END BEGIN
```

مثال: عملکرد برنامه زیر را توضیح دهد.

```
; ***                BEGIN OF THE PROGRAM                        ***
;***    PART(I):    CHANGE CHARACTERS TO DECIMAL INTEGERS  ***
;***                DEFINE THE VARIABLES                        ***
CODE_SEG   SEGMENT
BEGIN      PROC FAR
           ASSUME CS:CODE_SEG،DS:DATA_SEG،SS:STACK_SEG
START:     MOV AX،DATA_SEG
           MOV DS،AX
;***              DISPLAY THE MESSAGE1                          ***
           MOV AH،09H
           LEA DX،MESSAGE1
           INT 21H
;***              READ THE CHARACTERS                           ***
READ_CH:   MOV AH،01H
           INT 21H
;***              CHECK THE "RETURN"  INPUT                     ***
           CMP    AL،0DH
           JZ     PARTII
;***              JUMP IF THE CHARACTERS < ,,0"                 ***
           CMP    AL،30H
           JL     READ_CH
;***              JUMP IF THE CHARACTERS > ,,9"                 ***
           CMP    AL،39H
           JG     READ_CH
;***              CHANGE THE CHARACTERS TO DIGITS          ***
           ADD    AL،30H
           MOV    ADDER،AL
           MOV    AX،SUM
           MOV    SI،10
           MUL    SI
```

```
                ADD     AL،ADDER
                MOV     SUM،AX
                ADD     COUNT،1
;***                     CHECK THE 4 INPUTS                              ***
                CMP     COUNT،4
                JZ      PARTII
                JMP     READ_CH
;***                     END OF PART(I)                                  ***
;***     PART (II): CHANGE DECIMAL ENTEGER TO CHARACTERS                 ***
;***                     DISPLAY THE OUTPUT SIGNAL                       ***
PARTII:         MOV     AH،02H
                MOV     DX،000DH
                INT     21H
                MOV     DX،000AH
                INT     21H
                MOV     AH،09H
                LEA     DX،MESSAGE2
                INT     21H
                MOV     COUNT،0
;***                     CHANGE DECIMAL TO CHAR.                         ***
CONVERT:        MOV     AX،SUM
                MOV     DX،0
                MOV     SI،10
                DIV     SI
                PUSH    DX
                MOV     SUM،AX
                ADD     COUNT،1
                CMP     AX،0
                JG      CONVERT
;***                     ECHO THE STRING                                 ***
WRITE_MSG: POP  DX
                ADD     DX،30H
                MOV     AH،02H
                INT     21H
                ADD     COUNT،-1
                CMP     COUNT،0
                JG      WRITE_MSG
;***                     END OF PROGRAM                                  ***
END_SEG:        MOV AH،4CH
                INT 21H
BEGIN ENDP
CODE_SEG        ENDS
;***                     STACK SEGMENT                                   ***
STACK_SG        SEGMENT         STACK
                DW      40      DUP(?)
;
```

```
STACK_SEG   ENDS
;***                DATA SEGMENT                              ***
DATA_SEG    SEGMENT
MESSAG1     DB “Enter digits (max 4) :$”
MESSAG2     DB “ The output is:$”
SUM         DW 0
COUNT       DB 0
ADDER       DB 0
DATA_SEG    ENDS
            END BEGIN
```

در زیر مجموعه کامل دستورات پروسسور ۸۰۸۶ را ملاحظه می کنید. برخی از آنها به همرا ه توصیف عملکردشان در جدول صفحات بعد آمده اند.

Complete 8086 instruction set

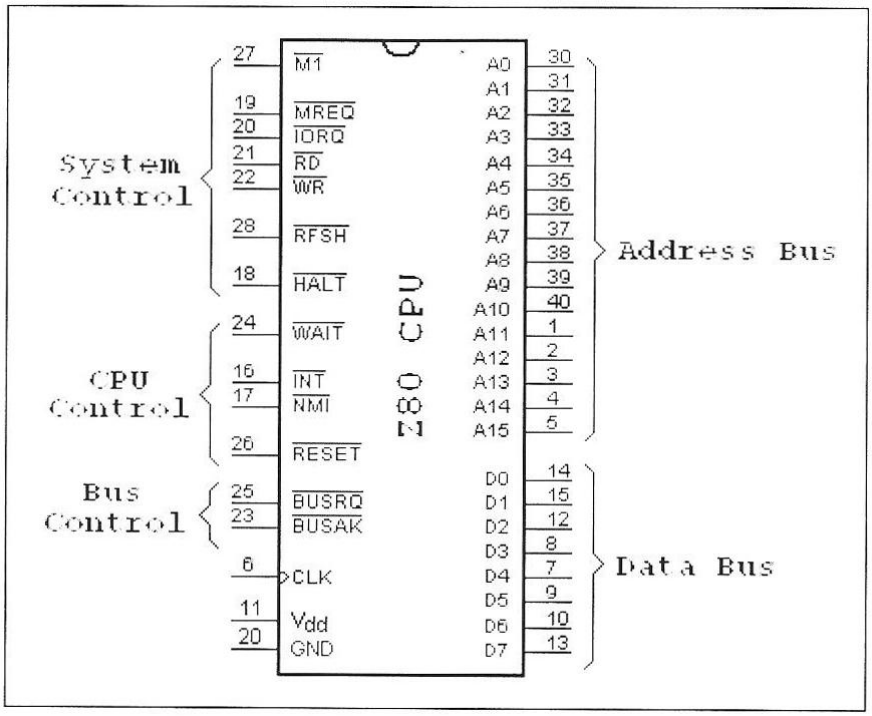
Quick reference:

	CMPSB				MOV		
AAA	CMPSW	JAE	JNBE	JPO	MOVSB	RCR	SCASB
AAD	CWD	JB	JNC	JS	MOVSW	REP	SCASW
AAM	DAA	JBE	JNE	JZ	MUL	REPE	SHL
AAS	DAS	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DEC	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DIV	JE	JNL	LEA	NOT	REPZ	STD
AND	HLT	JG	JNLE	LES	OR	RET	STI
CALL	IDIV	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IMUL	JL	JNP	LODSW	POP	ROL	STOSW
CLC	IN	JLE	JNS	LOOP	POPA	ROR	SUB
CLD	INC	JMP	JNZ	LOOPE	POPF	SAHF	TEST
CLI	INT	JNA	JO	LOOPNE	PUSH	SAL	XCHG
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA	SAR	XLATB
CMP	IRET	JNB	JPE	LOOPZ	PUSHF	SBB	XOR
	JA				RCL		

Instruction	Description
AAA	ASCII Adjust after Addition. Corrects result in AH and AL after addition when working with BCD values
AAD	ASCII Adjust before Division. Prepares two BCD values for division
AAM	ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values
AAS	ASCII Adjust after Subtraction. Corrects result in AH and AL after subtraction when working with BCD values
ADC	Add with Carry
ADD	Add. operand1 = operand1 + operand2
AND	Logical AND between all bits of two operands. Result is stored in operand1
CALL	Transfers control to procedure, return address is (IP) is pushed to stack. *4-byte address* may be entered in this form: 1234h: 5678h, first value is a segment, second value is an offset (this is a far call, so CS is also pushed to stack).
CBW	Convert byte into word
CLC	Clear Carry flag
CLD	Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW
CLI	Clear Interrupt enable flag. This disables hardware interrupts.
CMC	Complement Carry flag. Inverts value of CF
CMP	Compare. (operand1 - operand2) is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.
CMPSB	Compare bytes: ES:[DI] from DS:[SI].
CMPSW	Compare bytes: ES:[DI] from DS:[SI].
CWD	Convert Word to Double word
DAA	Decimal adjust After Addition. Corrects the result of addition of two packed BCD values.

DAS	Decimal adjust After Subtraction. Corrects the result of subtraction of two packed BCD values.
DEC	Decrement. operand = operand - 1
DIV	Unsigned divide.
HLT	Halt the System
IDIV	Signed divide. **when operand is a byte:** AL = AX / operand AH = remainder (modulus) **when operand is a word:** AX = (DX AX) / operand DX = remainder (modulus)
IMUL	Signed multiply. when operand is a **byte**: AX = AL * operand. when operand is a **word**: (DX AX) = AX * operand.
IN	Input from port into **AL** or **AX**. Second operand is a port number. If required to access port number over 255 DX register should be used.
INC	Increment. operand = operand + 1
INT	Interrupt numbered by immediate byte (0..255).
INTO	Interrupt 4 if Overflow flag is 1.
IRET	Interrupt Return.
JA	Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned.
JAE	Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned. if CF = 0 then jump
JB	Short Jump if first operand is Below second operand (as set by CMP instruction). if CF = 1 then jump
JBE	Short Jump if first operand is Below or Equal to second operand (as set by CMP instruction).

ساختار پایه ای Z80 : در شکل زیر نمای پایه ای و دسته بندی پایه های میکروپروسسور Z80 نشان داده شده است.



خطوط آدرس (Address Bus) : 16 پایه به این کار اختصاص دارد (A0 ~ A15) که توسط آن CPU توانایی آدرس دهی 65K خانه ی حافظه را دارد. این خطوط علاوه بر آدرس دهی خانه های حافظه، برای آدرس دهی پورت های ارتباطی نیز به کار می رود.

خطوط داده (Data Bus) : تعداد این خطوط 8 عدد می‌باشد. به عبارت دیگر خطوط داده ی CPU از نوع 8 بیتی می‌باشد و CPU توانایی پردازش اطلاعات را به صورت 8 بیتی دارد.

M1 (Machine Cycle 1) : این پایه زمانی فعال می‌شود که CPU از حافظه دستور بر می دارد (در پایان دریافت کد عملیاتی و آغاز اجرای آن فعال می‌شود). بنابراین فعال شدن این پایه نشانگر دریافت کد عملیاتی و اجرای آن توسط CPU است.

Mem Req (Memory Request) : زمانی که CPU قصد برقراری ارتباط با حافظه ی جانبی را داشته باشد، این خط فعال می‌شود. به عبارت دیگر فعال شدن این خط به این معنا است که CPU می خواهد با حافظه ی جانبی ارتباط برقرار کند.

IOReq (Input / Output Request) : این خط زمانی فعال می‌شود که CPU قصد برقراری ارتباط با خطوط I/O را دارد (پورت های ورودی / خروجی).

RD (Read) : این پایه زمانی فعال می‌شود که CPU قصد خواندن اطلاعات از حافظه ی جانبی و یا یک پورت را داشته باشد. این خط نیز از نوع فعال به صفر (active low) می‌باشد.

WR (Write) : زمانی که CPU بخواهد اطلاعاتی را در خانه ای از حافظه یا پورت خارجی بنویسد (ارسال کند)، این خط فعال می‌شود. این خط از نوع فعال به صفر (active low) می‌باشد. به این معنا که با فعال شدنش، به وضعیت صفر منطقی می رود.

RFSH (نوسازی Refresh) : این پایه برای زمانی به کار می رود که در سیستم حافظه ی دینامیکی استفاده شده باشد و برای نوسازی اطلاعات درون حافظه ی دینامیکی به کار می رود.

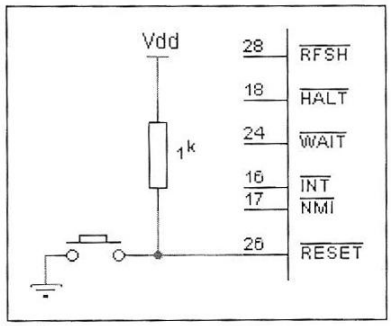
HALT : یک پایه ی خروجی است که وقتی دستور HALT توسط CPU پردازش می‌شود، فعال می‌شود. (HALT یک دستور نرم افزاری است که درآخر برنامه قرار می‌گیرد). این پایه از نوع فعال به صفر است.

Wait : برای همزمانی و هماهنگ کردن CPU با یک حافظه ی کندتر، درگاه جانبی و یا هنگام عیب یابی برنامه به کار می رود که با صفر شدن این خط CPU به حالت انتظار می رود تا درگاه یا حافظه ی جانبی اعلام آمادگی کند.

INT (درخواست وقفه Interrupt) : این پایه ی مخصوص درخواست وقفه ی ادوات جانبی از CPU می‌باشد. این نوع وقفه از نوع قابل چشم پوشی می‌باشد به این معنا که می‌توان آن را غیر فعال کرد (توسط نرم افزار). وقفه ی قابل چشم پوشی دارای 3 مد یا حالت می‌باشد که در ادامه به بحث در مورد آن می‌پردازیم. پایه ی مخصوص این وقفه نیز از نوع فعال به صفر است.

NMI (Non Maskable Interrupt) : این پایه مختص درخواست وقفه از CPU می‌باشد. تفاوت این پایه با پایه ی INT این است که این نوع وقفه غیر قابل چشم پوشی است و نمی‌توان آن را غیر فعال کرد. به این معنا که وقتی درخواست این وقفه شد (این پایه فعال شد یا به وضعیت صفر منطقی رفت)، بدون درنگ وقفه رخ می‌دهد.

RESET : این پایه ی مربوط به ریست کردن خارجی CPU می‌باشد. با این ریست شمارنده ی برنامه (PC) صفر می‌شود تا برنامه از آدرس صفر شروع به اجرا کند. ریست این CPU از نوع فعال به صفر است و در حالت معمولی باید توسط یک مقاومت بالا کش (Pull Up) به Vdd متصل شود تا در حالت عادی این پایه دارای وضعیت یک منطقی باشد.



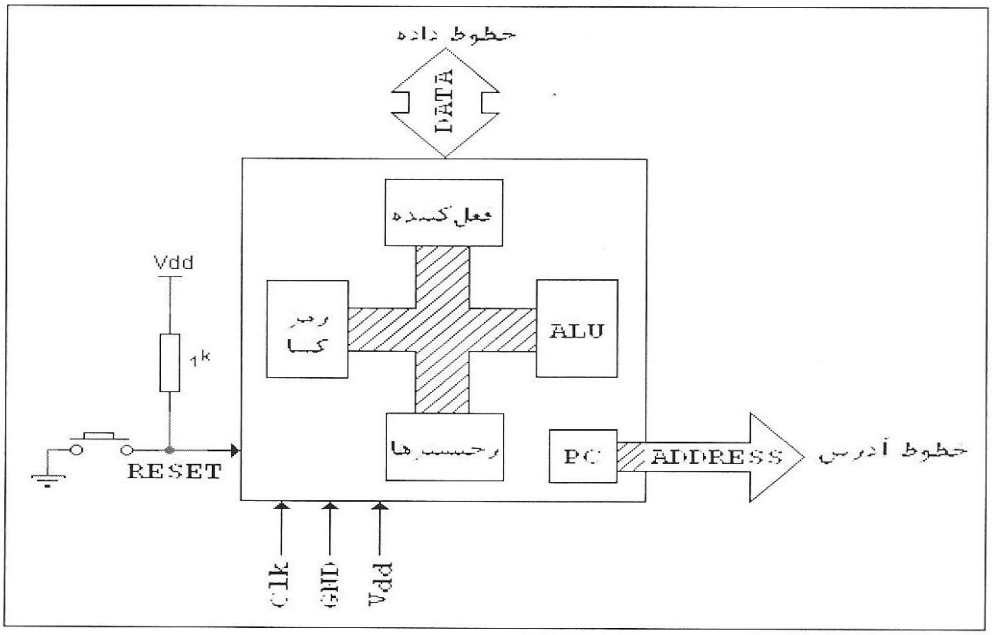
Bus RQ (Bus Request) و Bus AK (Bus Acknowledge) : از این دو پایه زمانی استفاده می‌شود که بخواهیم دو CPU باهم ارتباط برقرار کنند و به صورت مشترک به پردازش اطلاعات بپردازند. BusRQ برای درخواست گذرگاه می‌باشد و BusAK برای تأیید عمل درخواست می‌باشد. نحوه ی کار به این صورت است که زمانی که CPU1 در حال پردازش اطلاعات و ارتباط با خطوط اطلاعاتی است و CPU2 بخواهد وارد مدار شود و با خطوط اطلاعاتی ارتباط برقرار کند، پایه ی BusRQ از CPU1 را فعال می‌کند. در این زمان CPU1 پایه های خود را امپدانس بالا (های امپدانس) می‌کند و سپس پایه ی BusAK از CPU2 را فعال می‌کند تا به CPU2 بگوید که اجازه ی استفاده از خطوط را دارد.
به این روش **پردازش موازی** می‌گویند.

Vdd : پایه ی مربوط به تغذیه ی مثبت تراشه است و به 5 ولت متصل می‌شود.

GND: اتصال زمین برای تراشه است.

CLK : پایه ای است که کلاک سیستم باید به آن اعمال شود (در CPU همانند دیگر مدارات لاجیکی، عمل هماهنگی و زمانبندی اجرای دستورات توسط کلاک پالس تنظیم می‌شود). هر میزان فرکانس کلاک اعمالی به تراشه بالاتر باشد، سرعت پردازش اطلاعات نیز بیشتر می‌شود. حداکثر فرکانس کلاکی که می‌توان به Z80 اعمال کرد برابر با 2 MHZ می‌باشد.

ساختمان داخلی Z80: ساختمان داخلی این میکروپروسسور را می‌توان در شکل زیر خلاصه کرد.



نحوه ی کار سیستم : پس از وصل تغذیه، شمارشگر برنامه یاPC ، از صفر شروع به شمارش می‌کند و CPU از آدرس اول حافظه (عدد درون PC) شروع به خواندن اطلاعات می‌کند. به محض دریافت اطلاعات از حافظه، قفل کننده ی ورودی، اطلاعات دریافتی را در خود قفل می‌کند و پس از آن به رمزگشا می فرستد. بعد از عمل رمزگشایی، ALU با توجه به عدد رمزگشایی شده، عمل متناسب را انجام می‌دهد.

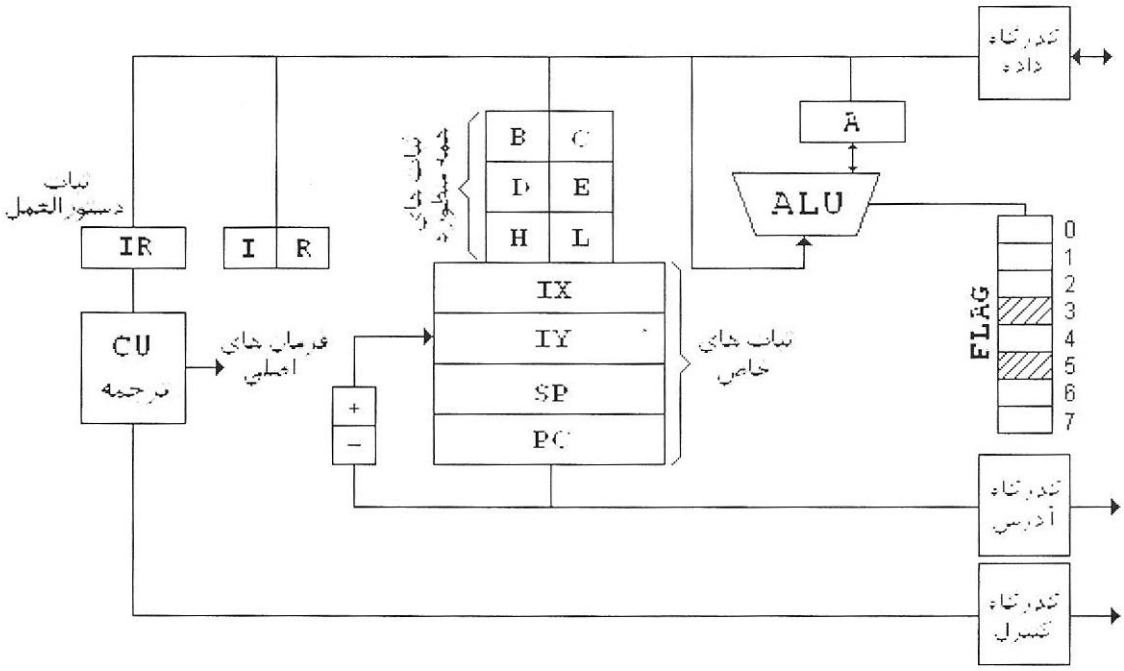
اجزای داخلی پردازنده Z80 :

ساختمان داخلی Z80 را به صورت کامل در شکل صفحه بعد آمده است که در ادامه به توضیح اجزای آن می‌پردازیم

ALU یا واحد محاسبات و منطق :کلیه ی عملیات جمع، تفریق و... و عملیات منطقی AND، OR و... در این واحد انجام می‌شود. این واحد را می‌توان به صورت ساده یک جمع کننده و تفریق کننده ی FULL فرض کرد البته با تغییرات و پیچیدگی های خاص خودش.

واحد کنترلCU(Control Unit) : واحد کنترل است.

.



رجیسترها (ثبات ها) :محلی برای ثبت اطلاعات می باشند که بر روی خود تراشه قرار دارند و از نظر عملکردی همانند RAM می باشند. Z80 دارای 208 بیت یا 26 بایت حافظه ثباتی می‌باشد که شامل 16 ثبات 8 بیتی و 5 ثبات از نوع 16 بیتی است. ثبات های داخلی Z80 را می‌توان بصورت زیر دسته بندی کرد.

الف- ثبات های همه منظوره :این رجیستر ها که شامل B، C، D، E، H و L نشان داده می‌شوند همگی 8 بیتی هستند.

ب- انباره یا آکومولاتور (Accumulator) : پر کاربردترین رجیستر می‌باشد که اصلی ترین کارها مانند انتقال، جمع و... با کمک این رجیستر صورت می‌گیرد.

پ- رجیستر های 16 بیتی معمولی: همان رجیسترهای 8 بیتی هستند که به صورت دو به دو در کنار هم قرار می‌گیرند و به رجیستر 16 بیتی تبدیل می‌شوند. این رجیسترها عبارتند از BC ، DE ، HL.

ت- رجسترهای مخصوص Special Purpose Registers : این رجیسترها که دارای عملکردهای خاص خود هستند، به شرح زیر می‌باشد :

شمارنده ی برنامه یا PC (Program Counter) : یک رجیستر 16 بیتی است و محتوای آن آدرس خانه ای است که قرار است اطلاعات درون آن برداشته شود. وقتی اطلاعات از خانه ی مورد نظر حافظه برداشته شد، به صورت اتوماتیک یک واحد به مقدار PC اضافه می‌شود (آدرس خانه ی بعدی تولید می‌شود).

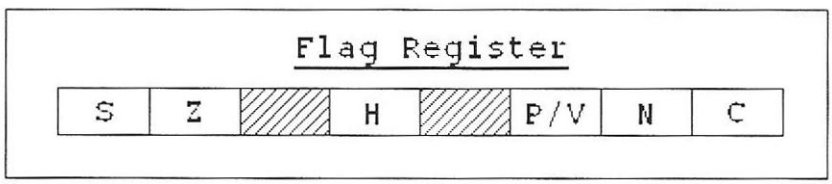
رجیستر اشاره گر پشته یا SP (Stack Pointer) : این رجیستر که 16 بیتی می‌باشد، همواره محتوی خانه ای از حافظه است که به صورت رزرو شده برای CPU می‌باشد و به اولین خانه ی خالی پشته اشاره می‌کند و برای اعمالی مانند Push و POP مورد استفاده قرار می‌گیرد.

رجیسترهای اشاره گر IY و IX (Index Registers) : دو رجیستر 16 بیتی هستند که به رجیسترهای اشاره گر معروفند. توسط چند دستورالعمل و این رجیسترها می‌توان اطلاعاتی را به صورت جدول و ستون جا بجا کرد (در کل برای جابجایی از آن استفاده می‌شود).

رجیستر وقفه یا I (Interrupt) : این رجیستر محتوی 8 بیت پر ارزش آدرس زیر برنامه وقفه است. با فعال شدن پایه ی وقفه، CPU اجرای برنامه ی اصلی را قطع می‌کند و به آدرس جدیدی می رود که 8 بیت پر ارزش آن توسط رجیستر I و 8 بیت کم ارزش آن توسط وقفه ی خارجی تأمین می‌شود.

رجیستر نوسازی یا R (Refresh Register) : یک رجیستر 7 بیتی است که در رابطه با نوسازی حافظه های دینامیکی به کار می رود. با دریافت هر دستور یک واحد به آن اضافه می‌شود و محتوای ان روی خطوط ادرس A6 – A0 قرار گرفته و در حافظه های دینامیکی سیستمی قرار گرفته که با این سیگنال ها عمل نوسازی خود به خود انجام می‌شود. عمل نوسازی با قرار دادن صفر بر روی پایه ی RFSH از CPU توسط حافظه ی دینامیکی شروع می‌شود و تنها زمانی که CPU مشغول کشف رمز یک دستور العمل می‌باشد، خطوط آدرس در اختیار عمل نوسازی قرار می‌گیرد.

رجیستر پرچم یا FLAG : این رجیستر از نوع 8 بیتی است که 2 بیت آن بدون استفاده می‌باشد وظیفه ی این رجیستر نشان دادن برخی از نتایج عملیات ها می‌باشد. ساختار این رجیستر به صورت زیر است :



بیت C(Carry) : زمانی که در حین عمل جمع دو عدد، رقم نقلی به وجود آید، این بیت 1 می‌شود (همچنین بعد از بیت قرضی در عمل تفریق).

بیت N (Negative) : بیت تفریق می‌باشد و زمانی یک می‌شود که آخرین دستورالعمل که در CPU اجرا شده، یک تفریق می‌باشد.

بیت P/V : این بیت دو وظیفه دارد. یکی نمایشگر سرریز که وقتی حاصل محاسبه از 8 بیت بزرگتر شود، این بیت 1 می‌شود. و وظیفه ی دوم به عنوان غلط یاب با استفاده از توازن، به این صورت که اگر تعداد یک های

موجود در نتیجه فرد باشد، این بیت 0 می‌شود (اصطلاحا Parity Odd) و اگر تعداد یک های موجود زوج باشد (Parity Even) این بیت 1 می‌شود.

بیت H (Half Carry) : این بیت مخصوص حامل میانی است. از این بیت خود ریز پردازنده برای تشکیل اعداد BCD در رجیسترها استفاده می‌کند.

بیت Z (Zero) : این بیت زمانی 1 می‌شود که نتیجه ی محاسبه صفر باشد.

بیت S (Sign) : بیت علامت می‌باشد. اگر نتیجه ی عملیات منفی باشد این بیت 1 می‌شود و اگر مثبت باشد این بیت صفر می‌شود (عدد منفی یعنی بیشتر از 127).

مدهای آدرس دهی در Z80 :

دستورهای Z80 معمولاً بر روی اطلاعاتی که درون رجیسترهای داخلی، حافظه های جانبی و یا تراشه های ورودی - خروجی، عملیاتی را انجام می‌دهند. آدرس به روشی گفته می‌شود که CPU آدرس این حافظه ها و... را در هنگام عمل مورد نظر تولید می‌کند. به طور کلی کد مربوط به هر دستور خوانده شده، علاوه بر نوع عملیات، مشخص می‌کند که آیا دستور مورد نظر مربوط به رجیسترهای داخلی است یا حافظه های جانبی و یا...

آدرس دهی فوری (Immediate Addressing) : دستورالعمل ها عموما به صورت یک کلمه ای، دو کلمه ای و یا سه کلمه ای می باشند. کلمه اول کد عملیاتی (Opcode) است که نوع عمل را مشخص می‌کند و کلمه ی دوم به صورت داده (Data) می‌باشد که آن را عملوند می نامند و در این نوع آدرس دهی عملوند (اپراند) به صورت 8 یا 16 بیت بلافاصله پشت کد عملیاتی آورده می‌شود.

آدرس دهی توسعه یافته (Extended Addressing) : در این نوع آدرس دهی معمولاً کل دستور شامل 3 کلمه می‌باشد، کلمه ی اول کد عملیاتی و کلمه های دوم و سوم آدرس های حافظه ای می‌باشد که داده ی مورد نظر در ان قرار گرفته است. با این نوع آدرس دهی در اصل دسترسی به آن داده امکان پذیر می‌شود.

آدرس دهی رجیستر (Register Addressing) : در آدرس دهی رجیستر همان طور که از نامش پیداست صرفا انتقال داده بین خود رجیسترها صورت می‌گیرد. مثلاً وقتی بخواهیم محتوای رجیستر (r) را به رجیستر دیگری انتقال دهیم از دستور LD D ،Sاستفاده می کنیم که در آن این جا S مبداء و D مقصد است. یعنی محتوای رجیستر S باید به D انتقال داده شود.

آدرس دهی نسبی (Relative Addressing) : این نوع ادرس دهی معمولاً در زمان جهش (Jump) مورد استفاده قرار می‌گیرد و شامل دو بایت دستور می‌باشد، یکی خود کد ماشین و دیگری عددی که مشخص کننده ی میزان پرش می‌باشد که معمولاً به صورت متمم 2 نوشته می‌شود.

آدرس دهی غیر مستقیم رجیستر (Register Indirect Addressing) : این نوع ادرس دهی تقریبا شبیه آدرس دهی توسعه یافته می‌باشد و زمانی به کار برده می‌شود که بخواهیم محتوای یک حافظه ی خارجی را با استفاده از رجیستر (HL) به عنوان اشاره گر به داخل A انتقال دهیم و بالعکس. مثلاً وقتی نوشته می‌شود ،A (HL) LD به این معنا است که محتوای A باید به آدرسی که در رجیستر HL مشخص شده است، منتقل شود.

آدرس دهی شاخص دار (Indexed Addressing) : در این نوع آدرس دهی از دو رجیستر IX وIY برای انتقال محتوای یک حافظه ی به رجیسترهای داخلی استفاده می‌گردد. شکل اسمبلی دستور آن به صورت (IX + d)، A LD می‌باشد که در این جا A رجیستر مقصد و d یک عدد ثابت و محتوای رجیستر IX که 16 بیت است در برگیرنده ی آدرس حافظه مورد نظر می‌باشد و کل دستور به این معنی است که محتوای آدرسی که در رجیستر (IX + d) است به A انتقال دهد.

آدرس دهی ضمنی (Implied Addressing) : آدرس دهی ضمنی به نوعی ادرس دهی گفته می‌شود که در آن کد عملیاتی خود به خود به یک یا چند رجیستر اشاره نماید. مثلاً در دستورهای محاسباتی، رجیستر A خود به خود مقصد حاصل جمع واقع می‌شود.

بخش دوم:

مختصری راجع به

میکروکنترلرهای AVR

میکروکنترلر یا میکروپروسسور؟

یکی از سؤالاتی که ذهن هر علاقمند به الکترونیک را به خود مشغول می‌کند، این است که چه تفاوتی بین میکروپروسسور و میکروکنترلر وجود دارد و یا اصلا " چرا با وجود میکروکنترلرهای قوی، هنوز در سیستمهای جدید از میکروپروسسور استفاده می‌شود؟! اگر به دنبال جواب این سؤال هستید، این قسمت را با دقت مطالعه کنید:

اجازه دهید در ابتدا تعریفی کلی از یک میکروپروسسور و یک سیستم میکروپروسسوری داشته باشیم. یک میکروپروسسور صرفنظر از ساختمان داخلی آن، اساسا " یک CPU (Central Processing Unit) است و همانگونه که از اسمش بر می‌آید، وظیفه پردازش اطلاعات را بر عهده دارد. یک میکروپروسسور برای کار نیاز به مکانی دارد که دستورالعملهای مورد نیاز آن، در آنجا ذخیره شده باشد که به آن حافظه برنامه می‌گویند و از نوع و خانواده ROM ، میباشد. همچنین CPU نیاز به مکانی دارد که نتایج حاصل از پردازش را در آنجا بریزد که به آن حافظه داده می‌گویند و از نوع و خانواده RAM است. به علاوه CPU برای ارتباط با دنیای خارج نیاز به آی سی ورودی، خروجی دارد. تمامی موارد ذکر شده، تشکیل یک سیستم میکروپروسسوری را می‌دهند. ملاحظه می‌کنید که یک سیستم میکروپروسسوری بسیار ساده و اولیه، برای کار نیاز به ۴ جزء دارد که باعث زیاد شدن حجم بورد و همچنین هزینه نهایی خواهد شد.

در یک تعریف کلی، یک میکروکنترلر ساده از تمامی امکانات ذکر شده برای یک سیستم میکروپروسسوری، به اضافه سیستم کلاک ، تایمر و کانتر بر روی یک آی سی، تشکیل شده است. میکروکنترلرهای جدید مانند میکروکنترلرهای خانواده ATMEGA از سری میکروکنترلرهای AVR، دارای امکاناتی مانند مبدل آنالوگ به دیجیتال، مقایسه کننده آنالوگ، مدولاتور PWM، اسیلاتور WATCHDOG، حافظه EEPROM داخلی و.....، نیز می‌باشند.

حال سؤال این است که چرا با وجود این همه امکانات در میکروکنترلرها هنوز از میکروپروسسورها استفاده می‌شود ؟

پاسخ این است که میکروپروسسورها علی رغم کم بودن امکاناتشان، دارای قدرت پردازشی به مراتب بیشتر از میکروکنترلرها می‌باشند، ضمن اینکه میکروپروسسورها پردازنده های همه منظوره هستند، بر خلاف میکرو کنترلرها که دارای قدرت پردازش کمتری می‌باشند و برای اهداف خاصی طراحی شده‌اند.

از این رو در کامپیوتر از یک میکروپروسسور یا به اصطلاح همان CPU، به دلیل قدرت بالای پردازش آن استفاده شده است اما در سیستم کنترل دما و رطوبت یک گلخانه، به دلیل عدم نیاز به قدرت پردازش زیاد و وجود تمام امکانات مورد نیاز برای کنترل دما و رطوبت، در یک میکروکنترلر، از یک آی سی میکرو کنترلر استفاده شده است.

خانواده میکروکنترلرها: تمام میکروکنترلرها جزو یکی از شش خانواده زیر هستند

الف- 8051

ب- PIC

پ- AVR

ت- 6811

ث- Z8

ج- ARM

البته مدلهای 6811 ساخت شرکت موتورولا و Z8 ساخت شرکت زایلوگ حداقل در ایران خیلی کم استفاده می‌شوند و رقابت اصلی بین چهار نوع دیگر است.

تا به امروز هر میکروکنترلری که ساخته شده زیر مجموعه یکی از ۶ خانواده فوق بوده است. اگرچه کارخانه های خیلی زیادی با مارکهای مختلف میکرو کنترلر تولید می‌کنند ولی همه آنها زیر مجموعه یکی از این ۶ خانواده هستنند. شما برای هر کدام از این۶ نوع میکروکنترلر می‌توانید میکروکنترلر های مختلفی از شرکتهای مختلف را پیدا کنید. (البته در بازار ایران کمی با مشکل مواجه می‌شوید).

اما خوشبختانه همه میکروکنترلر هایی که جزء هر کدام از ۶ خانواده بالا باشند از یک برنامه پیروی می‌کنند. بدین معنا که اگر شما کار با یکی از مدل های آن خانواده میکرو را یاد گرفته باشید مثل اینکه کار با تمام میکروکنترلرهای آن خانواده را یاد گرفته اید. مثلاً اگر شما یکی از مدل های میکروکنترلر AVR نظیر ATMEGA8 را یاد گرفته باشید دیگر با صد ها مدل دیگر میکروکنترلر AVR مشکلی ندارید و تقریباً بدون هیچ مشکلی می‌توانید با دیگر مدل های این میکرو هم کار کنید. اما یک مشکل که در میکروکنترلر ها وجود دارد این است که این ۶ نوع از لحاظ برنامه نویسی به هیچ وجه با هم دیگر سازگاری ندارند. به طور مثال اگر شما میکروکنترلر های AVR و ۸۰۵۱ را کامل یاد گرفته باشید حتی ساده ترین برنامه را روی یک میکروکنترلر PIC نمی‌توانید اجرا کنید و این یکی از بزرگترین عیب و مشکل برای یادگیری میکروکنترلر است. بنابراین از همان اول باید یک انتخاب درست داشته باشید و میکروکنترلر مناسب را برگزینید تا با یادگیری آن نوع میکروکنترلر بتوانید بعداً به سادگی پروژه های خود را اجرا کنید.

معایب و مزایای میکروکنترلر های مختلف نسبت به هم:

از آنجا که 6811 و Z8 در بازار ایران کمیاب هستند و خیلی کمتر استفاده می‌شوند به معرفی چهار نوع دیگر می‌پردازیم.

اول از ۸۰۵۱ که اولین میکروکنترلری بود که به دست بشر ساخته شد شروع می‌کنیم. این میکروکنترلر توسط شرکت بزرگ INTEL ساخته شد. اما بعداً INTEL این امکان را به دیگر شرکت ها داد که این میکروکنترلر را تولید کنند و شرکت هایی مانندATMEL، PHILIPS، SIEMENS ، DALLAS و... به تولید این میکروکنترلر پرداختند یکی از شرکت هایی که به صورت گسترده به تولید این تراشه پرداخت ATMEL بود که مدل های مختلف این میکروکنترلر، ساخت این شرکت، در سراسر جهان و در ایران به خوبی یافت می‌شود. اما اگر بخواهیم به صورت کلی سیر پیشرفت این نوع میکروکنترلر را در نظر بگیریم اولین میکروکنترلر هایی که ساخته شد با جدیدترین میکروکنترلرهای ۸۰۵۱ که الان تولید می‌شود با توجه به این پیشرفت شگفت در تمام زمینه ها که صنایع دیگر در دنیا دارند پیشرفت زیادی ندارد به طور مثال AT89S5Xکه میکروکنترلر ۸۰۵۱ جدید ساخت ATMEL است نسبت به مدل های اولیه ۸۰۵۱ پیشرفت آنچنانی ندارد. امکانات این میکرو نسبت به AVR و PIC قابل مقایسه نیست. به صورتی که همین مدل جدید ۸۰۵۱ تقریباً حافظه ای برابر یک صدم میکروکنترلر های AVR را دارد و سرعتش ۴ برابر کمتر از میکروکنترلر های PIC و ۱۲ بار کمتر از میکروکنترلر های AVR است. از لحاظ امکانات دیگر هم، چنین ضعفی احساس می‌شود. اما برای کارهای ساده تر که پیچیدگی زیادی در آن نباشد به خاطر قیمت بسیار پایینی که این میکروکنترلر دارد بسیار مناسب است. این میکرو کنترلر از زبان اسمبلی و C پشتیبانی می‌کند که زبان برنامه نویسی اصلی آن اسمبلی است که واقعا نوشتن با این زبان

برنامه نویسی نسبت به زبان های برنامه نویسی دیگر هم مشکل تر و هم طولانی تر است. در کل این میکروکنترلر امروزه دیگر توانای رقابت با AVR و PIC و ARM را ندارد و امروزه رقابت اصلی بین AVR ، PIC و ARM است.

میکروکنترلر PIC واقعا میکروکنترلر خیلی قوی است که بر اساس بعضی آمار ها بیشترین کاربر را به خود اختصاص داده است البته متذکر شوم که در ایران این آمار به نفع AVR است. این میکروکنترلر ساخت شرکت میکرو چیپ است که PIC را در مدلهای خیلی زیاد و با امکانات مختلف برای کارهای مختلف روانه بازار کرده است. این میکروکنترلر با مدل های مختلف PIC16XXX و PIC12XXXX که به جای X دوم از چپ به راست حروف C، X، E، F قرار می‌گیرد که هر کدام مفهوم خاصی دارد که چون بحث ما آموزش AVR است از روی آن سریع می‌گذریم. X های بعدی هم اعدادی هستنند که نشان دهنده مدل های مختلف این نوع میکرو هستنند.

میکروکنترلر AVR به نظر خیلی از کاربران، بهترین میکروکنترلر موجود در بازار ایران است. برخی مزایای آن را می‌توان بصورت زیر بیان کرد.

الف- سرعت این میکروکنترلر بسیار بالاست و دستورانش را در یک سیکل کلاک انجام می‌ده د در صورتی که این سیکل کلاک برای ۸۰۵۱ باید تقسیم بر۱۲شود و برای PIC تقسیم بر ۴. بنابراین AVR سریعترین میکروکنترلر موجود در بازار است(البته بعد از میکروهای ARM).

ب- AVR از زبان های برنامه نویسی سطح بالا یا به اصطلاح HLL (High Level Laanguage) پشتیبانی می‌کند که باعث تولید کدهای کوتاهتری می‌شود که در کل برنامه نوشته شده نسبت به برنامه هایی که برای ۸۰۵۱ و PIC نوشته می‌شود کوتاهتر است.

ج- امکانات جانبی این میکروکنترلر بسیار مناسب است و شما را از خرید بعضی لوازم جانبی مانند چیپ های آنالوگ به دیجیتال(ADC) ، مقایسه گر آنالوگ و... راحت می‌کند. در ضمن AVR از بسیاری از استاندارد های ارتباطی مانند SPI،UART،I2C ، JTAG پشتیبانی می‌کند که به راحتی می‌توان این میکروکنترلر را با میکروکنترلر دیگر یا و سایل دیگر وصل و به راحتی با آنها ارتباط برقرار کرد. قیمت این میکروکنترلر هم به نسبت امکانات فراوانی که دارد بسیار پایین است به طوری که یک میکروکنترلر AVR تقریباً پیشرفته را با قیمت حدود ۸ هزار تومان می‌توان خرید.

میکروهای AVR از میکروکنترلرهای قوی ۸ بیتی عرضه شده به بازار است که متعلق به شرکت ATMEL می‌باشد.

سرعت بالا، کاهش و بهینه سازی کدهای مورد استفاده، انجام بیشتر عملیاتها در یک ماشین سیکل (برخلاف میکروی ۸۰۵۱ که کلاک آن تقسیم شده کلاک ورودی بر ۱۲ است) و استفاده از ۳۲ رجیستر همه منظوره (که افزایش سرعت ۴ تا ۱۲ برابر میکروهای دیگر را باعث می‌شود)، برخی مزایای میکروهای AVR نسبت به سایر میکروهاست.

انواع میکروهای AVR:

میکروهای AVR به لحاظ امکانات به سه گروه Tiny AVR، AT90S (Classic) و MEGA AVR تقسیم می‌شوند که به ترتیب دارای قابلیتهای بیشتر، حافظه بیشتر، توان مصرفی بالاتر و تعداد ورودی/خروجی بیشتری هستند. تفاوت بین این سه نوع به امکانات موجود در آنها مربوط می‌شود. Tiny AVRها غالباً تراشه‌هایی با تعداد پین و مجموعه دستورات کمتری نسبت به mega AVRها هستند و از لحاظ پیچیدگی حداقل امکانات را دارند. Mega AVRها حداکثر امکانات در

بین این سه نوع را دارند. به عنوان مثال میکروی ATtiny28 دارای ۹۰ دستورالعمل با کارایی بالاست که اکثراً در یک ماشین سیکل انجام می‌شود. همچنین دارای ۲ کیلوبایت حافظه FASH است. میکروی AT90S8535 دارای ۱۱۸ دستورالعمل با کارایی بالاست که اکثراً در یک ماشین سیکل انجام می‌شود و همچنین دارای ۸ کیلوبایت حافظه FASH، ۵۱۲ بایت حافظه EEPROM داخلی و ۵۱۲ بایت حافظه SRAM است و بالاخره میکروی MEGA16 دارای ۱۳۱ دستورالعمل با کارایی بالاست که اکثراً در یک ماشین سیکل انجام می‌شود. همچنین دارای ۱۶ کیلوبایت حافظه FASH، ۵۱۲ بایت حافظه EEPROM داخلی و ۱۰۲۴ بایت حافظه SRAM است. جدول زیر برخی از اعضای خانواده AVR را به لحاظ امکانات نشان می‌دهد.

PART	PINS	SPEED	FLASH	EEPROM	RAM	UART	ADC
90S4414	40	10MHZ	4K	256	256	YES	NO
90S8515	40	8MHZ	8K	512	512	YES	NO
90S4434	40	10MHZ	4K	256	256	YES	YES
Mega103	64	6MHZ	128K	4096	4096	YES	YES
Mega603	64	6MHZ	64K	2048	4096	YES	YES
Tiny10	8	10MHZ	1K	64	0	NO	NO
Tiny12	8	10MHZ	1K	64	0	NO	NO
Tiny13	8	10MHZ	2K	128	128	NO	NO
Tiny22	8	10MHZ	2K	128	128	NO	NO

AVRها دارای سه نوع حافظه FLASH (برای ذخیره کدهای برنامه)، EEPROM و SRAM هستند.

AVR ها دارای ۳۲ رجیستر(R_0 تا R_{31}) هستند که مستقیماً به ALU متصل شده‌اند و تنها در یک کلاک سیکل به این واحد دسترسی پیدا می‌کنند. سه جفت از این رجیسترها می‌توانند به عنوان رجیسترهای ۱۶ بیتی X، Y و Z استفاده شوند (رجیسترهای R_{26} و R_{27} به عنوان رجیستر X، رجیسترهای R_{28} و R_{29} به عنوان رجیستر Y و رجیسترهای R_{30} و R_{31} به عنوان رجیستر Z). این رجیسترها ۳۲ مکان اول حافظه SRAM را اشغال کرده اند.

امکانات کلی یک AVR

- در حدود ۱۳۰ دستور که اکثر آنها در یک سیکل ساعت اجرا می‌شوند.
- ۳۲ رجیستر ۸ بیتی همه منظوره.
- ضرب کننده‌ی سخت افزاری با زمان اجرای ۲ سیکل ساعت.
- دارای سه نوع حافظه FLASH (برای ذخیره کدهای برنامه)، EEPROM وSRAM.
- برنامه ریزی تراشه در داخل مدار مورد نظر (ISP) یا وجود ارتباط سریالISP برای برنامه ریزی (پروگرام کردن) داخل مدار (هنگامی که میکرو داخل مداراست با پروگرامرISP میتوانید میکرورا برنامه ریزی کنید. برای برنامه ریزی ازچهارخط RESET، MISO ،SCK و MOSI استفاده می شود).

- قابلیت ارتباط ارتباط سریالSPI به صورت master/slave
- قابلیت ارتباطJTAG (یک نوع ارتباط که از طریق ان می‌توان حافظه های قابل برنامه ریزی میکرو را خواند یا نوشت).
- حفاظت از کدهای برنامه در مقابل خواندن(قفل برنامه داخل حافظهEEPROM و FLASH برای جلوگیری از خواندن آن).
- قابلیت تنظیم نوسانگر برای کار توسط کریستال خارجی، کریستال فرکانس پایین، نوسانگر RC خارجی، نوسانگر RC داخلی و فرکانس خارجی.
- شمارنده و تایمر ۸ بیتی و شمارنده و تایمر ۱۶ بیتی
- RTC یا نوسانگر جداگانه
- کانال های PWM
- امکان تنظیم تایمر به صورت CTC
- ADC های ۱۰ بیتی با ورودی تک سر و یا ورودی تفاضلی با بهره‌ی قابل تنظیم ۱، ۱۰ و ۲۰۰
- مجهز به پروتکل I^2C یا TWI
- ارتباط سریال USART با قابلیت برنامه ریزی.
- تایمر نگهبان قابل برنامه ریزی با نوسانگر مجزا.
- مقایسه کننده‌ی آنالوگ با امکان تعریف وقفه برای آن.
- RESET شدن در زمان اتصال به برق.
- منابع وقفه داخلی و خارجی.
- شش حالت مختلف برای کاهش توان مصرفی.
- کار با ولتاژهای ۴/۵ تا ۵/۵ ولت در مدلهای بدون پسوند L و ۲/۷ تا ۵/۵ ولت در مدلهای با پسوند L.

شماره گذاری روی آی سی میکروهایAVR در حالت کلی شامل ۵ فیلد بصورت زیر است.

AVRNAME		-	X X	P	C
	V			S	I
	L			M	U
				J	
				A	
				Y	
	۱		۲	۳	۴ ۵

۱- نام میکروکنترلر بطور مثال ATMEGA128 که معمولاً اعداد آنها مقدار حافظه FLASH را مشخص می‌کند.

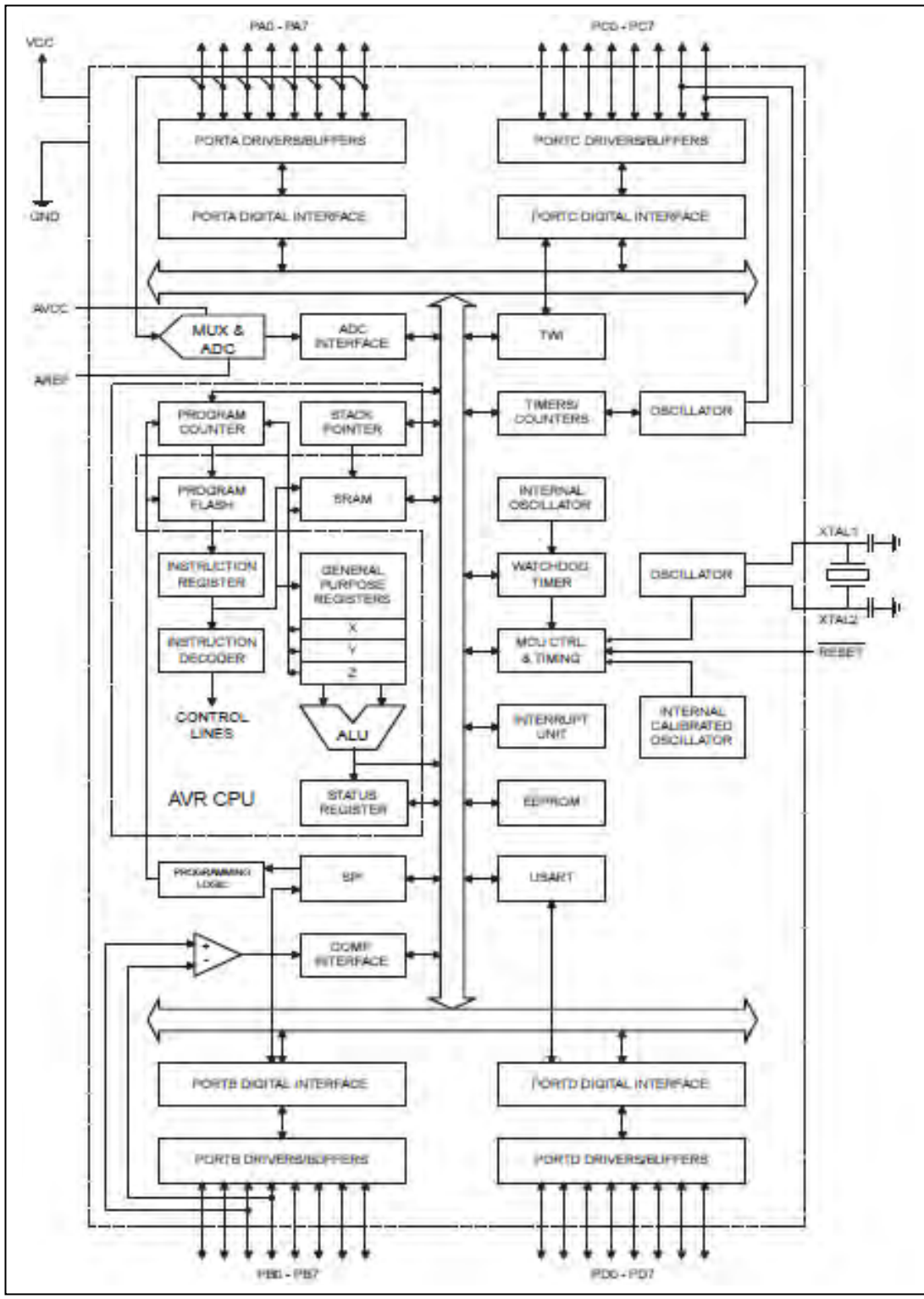
۲- مشخص کننده فرکانس و ولتاژ کاری میکرو است (بدون پسوند، ولتاژ ۴ تا ۵/۵ ولت و فرکانس ۰ تا ۱۶ مگاهرتز مانند ATMEGA32، پسوند V ولتاژ ۱/۸ تا ۵/۵ ولت و فرکانس ۰ تا ۴ مگاهرتز مانند ATTINY2313V، پسوند L ولتاژ ۲/۷ تا ۵/۵ ولت و فرکانس ۰ تا ۸ مگاهرتز مانند ATMEGA8L).

۳- مشخص کننده بیشترین فرکانس کاری میکرو است. به عنوان مثال 8 به معنای حداکثر فرکانس کار 8MHZ است.

۴- مشخص کننده نوع بسته بندی است.

۵- مشخص کننده محدوده دمای کار میکرو است (پسوند C برای تجاری یعنی دمای ۰ تا ۷۰ درجه سانتیگراد، پسوند I و U برای صنعتی یعنی دمای ۴۰- تا ۸۵ درجه سانتیگراد(جهت کسب اطلاعات بیشتر می‌توانید به کتا ب آقای علی کاهه فصل اول مراجعه کنید).

ساختمان داخلی یک میکروکنترلر AVR: ساختار داخلی یک میکروی AVR به شماره ATMEGA32 را در زیر مشاهده می‌کنید.



ساختمان داخلی یک میکروکنترلر

در شکل بالا ارتباط همه‌ی قسمت های داخلی و ارتباط آنها با هسته‌ی مرکزی cpu نشان داده شده است. برای کار کردن با AVR باید توانایی نوشتن با یکی از زبانهای برنامه نویسی را داشته باشیم.

تعریف واحد MIPS (Million Instruction Per Second): تعداد دستورات بر حسب میلیون که میکرو در مدت یک ثانیه انجام می‌دهد. به عنوان مثال وقتی می گوییم ATMEGA32 سرعت 16MIPS دارد یعنی قادر است حدود ۱۶ میلیون دستورالعمل در هر ثانیه را انجام دهد.

تایمر نگهبان (Watchdog Timer): یک تایمر داخلی میکرو است که از اسیلاتور جداگانه داخلی کلاک دریافت می‌کند. توسط این تایمر و با کنترل کلاک Watchdog و منبع تغذیه میکرو، می‌توان بعد از گذشت مدت زمان مشخصی میکرو را ری‌ست کرد (تایمر نگهبان یک تایمر قابل برنامه‌ریزی با نوسانگر مجزا است). ۸ زمان مختلف برای ری‌ست کردن میکرو وجود دارد که در محیط BASCOM با نوشتن دستور CONFIG WATCHDOG = TIME این تایمر پیکره بندی شده و با دستور START WATCHDOG شروع به کار می‌کند. مقادیر معتبر برای TIME عبارتند از ۱۶، ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۱۰۲۴ و ۲۰۴۸ میلی ثانیه. به عنوان مثال دستور زیر در محیط BASCOM باعث می‌شود که میکرو پس از گذشت زمان ۲۰۴۸ میلی ثانیه بعد از دستور START WATCHDOG ری‌ست شود.

```
CONFIG WATCHDOG = 2048
```

با دستور STOP WATCHDOG تایمر نگهبان متوقف شده و با دستور RESET WATCHDOG ری‌ست می‌شود. ری‌ست کردن تایمر نگهبان باید در بخش اصلی برنامه (main) انجام شود.

فیوز بیتها: بخشی از حافظه FLASH هستند که با برنامه‌ریزی آنها (صفر کردن آنها) امکاناتی از میکرو در اختیار کاربر قرار می‌گیرد. فیوز بیتها با پاک کردن (Erase) میکرو از بین نمی‌روند و می‌توانند توسط بیتهای قفل مربوطه قفل شوند. در فیوزبیتهای AVR یک بودن فیوزبیت به معنی برنامه‌ریزی نشدن (Unprogrammed) و صفر بودن به معنی برنامه‌ریزی شدن(Programmed) آن است. به عنوان مثال میکروی ATMEGA16 دارای دو بایت (۱۶ بیت) فیوز بیت است. جدول زیر بایت بالای فیوز بیتها را نشان می دهد.

Fuse High Byte

BIT	Bit No	Description	Default value
OCDEN	7	Enable OCD	1 (unprogrammed,OCD disabled)
JTAGED	6	Enable JTAG	0 (programmed,JTAG Enabled)
SPIEN	5	Enable SPI serial program and data downloading	0 (programmed,SPI prog. Enabled)
CKOPT	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the chip erase	1 (unprogrammed,EEPROM not preserved)
BOOTSZ1	2	Select boot size	0 (programmed)
BOOTSZ0	1	Select boot size	0 (programmed)
BOOTRST	0	Select reset vector	1 (unprogrammed)

که شرح بیتهای آن بصورت زیر است.

OCDEN: در صورتی که بیت های قفل برنامه‌ریزی نشده باشند برنامه‌ریزی این بیت به همراه بیت JTAGEN باعث می‌شودکه سیستم ON CHIP DEBUG فعال شود. برنامه‌ریزی شدن این بیت به قسمت هایی از میکرو امکان می‌دهد که در مدهای SLEEP کار کنند که این خود باعث افزایش مصرف سیستم می گردد. این بیت به صورتی پیش فرض برنامه‌ریزی نشده (1) است.

JTAGEN: بیتی برای فعال سازی برنامه‌ریزی میکرو از طریق استاندارد ارتباطی IEEE(JTAG) که در حالت پیش فرض فعال است و میکرو می‌تواند از این ارتباط برای برنامه‌ریزی خود استفاده نماید.

SPIEN: در حالت پیش فرض برنامه‌ریزی شده و میکرو از طریق سریال SPIبرنامه‌ریزی می‌شود.

CKOPT: انتخاب کلاک که به صورت پیش فرض برنامه‌ریزی نشده است. عملکرد این بیت به بیت های CKSEL بستگی دارد.

EESAVE: در حالت پیش فرض برنامه‌ریزی نشده و در زمان پاک شدن (ERASE) میکرو حافظه EEPROM پاک می‌شود ولی در صورتی که برنامه‌ریزی شود محتویات EEPROM در زمان پاک شدن میکرو محفوظ می‌ماند.

BOOTSZ1 , BOOTSZ0 : این بیتها برای انتخاب مقدار حافظه BOOT و طبق جدول زیر برنامه‌ریزی می‌شوند. در زمان برنامه‌ریزی شدن فیوز بیت BOOTRST اجرای برنامه از آدرس حافظه BOOT آغاز خواهد شد.

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application section	Boot Reset Address (start Boot Loader Section)
1	1	256 words	4	$0000 - $3EFF	$3F00 - $3FFF	$3EFF	$3F00
1	0	512 words	8	$0000 - $3DFF	$3E00 - $3FFF	$3DFF	$3E00
0	1	1024 words	16	$0000 - $3BFF	$3C00 - $3FFF	$3BFF	$3C00
0	0	2048 words	32	$0000 - $37FF	$3800 - $3FFF	$37FF	$3800

BOOTRST: بیتی برای انتخاب بردار ریست BOOT است که در حالت پیش فرض برنامه‌ریزی نشده و آدرس بردار ریست $0000 است و در صورت برنامه‌ریزی شدن، آدرس بردار ریست به آدرسی که فیوز بیت های BOOTSZ0 وBOOTSZ1 مشخص کرده‌اند تغییر می‌یابد.

BOOTRST	RESET ADDRESS
1(UNPROGRAMMED)	RESET VECTOR=APPLICATION RESET (ADDRESS $0000)
0(PROGRAMMED)	RESET VECTOR=BOOT LOADER RESET

جدول زیر بایت پایین فیوز بیتها را نشان می دهد.

Fuse Low Byte

Fuse high byte	Bit No	Description	Default value
BODLEVEL	7	Brown-out detector trigger level	1(unprogrammed)
BOODEN	6	Brown-out detector enable	1(unprogrammed,BOD disabled)
SUT1	5	Select start-up time	1(unprogrammed)
SUT0	4	Select start-up time	0(programmed)
CKSEL3	3	Select clock source	0(programmed)
CKSEL2	2	Select clock source	0(programmed)
CKSEL1	1	Select clock source	0(programmed)
CKSEL0	0	Select clock source	1(unprogrammed)

BODLEVEL : بصورت پیش فرض برنامه‌ریزی نشده است و به موجب آن اگر تغذیه ۵ ولت آی سی کمتر از ۲/۷ ولت شود RESET داخلی میکرو را فعال کرده و میکرو را ری‌ست خواهد کرد. اما اگر این بیت برنامه‌ریزی شود (صفر شود)، چنانچه ولتاژ تغذیه از ۴ ولت کمتر شود RESET داخلی میکرو، فعال شده و میکرو ری‌ست خواهد شد. البته برای آنکه این فیوز بیت عمل کند باید فیوز بیت BODEN نیز برنامه‌ریزی شده باشد.

BODEN: برای فعال کردن عملکرد مدار BROWN-OUT این بیت بایستی برنامه‌ریزی شده باشد. این بیت به صورت پیش فرض برنامه‌ریزی نشده است.

BODEN , BODLEVEL	BROWN- OUT DETECTION
11	DISABLE
10	DISABLE
01	AT VCC=2.7V
00	AT VCC=4.0V

CKSEL3 …CKSEL0: عملکرد این بیت ها برای تأمین منبع کلاک برای میکرو (مطابق جدول زیر) است و مقدار پیش فرض آن به صورت INTERNAL RC OSCILLATOR @ 1MHZ است.

Table 2. Device Clocking Options Select(1)

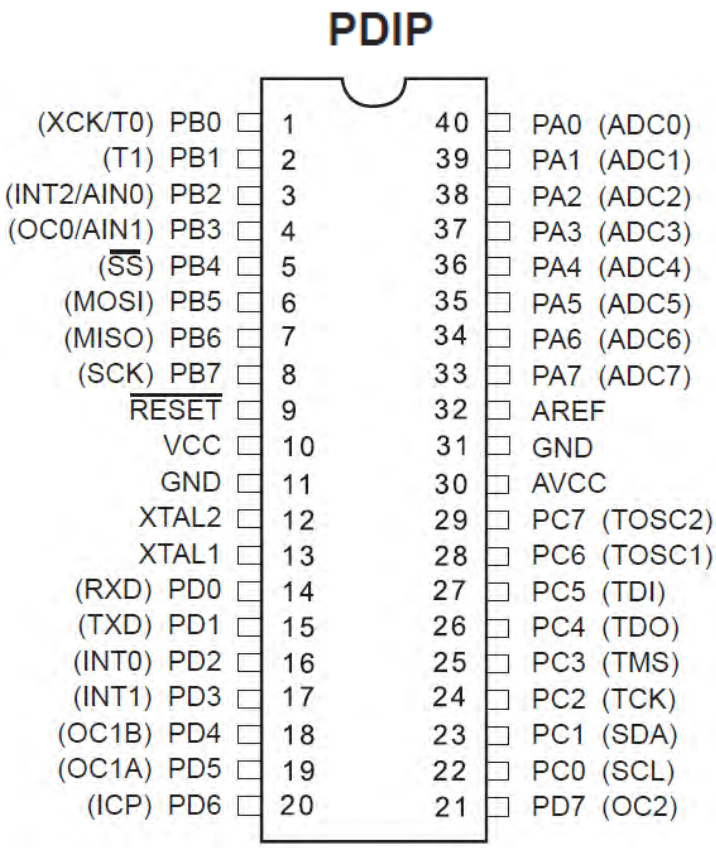
Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

SUT0, **SUT1**: عملکرد این دو بیت برای انتخاب زمان START-UP (مطابق جدول زیر) است.

Table 5. Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

پایه های میکروی ATMEGA32 :



انواع بسته بندی و تعداد پایه ها

۳۲ خط ورودی و خروجی تعبیه شده در چهار پورت ۸ بیتی A و B و C و D
۳ پایه مربوط به تغذیه ها در بسته بندی PDIP و ۷ پایه مربوط به تغذیه ها در بسته بندی MLF و TQFP
۲ پایه مربوط به کریستال در بسته بندی MLF و TQFP و PDIP
۱ پایه مربوط به RESET میکرو در بسته بندی MLF و TQFP و PDIP
۲ پایه مربوط به تغذیه ADC و ولتاژ مرجع آن در بسته بندی MLFو TQFP و PDIP
جمع پایه ۴۴ پایه در بسته بندی MLFو TQFP و ۴۰ پایه در بسته بندی PDIP

خصوصیات Atmega32

از معماری AVR RISC استفاده می کند.
کارایی بالا و توان مصرفی کم
دارای ۱۳۱ دستور که اکثرا آن ها در یک سیکل اجرا می شوند
۸*۳۲ رجیستر کاربردی عمومی
حداکثر کریستال مورد استفاده ۱۶ مگاهرتز برای Atmega32 و ۸ مگا هرتز برای Atmega32L
سرعتی تا 16MIPS در فرکانس 16MHZ
32K بایت حافظه فلش داخلی قابل برنامه ریزی که می‌تواند تا ۱۰۰۰۰ بار نوشته و پاک شود.
2K بایت حافظه SRAM داخلی
1024 بایت حافظه EEPROM داخلی برای ذخیره اطلاعات که می‌تواند تا ۱۰۰۰۰۰ بار نوشته شود و پاک شود.
قفل برنامه داخل حافظه FLASH و EEPROM برای جلوگیری از خواندن آن
قابلیت ارتباط JTAG
برنامه ریزی برنامه FLASH , EEPROM , FUSE BITS , LOCK BITS از طریق ارتباط JTAG
دو تایمر / کانتر ۸ بیتی با prescaler مجزا و دارای مد compare (تایمر / کانتر ۰ و۲)
یک تایمر / کانتر ۱۶ بیتی با prescaler مجزا و دارای مد compare ,capture (تایمر / کانتر ۱)
۴ کانال pwm
۸ کانال مبدل آنالوگ به دیجیتال ۱۰ بیتی single – ended
دارای ۷ کانال تفاضلی در بسته بندی TQFP (این نوع adc اختلاف بین دو ولتاژ را اندازه می گیرد در حالی که adc های معمولی ولتاژ ورودی را نسبت به زمین اندازه می‌گیرند)
دارای دو کانال تفاضلی با گین 1x , 10x , 200x
دارای RTC (نوعی ساعت است که زمان و تاریخ را مستقل از عملکرد میکرو محاسبه می کند) با اسیلاتور مجزا
یک مقایسه کننده آنالوگ داخلی
USART قابل برنامه ریزی
WATCHDOG قابل برنامه ریزی با اسیلاتور داخلی
ارتباط سریال ISP برای برنامه ریزی (پروگرام کردن) داخل مدار (هنگامیکه میکرو داخل مدار است با پروگرامر ISP می‌توانیم میکرو را برنامه ریزی کنیم برای برنامه ریزی از ۴ خط MISO , MOSI, SCK , RESET استفاده می‌شود.)
قابلیت ارتباط سریال SPI به صورت MASTER یا SLAVE
قابلیت ارتباط JTAG (یک نوع ارتباط است که از طریق آن می‌توان کلیه حافظه های قابل برنامه ریزی میکرو را خواند یا نوشت)

خصوصیات ویژه میکرو :

RESET شدن میکرو بعد از روشن شدن
دار ای ۵ مد در حالت بیکاری برای مصرف کمتر انرژی و راندمان بیشتر
منبع وقفه داخلی و خارجی
دارای نوسان ساز داخلی کالیبره شده (حداکثر فرکانس این نوسان ساز ۸ مگا هرتز است)
دارای اسیلاتور RC داخلی کالیبره شده .

شرح مختصر پایه ها:

پایه شماره1 - PB0/XCK/T0 : این پایه علاوه بر نقش پین ورودی یا خروجی (PORTB.0) دونقش دیگر نیز دارد. در نقش T0 به عنوان ورودی کلاک برای کانتر 0 و در نقش XCK به عنوان کلاک خارجی در ارتباط سریال سنکرون USART است. این پایه فقط زمانی که Usart در مد سنکرون کار میکند فعال می‌شود (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که یکی از امکانات گفته شده راه اندازی شود این پایه فقط کار مربوط به آن پایه را انجام میدهد و دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره2- PB1/T1 این پایه علاوه بر نقش پین ورودی و خروجی (PORTB.1) نقش ورودی کلاک برای کانتر 1 را هم عهده دار است(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که تایمر/ کانتر 1 با کلاک خارجی راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد).

پایه شماره3 - PB2/INT2/AIN0 این پایه علاوه بر نقش پین ورودی و خروجی (PORTB.2) نقش ورودی مثبت مقایسه کننده انالوگ را نیز به عهده دارد. نقش دیگر این پایه به عنوان منبع وقفه خارجی دو است (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که وقفه خارجی شماره دو یا مقاسه کننده انالوگ راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره4 - PB3/OC0/AIN1 این پایه علاوه بر نقش پین ورودی و خروجی (PORTB.3) نقش ورودی منفی مقایسه کننده انالوگ را نیز به عهده دارد. علاوه بر این، این پایه خروجی مد مقایسه ای تایمر/کانتر صفر را نیز به عهده دارد همچنین این پایه به عنو‌ن خروجی مد pwm تایمر/کانتر صفر مورد استفاده قرار میگیرد(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که وقفه خارجی یا مقاسه کننده آنالوگ یا تایمر دو در مد مقایسه ای یا pwm راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره5 –PB4/SS این پایه علاوه برنقش پین ورودی و خروجی (PORTB.4) زمانی که ارتباط SPIراه اندازی می‌شود این پایه در میکروی slave ورودی تعریف شده و با صفر شدن این پایه SPI فعال میگردد(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما در ارتباط SPI نمیتوان در میکروی slave از این پایه به عنوان خروجی استفاده کرد ولی در میکرو master میتوان آن را به عنوان ورودی یا خروجی تعریف کرد)

پایه شماره6 - PB5/MOSI این پایه علاوه بر نقش پین ورودی و خروجی b.5 , (PORTB.5) در ارتباط SPI به عنوان ورودی داده برایSLAVE و خروجی داده برای MASTER محسوب می‌شود (در حالت عادی این پایه به عنوان ورودی و خروجی دیجیتال (I/O) استفاده می‌شود اما زمانی که میکرو در ارتباط SPI به صورت MASTER شکل دهی می‌شود خروجی است و زمانی که به صورت SLAVE شکل دهی می‌شود این پایه ورودی است.)

پایه شماره7 – PB6/MISO این پایه علاوه بر نقش پین ورودی و خروجی (PORTB.6) در ارتباط SPI این پایه ورودی داده برای MASTER و خروجی داده برای SLAVE محسوب می‌شود(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود زمانی که میکرو در ارتباط SPI به صورت MASTER شکل دهی می‌شود ورودی است و زمانی که به صورت SLAVE شکل دهی می‌شود این پایه خروجی است.)

پایه شماره8 -PB7/SCK این پایه علاوه بر نقش پین ورودی و خروجی (PORTB.7) در ارتباط SPI این پایه برای کلاک خروجی میکرو MASTER و کلاک ورودی میکرو SLAVE استفاده می‌شود(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود زمانی که میکرو در ارتباط SPIبه صورت MASTER شکل دهی می‌شود خروجی است و زمانی که به صورت SLAVE شکل دهی می‌شود این پایه ورودی است.)

پایه شماره9 –RESET: نقش پایه RESET ، باز نشانی میکرو است. اگر به این پایه یک پالس یک به صفر داده شود برنامه از اول اجرا می‌شود.

پایه شماره10 - VCC :این پایه، یکی از پایه های تغذیه میکرو می باشد که باید به VCC (۵ ولت) مدار متصل شود (هر دو VCCمیکرو از داخل به هم متصل اند)

پایه شماره11 - GND این پایه یکی از پایه های تغذیه میکرو می باشد که باید به GND (صفر ولت) مدار متصل شود (هر سه GND میکرو از داخل به هم متصل اند)

پایه شماره12 XTAL1: کریستال خارجی بین این پایه و پایه ۱ xtal قرار میگیرد

پایه شماره13 - XTAL2: کریستال خارجی بین این پایه و پایه۲ xtal قرار میگیرد

پایه شماره14 - PD0/RXD: این پایه علاوه برنقش پین ورودی و خروجی d.0 , (portd.0) نقش پایه RXD (گیرنده اطلاعات)در ارتباط سریال رانیز به عهده دارد(این پایه و پایه TXD در ارتباط سریال با هم استفاده میشوند) و هنگامی که از ارتباط سریال استفاده می‌شود از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد

پایه شماره15 –PD1/TXD: این پایه علاوه برنقش پین ورودی و خروجی d.1 , (portd.1) نقش پایه TXD (فرستنده اطلاعات)در ارتباط سریال رانیز به عهده دارد(این پایه و پایه RXD در ارتباط سریال با هم استفاده میشوند) و هنگامی که از ارتباط سریال استفاده می‌شود از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد

پایه شماره16 PD2/INT0: این پایه علاوه بر نقش پین ورودی و خروجی d.2 , (portd.2) نقش منبع وقفه خارجی 0 را نیز به عهده دارد(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که وقفه خارجی راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره17 PD3/INT1: این پایه علاوه بر نقش پین ورودی و خروجی d.3 , (portd.3) نقش منبع وقفه خارجی 1 را نیز به عهده دارد(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که وقفه خارجی راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره18 PD4 /OC1B: این پایه علاوه بر نقش پین ورودی و خروجی d.4 , (portd.4) نقش خروجی دیگر مد مقایسه ای تایمر/کانتر 1 را نیز به عهده دارد همچنین این پایه به عنوان خروجی مد pwm تایمر/کانتر1 مورد استفاده قرار میگیرد (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود، اما وقتی که پایه به عنوان خروجی pwm تعریف

می‌شود، دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد اما در ارتباط SPI میتوان در میکرو MASTER آن را به عنوان ورودی یا خروجی تعریف کرد)

پایه شماره19 -PD5/OC1A : این پایه علاوه بر نقش پین ورودی و خروجی d.5 , (portd.5) نقش خروجی مد مقایسه ای تایمر/کانتر 1 را نیز به عهده دارد همچنین این پایه به عنوان خروجی مد pwm تایمر/کانتر 1 مورد استفاده قرار میگیرد(در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که پایه pd5 با یک شدن ddd5 برای خروجی مد مقایسه ای تایمر/کانتر1 ، راه اندازی می‌شود،یا به عنوان خرو جی pwm تعریف می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره20 -PD6/ICP1 : این پایه علاوه بر نقش پین ورودی و خروجی d.6 , (portd.6) نقش ورودی capture تایمر/کانتر 1را نیز به عهده دارد (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که capture تایمر/کانتر 1 راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره21 -PD7/OC2 : این پایه علاوه بر نقش پین ورودی و خروجی d.7 , (portd.7) نقش خروجی مد مقایسه ای تایمر/کانتر 2 را نیز به عهده دارد همچنین این پایه به عنوان خروجی مد pwm تایمر/کانتر 2 مورد استفاده قرار میگیرد (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که پایه pb3 با یک شدن ddd3 برای خروجی مد مقایسهای تایمر/کانتر2 ، راه اندازی می‌شود،یا به عنوان خرو جی pwm تعریف می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره22 - PC0/SCL: این پایه علاوه بر نقش پین ورودی و خروجی c.0 , (portc.0) در زمان ارتباط دو سیمه به عنوان خط کلاک استفاده می‌شود (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که پایه ارتباط wire -۲ راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد.)

پایه شماره23 –PC1/SDA : این پایه علاوه بر نقش پین ورودی و خروجی c.1 , (portc.1) در زمان ارتباط دو سیمه به عنوان خط داده استفاده می‌شود (در حالت عادی این پایه به عنوان ورودی وخروجی دیجیتال (I/O) استفاده می‌شود اما وقتی که پایه ارتباط دو سیمه راه اندازی می‌شود دیگر از این پایه نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره24 – PC2/TCK : این پایه علاوه بر نقش پین ورودی و خروجی c.2 , (portc.2) هنگامی که ارتباط jtag استفاده می‌شود این پایه به عنوان خط کلاک استفاده می‌شود و دیگر نمی‌توان از ان به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره25 –PC3/TMS : این پایه علاوه بر نقش پین ورودی و خروجی c.3 , (portc.3) هنگامی که ارتباط jtag استفاده می‌شود این پایه به عنوان خط فرمان استفاده می‌شود و دیگر نمی‌توان از آن به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره26 –PC4/TDO : این پایه علاوه بر نقش پین ورودی و خروجی c.4 , (portc.4) هنگامی که ارتباط jtag استفاده می‌شود این پایه به عنوان خط خروجی داده سریال استفاده می‌شود و دیگر نمی‌توان از ان به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره27 –PC5/TDI : این پایه علاوه بر نقش پین ورودی و خروجی c.5 , (portc.5) هنگامی که ارتباط jtag استفاده می‌شود این پایه به عنوان خط ورودی داده سریال استفاده می‌شود و دیگر نمی‌توان از آن به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره28 –PC6/TOSC1 : این پایه علاوه بر نقش پین ورودی و خروجی c.6 , (portc.6) وقتی که از تایمر /کانتر 2 در مد اسنکرون(میکرو به مد sleep میرود اما تایمر /کانتر 2 به شمارش ادامه می‌دهد)استفاده می‌شود به این پایه و پایه 29 کریستال ساعت متصل می‌شود و دیگر نمی‌توان از آن به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره29 –PC7TOSC2 : این پایه علاوه بر نقش پین ورودی و خروجی c.7 , (portc.7) وقتی که از تایمر /کانتر 2 در مد اسنکرون(میکرو به مد sleep میرود اما تایمر /کانتر 2 به شمارش ادامه می‌دهد)استفاده می‌شود به این پایه و پایه 28کریستال ساعت متصل می‌شود و دیگر نمی‌توان از آن به عنوان ورودی یا خرو جی استفاده کرد

پایه شماره30 AVCC این پایه برای تغذیه مبدل آنالوگ به دیجیتال استفاده می‌شود،زمانی که از مبدل آنالوگ به دیجیتال استفاده نمی‌شود این پایه آزاد است،و زمانی که از مبدل آنالوگ به دیجیتال (adc) استفاده می‌شود این پایه باید به vcc متصل کردد (اختلاف ولتاژ این پایه با vcc نباید بیشتر از 3.ولت باشد (در غیر این صورت محاسبات دقیق نخواهد بود))

پایه شماره31 - GND این پایه یکی از پایه های تغذیه میکرو می باشد که باید به gnd (صفر ولت) مدار متصل شود (هر سه gnd میکرو از داخل به هم متصل میباشد)

پایه شماره32 -AREF این پایه برای ولتاژ مبدل آنالوگ به دیجیتال استفاده می‌شود،زمانی که از مبدل آنالوگ به دیجیتال استفاده نمی‌شود این پایه آزاد است،و زمانی که از مبدل آنالوگ به دیجیتال (adc) استفاده می‌شود این پایه باید طبق برنامه نوشته شده به vcc یا ولتاژ مرجع متصل گردد (در صورتی که از ولتاژ مرجع استفاده نشود(ولتاژ مرجع زمین گرفته شود) این پایه آزاد خواهد بود)

پایه شماره33 –PA7/ADC7 : این پایه علاوه بر نقش پین ورودی و خروجی a.7 , (porta.7) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(7)) استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره34 –PA6/ ADC6: این پایه علاوه بر نقش پین ورودی و خروجی a.6 , (porta.6) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(6)) استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره35 –PA5/ ADC5 : این پایه علاوه بر نقش پین ورودی و خروجی a.5 , (porta.5) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(5)) استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره36 –PA4/ ADC4 : این پایه علاوه بر نقش پین ورودی و خروجی a.4 , (porta.4) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(4)) استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره ۳۷ –PA3/ ADC3 : این پایه علاوه بر نقش پین ورودی و خروجی a.3 , (porta.3) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(3)) استفاده می‌شود(زمانیکه مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره38 –PA2/ ADC2 : این پایه علاوه بر نقش پین ورودی و خروجی a.2 , (porta.2) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(2)) استفاده می‌شود(زمانیکه مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره39 –PA1/ ADC1 : این پایه علاوه بر نقش پین ورودی و خروجی a.1 , (porta.1) به عنوان ورودی مبدل آنالوگ به دیجیتال صفر ((adc(1)) استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل آنالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

پایه شماره40 –PA0/ADC0 : این پایه علاوه بر نقش پین ورودی و خروجی a.0 , (porta.0) به عنوان ورودی مبدل آنالوگ به دیجیتال کانال شماره صفر ((ADC(0)) نیز استفاده می‌شود(زمانی که مبدل آنالوگ به دیجیتال راه اندازی می‌شود از این پایه و سایر پایه های ورودی مبدل انالوگ به دیجیتال نمی‌توان به عنوان ورودی یا خروجی استفاده کرد)

نحوه‌ی کارکردن با بخش های مختلفAVR

AVR دارای قسمت های مختلفی است که دراین پروژه فقط در مورد سه بخش کار کردن با پورت ها، تایمر و ورودی آنالوگ توضیح داده می‌شود.

پورتهای میکروهای AVR

در AVR برای هر پورت سه رجیستر اختصاص داده شده است. ۱- رجیستر DDR که جهت پورت را تعیین می‌کند مثلاً برای پورت A رجیستر DDRA جهت پورت A را مشخص می‌کند بطوریکه دستور DDRA = 0x00 برای تعریف کل بیتهای پورت A به عنوان ورودی و دستور DDRA = 0xFF برای تعریف کل بیتهای پورت A به عنوان خروجی استفاده می‌شود(برای نمایش اعداد هگزادسیمال در زبان C از 0x استفاده می‌شود). ۲- رجیستر PIN که برای خواندن از پورت استفاده می‌شود. به عنوان مثال دستور PINB برای خواندن از پورت B استفاده می‌شود (مثلاً خواندن از صفحه کلید متصل شده به پورت B) ۳- رجیستر PORT که برای نوشتن در پورت استفاده می‌شود. (مثلاً برای ارسال داده از پورت A به LED ها از رجیستر PORTA استفاده می‌شود.

زمانی که بخواهید از پورتی بخوانید بایستی از رجیستر PIN پورت مربوطه استفاده کنید و در هنگام نوشتن در پورت بایستی در رجیستر PORT پورت مربوطه بنویسید.

در Bascom برای آنکه پورتی را به عنوان ورودی یا خروجی مشخص کنیم باید آن پورت را پیکره بندی کنیم. جهت هر کدام از پایه های پورت می‌تواند ورودی یا خروجی تعریف شود. در Bascom از دستورات زیر استفاده می‌شود.

Config portx = state

Config pinx. y = state

x و y بسته به میکرو می‌توانند به ترتیب پایه های ۰ تا ۷ پورتهای D،C،B،A برای میکروهای مختلف باشند.

به عنوان مثال دستور Config Porta=output در BASCOM باعث آماده سازی پورت A (هر هشت بیت) به عنوان خروجی و دستور Config PINA.3=output باعث آماده سازی بیت سوم (چهارمین بیت) پورت A به عنوان خروجی می شود. دستورات معادل این دو دستور در CODEVISION به ترتیب بصورت DDRA=0XFF و DDRA=0X08 است.

اگر بخواهیم کلیدی را به یکی از پایه‌های پورت A وصل کنیم، چون قصد خواندن از پورت را داریم از رجیستر PIN استفاده می‌کنیم، ضمن اینکه چون پورت باید به صورت ورودی باشد DDRA را برابر 0x00 قرار می‌دهیم. بر عکس مثلاً اگر بخواهیم LED ای را روشن کنیم(نوشتن در پورت) از رجیستر PORT استفاده می‌کنیم و DDRA را برابر 0xFF قرار می‌دهیم.

در جدول زیر عملکرد هر یک از رجیسترها با مثال آمده است.

نمایش عملکرد رجیسترهای DDRA و PORTA

شماره بیت	7	6	5	4	3	2	1	0
DDRA	1	0	1	1	1	1	0	1
PORTA	1	1	0	1	0	1	0	0
جهت داده	خروجی با سطح منطقی۱	ورودی با مقاومت pull-up	خروجی با سطح منطقی۰	خروجی با سطح منطقی1	خروجی با سطح منطقی۰	خروجی با سطح منطقی۱	ورودی بدون مقاومت pull-up	خروجی با سطح منطقی۰

تایمرهای میکروهای AVR

در داخل AVR ها سه تایمر/ کانتر وجود دارد (بجزMEGA128،MEGA64،MEGA256،MEGA162 که دارای ۴ تایمر اند). تایمر / کانتر صفر ۸ بیتی بوده و می‌تواند کلاک خود را از کلاک سیستم، تقسیمی از کلاک سیستم و یا از پایه خارجی T0 بگیرد. شامل سه رجیستر ۸ بیتی TCNT0 (Timer/Counter0) و (Timer/Counter Control Register) TCCR0 و OCR0(Output Compare Register0) است که اولی محتوای تایمر/ کانتر(مقدار شمارنده) را در خود جای می‌دهد و دومی (البته سه بیت وزن پایین آن مطابق جدول زیر) انتخابگر Prescale برای تایمر/ کانتر است و جهت پیکربندی تایمر استفاده می‌شود و سومی جهت مقایسه (مقداری را که مقایسه با آن صورت می‌گیرد در خود دارد) استفاده می‌شود. همچنین این تایمر در رجیسترهای TIFR و TIMSK که بترتیب رجیسترهای پرچم و وقفه تایمر هستند با دیگر تایمرها مشترک است.

CS02	CS01	CS00		
0	0	0	Stop	تایمر کار نمی کند
0	0	1	CK	
0	1	0	CK/8	
0	1	1	CK/64	
1	0	0	CK/256	
1	0	1	CK/1024	
1	1	0	Falling	
1	1	1	Rising	

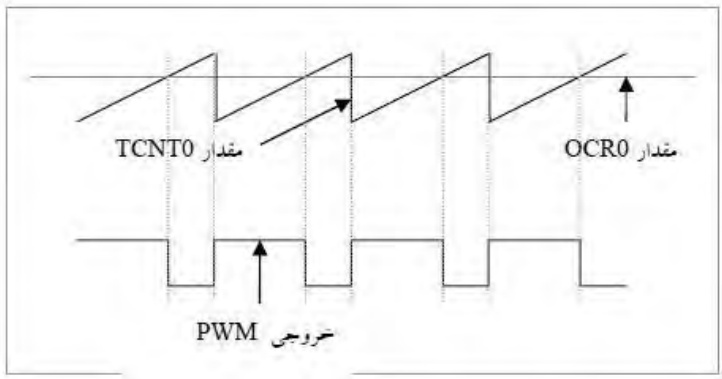
تایمر صفر بالا شمار بوده و قابلیت خواندن و نوشتن را دارد(هم مقدار شمارش را می‌دهد و هم می‌توان مقدار اولیه به آن داد. در پیکره بندی بصورت کانتر(تحریک از پین خارجی T0)، شمارش با لبه بالارونده یا پایین رونده انجام شده و اعمال لبه مورد نظر به پایه T0 باعث می‌شود که محتوای رجیستر TCNT0 یا متغیر COUNTER0 یک واحد افزایش یابد و با

رسیدن شمارش به عدد به FF پرچم سرریز OVF0 یک شود. امکان تنظیم تایمر به صورت عمل در مدت (Clear CTC Timer on compare match) نیز وجود دارد.

تایمر یک، یک تایمر ۱۶ بیتی است که دارای دو واحد مقایسه مستقل با خروجیهای OC1A و OC1B است. شامل ۴ رجیستر ۱۶ بیتی TCNT1، OCR1A، OCR1B، ICR1 و دو رجیستر کنترلی ۸ بیتی TCCR1A و TCCR1B (معرف Prescaler کلاک و چند بیتی بودن PWM) است. رجیستر پرچم TIFR و رجیستر چشم پوشی از اینتراپت TIMSK نیز بصورت مشترک با سایر تایمرها استفاده می شوند. تایمر یک نیز همانند تایمر صفر می‌تواند کلاک خود را از کلاک سیستم، تقسیمی از کلاک سیستم و یا از پایه T1 دریافت کند. همواره محتوای رجیسترهای OCR1A و OCR1B با محتوای TCNT1 مقایسه شده و در صورت تساوی خروجی OC1A یا OC1B فعال می‌شود. رجیستر ICR1 می‌تواند محتوای تایمر یک را تصرف کند در صورتی که تحریکی خارجی به پایه ICP1 وارد شود یا از طریق برنامه ریزی فیوز بیت دوم رجیستر ACSR که مستقیماً خروجی مقایسه کننده آنالوگ را به ICP1 متصل می‌کند این تحریک صورت گیرد.

مد PWM سریع (Fast PWM Mode)

تایمر / کانتر صفر یک تایمر ۸ بیتی است که دارای چهار Mode کاری Normal ، CTC، Fast PWM , Phase correct PWM است. مد PWM سریع مشابهمد نرمال است با این تفاوت که پین OC0 فقط در حالت برابری رجیسترهای TCNT0 و OCR0 تغییر حالت نمی‌دهد، بلکه در زمان سرریزی رجیستر TCNT0 نیز تغییر حالت می‌دهد. در این مد، تایمر از صفر تا مقدار TOPخود شروع به شمارش کرده و پس از سرریز، مجددا از صفر شروع به شمارش می‌کند.



مهمترین رجیستر تایمر صفر، رجیستر TTCR0 است که بیت های Clock Select آن، جهت انتخاب منبع کلاک تایمر و بیت های Wave Generation Mode برای تنظیم مدکاری تایمر و بیت های Compare Match Output پیکربندی پین OC0 را تعیین می‌کند.

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

مقدار ب

Bit	7	6	5	4	3	2	1	0
TCCR0	0	1	COM01	COM00	1	CS02	CS01	CS00

تایمر/کانتر صفر و یک دارای یک Prescaler مشترک بوده و وضعیت منبع کلاک با توجه به بیت های Clock Select و توسط جدول زیر تعیین می‌شود:

CS02	CS01	CS00	وضعیت منبع کلاک تایمر
0	0	0	بدون کلاک (متوقف)
0	0	1	کلاک سیستم (بدون تقسیم)
0	1	0	۸/کلاک سیستم
0	1	1	٦٤/کلاک سیستم
1	0	0	٢٥٦/کلاک سیستم
1	0	1	١٠٢٤/کلاک سیستم
1	1	0	لبه ی پایین رونده ی پالس خارجی (T0)
1	1	1	لبه ی بالا رونده ی پالس خارجی (T0)

در Modeهای PWM عملکرد بیت های COM00, COM01 به صورت زیر است.

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	رزرو شده
1	0	Clear دروضعیت تطابق، Set در زمان سرریز (PWM غیر معکوس)
1	1	Set در وضعیت تطابق، Clear در زمان سرریز (PWM معکوس)

برای محاسبه ی فرکانس موج PWM تولید شده می‌توان از فرمول زیر استفاد نمود:

$$f_{PWM} = \frac{f_{clk\_I/O}}{N.X}$$

N = Prescale = 1, 8, 64, 256, 1024 X = ٢٥٦ – مقدار بارگذاری شده در تایمر

از وقفه ی سرریز تایمر می‌توان برای مقدار اولیه دادن به TCNT0 و یا تغییر مقدار OCR0 استفاده نمود،

اگرچه بهتر است مقدار OCR0 در روتین وقفه ی مقایسه تغییر داده شود.

- با مقدار اولیه دادن به TCNT0 می‌توان فرکانس موج PWM را تغییر داد.
- با کمتر شدن OCR0 زمان وظیفه کمتر شده و تا حدی که مقدار صفر یک پالس سوزنی به عرض یک سیکل ایجاد خواهد کرد.
- با مقداردهی 256 به OCR0 مقدار مقایسه و سرریز برابر شده و پالس خروجی بسته به مقدار COM00 , COM01 همواره صفر یا یک خواهد بود.

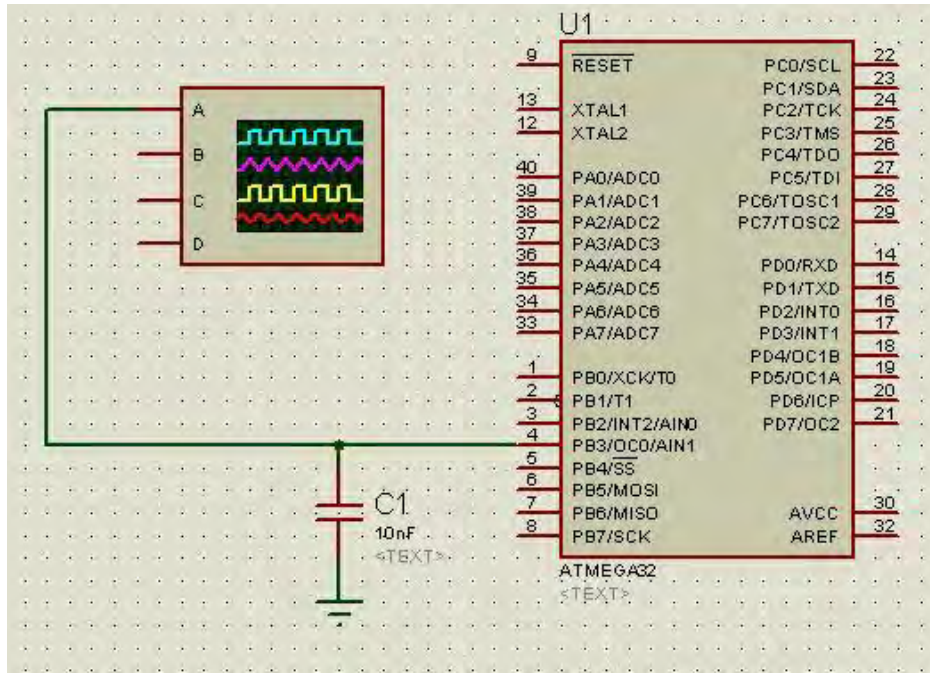
مثال۱: تولید موج PWM با فرکانس 4_{KHZ} و زمان وظیفه ۲۰ درصد؟

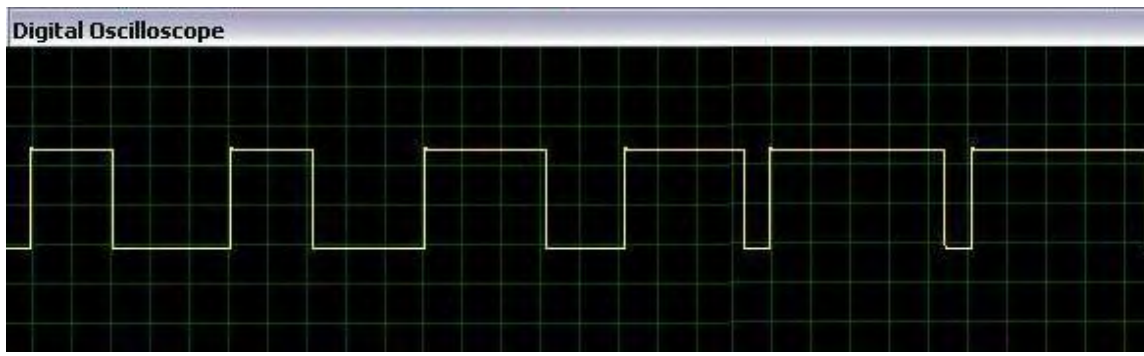
```
 #include <mega32.h>
#define xtal 8000000
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0x06;
}
void main(void)
{
PORTB=0x00;
    DDRB=0x08;
    // Timer/Counter 0 initialization
    // Clock source: System Clock
    // Clock value: 1000.000 kHz
    // Mode: Fast PWM top=FFh
    // OC0 output: Non-Inverted PWM
    TCCR0=0x6A; //0x7A for inverted PWM
    TCNT0=0x06;
    OCR0=0x38; //OCR0 = 56
    // Timer(s)/Counter(s) Interrupt(s) initialization
    TIMSK=0x01;
    // Global enable interrupts
    #asm("sei")
 while (1);
      }
```

$$f_{PWM} = \frac{۸۰۰۰۰۰۰}{۸(۲۵۶-۶)} = ۴۰۰۰Hz$$

$$DutyCycle = \frac{OCR۰}{۲۵۰} \times ۱۰۰\% = \frac{۵۰}{۲۵۰} \times ۱۰۰\% = ۲۰\%$$

مثال ۲: برنامه ای بنویسید که با استفاده از تایمر صفر در مد Fast PWM یک موج PWM تولید کند که پهنای High آن هر 20 میلی ثانیه بصورت متناوب از $\frac{0}{255}$ تا $\frac{200}{255}$ تغییر کند.(محیط کدویژن و بسکام)





نوشتن برنامه در محیط کدویژن:

```
#include <mega32.h>
#define xtal 8000000
#include <delay.h>
int i = 0;
int j = 0;
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
if (j == 0) i++;
if (i == 200) j = 1;
if (j == 1) i--;
```

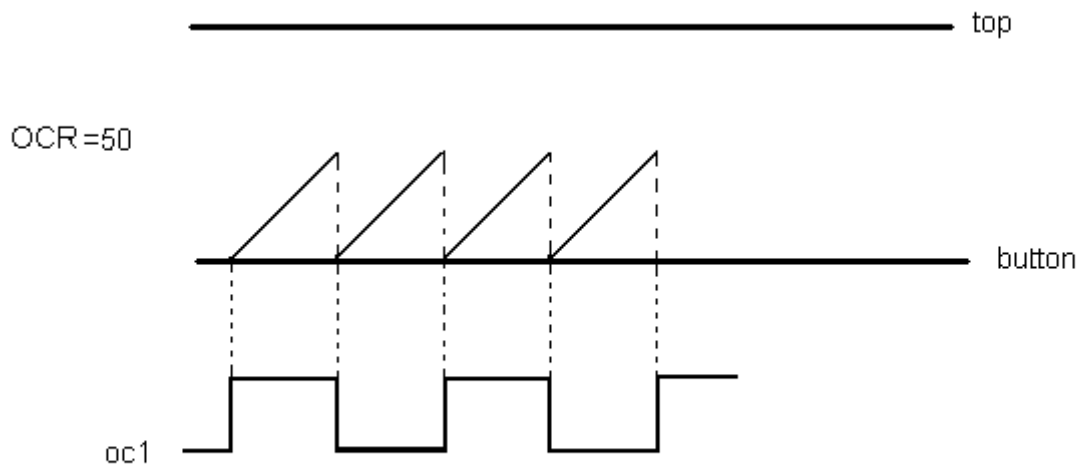
```
if (i == 1) j = 0;
OCR0 = i;
}
void main(void)
{
PORTB = 0x00;
DDRB = 0x08;
TCCR0 = 0x79;
TCNT0 = 0x00;
OCR0 = 0xff;
TIMSK = 0x02;
#asm ("sei")
while (1)
{
delay_ms(20);

}
}
```

$$f_{OC\backslash x} = \frac{f_{CLK}}{۲.N.(۱ + OCR\backslash A)}$$

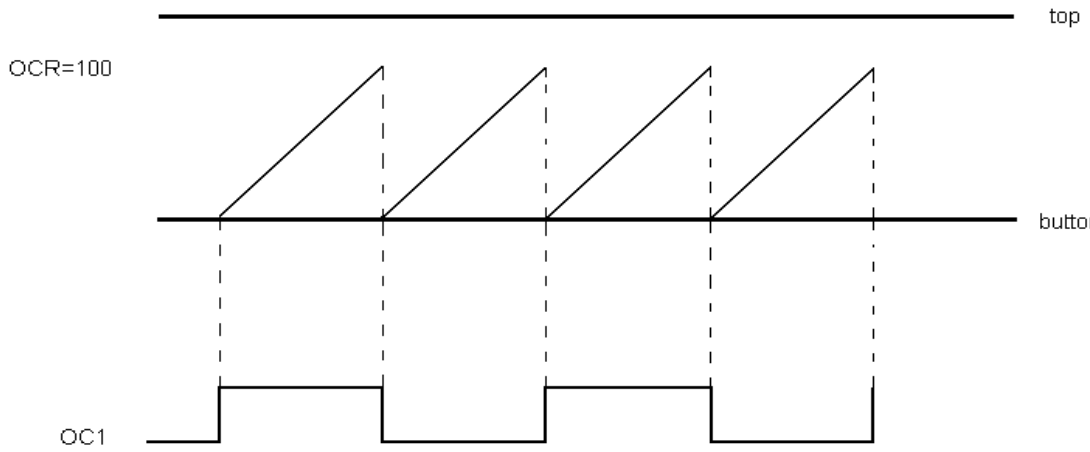
برنامه نوشته شده در محیط بسکام

```
$regfile = "m32def.dat"
$crystal = 8000000
Config PORTB.3 = Output
Dim I As Byte
Dim J As Byte
Tccr0 = &H79
Tcnt0 = &H00
Ocr0 = &HFF
Timsk = &H02
Enable Interrupts
Enable Int0
On Int0 Isr
Do
Waitms 20
Loop
End
Isr:
If  J = 0 Then Incr I
```

```
If  I = 200 Then  J = 1
If  J = 1 Then Decr I
```



showing timer1 in ctc mode with OCR=50



showing ctc mode with OCR=100 and width pulse of oc1 is double

```
If  I = 1 Then  J = 0
Ocr0 = I
Return
```

مد CTC

در این مد پالسهای خروجی مربعی متقارن بوده ولی بسته به مقدار OCR1 فرکانس ان متفاوت است زمانیکه شمارش به مقدار مورد نظر برسد (مقدار OCR) شمارش صفر می‌شود و پالس خروجی هم در هر بار رسیدن به عدد مورد نظر toggle می‌شود (یعنی اگر صفر است یک و اگر یک است صفر می‌شود)

در این حالت رجیستر TCNT1 به طور پیوسته با مقدار OCR1A یا ICR1 مقایسه می‌شود و در صورت برابری مقدار رجیستر TCNT1 برابر صفر می‌شود بنابراین در این حالت مقدار TOP تایمر را با توجه به مقدار موجود در بیتهای WGM مقدار رجیسترهای OCR1A یا ICR1 تعین می کند.

با رسیدن تایمر به مقدار TOP خود بر حسب اینکه مقدار ماکزیمم OCR1A یا ICR1 انتخاب شده باشد به ترتیب پرچمهای OCF1A یا ICF1 یک شده ودر صورت فعال بودن وقفه از ان می‌توان برای تغییر دادن مقدار مقایسه استفاده کرد. این عمل باید با دقت صورت گیرد زیرا رجیستر مقایسه تایمرها فقط در مدهای PWM دارای بافر دوبل می باشد. در این حالت فرکانس موج ایجاد شده روی پایه های OC1A یا OC1B مطابق رابطه زیر می باشد

مثال: تولید موج مربعی با فرکانس۱ کیلوهرتز روی پایه ی OC1A ؟

```
#include <mega32.h>
#define  xtal 8000000
Void main (void)
{
PORTD=0X00;
DDRD=0X30;
top =01F3h                                                    // mode: CTC
TCCR1A=0X40;                          N = Prescale = 1, 8, 64, 256, 1024
TCCR1B=0X0A;
OCR1AH=0X01;
OCR1AL=0XF3;   //OCR1A=499
While  (1);
}
```

برنامه ای بنویسید که در مد نرمال با استفاده از تایمر صفر یک موج مربعی با پریود 512us روی پین خروجی OC0 تولید کند.

```
#include <mega32.h>
void main(void)
{
DDRA=0x08;
PORTB=0x00;
DDRB=0x08;
TCCR0=0x1B;
TCNT0=0x00;
OCR0=0x0C;
ASSR=0x00;
MCUCR=0x00;
MCUCSR=0x00;
TIMSK=0x00;
```

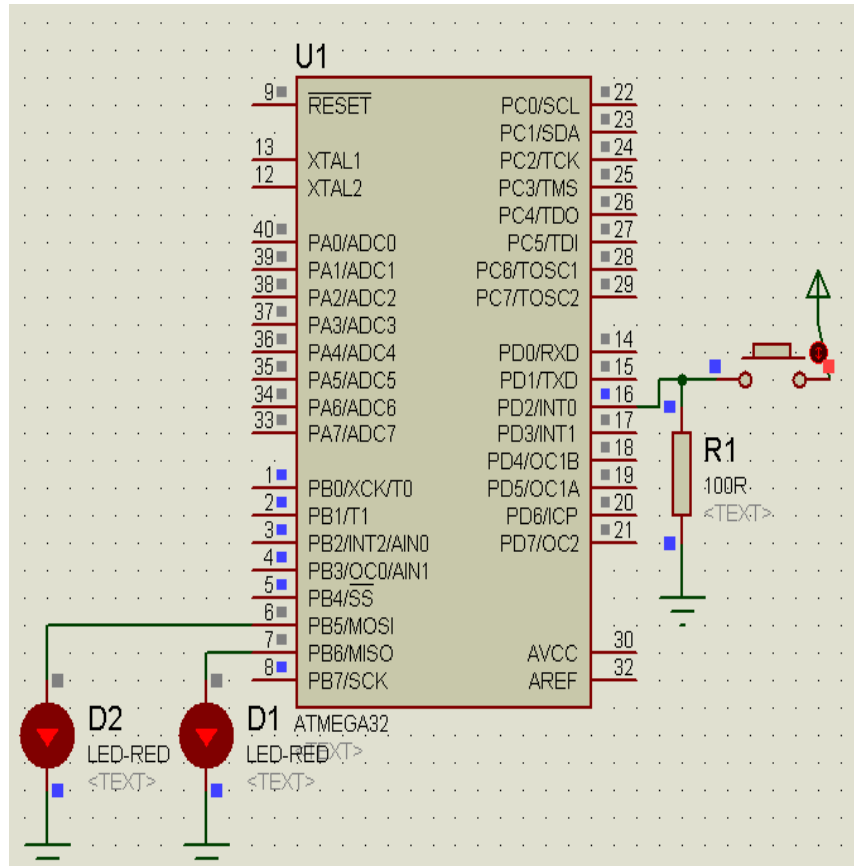
```
ACSR=0x80;
SFIOR=0x00;
while (1)
    {
    // Place your code here

    };
}
```

۲- برنامه ای بنویسید که در مد CTC با استفاده از تایمر صفر یک موج مربعی با فرکانس 5khz روی پین خروجی OC0 تولید کند.

```
#include <mega32.h>
void main(void)
{
PORTA=0x00;
DDRA=0x08;
PORTB=0x00;
DDRB=0x08;
PORTC=0x00;
DDRC=0x00;
PORTD=0x00;
DDRD=0x00;
TCCR0=0x1B;
TCNT0=0x00;
OCR0=0x0C;
MCUCR=0x00;
MCUCSR=0x00;
TIMSK=0x00;
ACSR=0x80;
SFIOR=0x00;
while (1)
    {

    };
{
```

برنامه ای بنویسید که در حالت عادی یک LED متصل به PB.5 با فرکانس 100HZروشن و خاموش شود و با وارد شدن لبه ی بالا رونده INT0 کار عادی خود را رها کرده وled متصل به PA.6 را به مدت 100mSروشن کند.

```
#include <mega32.h>
#include <delay.h>
// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
 PORTB.5=1;
 delay_ms(100);
 PORTB.5=0;
}
void main(void)
{
PORTB=0x00;
DDRB=0xFF;
MCUCR=0x03;
GICR=0X40;
#asm("sei")

while (1)
      {
       PORTB.6=1;
       delay_ms(5);
       PORTB.6=0;
       delay_ms(5);
      };
}
```



مبدل آنالوگ به دیجیتال در میکروهایAVR

قبلاً در درس اصول میکروکامپیوترها، متداول ترین انواع مبدلهای آنالوگ به دیجیتال را که شامل نوع موازی (همزمان)، نوع پله ای، نوع با تقریبات متوالی، نوع تبدیل ولتاژ به زمان تک شیبی، نوع تبدیل ولتاژ به زمان دوشیبی، نوع تبدیل ولتاژ به فرکانس بود بررسی کردیم و مزایا و معایب آنها را شناختیم. حال می خواهیم ADC موجود در AVR ها (از جمله در ATMEGA32) را بررسی کرده و از آن استفاده کنیم. مبدل ADC موجود در میکروهای AVR از نوع تقریبات متوالی است. چنانکه در درس اصول میکرو دیدیم در این نوع از A/D ها زمان تبدیل وابسته به مقدار ولتاژ آنالوگ ورودی است.

خصوصیات ADC در ATMEGA32

وضوح ۱۰بیتی، صحت مطلق LSB 1±، زمان تبدیل US ۲۶۰_۶۵، وضوح KSPS ۱۵ در بالاترین حد، کانالهای مولتی پلکس شده، مدهای تبدیل FREE و SINGLE، ولتاژ ورودی از صفر تا VCC، پرچم وقفه پایان تبدیل ADC

بسته به شماره میکرو، چند کانال آنالوگ می‌توانند به عنوان ورودی مبدل آنالوگ به دیجیتال عمل کنند. مبدل ADC برای تبدیل درست ADC نیاز به یک کلاک ورودی با فرکانسی بین ۵۰ تا۲۰۰کیلو هرتز دارد.

ADC دارای دو منبع ولتاژ مستقل است.AVCC و AGND که AGND باید به زمین وصل شود و AVCC نباید بیشتر از مثبت یا منفی ۰/۳ ولت با VCC اختلاف داشته باشد.

ولتاژ مرجع به پایه AREF وصل می‌شود که باید بین AVCC و AGND باشد در غیر این صورت پایه AREF به VCC وصل می‌شود. ADC مقدار آنالوگ ورودی را با تقریب متوالی به عدد دیجیتال ۱۰ بیتی تبدیل می‌کند. ولتاژ آنالوگ بین AGND تا AREF به عدد ده بیتی 0000000000 تا 1111111111 تبدیل می‌شود.

ADC دارای دو مد تبدیل SINGLE و FREE است. مد SINGLE باید توسط کاربر پیکره بندی شود. در مد ADC،FREE با یک ثابت نمونه برداری، رجیستر داده ADC راUPDATE می‌کند.

CONFIG ADC = SINGLE/FREE , PRESCALER = AUTO , REFERENCE= OPTIONAL

CONFIG ADC= SINGLE/FREE: استفاده از یکی از دو مد SINGLE و FREE

PRESCALER: کلاک ADC را مشخص می‌کند. با قرار دادن AUTO کامپایلر با توجه به فرکانس اسیلاتور، بهترین (بالاترین) کلاک را برای ADC انتخاب می‌کند. در غیر اینصورت تقسیم شده کلاک AVR بر عدد PRESCALER به عنوان کلاک مبدل ADC است.

REFRENCE = OPTIONAL: گزینه ای انتخابی برای ولتاژ مرجع است که می‌تواند سه مقدار **OFF**، **AVCC** یا INTERNAL را (با معانی زیر) داشته باشد.

OFF: جهت خاموش کردن ولتاژ مرجع داخلی است. به عنوان نمونه اگر در این حالت ولتاژ ۴ ولت به پایه AREF متصل شود ولتاژ آنالوگ ۴ ولت به مقدار ۱۰۲۴ و ولتاژ آنالوگ صفر ولت به مقدار ۰ تبدیل خواهد شد.

AVCC: زمانی که ولتاژ پایه AVCC به عنوان مرجع باشد. این ولتاژ می‌تواند از VCC تأمین شود. در این حالت AVCC به عنوان AREF منظور می‌شود که در اینصورت ولتاژ آنالوگی معادل VCC ولت به مقدار ۱۰۲۴ و ولتاژ آنالوگی برابرصفر ولت به مقدار ۰ تبدیل خواهد شد. (در هر صورت ولتاژ AVCC نباید بیش از ۰/۳ ولت با VCC اختلاف داشته باشد.)

INTERNAL: زمانی که ولتاژ مرجع داخلی با خازن خارجی بر روی پایه AREF استفاده شود. در این حالت ولتاژ مرجع AREF بصورت داخلی تأمین شده و برابر ۲/۵۶ ولت است که در اینصورت ولتاژ آنالوگی معادل ۲/۵۶ ولت به مقدار ۱۰۲۴ و ولتاژ آنالوگی برابرصفر ولت به مقدار ۰ تبدیل خواهد شد.

رجیستر های واحدADC

الف - ADC multiplexer selection register

ADMUX	۷	۶	۵	۴	۳	۲	۱	0
نام بیت	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

[4:0] Analog channel and gain selection bits: این بیت ها تعیین می‌کنندکه چه ترکیبی از ولتاژهای آنالوگ ورودی به واحد ADC متصل شده و همچنین بهره ورودی تفاضلی را نیز تعیین می‌کنند. در حالت های SINGLE-ENDED دقت مبدل ۱۰بیتی بوده و در حالت ورودی دیفرانسیل با بهره ی x1 و x10 این مقدار به ۸ بیت و در بهره ی x200 به ۷ بیت کاهش می‌یابد. درصورتی کهADC مشغول انجام یک تبدیل بوده واین بیت ها تغییر کنند تا اتمام تبدیل جاری این تغییر انجام نخواهد شد. تنظیمات این ۴ بیت مطابق جدول زیر است (عملکرد تفاضلی فقط بر روی PACKAGEهای TQFPوMLF آزمایش شده است.)

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

ADC LEFT adjust result: بیت ADLR بر نحوه نمایش نتیجه ی تبدیل در رجیستر داده ی ADC تاثیر می‌گذارد. نوشتن یک در این بیت آن را به صورت left adjust تنظیم می‌کند و در غیر اینصورت نتیجه به صورت right adjust خواهد بود. تغییر این بیت به صورت آنی بر روی رجیستر داده تاثیر می‌گذارد.

Reference selection bits: این بیت ها مرجع ولتاژ adc را مطابق جدول زیر تعیین می‌کنند. در صورتی که این بیت ها در حین تبدیل تغییر کنند تا انجام تبدیل تغییر اعمال نخواهد شد. در صورتی که از مرجع ولتاژ داخلی استفاده شود نباید ولتاژ خارجی به پین AREF اعمال شود. زمانی که یکی از دو ولتاژ AREF یا ولتاژ مرجع داخلی ۲/۵۶ ولت به عنوان مرجع انتخاب شده باشند با اتصال یک خازن ۱۰۰نانو بین پین AREF و زمین می‌توان مقدار نویز را کاهش داد.

REFS1	REFS0	ولتاژ مرجع
0	0	ولتاژ پایه ی AREF
0	1	ولتاژ پایه ی AVCC
1	0	رزرو شده
1	1	ولتاژ داخلی ۲/۵۶ ولت

Example:

ADMUX=0XED (ADC3-ADC2-10X GAIN , 2.56V REFRENCE, LEFT adjusted result)
Voltage on ADC3 is 300 mv, voltage on ADC2 is 500mv.
ADCR=512*10*(300-500)/2560 = -400 = 0x270
ADCL will thus read 0x00, and ADCH will read 0x9c.
writing zero to ADLAR right adjusts the result: ADCL =0x70,ADCH=0x02.

If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where V_{POS} is the voltage on the positive input pin, V_{NEG} the voltage on the negative input pin, GAIN the selected gain factor, and V_{REF} the selected voltage reference. The

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

ب – ADC control and status register

ADCSRA	7	6	5	4	3	2	1	0
نام بیت	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADC PRESCALER SELECT BIT[2:0]: این بیت ها ضریب پیش تقسیم کننده ای را که از کلاک سیستم برای واحدADC کلاک تامین می‌کند را مشخص می‌کند

Table 85. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC interrupt enable: درصورت یک بودن بیت فعال ساز عمومی وقفه(I) و یک بودن این بیت اتمام یک تبدیل می‌تواند باعث ایجاد وقفه شود.

ADC interrupt flag: با اتمام یک تبدیل این پرچم یک شده و در صورت فعال بودن وقفه، اجرای سابروتین وقفه می‌تواند باعث پاک شدن آن شود و در غیر اینصورت با نوشتن یک در محل این بیت می‌توان آنرا پاک نمود.

ADC auto trigger enable: عملیات تبدیل به دو صورت می‌تواند راه اندازی شودsingle وauto trigger که حالت اول با هر بار راه اندازی ADCیک تبدیل انجام شده ودر وضعیت دوم ADC به صورت خود کار از طریق یکی از منابع داخلی تحریک می‌شود. برای قرار دادن ADC در وضعیت trigger auto باید این بیت یک شود. نوع منبع تریگر کننده بوسیله ی بیت های ADTS[2:0] از رجیستر SFIOR و مطابق جدول زیر انتخاب می‌شود.

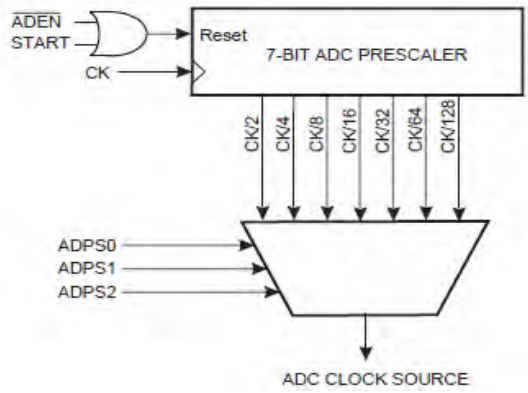
Table 86. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADC start conversion: در وضعیت راه اندازی single برای آغاز هرتبدیل باید این بیت یک شود و در وضعیت تبدیل پیوسته(free running) نوشتن یک روی این بیت اولین تبدیل را موجب می‌شود.

ADC enable: این بیت فعال ساز ماژول ADC بوده و با یک کردن آن می‌توان ADC را فعال نمود. نوشتن صفر روی این بیت در حالی که ADC مشغول تبدیل است باعث می‌شود که عملیات تبدیل نیمه کاره رها شود.

ADC Prescaler



پ - ADC Special function IO register

SFIOR	۷	۶	۵	۴	۳	۲	۱	0
نام بیت	ADTS2	ADTS1	ADTS0	0	ACME	PUD	PSR2	PSR10

ADC auto trigger source: در صورتی که بیت ADATE از رجیستر ADCSRA مقدار یک داشته باشد بیت های ADTS تعیین می‌کنند که کدام منبع به صورت خودکار ADC را راه اندازی کند. لحظه این راه اندازی لبه ی بالا رونده ی پرچم وقفه آن منبع بوده و در صورتی که بخواهیم اتمام تبدیل خود ADC منبع تریگر بعدی باشد وADC به صورت پیوسته عملیات تبدیل را انجام دهد از حالت FREE running استفاده می‌کنیم.

ت- ADC data register

ADLR=0

BIT	۷	۶	۵	۴	۳	۲	۱	0
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

ADCH	0	0	0	0	0	0	ADC9	ADC8

ADLR=1

BIT	۷	۶	۵	۴	۳	۲	۱	0
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	0	0	0	0	0	0

با پایان عملیات تبدیل، نتیجه در این رجیستر قرار می‌گیرد و در صورتی که ورودی ADC به صورت دیفرانسیل باشد نتیجه به فرم مکمل ۲ نمایش داده می‌شود.

وقفه های میکروهای AVR

وقفه چیست ؟

اصطلاح وقفه یا interrupt رخداد خارجی یا داخلی است که منجر به توقف اجرای عادی برنامه گشته و کنترل اجرا را به روال سرویس دهی به یکی از اجزای میکروکنترلر منتقل می‌کند .

عاملی که در اجرای برنامه اصلی وقفه ایجاد می‌کند را به عنوان منبع وقفه می شناسیم. منابع وقفه در میکروکنترلرها به دو دسته داخلی و خارجی تقسیم می شوند. وقفه های داخلی توسط ماژول های داخل میکرو ایجاد می شوند. به عنوان مثال در صورتی که شمارنده Timer/Counter0 فعال باشد پس از رسیدن شمارش به عدد 0xFF مجددا صفر شده و شمارنده هنگام تغییر حالت از 0xFF به صفر امکان ایجاد وقفه را دارد. تغییر حالت از 0xFF به صفر، شرط ایجاد وقفه در تایمر است البته باید توجه داشت که شمارنده زمانی وقفه ایجاد می‌کند که علاوه بر ایجاد شرط وقفه بیت I در رجیستر SREG و بیت فعال ساز اختصاصی وقفه شمارنده نیز یک باشد. اگر فقط از بیت اختصاصی وقفه هر ماژول استفاده می شد امکان فعال شدن ناخواسته وقفه ها وجود داشت.

از آنجا که بررسی و ارائه مثال در مورد وقفه های داخلی مستلزم شناخت منابع آنها و همچنین شناخت روشهای پیکره بندی منابع وقفه داخلی است ابتدا منابع وقفه خارجی را بررسی می‌کنیم.

روشهای بررسی یک رخداد از جانب CPU:

بطور کلی cpu برای تشخیص رویداد های داخلی و رویداد های خارجی می‌تواند از دو روش زیر استفاده کند:

۱ - روش سرکشی(polling):

در این روش کاربر توسط برنامه نویسی با فواصل زمانی مشخص و دایما رویداد مورد نظر را بررسی می‌کند. بطور مثال شما در حال درس خواندن، منتظر یک شخص هستید اگر شما هر یک دقیقه یکبار از پنجره بیرون را نگاه کنید که ببینید آن شخص آمده است یا خیر، به این روش سرکشی یا polling گویند. حال اگر در برنامه نویسی برای بررسی فشردن یک کلید یا فعال شدن یک پرچم خاص هر چند لحظه بخواهیم بررسی را انجام دهیم در واقع به روش polling عمل کرده ایم عیب این روش تلف کردن وقت cpu است اما به این معنی نیست که این روش فایده و کاربردی ندارد.

۲ - روش وقفه(interrupt):

در مثالی که طرح کردیم یک راه ساده تر آن است که شما به درس خواندن خود ادامه دهید و منتظر زنگ وی بمانید و هر موقع زنگ را زد بروید و به او پاسخ دهید و پس از اتمام کارتان برگردید و به درس خواندن ادامه دهید. در این روش وقت کمتری نسبت به مرحله قبل (روش سرکشی) از شما گرفته می‌شود. این روش همان روش وقفه (Interrupt) نامیده می‌شود.

در این روش cpu بدون توجه به وقوع رویداد، به کارهای دیگر مشغول است و با وقوع رویداد مورد نظر، پس از انجام خط جاری، برنامه را متوقف کرده و به بردار وقفه مربوطه پرش می‌کند و زیر روال سرویس وقفه (Interrupt Service Routin) ISR را اجرا می‌کند. پس از اجرای این زیر روال، به خطی از برنامه که اجرای آنرا قطع کرده بود برگشته و ادامه برنامه را اجرا خواهد کرد.

برای درک بهتر مفهوم وقفه مثال زیر را د رنظر بگیرید

فرض کنید که دائماً می خواهید داده های موجود بر روی پورت A را خوانده و عملیاتی را بر روی این داده ها انجام دهید. در عین حا ل، در نظر دارید که داده های رسیده از طرف پورت سریال را هم دریافت کنید. اگر بخواهید این دو عمل را در یک برنامه و به صورت ادغام شده در داخل هم انجام دهید زمانی که اجرای برنامه بر روی دستوری مانند Waitkey (یکی از دستورات دریافت داده از پورت سریال در BASCOM) متوقف می‌شود، حجم زیادی از داده های موجود بر روی پورت A را ممکن است از دست بدهید. بنابراین باید چاره ای اندیشید تا فقط زمانی که داده سریال دریافت شد روند اجرای برنامه متوقف شود. به این منظور سرویسهای اعلام وقفه در داخل میکروکنترلر گنجانده شده است تا هر یک از اجزای سیستم میکروکنترلر ، با اعمال یک سیگنال وقفه، اجرا را به روال وقفه مورد نظر منتقل کرده و پس از سرویس دهی به وقفه مورد نظر کنترل اجرا به همان نقطه ای از برنامه اصلی که متوقف شده بود برگردد.

لازم به ذکر است که اگر در حین اجرای یک دستور العمل از بدنه اصلی برنامه، به CPU وقفه داده شود دستور جاری به طور کامل اجرا خواهد شد و سپس روال وقفه اجرا می‌شود همچنین پس از بازگشت از یک روتین وقفه، در صورتی که وقفه دیگری بلافاصله رخ دهد. حداقل یک دستور از بدنه اصلی برنامه اجرا خواهد شدو سپس به وقفه پاسخ داده می‌شود. پاسخ اجرای وقفه برای همه ی اینتراپت های فعال AVR حداقل ۴ کلاک سیکل است.(در طی ۴ پالس ساعت آدرس بازگشت،در پشته ذخیره و برنامه سرویس وقفه اجرا می‌شود).

اگر در حین سرویس دهی به یکی از وقفه ها، وقفه دیگری رخ دهد چه خواهد شد؟

در این مورد باید گفت که معماری میکروکنترلرهایAVR به نحوی طراحی شده است که وقوع وقفه های متداخل ثبت شده و پس از خروج از روتین وقفه جاری و اجرای حداقل یک دستور از برنامه اصلی به وقفه های رخ داده معلق به ترتیب اولویت پاسخ داده می‌شود.

اصطلاح روال وقفه یا روتین سرویس دهی چیست؟

برای هر وقفه موجود در معماری میکروکنترلر مورد استفاده شما، یک روتین سرویس دهی به وقفه وجود دارد. در واقع اصطلاح روتین چیزی غیر از یک بلوک از برنامه نیست. هر بلوک برنامه متعلق به یک وقفه، دارای یک نقطه خروجی و یک نقطه ورودی است. هنگامی که یک وقفه رخ دهد، کنترل اجرای دستورات، به آدرس مربوط به روتین آن وقفه پرش می‌کند و دستورات موجود در بلوک مربوطه را اجرا خواهد کرد و پس از آن به محلی که قبلا برنامه اصلی در حال اجرا بود بر می گردد. در هر میکروکنترلر

قسمتی از حافظه Flash که معمولا قسمت ابتدای آن است، آدرس روتین ها مربوط به هر وقفه را نگهداری می‌کند که به این قسمت از حافظه جدول بردار وقفه[1] گفته می‌شود. در جدولی که مشاهده می کنید نام منابع وقفه موجود در میکروکنترلر AVR به شماره ATmega32ذکر شده است. توجه به این نکته حائز اهمیت است که همه این منابع ممکن است در یک میکروکنترلر خاص موجود نباشند. این لیست همچنین اولویت اینتراپت های مختلف را نشان می‌دهد. پایین ترین آدرس، بالاترین اولویت را دارد. RESET بالاترین اولویت را دارد و بعد از آن اینتراپت خارجی صفردر اولویت قرار دارد. از این وقفه ها سه نوع آن خارجی بوده (INT0,INT1,INT2) و بقیه داخلی اند.

مراحل اجرای یک وقفه:

میکرو کنترلر با دریافت وقفه خارجی یا داخلی مراحل زیر را انجام می‌دهد:

۱ - دستوری که در حال اجرای آن میباشد را به پایان رسانده و آدرس دستور العمل بعدی را در حافظه پشته ذخیره میکند.

۲ - به بردار وقفه مربوطه پرش میکند و برنامه یا زیر روال سرویس وقفه (ISR) را انجام می‌دهد.

۳ - پس از اتمام زیر روال وقفه، توسط دستور اسمبلیRETI از وقفه برمیگردد. البته در زبان C یا basic دستور RETI را بکار نمی بریم اما کامپایلر پس از تفسیر برنامه از آن استفاده میکند.

۴ - آدرس دستور العملی که در مرحله یک در حافظه پشته ذخیره کرده بود را بر می دارد و با قرار دادن ان آدرس، در شمارنده برنامه (PC) به ادامه برنامه بر میگردد.

1 - IVT = Interrupt Vector Table

Vector NO	Program Adress(2)	Source	Interrupt definition
1	$000(1)	RESET	External Pin, Power-on RESET, Brown-out RESET, Watchdog RESET, and JTAG AVR RESET
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COPM	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COM	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI,STC	Serial Transfer Complete
14	$01A	USART,RXC	USART, Rx Complete
15	$01C	USART,UDRE	USART Data Register Empty
16	$01E	USART,TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_COMP	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

به منظور قبول اینتراپت توسط میکروکنترلر باید بیت متناظر با وقفه عمومی که در رجیستر وضعیت SREG قرار دارد فعال شده باشد. بیتهای این رجیستر به صورت زیر است.

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

بیت هفتم این رجیستر با عنوان I یا Global Interrupt Enable است. اگر این بیت را یک کنیم وقفه کلی(سراسری یا همگانی) فعال میگردد و در این حالت است که دیگر وقفه ها در صورت فعال بودن و تحریک شدن می‌توانند مورد پذیرش میکرو قرار گیرند. در صورتی که این بیت صفر باشد و دیگر وقفه ها فعال باشند و حتی تحریک هم شوند قابل اجرا توسط CPU نخواهند بود. با وقوع هر وقفه ی داخلی یا خارجی این بیت پاک شده و در نتیجه تمام وقفه های دیگر غیر فعال می‌شوند. دراین حالت نرم افزار می‌تواند با نوشتن یک روی این بیت آن را مجددا فعال کند و باعث ایجاد وقفه های تو در تو شود. با بازگشت از ISR این بیت

مجددا یک می‌شود. لازم بذکر است که وقفه کلی توسط دستور اسمبلی SEI فعال و با دستور اسمبلی CLI غیر فعال می‌شود. این دو دستور معادل یک یا صفر کردن بیت GIE در رجیستر وضعیت SREG است.

وقفه های خارجی(External Interrupts):

برای درک علت استفاده از وقفه های خارجی درنظر بگیرید که میکروکنترلی در حال کنترل یک پروسه بزرگ باشد و علاوه بر این، کنترل یک وضعیت اضطراری پروسه را بر عهده داشته باشد.چگونه باید هر دو برنامه را اجرا کند ؟با فرض اینکه سنسوری بکار برده شده باشد که هنگام تشخیص وضعیت اضطراری، یک لبه بالا رونده ایجاد و به پایه میکرو کنترلر اعمال کند ایا میکرو کنترلر با توجه به حساسیت پروسه،میتواند دائما مقدار پایه میکرو کنترلر را خوانده و ایجاد لبه را تشخیص دهد ؟ در صورت انجام چنین کاری قادر به اجرای برنامه کنترل پروسه نخواهد بود زیرا دایما باید مقدار پایه را بررسی کند برای رفع این مشکل از وقفه خارجی استفاده و سیگنال خروجی سنسور به پایه ای از میکرو کنترلر که برای این کار در نظر گرفته شده است، اعمال می گردد در این صورت میکرو کنترلر در حالت عادی برنامه کنترل پروسه را اجرا میکند درصورت تشخیص وضعیت اضطراری توسط سنسور و اعمال یک لبه بالا رونده به پایه وقفه خارجی میکرو کنترلر، بلافاصله برنامه سرویس وقفه که وظیفه آن برطرف کردن وضعیت اضطراری است، اجرا و مجددا ادامه برنامه اصلی اجرا می‌شود.

بنابراین از وقفه های خارجی معمولا برای تشخیص پالس استفاده می‌شود حال این پالس می‌تواند حاوی اطلاعات گوناگونی باشد بطور مثال پالس هایی که از طرف یک سنسور خاص ارسال می گردد یا پالسی که در اثر وصل یک میکرو سوئیچ ایجاد می گردد یا پا لسی که از طرف یک مبدل انالوگ به دیجیتال بیرونی ارسال میشود یا پا لسی که از هر تراشه و رویدادی ارسال می‌شود که ما بخواهیم توسط میکرو کنترلر خیلی سریع به ان پاسخ بگوییم از وقفه خارجی بهره می گیریم. برای استفاده از هریک از وقفه های خارجی باید با یک کردن بیت مربوطه در رجیستر GICR(General Interrupts Control Register) آن را فعال نمود.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

وقفه های خارجی ۱و۲و۳ به ترتیب از پین های PD2 و PD3 وPB2 تریگر می‌شوند. نوع تریگر شدن هریک از وقفه های خارجی 0 و 1 بوسیله ی چهاربیت اول رجیستر MCUCR و به صورت مندرج در جدول زیر تعیین می‌شوند:

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISC11	ISC10	Discription
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC11	ISC10	نوع تریگر شدن وقفه ی یک
0	0	سطح منطقی صفر در پین INT1
0	1	تغییر در سطح منطقی پین INT1
1	0	لبه ی پایین رونده در پین INT1
1	1	لبه ی بالا رونده در پین INT1

ISC01	ISC00	نوع تریگر شدن وقفه ی صفر
0	0	سطح منطقی صفر در پین INT0
0	1	تغییر در سطح منطقی پین INT0
1	0	لبه ی پایین رونده در پین INT0
1	1	لبه ی بالا رونده در پین INT0

نوع تریگر شدن وقفه ی خارجی 2 بوسیله بیت 6 از رجیستر MCUCSR تعیین می‌شود:

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

وقفه ی 2 خارجی بر خلاف وقفه ی 0 و 1 تنها در دو حالت لبه ی بالا رونده و پایین رونده قابل پیکربندی است نوشتن صفر در بیت ISC 2 باعث تریگر شدن این وقفه با لبه ی پایین رونده و نوشتن یک باعث تریگر شدن آن با لبه ی بالا رونده می‌شود.

هر یک از وقفه های خارجی دارای یک بیت پرچم هستند که در صورت تریگر شدن از پین وقفه ی خارجی و فعال بودن بیت مربوطه در رجیستر GICR (General Interrupt Control Register) و فعال بودن بیت فعال ساز وقفه(I) علاوه بر یک شدن پرچم، می‌تواند باعث ایجاد وقفه شود .در این حالت پس از اجرایISR پرچم آن وقفه به صورت سخت افزاری پاک می‌شود.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

پیکر بندی درCodeVision :

```
(void)  نام روتین سرویس وقفه void ]شماره بردار وقفه[ Interrupt
{
برنامه سرویس وقفه
}
```

شماره بردار را از جدول بردار وقفه ذکر شده در صفحات قبل میتوان بدست اورد.

پیکر بندی درBASCOME:

با استفاده از دستور زیر می‌توان منابع وقفه خارجی را راه اندازی کرد CONFIG

```
INTx = state
Enable Interrupts
Enable Intx
On Intx lable[No Save]
برنامه اصلی
Lable:
برنامه سرویس وقفه
Return
```

INTx: x نام پایه وقفه است که در میکروکنترلر ATmega32به نامINT0وINT1و INT2موجود است

State :**نوع** پالس اعمالی برای فعال شده وقفه را معین می‌کند، Stateمی‌تواند یکی از موارد زیر باشد:

FALLING: با اعمال یک پالس پایین روند (یک به صفر) به پایه مورد نظر وقفه فعال می‌شود

RISING: با اعمال یک پالس بالا رونده(صفر به یک) به پایه مورد نظر وقفه فعال می‌شود.

LOW LEVEL :با اعمال سطح صفر به پایه مورد نظر وقفه فعال می‌شود.

Change:با تغیرات لبه سطح اعمالی وقفه فعال می‌شود.

با دستورEnable Interrupts وقفه سراسری و بادستورEnable Intx وقفه پیکر بندی شده فعال می‌شود(وبا دستورDisableغیر فعال می‌شود) در نهایت شما می‌توانید با دستور[On Intx lable[No Save به هنگام اعمال پالس به lable مورد نظر پرش کنید. با فعال شدن وقفه، پایه های ورودی وقفه در همه حالت ها چک می‌شود و نیازی به اوردن دستورات در حلقه اصلی ودیگر حلقه ها نیست.باز گشت از زیر برنامه با دستور Return انجام می‌شود.

تذکر:با نوشتن عبارت No Saveهیچ یک از رجیسترهای وضعیت[R0…R11,R16…R31]ذخیره نمی شوندودر برنامه وقفه تغییری نمیکنندودر صورت استفاده نکردن از انتمام رجیستر های استفاده شده در برنامه وقفه تغییر نمیکنند.

نکته: قسمت

```
Lable:
        برنامه سرویس وقفه
Return
```

بعد از end برنامه اصلی نوشته می‌شود.

آموزش برنامه Code vision توسط برنامه نویسی C

در این بخش الگوی دستورهای عمومی کامپایلر C برای Code vision شرح داده خواهد شد و فقط دستورهای ویژه ای از زبان C که در این کامپایلر استفاده شده است، بیان می گردد.

نکات مورد استفاده در برنامه نویسی با زبانC با کامپایلر(Code vision)

۱) در زبان C بین حروف کوچک و بزرگ تفاوت وجود دارد. در این زبان ، دستورات و کلمات کلیدی با حروف کوچک نوشته می‌شود. به عنوان مثال کلمه کلیدی void با کلمه VOID فرق دارد و اگر دستورات و کلمات کلیدی را با حروف بزرگ بنویسیم کامپایلر پیغام خطا می دهد. به عنوان نمونه ای دیگر short یک کلمه کلیدی است ولی SHORT یا shoRT کلمات کلیدی نیستند. توصیه می‌شود که تمام برنامه های این زبان با حروف کوچک نوشته شوند.

دستورات مخصوص میکرو باید با حروف بزرگ نوشته شودمثل DDRC=0x23.

مثال:

```
DDRB=0b00110110 ;
```

یک بودن بیت رجیستر DDRB به منزله خروجی بودن و صفر بودن آن به منزله ورودی بودن پایه متناظر پورت است. به عنوان مثال در مثال فوق پایه ۵ از پورت B خروجی و پایه صفر از پورت B ورودی است.

۲) اعداد هگزاسیمال را در زبان C با پیشوند 0x و اعداد باینری را با پیشوند 0b معرفی می‌کنیم . برای نمایش اعداد دسیمال پیشوند نیاز نیست.

مثال:

```
K=100 ;          در سیستم دسیمال
K=0B100 ;   k= 4          در سیستم باینری
K=0X100 ;          در سیستم هگزادسیمال
```

۳) توضیحات در یک سطر می‌تواند بعد از // قرار بگیرد و همچنین توضیحات گروهی (بیش از یک سطر) با */ شروع می‌شود و به /* ختم می‌شود.

مثال:

```
K= 100 ; // k is the number of students
G= 250 ; /*g is…….                    (برای توضیحات بیشتر)
……
                ……*/
```

۴) در آخر تمامی دستورات زبان C باید عبارت سیمی کالون(;) قرار داد که نشان دهنده آن است که خط دستوری پایان یافته است(به غیر از دستوراتی که با علامت مخصوص کمپایلر (#) شروع می‌شود).

۵) حداکثر طول یک دستور، ۲۵۵ کلراکتر است.

۶) در هر سطر می‌توان یک یا چند دستور نوشت.

۷) هر برنامه ی C حداقل یک تابع ()main دارد. یک برنامه ی میکروکنترلری درساده ترین حالت خود به شکل زیر است:

```
# include< mega32.h>
Voidmain( )
{
While(1) {
/*place your program here*/
•
•
}
}
```

خطوطی که با علامت پوند(#) شروع می شوند دستوراتی برای پیش پردازنده بوده و جزء خطوط اجرایی برنامه نمی باشند بلکه فقط نشانه هایی برای کامپایلراند. به عنوان مثال دستور # include < mega32.h> در دستورات فوق به پیش پردازنده می گوید که میکروی مورد استفاده Atmega32 است.

گاهی اوقات در برنامه نویسی به توابعی نیاز پیدا می کنیم که نیاز ندارند چیزی را به عنوان خروجی تابع برگردانند و یا توابعی که نیاز به آرگومان و ورودی ندارند و یا هر دو. زبان C برای امکان استفاده از چنین توابعی، کلمه void را در اختیار ما قرار داده است. اگر بخواهیم تابعی بدون خروجی ایجاد کنیم کافی است به جای نوع داده خروجی تابع کلمه void را قرار دهیم. تابع main را از نوع void تعریف کرده ایم چون چیزی را به عنوان خروجی برنمیگرداند.

دستور void main( ) شروع بدنه اصلی برنامه را مشخص می‌کند. تابع main اولین جایی از برنامه است که کامپایلر شروع به اجرای آن می‌کند. فرقی ندارد که تابع main را در کجا مورد استفاده قرار دهیم. ابتدا، وسط یا انتهای کدهای برنامه نویسی. در هر کجا که تابع main را قرار دهیم، زبان C ابتدا به این تابع مراجعه می‌کند. بعد از کلمه main یک جفت پرانتز ( ) قرار می دهیم، چون main یک تابع است. در زبان C تمام توابع دارای یک جفت پرانتز می‌باشند.

برای ایجاد توابع بدون آرگومان می‌توانید در پرانتز تابع کلمه void را بنوسید یا آنکه این پرانتز را خالی گذاشته و در آن چیزی ننویسید. محتویات تابع main همانطور که در برنامه دیدید بین دو علامت { } قرار می گیرند.

تابع با نام func2 که دارای چهار آرگومان ورودی است که ورودی اول آن یک عدد ۱۶ بیتی و سه ورودی دیگر آن اعداد ۸ بیتی هستند ولی متغیری را به عنوان خروجی برنمی گرداند به صورت زیر تعریف می کنیم.

```
void func2(int, char, char, char);
```

توابعی که چیزی را برنمی گردانند بدون دستور Return نوشته می شوند.

توجه داشته باشید که بعد از هر دستور زبان C ملزم به استفاده از علامت (;) می باشیم.

انواع داده در زبان C:

در زبان C چهار نوع داده اصلی وجود دارد که عبارتند از:

۱- char: این نوع داده برای ذخیره داده های کاراکتری مانند 'a' ,'1' ,'0' به کار می‌رود و بازه قابل قبول آن از ۱۲۸- تا ۱۲۷+ است. در حقیقت خانه های Char نیز از نوع اعداد صحیح می باشند که یک بایت طول دارند و کداسکی کاراکتر مورد نظر را در خود حفظ می‌کنند. به عنوان مثال کداسکی کاراکتر A عدد ۶۵ دسیمال(یا ۴۱ هگز) است.

۲- int: این نوع داده برای ذخیره اعداد صحیح تا ۱۶ بیتی مانند ۱۳۰۰، ۳۲۰۰، ۸۵۰- به کار می‌رود و بازه قابل قبول آن ۳۲۷۶۸- تا ۳۲۷۶۷ می‌باشد.

۳- float: این نوع داده برای ذخیره اعداد اعشاری مانند ۵۲۴۱/۱۲، ۳/۱۵۰۱-، ۱۲۳۴/۱۴۱۵ به کار می‌رود و دقت آن تا ۷ رقم اعشاری است.

۴- double: این نوع داده برای ذخیره سازی اعداد اعشاری بزرگ به کار می‌رود و دقت آن از float بیشتر است. با کلماتی مانند signed(علامت دار)، unsigned(بدون علامت)، short(کوتاه) و long(بلند) انواع داده های جدید نیز می‌توان ایجاد کرد. مانند Unsigned char k که تعریف یک داده بدون علامت ۸ بیتی با نام k را انجام می‌دهد.

در جدول زیر انواع داده ها همراه با حافظه ای که اشغال می‌کند، آمده است.

type	Size(bits)	range	
bit	1	0 و 1	2^1-1
(signed) char	8	-128 to +127	$-2^7 \leq x < +2^7$
unsigned char	8	0 to 255	2^8-1
(signed) int	16	-32768 to +32767	$-2^{15} \leq x < +2^{15}$
unsigned int	16	0 to 65535	2^{16}-1
short int	16	-32768 to +32767	$-2^{15} \leq x < +2^{15}$
(signed) long int	32	-2147483648 to +2147483647	$-2^{31} \leq x < +2^{31}$
unsigned long int	32	0 to 4294967295	2^{32}-1
float	32	1.175e-38 to 3.402e+38	تا ۷ رقم اعشار
double	32	1.175e-38 to 3.402e+38	بیش از ۷ رقم اعشار

Signed (علامتدار بودن) پیش فرض برای داده ها است.

متغیرها:

تعریف متغیر یعنی انتخاب نام مستعار برای مکانی از حافظه. فرم تعریف متغیر بصورت زیر است.:

نوع متغیر(نوع داده) نام متغیر ;

```
char a;
unsigned int k;
```

برای معرفی متغیر ها فقط می‌توان از حروف، اعداد و... استفاده کرد و نیز حرف اول نام یک متغیر باید حتماً یک حرف باشد. به عنوان مثال نامهای 1test و test!num و mark.1 اسامی غیر مجاز هستند.

بین حروف نام متغیر نمی‌توان از کاراکتر فاصله استفاده کرد.
زبان C همانند سایر زبانهای برنامه نویسی دارای تعدادی کلمات کلیدی است که نمی‌توان از این کلمات به عنوان نام متغیر استفاده کرد. برخی از این کلمات عبارتند از:

Char , case , break , auto , asm , delete , class , enum , float , goto , double , …

برای تعیین محل تعریف متغیر از عملگر @ استفاده می‌کنیم به عنوان مثال:

```
Int b@0xA3;
```

برای تعریف متغر در حافظه‌ی EEPROM از کلمه‌ی کلیدی eeprom قبل از نام متغیر استفاده می‌کنیم به عنوان مثال:

```
eeprom int code;
```

می‌توان در زمان تعریف متغیر، به متغیر مقدار اولیه نسبت داد. مثال: char s="a";

اگر مکان ذخیره متغیر را ذکر نکنیم بصورت پیش فرض متغیر در حافظهSRAM تعریف می شود. اگر مقدار اولیه به متغیر ندهیم بصورت پیش فرض مقدار صفر برای آن در نظر گرفته می شود.

انواع متغیر در زبان C:

۱- متغیرهای کلی یا عمومی(global): این متغیرها بعد از معرفی پیش پردازنده ها و شناسه ها در ابتدای برنامه و خارج از بدنه های توابع می آیند و مقدار آنها در تمامی توابع قابل دسترسی می‌باشد و مقدار خود را در تمامی توابع حفظ می‌کنند.

۲- متغیرهای محلی(local): این متغیرها در داخل بدنه توابع تعریف می شوند و مقدار آنها باخارج شدن از بدنه توابع از بین می‌رود یعنی صفر می‌شود.

تفاوت ذخیره متغیرها در حافظه های مختلفAVR:

بر روی هر یک از حافظه های سه گانه AVR (حافظه FLASH، حافظه EEPROM و حافظه SRAM) می‌توان متغیر را معرفی کرد. برای معرفی متغیر بر روی حافظه FLASH و EEPROM از خود این عبارات در ابتدای معرفی متغیر استفاده می‌شود. اگر در ابتدای معرفی متغیر اسمی قرار داده نشود، متغیر بر روی حافظه SRAM ذخیره خواهد شد.
برای تعریف متغیر درحافظه ی EEPROM از کلمه ی کلیدی eeprom قبل از نام متغیر استفاده می‌کنیم. به عنوان مثال:

```
eeprom char ali ;
```

متغیری به نام علی از نوع char تعریف می‌کند با محدوده ۸ بیت و بر روی حافظه eeprom.
می‌توان در زمان تعریف متغیر، به متغیر مقدار اولیه نسبت داد به عنوان مثال

```
Char s='a' ;
```

برای ذخیره سازی در FLASH و SRAM به عنوان نمونه داریم:

```
Flash int ali ;
```

متغیری به نام علی از نوع int تعریف می‌کند با محدوده ۱۶ بیت و بر روی حافظه flash.

Float ali ;

متغیری به نام علی از نوع float تعریف می‌کند با محدوده ۳۲ بیت و بر روی حافظه sram.

اگر متغیری در حافظه sram ذخیره شده باشد چنانچه میکرو ریست یا خاموش شود مقدار متغیر به مقدار صفر تغییر خواهد کرد(حافظه sram حافظه ذخیره موقتی است). اگر متغیری در حافظه eeprom تعریف شده باشد با قطع شدن تغذیه یا ریست شدن میکرو آخرین مقداری که به متغیر تخصیص(assign) داده شده بود در متغیر باقی خواهد ماند. اگر متغیری در حافظه flash تعریف شده باشد نه تنها با قطع شدن تغذیه میکرو مقدار آن تغییر نمی‌کند بلکه مقدار آن را برنامه نویس نیز نمی‌تواند در طول اجرای برنامه تغییر دهد مگر آن که مقدار جدید متغیر برنامه نویسی شده و دوباره به داخل IC پروگرام شود به همین علت به متغیرهایی که در حافظه flash ذخیره می‌شود ثابت ها گویند.

سه روش برای تعیین ثابت ها در برنامه میکرو وجود دارد:

۱- تعریف به عنوان متغیر در حافظه flash:

flash float pi=3.14 ;

۲- استفاده از کلمه کلیدی Const(ثابت):

Const float pi=3.14;

در حا فظه SRAM ذخیره می‌شود.

۳- استفاده از دستور # define:

#define pi 3.14 ;

معرفی کریستال میکرو با مقدار 4MHZ #define xtal 4 ;

نکات قابل توجه:

برای معرفی متغیرها فقط می‌توان از حروف، اعداد و ... استفاده کرد و نیز حرف اول نام یک متغیر باید یک حرف (و نه یک رقم) باشد. به عنوان مثال نام های mark.1 و test!num و 1test اسامی غیرمجازند. بین حروف نام متغیر نمی‌توان از کاراکتر فاصله استفاده کرد.

زبان C همانند سایر زبان های برنامه نویسی دارای تعدادی کلمات کلیدی است که نمی‌توان از این کلمات به عنوان نام متغیر استفاده کرد. برخی از این کلمات عبارتند از:

Char, case, break, auto, asm, delete, class, enum, float, go to, double,…

آرایه:

یک آرایه مجموعه ای از خانه های متوالی حافظه است که دارای یک نام و یک نوع می باشند که به هر یک از این خانه ها یک عنصر آرایه گفته می‌شود.

برای دستیابی به یک عنصر آرایه، باید نام آرایه و شمارنده آن خانه را مشخص کنیم. لذا عناصر آرایه توسط متغیری به نام اندیس مشخص می شوند به همین دلیل، آرایه ها را متغیرهای اندیس دار نیز می گویند. نام آرایه، از قواعد نام گذاری متغیرها پیروی می‌کند. نوع آرایه نیز یکی از انواع داده است. اعلان آرایه ها به صورت زیر است:

نوع داده آرایه **نام آرایه** [طول آرایه] ;

به عنوان مثال دستور زیر آرایه ای به طول ۴ با نام num را از نوع int ایجاد می‌کند.

```
Int  num[4] ;
```

25	num[0]
4A	num[0]
C3	num[1]
56	num[1]
0D	num[2]
50	num[2]
3B	num[3]
A3	num[3]

توجه داشته باشید که چون هر مکان حافظه قابلیت ذخیره یک بایت را دارد و متغیرهای تعریف شده در آرایه فوق دوبایتی هستند لذا برای آرایه چهار عضوی هشت مکان از حافظه تخصیص داده شده است.

توجه داشته باشید که **تمام عناصر آرایه دارای نام یکسانی بوده** و تنها با اندیس از هم تفکیک می شوند. ضمناً اندیس عناصر از صفر شروع می‌شود. برای تقسیم عنصر چهارم آرایه بر ۲ و قرار دادن حاصل در متغیر x از دستور زیر می‌توان استفاده کرد.

```
x=num[3]/2;
```

نکته : توجه داشته باشید که عنصر چهارم آرایه با عنصر شماره چهار (با اندیس چهار) متفاوت است. همانطور که در دستور فوق دیدید عنصر سوم دارای اندیس دو می‌باشد، دلیل این امر این است که اندیس گذاری از صفر شروع می‌شود. در آرایه فوق عنصر سوم آرایه num[2]=0D50 است ولی عنصر شماره سه (با اندیس سه) num[3]=3BA3 است.

همانند متغیرها چند آرایه را می‌توان توسط یک دستور تعریف کرد:

```
Int  b[100] , x[27];
```

دستور فوق ۱۰۰ خانه از نوع عدد صحیح را برای آرایه با نام b و ۲۷ خانه از نوع عدد صحیح را برای آرایه با نام x در نظر می گیرد.

برای **مقدار دهی اولیه به هر یک از عناصر آرایه**می‌توانید از شیوه زیر استفاده کنید:

```
int n[5]= {32,27,64,18,95}
```

دستور فوق عناصر آرایه n را به صورت فوق مقدار دهی می‌کند.

اگر طول آرایه هنگام تعریف آرایه تعیین نشده باشد و لیست مقدار عناصر نوشته شود، همانند دستور زیر:

```
char n[ ]={1,2,3,4,5}
```

در این صورت کامپایلر به تعداد عناصر لیست، خانه حافظه برای آرایه در نظر می گیرد، مثلاً دردستور فوق ۵ خانه حافظه برای آرایه n در نظر گرفته می‌شود.

راه دیگری که برای مقدار دهی اولیه به عناصر آرایه وجود دارد، استفاده از روش زیر است:

```
int num[10]={0}
```

دستور فوق ۱۰ خانه حافظه برای آرایه num در نظر می گیرد و مقادیر همه آن ها را صفر می‌کند. توجه داشته باشید که اگر از دستور زیر استفاده کنیم، تمامی عناصر مقدار ۱ را نمی گیرند بلکه عنصر اول آرایه یک می‌شود وبقیه عناصر مقدار صفر را می گیرند.

```
int num[10]={1}
```

آرایه های چند بعدی: آرایه ها در C می‌توانند بیش از یک اندیس داشته باشند. بدین صورت یک آرایه چند اندیس یا چند بعدی خواهیم داشت. کاربردی ترین آرایه چند بعدی، آرایه دوبعدی می‌باشد که توسط آن می‌توان جدولی حاوی مقادیر مختلف را شبیه سازی کرد. به دستور زیر توجه کنید:

```
int a[3] [4] ;
```

دستور فوق یک آرایه دو بعدی ۳ در ۴ را به صورت زیر ایجاد می‌کند:

a[0] [0]	a[0] [1]	a[0] [2]	a[0] [3]
a[1] [0]	a[1] [1]	a[1] [2]	a[1] [3]
a[2] [0]	a[2] [1]	a[2] [2]	a[2] [3]

هر عنصر آرایه به صورت a[i] [j] که در آن i شماره سطر و j شماره ستون می‌باشد، قابل دسترسی است. برای مقدار دهی اولیه به عناصر آرایه می‌توانید مانند دستور زیر عمل کنید:

```
int b [2] [2]={{1,2},{3,4}};
```

دستور فوق آرایه b را به صورت زیر مقدار دهی می‌کند:

2	1
4	3

عملگرها:

الف- عملگرهای حسابی و بیتی:

عملگر	عملکرد	مثال	نتیجه
*	ضرب	3*2	6
/	تقسیم	5/2	2.5
+	جمع	3+6	9
-	تفریق	8-3	5
٪	باقیمانده	10%3	7
&	AND	0xF0 & 0x0F	0x00
l	OR	0x00 l0x03	0x03
^	XOR	0x0F ^0xFF	0xF0
~	مکمل یک	~(0xF0)	0x0F
>>	شیفت به راست	0xF0 >> 4	0x0F
<<	شیفت به چپ	0x0F << 4	0xF0

ب- عملگر های منطقی:

عملگر	عملکرد	مثال	نتیجه
&&	AND منطقی	(2>3)&&(11=3)	False
ll	OR منطقی	('a'<3) ll (10)	True
!	نقیض	!(7>5)	False

ج- عملگرهای انتساب:

عملگر= عملگر انتساب است که مقدار سمت راست را در متغیر سمت چپ قرار می دهد. دقت کنید که این عملگر با عملگر= = که عملگر برابر بودن دو متغیر است متفاوت است. به عنوان مثال در عبارت a (== (b=2)) ابتدا مقدار ۲ به متغیر b نسبت داده شده و سپس چک می‌شود که آیا a با b برابر هست یا خیر

عملگر	عملکرد	مثال	نتیجه
=	انتساب	a=b	a←b
*=	ضرب و انتساب	a*=b	a=a*b
/=	تقسیم و انتساب	a/=b	a=a/b
+=	جمع و انتساب	a+=b	a=a+b
-=	تفریق و انتساب	a-=b	a=a-b
a=(value)?x:y	انتساب شرطی	اگر value مقدار صحیح باشد a برابر x و درغیراینصورت برابر y می‌شود	

د- عملگرهای یکانی:

شامل عملگرهای افزایش یک واحد به متغیر، کاهش یک واحد به متغیر و قرینه کردن یک متغیر است. عملگر افزایش به صورت ++ و عملگر کاهش به صورت - - است. عملگر افزایش(++) یک واحد به مقدارقبلی که در متغیر اضافه می‌کند و عملگر کاهش (- -) یک واحد از مقدار قبلی که در متغیر بود کم می‌کند. a++; a+=1; ++a; a=a+1;

هر چهار دستور فوق یک واحد به مقدار قبلی متغیی اضافه می‌کنند.

--a; a--; a-=1; a=a-1;

هر چهار دستور فوق یک واحد از مقدار قبلی متغیر کم می‌کنند.

اگر دستورات ++a و a++ به تنهایی استفاده کنیم فرقی ندارد که ++ قبل از متغیر قرار گیرد یا بعد از متغیر. اما اگر از++ در کنار عملگرهای دیگر استفاده شود، اگر قبل از متغیر قرار گیرد ابتدا یک واحد به متغیر اضافه شده سپس در محاسبه استفاده می‌شود، ولی اگر ++ بعد از متغیر قرار گیرد ابتدا متغیر در محاسبه استفاده می‌شود سپس یک واحد به آن اضافه می‌شود. همین روال برای عملگر- -نیز برقرار است. به عنوان مثال:

```
b=3;
a= b++;    a=++b;
```

در مثال سمت چپ ابتدا یک واحد به b اضافه می‌شود، یعنی b مقدار ۴ را می گیرد سپس عدد ۴ در a قرار می گیرد. اما در مثال سمت راست ابتدا مقدار b یعنی عدد ۳ در a قرار می گیرد سپس یک واحد به b اضافه می‌شود و مقدار ۴ را می گیرد. درمثال زیر عدد ۵ در m قرار می گیرد:

```
a=2;
b=3;
```

m=++a+b-- ;

b مقدار ۲ و a مقدار ۳ را می گیرد.

عملگر	عملکرد	مثال	نتیجه
-	قرینه	-a	a= a×-1
++	افزایش یک واحد	a++	a= a+1
--	کاهش یک واحد	a--	a= a-1

ه- عملگرهای مقایسه ای

عملگر	عملکرد	مثال	نتیجه
>	بزرگتر	2>3	False
<	کوچکتر	'm' > 'e'	True
>=	بزرگتر یا مساوی	5>=5	True
<=	کوچکتر یا مساوی	2.5< = 4	True
= =	تساوی	'A'= = 'B'	False
!=	نامساوی	21=3	True

دستورات تصمیم گیری و انتخاب:

ساختار if:

if(شرط مورد نظر)

دستور مورد نظر ;

به مثال زیر توجه کنید:

```
If(x==50)
x+=y;
```

اگر از دستورفوق در برنامه استفاده کنیم، اگر مقدار متغیر x قبل از رسیدن به شرط فوق برابر ۵۰ باشد، مقدار x+y در متغیر x قرار می گیرد وگرنه این دستور نادیده گرفته می‌شود و برنامه خط بعدی را اجرا می‌کند.

اگر بخواهیم هنگامی که شرط برقرار می‌شود، چند دستور اجرا شود باید دستورات مورد نظر را با علامت { } دسته بندی، به مثال زیر توجه کنید:

```
if(x==50)
{
```

```
  y++;
x+=y;
}
```

قطعه کد فوق هنگامی که مقدار x عدد ۵۰ باشد، ابتدا یک واحد به y اضافه نموده و سپس حاصل x+y را در متغیر x قرار می دهد.

ساختارانتخاب if-else:

گاهی اوقات نیاز داریم که در صورت برقرار بودن شرط خاصی یک سری دستورات اجرا و در صورت برقرار نبودن شرط، دسته ای دیگراز دستورات اجرا گردند. به عنوان مثال اگر فردا باران بیاید من به کوه نمی روم در غیر این صورت من به کوه خواهم رفت. زبان C برای پیاده سازی چنین ساختاری شیوه زیر را در اختیار ما قرار داده است.

```
if(شرط مورد نظر)
دستور۱ ;
else
دستور۲ ;
```

اگرشرط برقرار باشد دستور ۱ اجرا می گردد و در غیر این صورت دستور ۲ اجرا می‌شود. به مثال زیر توجه کنید:

```
if(x==50)
x+=x;
else
x-=x;
```

اگر مقدار متغیر قبل از رسیدن به شرط فوق برابر ۵۰ باشد یک واحد به متغیر x اضافه می‌شود و اگر برابر ۵۰ نباشد یک واحد از آن کم می‌شود.

به یاد داشته باشید اگر می خواهید از بیش از یک دستور استفاده کنید، حتماً آنها را با { } دسته بندی نمایید.

به عنوان مثال:

```
if(x > 50)
{
x+=x;
y-=y;
}
else
{
x-=x;
y+=y;
}
```

اگرمقدار متغیر x بزرگتر از ۵۰ باشد، یک واحد به متغیر x اضافه شده و یک واحد از متغیر y کم می‌شود و در غیر اینصورت یک واحد از متغیر x کم شده و یک واحد به متغیرy اضافه می‌شود.

ساختار چند انتخابی switch

در برنامه نویسی گاهی به الگوریتمی نیاز پیدا می‌کنیم که در آن متغیری به ازای هر مقدار صحیح ثابتی، باعث اجرای یک دستور خاص شود و به ازای هر مقدار اعمال مختلف انجام پذیرد. برای نیل به این هدف C ساختار چندانتخابی switch راکه به صورت زیر است، دارا می‌باشد:

```
switch(عبارتی که مورد بررسی قرارگیرد)
{
case ۱ مقدار ثابت :
مجموعه دستورات۱
break;
case ۲مقدار ثابت :
مجموعه دستورات۲
break;
.
.
.
case nمقدار ثابت :
مجموعه دستوراتn
break;
default :
مجموعه دستورات حالت پیش فرض
}
```

حلقه های تکرار: ساختار تکرار به برنامه نویس این امکان را می دهد که برنامه، قسمتی از دستورات را تا هنگامی که شرط خاصی برقرار است تکرار کند.

ساختار while:

```
while(شرط)
{
دستورات ;
}
```

ساختار do/while: ساختار تکرار do/while مشابه ساختار تکرار while است. در ساختار تکرار do/while شرط در انتهای حلقه بررسی می‌شود(برعکس while).

```
do {
مجموعه دستورات
} while (شرط مورد نظر) ;
```

حلقه های For: ساختار تکرار for نیز مانند دو ساختار قبلی یک حلقه تکرار می سازد. از ساختار تکرار for معمولاً هنگامی که دفعات تکرار حلقه مشخص است استفاده می‌شود. ساختار تکرار for به صورت زیر می‌باشد:

```
for (مقدار ابتدای حلقه ; شرط پایان ; گام)
{
دستورات ;
}
```

دستور break: هرگاه در ساختار های do/while و while و for یا switch اجرا گردد، باعث خروج فوری برنامه از آن ساختار خواهد شد و برنامه اولین دستور بعد از آن ساختار را اجرا خواهد کرد.

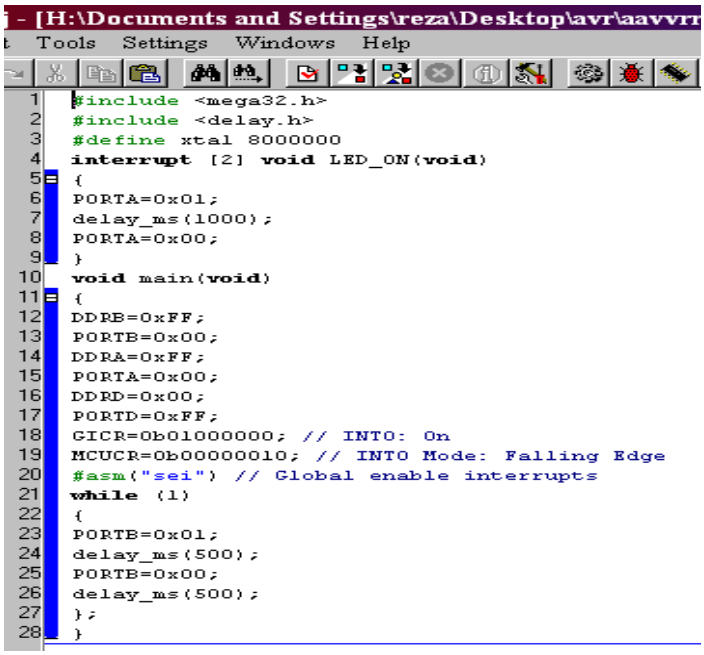
دستور continue: این دستور هرگاه در ساختار های do/while و while و for اجرا گردد دستورات بعدی آن ساختار نادیده گرفته می‌شود و بار بعدی حلقه تکرار اجرا می‌شود. مثال برنامه زیر مجموع اعداد ۱ تا ۲۰ به جز ۱۰ را محاسبه می‌کند.

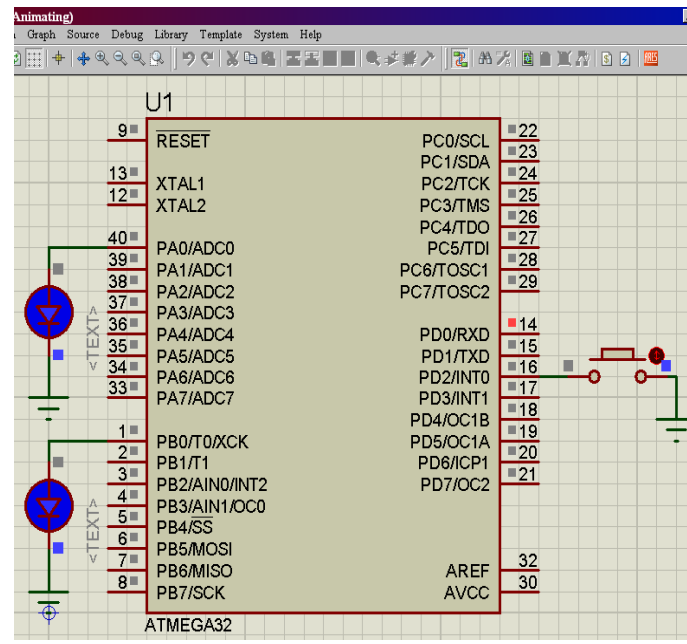
```
# include<isotream.h>
int main ( )
{
int n= 0, sum=0;
while(n<20)
{
++n; // n= n+1;
if(n==10)continue;
sum+=; // sum= sum +n;
}
Cout << "1+2+…(except 10)…+ 20 = " << sum << endl;
return0;
}
```

خروجی برنامه به صورت زیر می‌باشد.

```
1+2+…(except 10)… +20=200
```

مثال: برنامه ای بنویسید که بطور عادیPB.0را با تاخیر زمانی خاموش و روشن کند و با امدن اینتراپت صفرPA.0رابا تاخیریک زمانی روشن و خاموش کند:

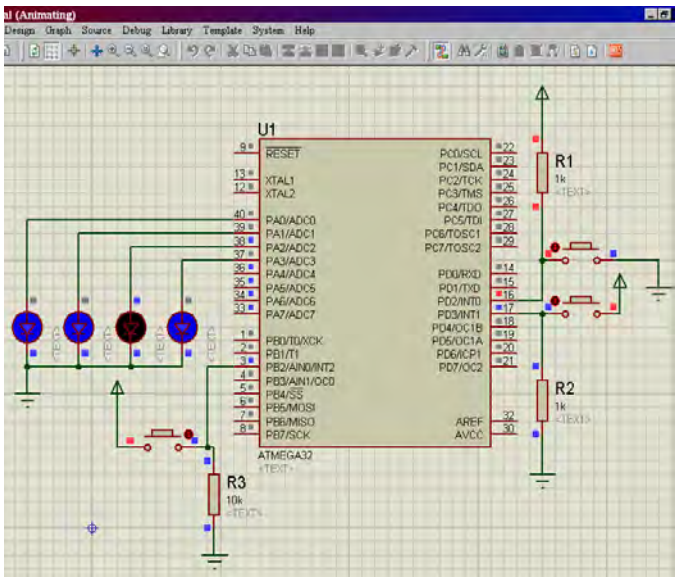




که اگر برنامه را به کمک codewizard ایجاد کرده باشیم باید در codewizardتنظیمات اینتراپت ها را بصورت شکل زیر انجام داد:

که بعد در برنامه ای که بازمیشود در قسمت whileبرنامه اصلی ودر قسمت اینتراپت برنامه سرویس وقفه را مینویسیم

مثال:




```
R IDE [1.11.9.0]
Program  Tools  Options  Window  Help
Label
e = "m32def.dat" : $crystal = 8000000
PORTA = Output
INT0 = Low Level :Config INT1 = Change :Config INT2 = Rising
Int0 : Enable Int1 : Enable Int2
Interrupts : On INT0 Q : On INT1 W : On INT2 Y
TA.0 : Waitms 50 : Reset PORTA.2 : Waitms 50
.1 = 0 Then : PORTA.1 = 1 : Else : PORTA.1 = 0
.2 = 0 Then : PORTA.2 = 1 : Else : PORTA.2 = 0
.3 = 0 Then : PORTA.3 = 1 : Else : PORTA.3 = 0
```

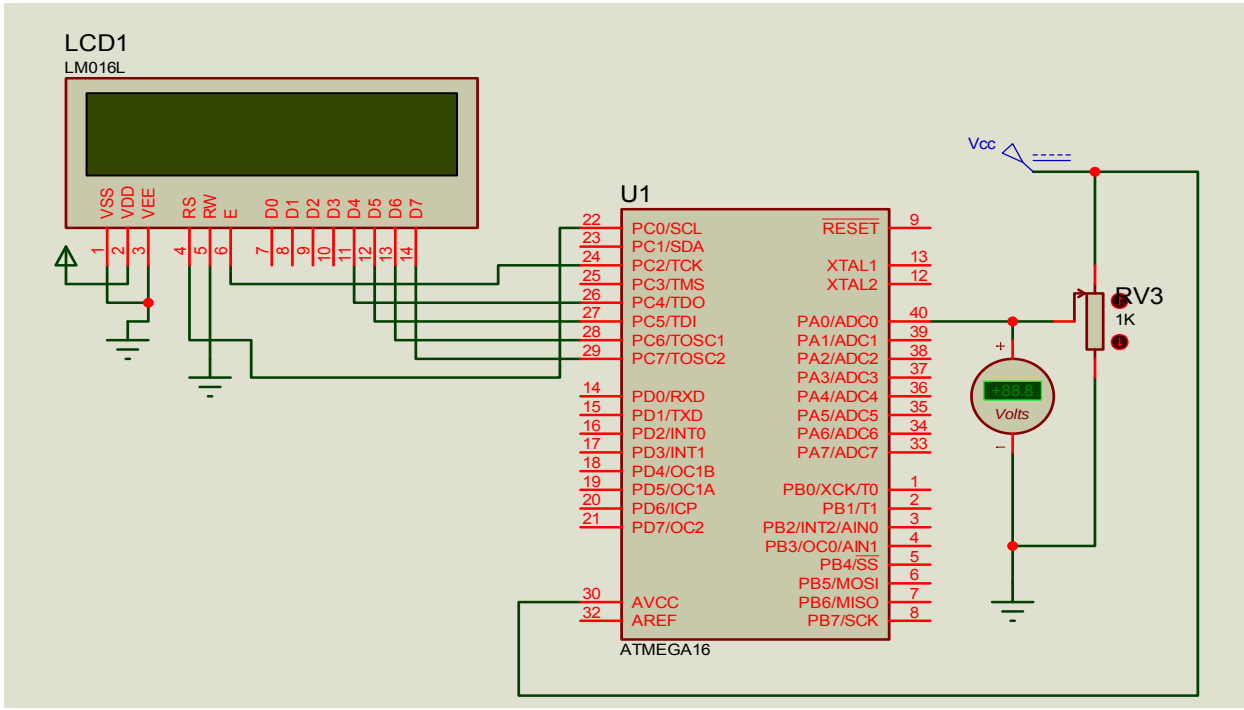
در برنامه بالا از۳ منبع وقفه int0,int1,int2استفاده شده است، همانگونه که میبینید به یکی از پایه ها یک پالس پایین رونده و به دیگران پالس بالا رونده اعمال می‌شوددر حلقه ی اصلی مدام یکی از پایه هاخاموش وروشن می‌شود، ۳ منبع وقفه مدام چک می‌شوند، هنگامی که پالس مشخص شده به پایه وقفه اعمال شد به زیر برنامه تعریف شده(qبرای منبع وقفه صفر و Wبرای منبع وقفه۱وYبرای منبع وقفه ۲) پرش می‌شود و وضعیت سهLEDتغییر می‌کند.مدار مورداستفاده رادر بالا مشاهده میفرمایید.
در صورتی که پین وقفه به صورت خروجی تعریف شود، آنچه روی پورت نوشته می‌شود می‌تواند باعث ایجاد وقفه شود، به این ترتیب می‌توان وقفه ی نرم افزاری ایجاد کرد

به این نکته نیز توجه شودکه اگر وقفه خارجی را به صورت حساس به سطح انتخاب کنید هر بارکه کنترل از روتین وقفه خارج شود یک دستور از بدنه برنامه اصلی اجرا شده و در صورتی که هنوز مقدار پایه مربوطه ۰باشد دوباره به روتین وقفه پرش می‌شود.

فیلتر کاهش نویز ورودی

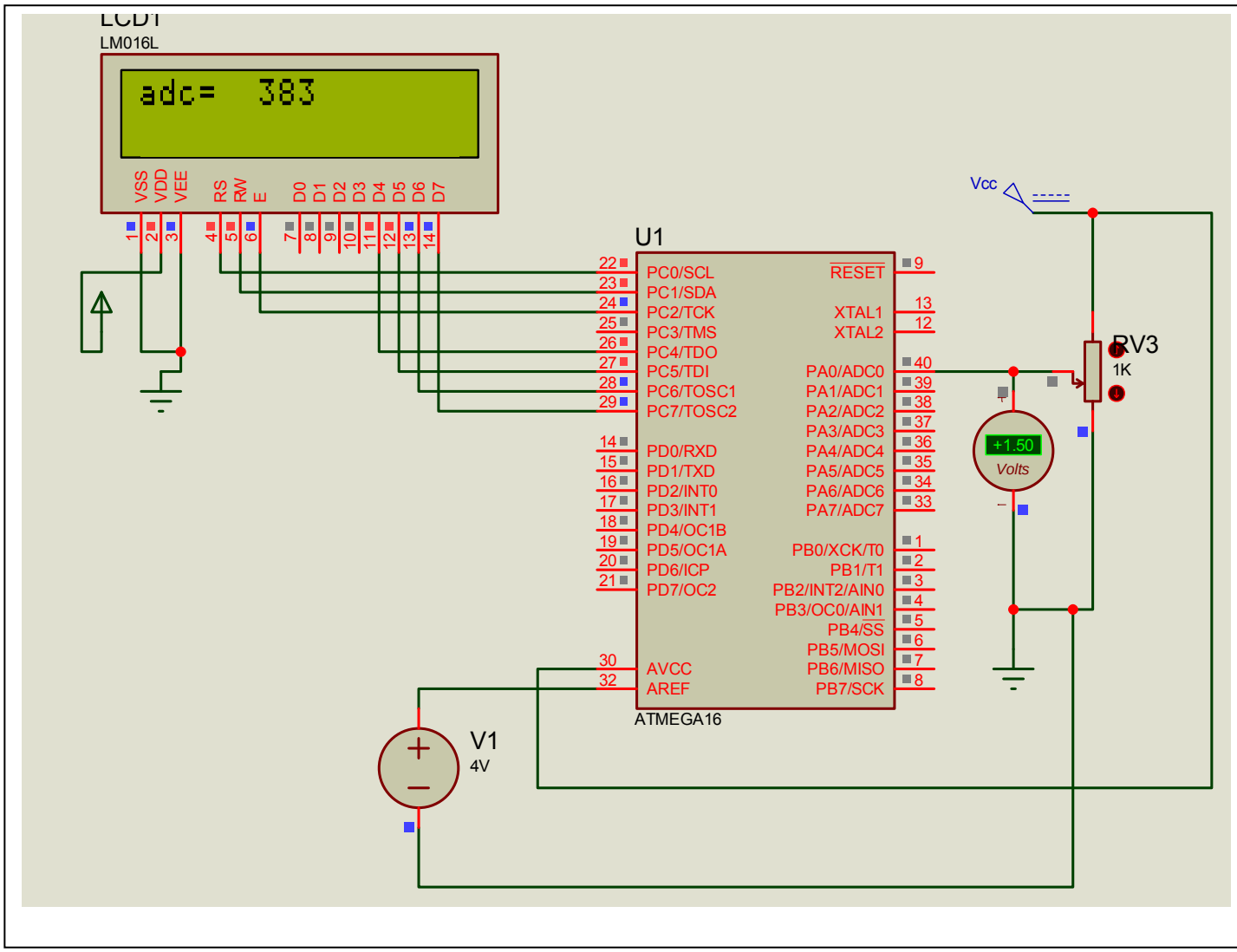
مدارات داخلی میکرو کنترلر با ایجاد نویز با عث کاهش دقت مقدار خوانده شده توسطADCمی‌شوند. برای بهبود این مشکل می‌توان در زمان تبدیل میکروکنترلر را به یکی ازمد های IDLEیاADC NOISE REDUCTION برد تا عملیات تبدیل بعد از خاموش شدن CPU انجام شود.

مثال: برنامه‌ای بنویسید که با استفاده از کانال صفر مبدل آنالوگ به دیجیتال یک مقدار آنالوگ را به دیجیتال تبدیل کرده و بر روی یک LCD که به پورت C متصل است نمایش دهد.




```
$regfile = "M16DEF.DAT"
$crystal = 12000000
Config Lcd = 16 * 2
Config Adc = Single , Prescaler = Auto , Reference = Avcc       'Internal ,  AVCC , OFF
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.2 , Rs =
Portc.0
Dim W As Word , Channel As Byte
Cursor Off
Locate 1 , 2
Lcd "ADC PROGRAM"
Cls
  Channel = 0
    Do
     Start Adc
     W = Getadc(channel)                          'read A/D value from channel 0
     Print "Channel " ; Channel ; " value " ; W
     Lcd W
     Waitms 200
     Cls
   Loop
End
```

۱ -برنامه‌ای بنویسید که یک ولتاژ آنالوگ بین صفر تا ۴ ولت را به عدد دیجیتال تبدیل کرده و بر روی LCD نشان دهد.

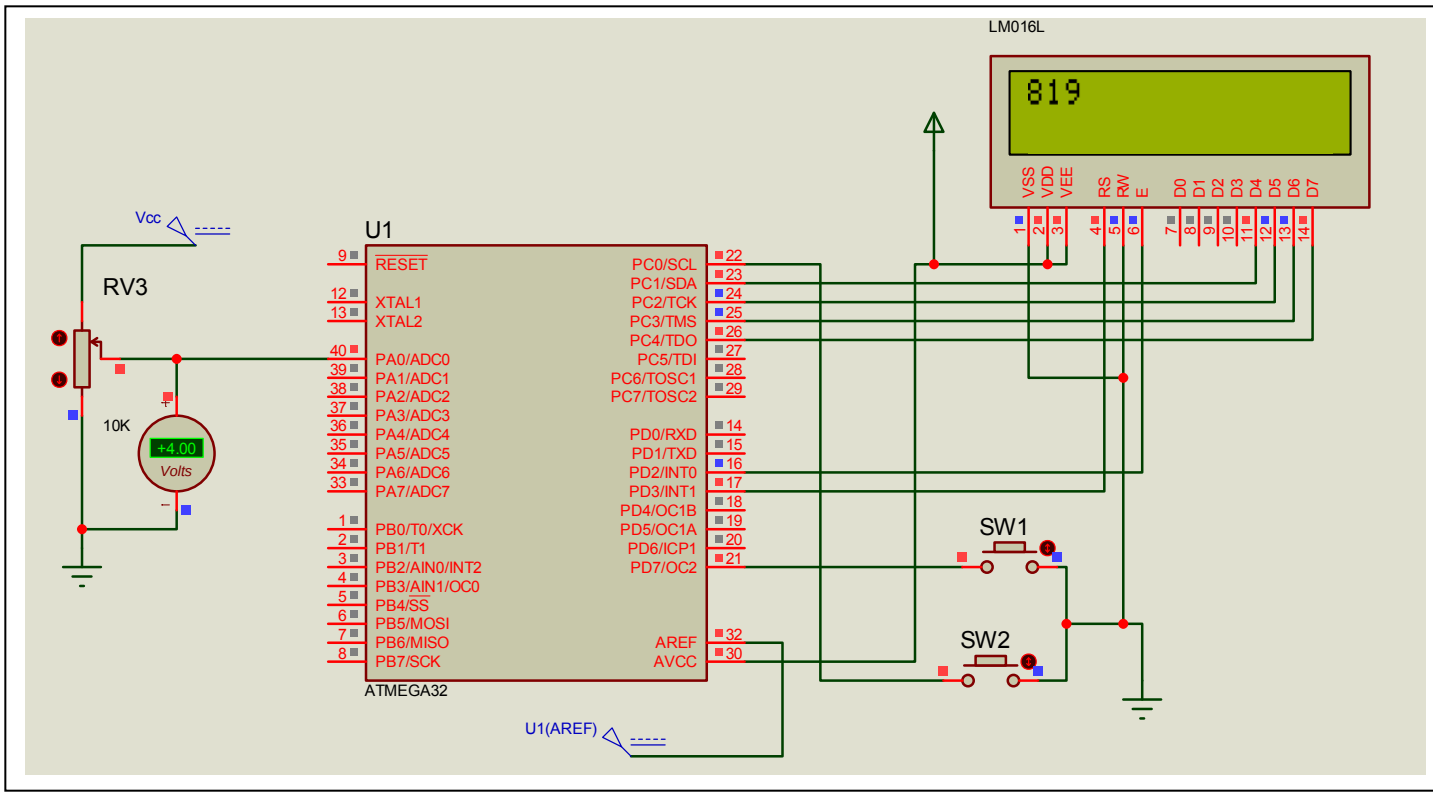


**********/

```
#include <mega16.h>
#include <stdio.h>
// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
#include <delay.h>
#define ADC_VREF_TYPE 0x00
// Read the AD e dconversion result
unsigned int read_adc(unsigned char adc_input)
```

```
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
delay_us(10);    // Delay needed for the stabilization of the ADC input voltage
ADCSRA|=0x40;    // Start the AD conversion
while ((ADCSRA & 0x10)==0);     // Wait for the AD conversion to complete
ADCSRA|=0x10;
return ADCW;
}
// Declare your global variables here
char lcd_buff[2];
unsigned int adc_in;
char msg[]="ADC Program";
void main(void)
{
// ADC initialization    // ADC Clock frequency: 750.000 kHz
// ADC Voltage Reference: AVREF   // ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE & 0xe0;
ADCSRA=0x84;
// LCD module initialization
lcd_init(16);
lcd_puts(msg);     // OR   lcd_putsf("ADC Program ");
delay_ms(100);
while (1)
     {
     adc_in=read_adc(0);
     sprintf(lcd_buff,"adc=%4d",adc_in);
     lcd_clear();
     lcd_gotoxy(0,0);
     lcd_puts(lcd_buff);
     delay_ms(50);
     };
}
```

۲ -برنامه‌ای بنویسید که در آن متناظر با فشرده شدن کلید sw1 تعداد بار فشرده شدن این کلید را تا آن لحظه بر روی LCD نمایش دهد و با فشرده شدن کلید SW2 مقدار ولتاژ آنالوگ متصل شده به کانال صفر را به دیجیتال تبدیل کرده و بر روی LCD نشان دهد.



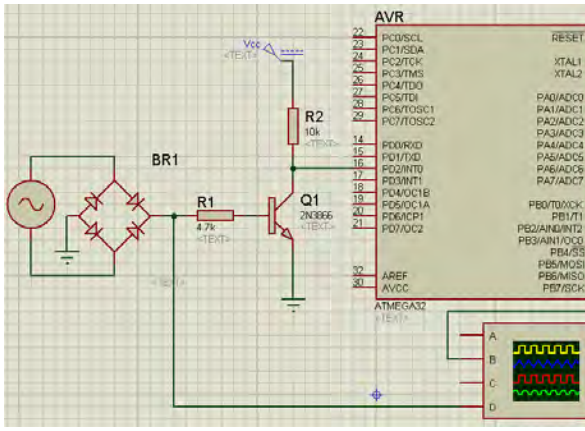

```
$regfile = "M32DEF.DAT"                    'mega32 used
$crystal = 8000000
Config Adc = Single , Prescaler = Auto , Reference = Off    'aaref as reference voltage
'internal or external crystal
Config Lcdpin = Pin , Db4 = Portc.1 , Db5 = Portc.2 , Db6 = Portc.3 , Db7 = _
Portc.4 , E = Portd.2 , Rs = Portd.3              'lcd connection
Config Lcd = 16 * 2
Config Debounce = 30                       'config debounce delay
Dim A As Word                             'dimension a
Dim B As Word
Dim Lsb_a As Byte                          'dimension lsb_A(use to keep lsb of a)
Dim Msb_a As Byte                          'dimension lsb_A(use to keep msb of a)
Ddrc.0 = 0: Portc.0 = 1                   'pinc.0 pull up resistor active
Ddrd.7 = 0: Portd.7 = 1                   'pinc.0 pull up resistor active
Declare Sub Increment                      'called when increment button pressed
Declare Sub Adc_display
Readeeprom Lsb_a , 10                      'read lsb of a from add 10 of eeprom
Readeeprom Msb_a , 11                      'read msb of a from add 11 of eeprom
A = Makeint(lsb_a , Msb_a)                 'attching lsb and msb due to make A
Cls                                'lcd cls
```

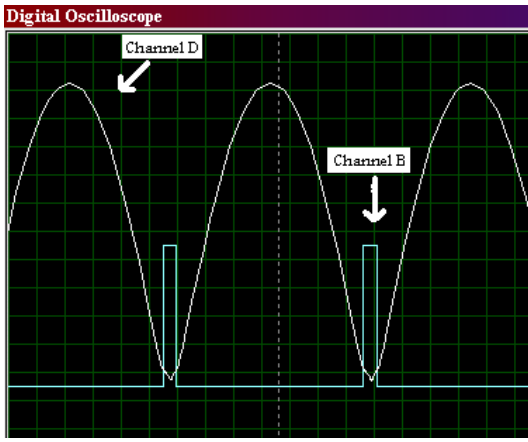
```
Lcd A                                   'show A
Do
   Debounce Pind.7 , 0 , Increment , Sub          'Read Push Button repeatedly
   Debounce Pinc.0 , 0 , Adc_display , Sub        'START ADC when this key is pressed
Loop
End                                     'end program


Sub Increment
  Clss
  Incr A
  cursor off
  Lcd A ; "  "
  Home
  Lsb_a = Low(a)                        'load lsb of A to variable lsb_a
  Writeeeprom Lsb_a , 10                'write lsb of a to add 10 of eeprom
  Msb_a = High(a)                       'load msb of A to variable msb_a
  Writeeeprom Msb_a , 11                'write lsb of a to add 11 of eeprom
  Waitms 4
End Sub
Sub Adc_display
  Start Adc
  B = Getadc(0)
  Cls
  cursor off
  Lcd B
  Waitms 50
End Sub
```

مثال: مدار اشکار ساز گذر از سطح صفر:

برنامه ای بنویسید که با رسیدن یک موج سینوسی یکسو شده با پل،به نقطه صفر PORTA.0 به مدت یک MS یک شود.

B






```
\Documents and Settings\reza\Desktop\1\a.c]
ols  Settings  Windows  Help
//Project : Zero Cross Detector
#include <mega32.h>
#include <delay.h>
#define xtal 8000000

interrupt [2] void switch_(void)
{
        PORTA=0x01;
        delay_ms(1);
        PORTA=0x00;
}
void main(void)
{
        DDRA=0xFF;
        PORTA=0x00;
        DDRD=0x00;
        PORTD=0xFF;

        GICR=0b01000000;     // INT0: On
        MCUCR=0b00000011;    // INT0 Mode: Rising Edge
        #asm("sei")          // Global enable interrupts

        while (1);
}
```

در این برنامه از اینتراپت خارجی صفر حساس به لبه بالا رونده استفاده شده است.درست زمانی که خروجی پل صفر میشوددر خروجی Trپالس مربعی بوجود می ایدکه INT0را که حساس به لبه بالارونده هست فعال می‌کند با لبه بالا رونده این پالس در portA.0نیز پالسی به مدت 1msتولید می‌شود.

پروتکل SPI

ارتباط سریال SPI یک پرتکل ارتباطی سریال سنکرون با سرعت بالاست که می‌تواند برای عملیاتی همچون پروگرام کردن حافظه هایFLASHوEEPROMو ارتباط میکروهای AVR با یکدیگر و یا با وسیله های دیگر که قابلیت ارتباط با این پرتکل را دارا هستند به کاربرده شود.این واسط برای فواصل کوتاه به کار می رود.واسط SPI در میکروکنترلر های AVR دارای خصوصیات زیر است:

- ارسال اطلاعات از طریق سه خط به صورت همزمان و تبادل دو طرفه (FULL- DUPLEX)
- عملکرد به صورت MASTER، SLAVE
- الویت در ارسال MSB یا LSB
- قابلیت انتخاب سرعت انتقال

پین های مورد استفاده برای این ارتباط به صورت جدول زیر است :

جدول ۱_۱_پ_پین های مورد استفاده در ارتباط SPI

PIN	کاربرد
MISO	ورودی داده MASTER و خروجی دادهSLAVE
MOSI	ورودی داده SLAVE و خروجی داده MASTER
SS	انتخاب MASTER یا SLAVE بودن در ارتباط دو میکرو
SCK	کلاک ورودی و خروجی
RESET	استفاده در پروگرام کردن تراشه

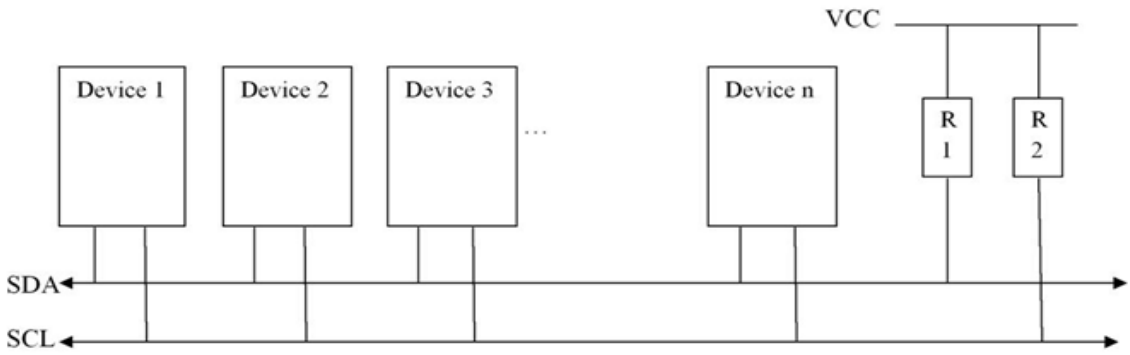
پروتکل ارتباط سریال I2C

رابط سریال دو سیمه (Two Wired Serial Interface) که در برخی مواقع به آن رابط I2C نیز می گویند اولین بار توسط شرکت Philips طراحی شد. این واسط از طریق دو سیم SDA,SCLکه یکی برای انتقال اطلاعات و دیگری برای انتقال پالس ساعت، با وسایل جانبی ارتباط برقرار می کند. از خصوصیات این رابط می‌توان به موارد زیر اشاره کرد:

۱) رابطی انعطاف پذیر و بسیار ساده و در عین حال قدرتمند که فقط نیاز به دو خط انتقال دارد.
۲) قابلیت پشتیبانی از مدهای Master وSlave در حالت های فرستده و گیرنده .
۳) سرعت انتقال اطلاعات تا 400KHz

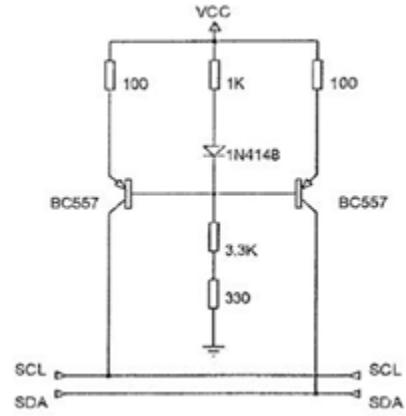
بررسی گذرگاه ارتباط دوسیمه

در ارتباط سریال دو سیمه یکی از خطوط برای پالس ساعت (SCL) و دیگری برای داده (SDA) می باشد هریک از خطوط (SDA,SCL) از خارج با یک مقاومت بالا کش (Pullup) به منبع تغذیه متصل می شوند. این پروتکل ارتباطی قابلیت شبکه شدن را نیز دارد. در شکل زیر چندین دستگاه که با هم شبکه شده اند نشان داده شده است. در این شکل دستگاه ها به حالت slave در باس قرار می‌گیرند.



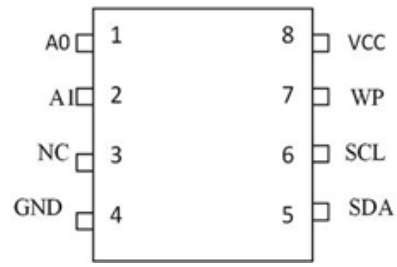
اتصال وسایل I2C در یک باس

در ساده ترین حالت خطوط I2C با مقاومت های 10K، PULL-UP می شوند تا خط را در سطح HIGH نگه دارند. ولی این حالت برای فواصل کوتاه مناسب است و در خطوط طولانی که دارای یک ظرفیت خازنی اند یک ثابت زمانی RC ایجاد می‌شود که جهت رفع آن به جای مقاومت از منابع جریان به شکل زیر استفاده می‌شود.



مدار ارتباطی وسایل I2C برای فواصل دور

کلاک ارتباط I2C بستگی به کلاک سیستم دارد و بالاترین فرکانس کلاک آن می‌تواند 400KHZ باشد.
برای ارسال داده، فرستنده اصلی، بیتها را به ترتیب از با ارزشترین بیت (MSB) تا کم ارزشترین بیت (LSB) روی خط داده قرار می‌دهد، سپس با ایجاد پالس ساعت روی خط SCL داده های فوق انتقال می یابند.
برای ارتباط میکرو با وسایل جانبی دیگر از طریق این پروتکل لازم است که پایه های SDA و SCL پیکر بندی شوند.
دستورا ت لازم برای کار با این پروتکل در محیط بسکام در کتاب میکروکنترلر های AVR تالیف مهندس علی کاهه موجود است.
یک EEPROM سریال با شماره 24C256 که با این پروتکل کار می کند در زیر معرفی می‌شود.



EEPROM سریال

جدول زیر پایه های این حافظه را معرفی می کند.

Pin Name	Function
A0 to A1	Address Inputs
SDA	Serial data
SCL	Serial Clock Input
WP	Write Protect
NC	No Connect

جهت ارتباط با این حافظه سریال از پروتکل I2C استفاده می‌شود.

معرفی پایه ها:

- **پایه کلاک سریال (SCL)**

از لبه مثبت SCL برای ورودی داده به هر یک از EEPROM ها و از لبه منفی برای خروج داده از EEPROM ها استفاده می‌شود.

- **پایه داده سریال (SDA)**

SDA یک پایه دو طرفه برای انتقال داده هاست. این پایه بصورت درین باز عمل می کند و باید از خارج با یک مقاومت بالاکش به VCC متصل شود.

- **پایه های آدرس سخت افزاری EEPROM (A0,A1)**

این پایه ها آدرس سخت افزاری EEPROM را مشخص می کنند.در صورتی که بخواهیم از چند حافظه EEPROM بر روی گذرگاه استفاده کنیم با توجه به نوع EEPROM تا هشت حافظه را می‌توان روی گذرگاه قرار داد. در این نوع تراشه می‌توان حداکثر چهار EEPROM را آدرس دهی کرد.

- **پایه حفاظت از نوشتن (WP)**

این پایه ورودی بوده و چنانچه آن را زمین کنیم عملیات خواندن و نوشتن به صورت عادی انجام می‌شود. اما اگر این پایه را یک کنیم عملیات نوشتن در EEPROM متوقف می‌شود.

این حافظه سریال (AT24C256) دارای 256kbit حافظه است که قابلیت ارتباط با پروتکل سریال (I2C) را دارد.

در صورتی که بخواهیم در حافظه بنویسیم بیت R/W را صفر و در صورتی که قصد خواندن داشته باشیم آن را یک قرار می دهیم . آدرس سخت افزاری نوشتن حافظه ای که پایه های A0 وA1 آن زمین شده است برابر &B10100000=160 است و آدرس سخت افزاری خواندن حافظه ای که پایه های A0 وA1 آن زمین شده است برابر &B10100001=161 است.

حافظه های سریال EEPROM

زمانی که نیاز به ارسال داده با سرعت بالا نباشد از حافظه های سریال که حجم کمی نسبت به حافظه های موازی اشغال می کنند،استفاده می گردد. تعداد پایه های کم حافظه سریال نسبت به حافظه موازی باعث می‌شود که پایه های بیشتری از میکرو آزاد بماند.

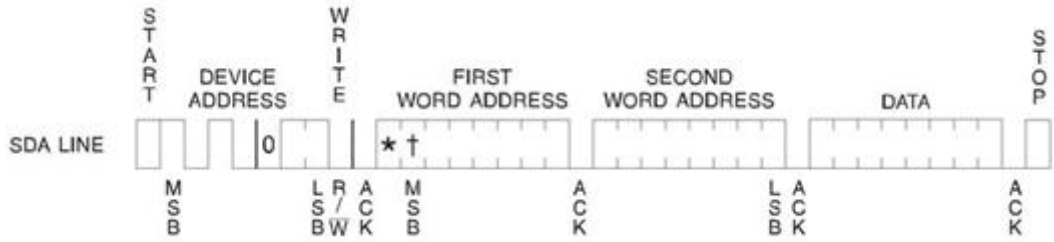
EEPROM های سریال برای ارتباط از دو سیم استفاده می کنند و به لحاظ تعداد پایه های کم، در پروژه های الکترونیکی از جایگاه ویژه ای برخوردار هستند.از مشخصات این حافظه ها می‌توان به موارد زیر اشاره کرد:

- مصرف کم و ولتاژهای کاری استاندارد
5.0 (VCC = 4.5V to 5.5V
2.7 (VCC = 2.7V to 5.5V)
1.8 (VCC = 1.8V to 3.6V)
- سازماندهی حافظه داخلی به طور مثال برای AT24C256 بصورت 8*16.384 و 8*32.768 می باشد.
- قابلیت ارتباط بصورت 2_WIRE
- دارای اشمیت تریگر و فیلتر ورودی جهت کاهش نویز
- قابلیت ارتباط دو طرفه
- ارتباط 1 MHZ در ولتاژ ۵ولت, 400 kHz در ولتاژ ۲/۷ ولت , 100 kHz در ولتاژ ۱/۸ ولت
- مد نوشتاری ۸ بایتی و ۶۴ بایتی
- دارای پایه حفاظت از نوشتن wp (write protect) برای استفاده نرم افزاری یا سخت افزاری
- نگهداری داده تا ۴۰ سال
- قابلیت نوشتن تا ۱۰۰۰۰۰ بار
- حد اکثر زمان نوشتن در ۱۰ میلی ثانیه
- قابلیت نوشتن تا ۱۰۰۰۰۰۰ بار و نگهداری اطلاعات تا ۱۰۰ سال

این وسایل برای کار در محیط های تجاری و صنعتی و در مکان هایی که استفاده از ولتاژ و توان های کم احتیاج باشد بهینه سازی شده اند.

۳_۲_پ_نوشتن در حافظه

عملیات نوشتن مطابق دیاگرام زمانی زیر انجام می‌شود:



- Start : ایجادیک start condition می کند که می‌توان با دستور I2CSTART آن را ایجاد کرد.
- Device address : منظور همان آدرس سخت افزاری است که توسط ارسال بایت آدرس دهی سخت افزاری برای نوشتن (۱۶۰) بدست آمده و با دستور I2CWBYTE address می‌توان آن را به خط یا باس SDA ارسال کرد .باید توجه داشت که بیت R/W=0 باشد .
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.
- ارسال ابتدای بایت بالای آدرس حافظه دلخواه توسط دستور I2CWBYTE .این بایت می‌تواند توسط دستور HIGH از یک متغیر دو بایتی بدست آید.
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.

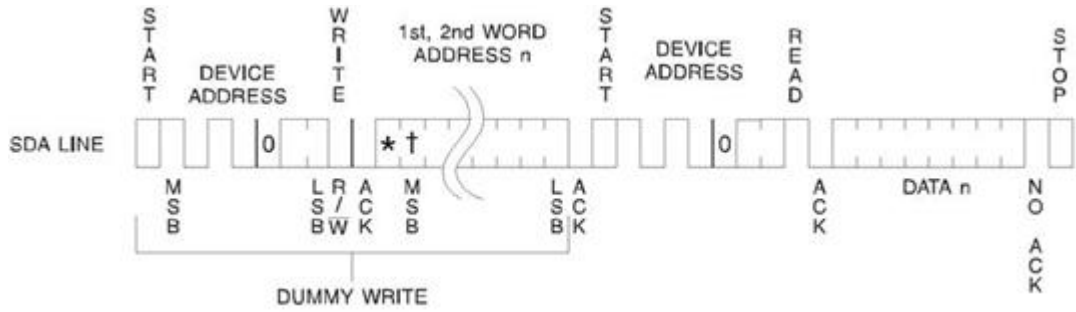
- ارسال بایت پایین آدرس حافظه دلخواه توسط دستور I2CWBYTE این بایت می‌تواند توسط دستور LOW از یک متغیر دو بایتی بدست آید.
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.
- ارسال داده به چیپ آدرس دهی (سخت افزاری) شده.
- Stop : ایجاد stop condition که توسط دستور I2CSTOP به خط یا باس SDA ارسال می گردد.
- ایجاد تاخیر توسط دستور WAITms به مقدار 5ms برای تکمیل نوشتن.

خواندن از حافظه

برای خواندن از حافظه به صورت بایتی باید مراحل زیر بر روی پایه SDA طی شود.

- ایجاد STARTCONDITION توسط دستور I2CSTART
- ارسال بایت آدرس دهی سخت افزاری برای
- DEVICE ADDRESS: منظور همان آدرس سخت افزاری است که توسط ارسال بایت آدرس دهی سخت افزاری برای نوشتن (۱۶۰) بدست آمده و با دستور I2CWBYTE address می‌توان آن را به خط یا باس SDA ارسال کرد .باید توجه داشت که بیت R/W=0 باشد .
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.
- ارسال ابتدای بایت بالای آدرس حافظه دلخواه توسط دستور I2CWBYTE .این بایت می‌تواند توسط دستور HIGH از یک متغیر دو بایتی بدست آید.
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.
- ارسال بایت پایین آدرس حافظه دلخواه توسط دستور I2CWBYTE این بایت می‌تواند توسط دستور LOW از یک متغیر دو بایتی بدست آید.
- ACK : در تمام موارد مورد نیاز این سیگنال توسط کامپایلر به صورت خودکار ایجاد شده و نیازی به نوشتن برنامه اضافه برای آن نیست.
- ایجاد STARTCONDITION مجدد توسط دستور I2CSTART
- ارسال بایت آدرس دهی سخت افزاری برای خواندن (۱۶۱) توسط دستور I2CWBYTE که توسط بایت آدرس دهی سخت افزاری بدست آمده است.
- دریافت داده از چیپ آدرس دهی (سخت افزاری) شده برای خواندن توسط دستور I2CRBYTE data,NACK
- ایجاد STOP CONDITION توسط دستور I2CSTOP

شکل زیر مراحل خواندن از حافظه سریال رانشان می‌دهد.



توابع USART در codevision

اعلان این توابع در فایلstdio.h قراردارد و قبل از استفاده از آن باید تنظیمات اولیه USART انجام شود.

تابع( )getchar : این تابع به روش سرکشی(polling) منتظر می ماند تا پرچم RXC یک شده و بعد از آن مقدار UDR را می خواند.

تابع( )putchar : این تابع به روش سرکشی(polling) منتظر می ماند تا پرچمUDRE یک شده و بعد ازآن یک کاراکتر را درUDR کپی می کند.

مثال: برنامه ای بنویسید که در آن میکروی دوم به روش سرکشی(polling) منتظر دریافت یک کاراکتر از سوی میکروی اول بماند و میکروی اول پس از گذشت ۳ ثانیه عدد ۷ را برای میکروی دوم ارسال کند و میکروی دوم کاراکتر دریافت شده را پس از دریافت در پورت A بریزد.

برنامه ی میکرو اول:

```
#include <mega32.h>
#include<delay.h>
#include<stdio.h>
#define xtal 8000000
Void main(void)
{
UCSRA=0x00;
UCSRB=0x08;    //usart  transmitter : on
UCSRC=0X86;    //8 DATA,1 STOP, NO parity
UBRRH=0x00;
UBRRL=0X33;    //usart baud rate= 9600
      delay_ms(3000);
      Putchar(7);
      While(1);
}
```

برنامه میکرو دوم:

```
#include <mega32.h>
#include<stdio.h>
#define xtal 8000000
Void main(void)
{
```

```
Char a;
DDRA=0XFF;      //port A as output
PORTA=0XFF;    //all pins of port A are high
UCSRA=0X00;
UCSSRB=OX10;   //USART receiver: on
UCSRC=0X86;     // 8 DATA, 1 STOP, NO PARITY
UBRRH=0X00;
UBRRL=0X33;    //USART BAUD RATE=9600
a=getcher();        //polling  Untill receive is complete
PORTA=a;
While(1);
}
```

توابع سطح بالای USART تنها برای صرفه جویی در وقت برنامه نویس می باشد. در صورت نیاز می توان به صورت مستقیم با رجیسترها کار کرد. به عنوان مثال حلقه ی اصلی برنامه ی میکرو دوم را به صورت زیر میتوان نوشت:

```
While(!(UCSRA&0X80));
PORTA=UDR;
While(1);
```

توابع ( )puts و( )putsf : این توابع یک رشته را توسط USART به پورت سریال ارسال می کنند. تابع ()puts رشته ای را که در SRAM است و تابع ()putsf رشته ای را که در flash است به خروجی ارسال می کند. اساس تمام توابع سطح االيUSART توابع ()putchar و ()getchar است.

مثال: برنامه ای بنویسید که رشته های" KAVOSH و"AVR" را به پورت سریال ارسال کند.

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000
Flash char string1[7]="kavosh";
Char string2[7]="avr";
Void main(void
{
UCSRA=0X00;
UCSRB=0X08;  //USART TRANSMITTER: ON
UCSRC=0X86; //8 DATA,1 STOP, NO PARITY
UBRRH=0X00;
UBRRL=0X33; //USART BAUD RATE= 9600
Putsf(string1);
Puts(string2);
While(1);
}
```

تابع ( )gets : این تابع دارای دو آرگومان نام متغیر و طول است که منتظر می ماند تا کاراکتر های دریافتی به اندازه ی طول مورد نظر شده و سپس آنها را داخل متغیر می‌ریزد.

تابع( )printf: این تابع یک رشته ی قالب بندی شده را به پورت سریال ارسال می کند.کاراکتر فرمت می‌تواند مطابق جدول زیر باشد:

کاراکتر فرمت	نوع اطلاعات
%c	یک کارکتر
%d و%1	عدد صحیح علامتدار در مبنای ۱۰
%E و%e	عدد اعشاری به صورت نماد علمی
%F	عدد اعشاری
%S	رشته ی ختم شده به null واقع درSRAM
%P	رشته ی ختم شده به null واقع در FLASH
%u	عدد صحیح بدون علامت در مبنای ۱۰
%X و%x	عدد صحیح بدون علامت در مبنای۱۶
%%	کاراکتر٪

مثال : یک رشته ی قالب بندی شده را به پورت سریال ارسال می‌کند.

```
#include <mega16.h>
#include <delay,h>
#include < stdio.h >
#define xtal 8000000
Int a=100;
Char b='A';
Float pi=3.14;
Void main (void)
{
UCSRA=0X00;
UCSRB=0X08;//usart transmitter: on
Ucsr c=0x86;//8 data, 1stop. No parity
Ubrr h =0x00;
UBRRL=0X33;//USART baud rate:9600
Print f("a=%d b=%c pi=%f" ,a ,b ,pi );
While (1);
}
Isr_int0:
P0rta=&H01
Waitms100
Port a=&H00
Return
```

محیط برنامه نویسی BASCOM

آشنایی با محیط BASCOM برای برنامه نویسی میکروهای AVR

بدنه یک برنامه بیسیک در محیط BASCOM شامل موارد زیر است. تعیین میکروی مورد استفاده، تعیین فرکانس کریستال استفاده شده، تعریف متغیرها، پیکره بندی (مانند پیکره بندی پورتها، تایمرها، LCD، اینتراپتها و...)، نوشتن برنامه (کد) مورد نظر، مشخص کردن پایان برنامه

۱- معرفی میکرو: برای شروع برنامه در محیط BASCOM ابتدا باید نوع میکرو معرفی شود.

$REGFILE = VAR

در دستور فوق VAR نام چیپ مورد استفاده است که به عنوان مثال می‌تواند شامل موارد زیر باشد.

```
$REGFILE = "Attiny2313. dat"     'ATtiny2313 MCU
$REGFILE = "8535def. dat"        'AT90s8535 MCU
$REGFILE = "M32def. dat"         'MEGA 32 MCU
```

علامت $ در ابتدای دستور نشان می‌دهد که این دستور از دستورات مربوط به کمپایلر است.

۲- معرفی فرکانس کریستال: برای مشخص کردن فرکانس کریستال استفاده شده، از دستور زیر استفاده می‌شود.

$CRYSTAL = X

در این دستور، X فرکانس کریستال استفاده شده بر حسب هرتز است. به عنوان مثال:

```
$CRYSTAL =  14000000          '14MHZ external osc
$CRYSTAL =  8000000           '8MHZ external osc
```

۳- تعریف متغیرها:

DIM var as [XRAM/SRAM/ERAM] data_type [at location] [overlay]

دستور فوق متغیرهایی را که در برنامه استفاده می‌شود تعریف می‌کند. Var نام متغیری است که در برنامه به کار برده می‌شود. در صورت استفاده از حافظه جانبی آنرا با XRAM مشخص کنید SRAM را زمانی انتخاب کنید که می‌خواهید متغیر را در حافظه SRAM قرار دهید و ERAM متغیر مورد نظر را در EEPROM داخلی قرار می‌دهد. Data_type نوع داده است که می‌تواند طبق جدول زیر باشد.

Data_type	Store as
BIT	بیت
BYTE	بایت (عدد ۸ بیتی بدون علامت)
WORD	عدد ۱۶ بیتی بدون علامت
INTEGER	عدد ۱۶ بیتی علامتدار
LONG	عدد ۳۲ بیتی علامتدار
SINGLE	عدد ۳۲ بیتی علامتدار
STRING	صفر تا ۲۵۴ بایت

(SRAM پیش فرض RAM داخلی میکرو است). در صورت استفاده از متغیر STRING، بیشترین طول آن نیز باید ذکر شود. گزینه اختیاری OVERLAY متغیر تعریف شده را بصورت Pointer در نظر گرفته و فضایی را برای متغیر در نظر نمی‌گیرد.

نکته: اعداد هگزادسیمال را با علامت &H و اعداد باینری را با علامت &B و اعداد دسیمال با علامت & مشخص می‌شوند.

مثال:

متغیر S از نوعSTRING با طول حداکثر ۱۰ کاراکتر

```
DIM S AS STRING*12
S = " Hello World"
PRINT S
```

متغیر W از نوع دو بایتی بدون علامت

```
DIM W AS WORD
W=20000
Dim k as integer at 100
K = -15
Dim H as Xram byte at &H300
```

۴- یادداشت (اختیاری)[۲]: گاهی نیاز است برای اطلاعات بیشتر، یادداشتهایی در برنامه اضافه کنیم که این کار با استفاده از دستور زیر انجام می‌شود.

REM یا ‘

یادداشتها و نوشته های بعد از دستور فوق غیرفعال بوده و کامپایل نخواهند شد و به رنگ سبز در می‌آیند.

مثال:

```
Dim c as byte          REM c is the number of students  or  ,c is the number of students
Print c                'send c to serial port
```

۵- آدرس شروع برنامه ریزی حافظهFlash (اختیاری):

گاهی نیاز است که برنامه خود را از آدرس دلخواهی به بعد در حافظه داخلی میکرو قرار دهید. به این منظور بصورت زیر عمل کنید.

$Romstart = Address

که در آن Address مکانی از حافظه است که برنامه HEX از آن آدرس به بعد در حافظه داخلی میکرو قرار می‌گیرد. در صورتی که از این دستور استفاده نشود، کمپایلر بصورت خودکار برنامه را از آدرس 0000H حافظه (FLASH ROM) به بعد قرار خواهد داد.

مثال:

```
$Romstart = &H1000    REM place hex code from 1000 hex  address
```

۶- تقسیم فرکانسی کلاک (اختیاری):

با دستور زیر می‌توان در بعضی میکروهای سری MEGA AVR مانند MEGA103 یا MEGA603 فرکانس کلاک را بصورت نرم افزاری بر یک عدد مشخص تقسیم کرد.

2 - Comment

Clockdivision = var

Var مقادیر معتبر بین اعداد ۲ تا ۱۲۸ را دارد. توسط دستور فوق کلاک اصلی میکرو بر var تقسیم شده و سپس توسط میکرو استفاده می‌شود.

مثال:

```
Clockdivision = 4     REM divide clock by 4
```

۷- دستور CONST:

برای تعریف یک بایت ثابت از این دستور استفاده می‌شود.

CONST SYMBOL = NUMCONST
CONST SYMBOL = STRINGCONST
CONST SYMBOL =EXPRESSION

مثال:

```
CONST S = “TEST”
CONST A =  52
CONST B = &B10010
CONST C = (B*3) + 2
CONST D = SIN(1)
```

۸- دستور ALIAS:

برای تغییر نام متغیر استفاده می‌شود.

مثال:

```
LED  ALIAS PORTB. 3
SET LED          Rem set PORTB. 3 or turn on LED
```

۹- اسمبلی و بیسیک (اختیاری): برای نوشتن برنامه اسمبلی در بین برنامه بیسیک از دستور زیر استفاده می‌شود.

$ASM
ASSEMBLY PROGRAM
$END ASM

مثال:

```
Dim  c as byte
Loadadr c،x          'load address of variable c into register x
$asm                  'start assembly program
Ldi R24،1             'load  register R24 with  the constant 1
St x،R24              'store 1 in variable c
$END ASM
Print c               'send c to serial port
End
```

تعیین پایان برنامه:

END

دستور END که در انتهای برنامه قرار می‌گیرد اجرای برنامه را متوقف و تمام وقفه‌ها را غیرفعال می‌کند.

مثال:

```
Print “hello”   REM print this
END             REM end program execution and disable all interrupt
```

۱۰- پیکره بندی AVR:

این پیکره بندی شامل پیکره بندی پورتها، تایمرها، LCD، اینتراپتها، صفحه کلید و... است که بصورت زیر می‌باشد.

۱۰-۱- پیکره بندی پورتها

برای آنکه پورتی را به عنوان ورودی یا خروجی مشخص کنیم باید آن پورت را پیکره بندی کنیم. جهت هر کدام از پایه های پورت می‌تواند ورودی یا خروجی تعریف شود.

Config portx = state
Config pinx. y = state

x و y بسته به میکرو می‌توانند به ترتیب پایه های ۰ تا ۷ پورتهای G،F،E،D،C،B،A برای میکروهای مختلف باشند.

در AVR برای هر پورت سه رجیستر اختصاص داده شده است. مثلاً برای پورت A رجیسترهای DDRA (که جهت پورت را تعیین می‌کند بطوریکه DDRA = 0x00 برای ورودی و DDRA = 0xFF برای خروجی کردن پورت است)، PINA (که هنگام نوشتن در پورت استفاده می‌شود (مثلاً خواندن صفحه کلید متصل شده به پورت A) و PORTA که هنگام خواندن از پورت استفاده می‌شود (مثلاً ارسال DATA از پورت A به LED). به همین علت **زمانی که بخواهید از پورتی بخوانید بایستی از رجیستر PIN پورت مربوطه استفاده کنید و در هنگام نوشتن در پورت بایستی در رجیسترPORT پورت مربوطه بنویسید** (به مثال زیر دقت کنید).

مثال:

```
Config portd = input        'configure port D  for input mode
Config PORTB = output       'configure port B  as output port
K = PIND                    ' Read port D and place it in K variable
Wait pind. 5،RESET              ' تا زمانیکه بیت پنجم پورت دی صفر است صبر کن
Incr PORTB                      ' یک واحد به پورت بی اضافه کن
Set PORTB. 4                    ' بیت چهارم پورت بی را یک کن
```

۱۰-۲- پیکره بندی تایمر/کانترها:

پیکره بندی تایمر/ کانتر صفر بصورت زیر است.

CONFIG TIMER0 = TIMER، PRESCALE = 1|8|64|256|1024 'configure AS timer
CONFIG TIMER0 = COUNTER، EDGE = RISING/FALLING 'configure AS counter

تایمر صفر بالا شمار بوده و قابلیت خواندن و نوشتن را دارد(هم مقدار شمارش را می‌دهد و هم می‌توان مقدار اولیه به آن داد. با دستور var = timer0/counter0 می‌توان محتوای آنرا خواند و با دستور timer0/counter0 = value یا دستور TCNT0 = value می‌توان به آن مقدار دهی اولیه کرد). با دستور Start timer0 تایمر شروع به شمارش کرده و با دستور

Stop Timer0 از شمارش متوقف می‌شود. پس از رسیدن شمارش شمارنده به عدد FF پرچم سرریز OVF0 یک شده و می‌توان با استفاده از وقفه عملیات مورد نظر را انجام داد.

```
Enable Interrupts        ' Global interrupts should be enable
Enable Timer0            ' Enable Timer0 Interrupt
On  ovf0 timer0_isr   یا   On Timer0  timer0_isr
DO
     'your program goes here
Loop
End

Timer0_isr:
   Print  " interrupt service routine "
Return
```

در پیکره بتدی بصورت کانتر، شمارش با لبه بالارونده یا پایین رونده انجام شده و اعمال لبه مورد نظر به پایه T0 باعث می‌شود که محتوای رجیستر TCNT0 یا متغیر COUNTER0 یک واحد افزایش یابد و با رسیدن شمارش به عدد به FF پرچم سرریز OVF0 یک شود. محتوای کانتر صفر را می‌توان با دستور VAR = COUNTER0 خواند.

امکان تنظیم تایمر بصورت CTC (Clear Timer on compare match) نیز وجود دارد.

تایمر یک، یک تایمر ۱۶ بیتی است که دارای دو واحد مقایسه مستقل با خروجیهای OC1A و OC1B است. شامل ۴ رجیستر ۱۶ بیتی TCNT1، OCR1A، OCR1B، ICR1 و دو رجیستر کنترلی ۸ بیتی TCCR1A و TCCR1B (معرف Prescaler کلاک و چند بیتی بودن PWM) است. رجیستر پرچم TIFR و رجیستر چشم پوشی از اینتراپت TIMSK نیز بصورت مشترک با سایر تایمرها استفاده می شوند. تایمر یک نیز همانند تایمر صفر می‌تواند کلاک خود را از کلاک سیستم، تقسیمی از کلاک سیستم و یا از پایه T1 دریافت کند. همواره محنوای رجیسترهای OCR1A و OCR1B با محتوای TCNT1 مقایسه شده و در صورت تساوی خروجی OC1A یا OC1B فعال می‌شود. رجیستر ICR1 می‌تواند محتوای تایمر یک را تصرف کند در صورتی که نحریکی خارجی به پایه ICP1 وارد شود یا از طریق برنامه ریزی فیوز بیت دوم رجیستر ACSR که مستقیماً خروجی مقایسه کننده آنالوگ را به ICP1 متصل می‌کند این تحریک صورت گیرد.

۱۰-۳- پیکره بندی صفحه کلید:

اسکن صفحه کلید در BASCOM کار ساده ای است. کافی است صفحه کلید را به یکی از پورتهای میکرو وصل کرده و توسط CONFIG KBD آنرا پیکره بندی کنید (البته صفحه کلید ۴*۴).

Config KBD = PortX، Debounce = Value [، Delay = value]

PortX مشخص کننده پورتی است که صفحه کلید به آن متصل است و Debounce مدت زمانی که بعد از فشرده شدن کلید اگر هنوز کلید فشرده باشد آنرا معتبر می‌داند (بصورت پیش فرض 20msec است و بیشترین مقدار آن می‌تواند ۲۵۵ باشد). Delay نیز پارامتری اختیاری است و مشخص کننده تأخیر بر حسب میلی ثانیه است و مدتی است که بعد از معتبر بودن یک کلید تأخیر می اندازد و توسط دستور ()getkbd کلید را می‌خواند (معمولاً جهت کاهش نویزهای محیط از Delay=100 استفاده می‌شود).

زمانی که صفحه کلید ۶ سطر داشته باشد از دستور زیر استفاده می‌شود.

Config KBD = portx، debounce = value، rows = 6، row5 = pinx. y، row6 =pina. b

که در این حالت سطر پنجم به پایه y از پورت x و سطر ششم به پایه b از پورت a وصل شده است.

مثال:

Config KBD = portc، debounce = 50، rows = 6، row5 = pind. 6، row6 =pind. 7

با توجه به این پیکره بندی سطر پنجم و ششم صفحه کلید به ترتیب به پایه های ۶ و ۷ از پورت D متصل شده‌اند.

دستور ( )GETKBD:

Source = getkbd( )

توسط این دستور، میکرو صفحه کلید را خوانده و عدد متناظر با کلید فشرده شده را در متغیر Source قرار می‌دهد. این دستور زمانی که کلیدی فشرده نشده است عدد ۱۶ را بر می‌گرداند.

تذکر: نحوه اتصال کیبورد کامپیوتر به میکرو را می‌توانید در صفحه ۲۱۲ کتاب آقای علی کاهه چاپ سیزدهم ببینید.

۱۰-۴- پیکره بندی LCD:

برای استفاده از LCD ابتدا باید نوع آنرا (تعداد سطر و ستون LCD را) مشخص کنید. برای تعیین نوع LCD می‌توانید از دستور CONFIG LCD = LCDtype استفاده کنید که در آن LCDtype حاصلضرب تعداد ستون و سطر LCD است که می‌تواند 1*16، 2*16، 4*16، 2*20، 4*20، 4*40و... باشد. پیش فرض میکرو برای نوع LCD نوع 2*16 است.

مثال:

```
Config  LCD = 40*4
LCD "Hello"              ' display on LCD
Fourthline               ' select  line 4
LCD "4"                  ' display 4
End
```

پس از مشخص کردن نوع نمایشگر باید مشخص کنید که پایه های نمایشگر به کدام پایه های میکرو متصل می‌شود(هر پایه LCD می‌تواند به هر پایه ای از پورتها متصل شود). پایه های LCD برای اتصال به میکرو با استفاده از دستور CONFIG LCDPIN و بصورت زیر پیکره بندی می‌شوند.

Config LCDPIN = PIN، DB4=PN،DB5=PN، DB6=PN،DB7=PN، E = PN، RS=PN

PN پایه ای دلخواه از میکرو است که پایه LCD به آن اتصال می‌یابد به عنوان مثال PORTB. 7

در ساده ترین حالت، LCD توسط شش خط به میکرو متصل می‌شود که شامل ۴ پایه دیتا و دو پایه کنترلی E(Enable) و RS(Row Select) است. پیکره بندی پایه های LCD باید در یک خط و یا ادامه آن با علامت (Underline) _ در خط بعد نوشته شود.

مثال:

```
Config  LCDPIN = PIN، DB4=PORTB. 4،DB5=PORTB. 5، DB6=PORTB. 6، DB7= PORTB. 7، E =_
PORTB. 3، RS=PORTB. 2
```

پایه های LCD با توجه به پیکره بندی بالا باید بصورت جدول زیر متصل شوند.

LCD DISPLAY	LCD PIN	PORT
DB7	14	PORTB. 7
DB6	13	PORTB. 6
DB5	12	PORTB. 5
DB4	11	PORTB. 4
E (ENABLE)	6	PORTB. 3
R/W	5	GROUND
RS	4	PORTB. 2
VO (CONTRAST)	3	GROUND OF POT (5K)
VDD	2	VCC
VSS	1	GROUND

تعیین باس LCD: انتقال داده به LCD بصورت ۴ بیتی یا ۸ بیتی است که توسط دستور زیر این معرفی صورت می‌گیرد.

```
Config  LCDBUS = 8
```

بصورت پیش فرض، انتقال داده به LCD بصورت ۴ بیتی است و به همین خاطر، در انتقال ۴ بیتی نیاز به تعیین باس نیست.

دستورات و توابع مربوط بهLCD:

دستورLCD: جهت نمایش پیغام روی LCD استفاده می‌شود. هم متغیرها و هم ثابتها را می‌توان با این دستور نمایش داد.

دستورCLS: باعث پاک شدن صفحه LCD می‌شود.

دستورLCD ON/OFF: با این دستور صفحه LCD را روشن و خاموش می‌کنیم.

دستورCURSOR ON/OFF،CURSOR BLINK/NOBLINK : مکان نما را روشن/خاموش، چشمک زن یا چشمک نزن قرار می‌دهد. پیش فرض روشن و چشمک زن است.

دستورHOME: مکان نمای LCD را به سطر و ستون اول می برد.

Home U/L/T/F مکان نما به ترتیب در اولین ستون سطر اول، دوم، سوم و چهارم قرار می‌گیرد

دستورLOCATE: تعیین مکان قرارگیری مکان نما در سطر X و ستون Y **(Locate x ، y)** بنابراین X از یک تا ۴ و Y از ۱ تا ۶۴ می‌تواند متغیر باشد.

دستورShift Cursor left/right: مکان نما را یک واحد به چپ یا یک واحد به راست شیفت می‌دهد.

دستورShift LCD left/right: کل صفحه نمایش به راست یا به چپ شیفت داده می‌شود.

دستورLower Line: این دستور مکان نما را به سطر پایین تر می‌فرستد.

دستورUpper Line: این دستور مکان نما را به سطر بالاتر می‌فرستد.

دستورThird Line: این دستور مکان نما را به سطر سوم می برد.

دستورFourth Line: این دستور مکان نما را به سطر چهارم می برد.

۱۰-۵- پیکره بندی اینتراپتها:

AVR دارای وقفه های متعددی است که شامل ۱- وقفه های خارجی شامل INT0، INT1 و در بعضی میکروها INT2 تا INT7 ۲- وقفه تایمرها/کانترها ۳- وقفه های ADC، EEPROM و مقایسه کننده آنالوگ ۴- وقفه پورت سریال که شامل وقفه دریافت کامل داده توسط UART (URXC)، وقفه ارسال کامل داده توسط UART (UTXC)، وقفه خالی بودن رجیستر داده UART (UDRE) می باشد.

Config INTX = نام نوع فعال شدن وقفه

نام نوع فعال شدن وقفه شامل LOW LEVEL (اعمال سطح منطقی صفر باعث رخ دادن INTX می‌شود)، FALLING (اعمال یک لبه پایین رونده باعث وقوع INTX می‌شود) و RISING (اعمال یک لبه بالا رونده باعث رخ دادن INTX می‌شود).

دستور Disable برای غیرفعال کردن وقفه ها و **دستور Enable** جهت فعال کردن وقفه ها استفاده می‌شود.

برای فعال کردن وقفه مربوط به هر دستگاه، علاوه بر فعال کردن وقفه مربوطه، باید کل وقفه ها نیز باید فعال شود به عنوان مثال برای فعال کردن وقفه مربوط به اینتراپت صفر باید بنویسیم:

```
Enable interrupts
Enable INT0
```

تذکر: در AVR وقفه ها اولویت ندارند و هر کدام زودتر نوشته شوند اولویت با آن است.

دستور On Interrupt: با این دستور می‌توان مسیر اجرای برنامه را به روتین مربوط به اینتراپت هدایت کرد. به عنوان **مثال:**

```
Enable interrupts
Enable timer0
On timer0 timer0_isr   OR    On OVF0 timer0_isr
```

به معنی این است که چنانچه اینتراپت مربوط به تایمر صفر رخ دهد، اجرای برنامه متوقف شده و پردازشگر سراغ اجرای روتین مربوط به این تایمر (Timer0_isr) می رود و بعد از بازگشت، اجرای برنامه ادامه پیدا می‌کند.

تذکر: با پرش به یک وقفه، بقیه وقفه ها غیرفعال می شوند تا زمانی که از آن وقفه خارج شویم. در انتهای روتین مربوط به وقفه، جهت بازگشت به برنامه اصلی باید از عبارت Return استفاده شود.

تذکر: در صورتی که از عبارت No Save استفاده شود، مقادیر رجیسترهای وضعیت R0 تا R31 در برنامه وقفه تغییری نمی‌کند ولی چنانچه این عبارت نوشته نشود، تمام رجیسترهای استفاده شده در برنامه وقفه تغییر می‌کنند.

مثال: برنامه ای برای وقفه خارجی صفر بنویسید که چنانچه یک لبه پایین رونده به پایه INT0 وارد شود، عبارت BASCOM روی LCD نوشته شود.

```
Config INT0 = Falling
Enable interrupts
Enable INT0
```

```
On INT0 int0_isr No Save
Do
Loop
End
int0_isr:
print “BASCOM”
RETURN
```

۱۰-۶- پیکره بندی تایمر نگهبان:

در محیط BASCOM با نوشتن دستور CONFIG WATCHDOG = TIME این تایمر پیکره بندی شده و با دستور START WATCHDOG شروع به کار می‌کند. مقادیر معتبر برای TIME عبارتند از ۱۶، ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۱۰۲۴ و ۲۰۴۸ میلی ثانیه.

با دستور STOP WATCHDOG تایمر نگهبان متوقف شده و با دستور RESET WATCHDOG ری‌ست می‌شود.

مثال:

```
$BAUD = 1200
CONFIG WATCHDOG = 2048          ' RESET after 2048 msec
Start watchdog                  ' start the watchdog timer
Dim I As word
For I=1 TO 10000
Print I                         ' print value
Next
RESET watchdog
```

توجه داشته باشید که چون زمان اجرای حلقه از ۲۰۴۸ میلی ثانیه بیشتر است لذا حلقه بطور کامل اجرا نشده و میکرو ری‌ست می‌شود.

۱۰-۷- پیکره بندی ADC:

مبدل A/D برای میکروهایی که ADC دارند (مثلاً ATMEGA32) با دستور زیر پیکره بندی می‌شود.

```
CONFIG ADC = SINGLE | FREE، PRESCALER = AUTO، REFERENCE = OPTIONAL
```

مبدل داخلی میکرو ۱۰ بیتی بوده و نیاز به کلاکی بین ۵۰ تا ۲۰۰ کیلوهرتز دارد که این کلاک از تقسیم کلاک سیستم تأمین می‌شود. زمانی که مد SINGLE انتخاب شود، از دستور Var = GETADC(channel) جهت خواندن استفاده شود که در آن channel شماره کانال ADC است که می‌تواند بین صفر تا ۷ باشد.

PRESCALER = AUTO باعث می‌شود که کامپایلر با توجه به فرکانس اسیلاتور، بهترین (بالاترین) کلاک را برای ADC تعیین کند.

REFERENCE = OPTIONAL گزینه‌ای اختیاری برای ولتاژ مرجع (در بعضی میکروها از جمله MEGA8) است. در حالت کلی A/D ولتاژ آنالوگ بین AGND تا AREF را به عدد 0000000000B تا عدد 1111111111B تبدیلمی‌کند. OPTIONAL می‌تواند گزینه‌های OFF، AVCC و INTERNAL را داشته باشد. OFF به معنی خاموش کردن ولتاژ

مرجع داخلی و استفاده از ولتاژ خارجی موجود بر روی پایه AREF به عنوان ولتاژ مرجع است. به عنوان مثال اگر در این حالت، ولتاژ ۴ ولت به پایه AREF متصل شود A/D ولتاژ آنالوگ بین صفر ولت تا ۴ ولت را به عدد 0000000000B تا عدد 1111111111B تبدیلمی‌کند. گزینه AVCC وقتی است که ولتاژ پایه AVCC به عنوان ولتاژ مرجع در نظر گرفته شود. در این حالت، A/D ولتاژ آنالوگ بین صفر ولت تا AVCC ولت را به عدد 0000000000B تا عدد 1111111111B تبدیلمی‌کند. اگر OPTIONAL بصورت INTERNAL انتخاب شود، ولتاژ مرجع داخلی 2. 56V به عنوان ولتاژ مرجع در نظر گرفته شده و در این حالت، A/D ولتاژ آنالوگ بین صفر ولت تا 2 .56 ولت را به عدد 0000000000B تا عدد 1111111111B تبدیلمی‌کند. استفاده از این گزینه برای میکروهایی که ولتاژ مرجع داخلی ندارند تأثیری ندارد.

نکته: توسط دستورات START ADC، مبدل A/D شروع به نمونه برداری از سیگنال آنالوگ کرده و توسط STOP ADC تغذیه از ADC قطع شده و عملیات تبدیل پایان می‌یابد. این دستورات برای شروع و توقف ADC لازم اند.

۱۰-۸- پیکره بندی مقایسه کننده آنالوگ:

۱۱- دستور CHR:

برای تبدیل یک متغیر عددی به یک ثابت یا یک کاراکتر استفاده می‌شود. هنگام نمایش یک کاراکتر بر روی LCD می‌توان از این دستور استفاده کرد. دستور PRINT CHR (var) کاراکتر اسکی متغیر var را به پورت سریال ارسال می‌کند.

مثال:

```
Dim a as byte      Rem a is variable
A=65               ,, assign variable
Print a            Rem print value 65
Print hex(a)       Rem print hex value 65 = 41
Print chr(a)       Rem print asscii character 65 = A
```

۱۲- دستور INCR (DECR):

INCR (DECR) Var

توسط این دستور یک واحد به متغیرعددی Var اضافه (کم) می‌گردد.

مثال:

```
Dim A as byte      Rem a is variable
A=0                ,, assign variable
Print  A           Rem print 0
INCR A
Print  A           Rem print 1
```

۱۳- دستور LCASE (UCASE):

Target = LCASE(source) or Target = UCASE(source)

توسط این دستور تمام حروف رشته مورد نظر تبدیل به حروف کوچک (بزرگ) می شوند به عبارت دیگر تمام حروف رشته source کوچک (بزرگ) شده و در Target قرار می‌گیرند.

مثال:

```
Dim s as string*12، t1 as string*12 ،t2 as string*12
s = ”Hello Hasan”
t1 = Lcase(s)
t2 = Ucase(s)
print s          Rem print Hello Hasan
print t1         Rem print hello hasan
print t2         Rem print HELLO HASAN
end
```

۱۴- دستور LEN:

Var = LEN(String)

توسط این دستور، طول یا به عبارتی تعداد کاراکترهای رشته String در متغیر Var قرار می‌گیرد. رشته String نهایت طول ۲۵۵ بایت را دارد. فضای خالی در داخل رشته نیز یک کاراکتر به حساب می‌آید.

مثال:

```
Dim s as string*12
Dim a as byte
s = “test”
a=Len(s)
print a          Rem print 4
s= “  test”
a=Len(s)         Rem print 6
print a
```

۱۵- دستور SWAP:

SWAP Var1،Var2

با اجرای این دستور محتوای متغیر Var1 در متغیر Var2 و محتوای متغیر Var2 در متغیر Var1 قرار می‌گیرد. متغیر Var1 و متغیر Var2 باید از یک نوع باشند.

مثال:

```
Dim a as integer، b as integer
A=1:  b=2          ,,assign two integers
Swap a،b           ,, swap them
Print a            ,,prints 2
Print b            ,,prints 1
end
```

۱۶- دستور ROTATE:

ROTATE Var،LEFT/RIGHT [،shifts]

با اجرای این دستور تمام بیتها به چپ یا راست منتقل می شوند ولی هیچ بیتی به بیرون فرستاده نمی‌شود. گزینه shifts که اختیاری است تعداد چرخش بیتها را مشخص می‌کند که در صورت قید نکردن آن بصورت پیش فرض مقدار یک برای آن فرض می‌شود.

مثال:

```
Dim a as byte
A=128                    ,,assign variable
Rotate a،left،2          ,, swap them
Print a                  ,,prints 2
```

توابع ریاضی و محاسباتی

مشابه همه زبانهای برنامه نویسی توابع مربوط به جمع، تفریق، ضرب، تقسیم، کوچکتر، بزرگتر و.... و همچنین عملیات منطقی and، or، xor، not در محیط bascom پیش بینی شده است. برخی توابع ریاضی دیگر عبارتند از:

۱۷- تابع ABS:

Var1 = ABS(Var2)

قدر مطلق متغیر Var2 در متغیر Var1 قرار می‌گیرد.

مثال:

```
Dim a as integer، b as integer
a = -120                 ,,assign variable
b = abs(a)               ,, b = |a|
Print b                  ,,prints 120
end
```

۱۸- تابع EXP:

Target = exp (source)

در دستور فوق Target برابر است با e به توان source. Target متغیری از نوع داده single است.

مثال:

```
Dim a as single
a = exp(1. 1)            ,,assign variable
print a                  ,,prints 3. 004164931
```

۱۹- تابع LOG10:

Target = log10 (source)

لگاریتم مبنای ۱۰ متغیر یا ثابت source در متغیر Target قرار می‌گیرد. source و Target هر دو از نوع داده single هستند.

مثال:

```
Dim s1 as single، s2 as single
S1=0. 01
```

```
S2=log10(s1)
print s2
        for s1=1 to 100
              s2=log10(s1)
              print s1 ;  ;s2
        next
end
```

۲۰- تابع LOG:

Target = log (source)

لگاریتم طبیعی (Ln) متغیر یا ثابت source در متغیر Target قرار می‌گیرد. source و Target هر دو از نوع داده single هستند. این تابع برای اجرا وقت زیادی می برد و وقتی اعداد بزرگ باشند دقت پایین می‌آید.

مثال:

```
Dim s1 as single
S1=log(100)
print s1                ; print 4. 605170
```

۲۱- تابع RND:

Var = RND (limit)

دستور فوق یک عدد تصادفی بین صفر و limit را در متغیر Var بر می‌گرداند. با هر بار استفاده از این دستور عدد مثبت تصادفی دیگری بدست خواهد آمد.

مثال:

```
Dim I as single
 Do
   I = Rnd( 100)       ;get random number
   Print I
   Wait 1
 Loop
End
```

۲۲- توابع SIN و COS و TAN و SINH و COSH و TANH:

Var = SIN (source)

دستور فوق نسبت مثلثاتی مورد نظر متغیر source را در متغیر Var بر می‌گرداند. source و Var هر دو از نوع داده single هستند.

مثال:

```
Dim s1 as single ،s2 as single
Const pi=3. 141592
S1= pi/2
S2 = cos(s1)
Print  s2                ;prints cos(pi/2)=-0. 000006613
```

۲۳- توابع ASIN و ACOS و ATN و ATN2:

Var = ASIN (source)

دستور فوق Arcsin یک ثابت یا متغیر از نوع single را به رادیان بر می‌گرداند. تابع ATN2 بصورت ATN2(x،y) دو آرگومان ورودی x و y را به عنوان مختصات نقطه مورد نظر در صفحه دریافت کرده و بسته به ربعی از دایره که نقطه در آن قرار دارد جواب را در متغیر Var بر می‌گرداند.

مثال:

```
Dim s1 as single ،x as single، y as single، s2 as single
S1 = atn(1)*4
Print s1
X=-0. 5: y=-0. 5
S2=atn2(x،y)
Print  s2                ;prints arctang(-0. 5/-0. 5) = - 2. 35618 rad = -135 deg
```

۲۴- توابع DEG2RAD و RAD2DEG:

Var = DEG2RAD (single)

دستور فوق درجه را به رادیان (رادیان را به درجه) تبدیل کرده و جواب را در متغیر Var بر می‌گرداند.

مثال:

```
Dim s1 as single ،s2 as single
S1 =deg2rad(180)
Print s1                 ' 3. 141592498 will print
S2 =rad2deg(3. 141592498)
Print s2                 ' 179. 999984736 will print
```

۲۵- تابع ROUND:

Var = ROUND (x)

دستور فوق متغیر یا داده x از نوع single را روند کرده و در متغیر var از نوع single قرار می‌دهد.

مثال:

```
Round (2. 3) = 2 ، round(2. 8) = 3
Round (-2. 3) = -2 ، Round (-2. 8) = -3
```

۲۶- دستور HEX و HEXVAL:

Var1 = HEX(x) ، Var2 = HEXVAL(y)

دستور فوق داده x را به مقدار هگزادسیمال آن تبدیل کرده و در متغیر var1 قرار می‌دهد. همچنین دستور بعد داده هگزادسیمال y را به مقدار دسیمال آن تبدیل کرده و در متغیر var2 قرار می‌دهد.

مثال:

```
Dim a as byte ،s1 as string*10 ،s2 as string*10
a =123
s1=hex(a)
s2=  hexval(s1)
print s1              ' 7B will print
print s2              ' 123 will print
```

۲۷- دستور MAKEBCD و MAKEDEC:

Var1 = MAKEBCD(x) ، Var2 = MAKEDEC(y)

دستور فوق داده x را به مقدار BCD اش تبدیل کرده و در متغیر var1 قرار می‌دهد. دستور بعدی داده y را به مقدار دسیمالش تبدیل کرده و در متغیر var2 قرار می‌دهد.

مثال:

```
Dim a as byte ،b as byte، s2 as string*10
a =65
b=makebcd(a)
lcd b                 '101 will show
```

۲۸- دستور DO ... LOOP:

DO

Your task

LOOP until condition

تا زمانیکه condition فراهم باشد، Your task اجرا خواهد شد.

مثال:

```
Dim B as byte
B =0
DO
    B = B +1
    Print "B = " ; B
    Wait 2
LOOP until B = 10
```

حلقه فوق در هر بار اجرا یک واحد به متغیر B اضافه می‌کند و آنرا روی LCD نمایش می‌دهد. تا زمانی که مقدار B به ۱۰ نرسد این حلقه تکرار می‌شود.

اگر condition (شرط) وجود نداشته باشد، حلقه تا بی نهایت اجرا می‌شود و معادل while(1) در زبان C است.

زیربرنامه و تابع:

تابع هم مقدار ورودی(ارسالی) و هم مقدار برگشتی دارد اما زیربرنامه فقط مقدار ورودی دارد و مقدار برگشتی ندارد. زیر برنامه و تابع باید حتماً در ابتدای برنامه (با دستور Declare) معرفی شوند سپس در طول برنامه با دستور Call فراخوانی شوند. برای فراخوانی می‌توان از دستور استفاده نکرد و فقط نام زیربرنامه یا تابع را نوشت و در صورت داتن متغیر ورودی و خروجی نام آنها را داخل پرانتز بنویسیم.

معرفی تابع:

Declare function test1 [([BYVA/BYREF] invar as type1)] As typ2

test1 نام تابع مورد نظر است که type1 نوع متغیر ورودی به تابع و typ2 نوع متغیر برگشتی از تابع است که می‌توان STRING، LONG،WORD،INTEGER،BYTE باشد. انتقال داده بصورت BYVAL باعث می‌شود که یک کپی از متغیر به تابع فرستاده شود و در محتوای آن هیچ تغییری ایجاد نشود ولی در حالت BYREF آدرس متغیر ارسال و تغییرات در آن تأثیر می گذارد و ممکن است مقدار اولیه آن تغییر کند. بصورت پیش فرض داده بصورت BYREF فرستاده می‌شود.

مثال:

```
Declare function test1 ( byval I as integer، s as string ) As integer
Dim k as integer
Dim z as string*10
Dim t as integer
K = 5
Z = “abc”
T = test1(K،Z)
Print t
End
function test1 ( byval I as integer، s as string ) As integer
 local P as integer
P = I
I = 10
P =VAL(s) +I
Test1 = P
End test1
```

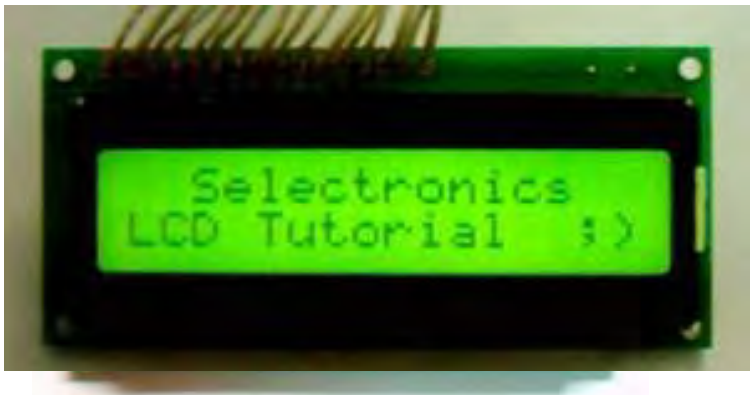
توضیحاتی در مورد LCD

LCD ها ابزاری برای نمایش اطلاعاتی هستند که شامل حروف و اعداد و همچنین برخی کاراکترهای گرافیکی می‌شود. بطور معمول در تجربیات اولیه برای نمایش اطلاعات دیجیتال از نمایشگر های هفت قسمتی (seven segment) استفاده می‌شودکه این نمایشگرها فقط ارقام (۰ تا ۹) و بعضی حروف مثل A b C را بصورت نه چندان زیبا نمایش می‌دهند. اما با بکار گیری LCD اطلاعات را بصورت زیبا و کاملتر می‌توان نمایش داد. البته استفاده از LCD برای مدارات ساده توصیه نمی‌شودو عموما آنرا همرا با میکروکنترلر یا CPU ها بکار می برند.

چیزی که از آن بعنوان LCD یاد می‌شوددرواقع یک صفحه نمایشگر LCD مانند صفحه ماشین حساب است که همراه با آی سی کنترلر و مدارهای جانبی اش و عموما با لامپ پشت صفحه در یک بسته پیش ساخته عرضه می‌شود.

LCD دارای یک کنترلر است که با ارسال اطلاعات به آن، این اطلاعات را در صفحه ای که عموما به چند سطر و ستون تقسیم شده نمایش می‌دهد. مثلا برای نمایش حرف "M" کافیست کد اسکی این حرف را طبق یک پروتکل ساده به LCD ارسال کنیم. همچنین می‌توان دستوراتی از قبیل پاک کردن صفحه نمایش، جابجایی مکان نما، خاموش روشن کردن مکان نما و غیره را نیز به LCD ارسال کرد.

LCD ها از طریق مقدار اطلاعاتی که می‌توانند در صفحه نمایش بدهند انتخاب و خریداری می‌شوند. انواع معمول آن عبارتند از ۱۶، ۲۰، ۳۲ و ۴۰ کاراکتر در هر خط و دارای ۱ یا ۲ یا ۴ سطر. مثلا ۲ در ۱۶ یعنی صفحه دارای دو خط و هر خط ۱۶ کاراکتر است. همچنین LCD مورد نظر می‌تواند همراه با لامپ پشت صفحه (Back light) یا بدون آن انتخاب شود. LCD ها، کاراکتر ها را در ماتریس های pixel 5*8 نمایش می‌دهند. در تصویر زیر یک نمونه ۲ در ۱۶ مشاهده می‌شود:



نمای جلویی LCD


نمای پشتی LCD

LCD ها دارای ۱۶ پایه هستند که ۸ خط آن مربوط به فرستادن یا خواندن داده ها یا دستورالعمل ها می‌باشد. پایه های دیگر خطوط کنترل و ولتاژهای تغذیه می‌باشند. لیست کامل پایه های LCD بصورت زیر است:

شماره و نام خط	عملکرد
1- Vss	زمین
2- V_{DD}	ولتاژ ۵ ولت برای کنترلر
3- V_{EE}	ولتاژ تنظیم درخشندگی(contrast)
4- RS	انتخابگر ثبات دستور / داده
5- R/W	انتخابگر خواندن / نوشتن
6- Enable	فعال کننده
7-14 Bus	8خط گذرگاه داد یا دستور
15	ولتاژ ۵ ولت برای لامپ پشت صفحه
16	زمین برای لامپ پشت صفحه

V_{EE}: برای تنظیم درخشندگی کاراکترها بکار می‌رود که باید ولتاژی بین صفر تا ۵ ولت به این پایه اعمال نمود. برای بیشترین درخشندگی این پایه را به زمین متصل کنید.

انتخابگر ثبات داده / دستور (**پایه RS**) مشخص می‌کند که آنچه به LCD فرستاده شده است داده یا دستور است. اگر این خط صفر باشد کنترلر LCD بایت موجود روی خطوط ۷ تا ۱۴ را بعنوان یک دستور تلقی کرده و اگر این پایه یک باشد اطلاعات را بعنوان یک کد اسکی که باید کاراکتر معادل آنرا نمایش دهد در نظر می‌گیرد. در LCD های گرافیکی این پایه با عنوان C/D(Code/Data) مشخص می‌شود.

انتخابگر خواندن / نوشتن (**پایه R/W**) جهت اطلاعات را نشان می‌دهد. اگر این پایه صفر باشد اطلاعات به LCD ارسال می‌شودو اگر یک باشد عمل خواندن از LCD صورت می‌گیرد.

فعال کننده (**پایه Enable**): برای هر دستور یا داده ای که به LCD میفرستیم یا میخواهیم از آن بخوانیم باید یک پالس پائین رونده (یعنی تغییر از سطح یک به صفر) را به این پایه اعمال کنیم تا دستور یا داده بوسیله کنترلر LCD پردازش شود.

در خطوط داده ی ۷ تا ۱۴، خط ۷ کم ارزشترین بیت (LSB) و خط ۱۴ پر ارزش ترین بیت (MSB) می‌باشد.

در صورت تمایل به روشن کردن لامپ پشت صفحه ولتاژ ۵ ولت را به پایه ۱۵ اعمال و پایه ۱۶ را به زمین متصل می‌کنیم.

برای آزمایش می‌توان LCD را به پورت چاپگر متصل و اطلاعاتی را به آن ارسال نمود. در این حالت بطور معمول خطوط داده پورت به خطوط ۷ تا ۱۴ و سه خط کنترلی به پایه های ۴ تا ۶ اتصال داده می‌شوند. توجه داشته باشید که ولتاژ تغذیه و لامپ پشت صفحه LCD توسط منبع خارجی تامین می‌شود.

روش فرستادن یک کاراکتر:

۱- خط خواندن نوشتن را صفر کنید تا نوشتن انتخاب شود.

۲- خط داده / دستور را یک کنید تا داده انتخاب شود.

۳- کد اسکی کاراکتر مورد نظر را روی خطوط D0 تا D7 قرار دهید.

۴- خط انتخاب را ابتدا یک و سپس صفر کنید. حداقل ۴۵۰ نانو ثانیه باید این خط را صفر نگه دارید تا داده پردازش شود. بعد از آن حالت خط تاثیری نخواهد داشت.

LCD گرافیکی:

پیکر بندی T6963 Graphical Lcd display:

برای راه اندازی LCD گرافیکی از پیکربندی زیر استفاده می کنیم.پیکربندی LCD بر اساس چیپ T6963C که در اکثر LCD های گرافیکی استفاده می‌شود،طراحی شده است.

pin=pin,RESET=pin,rd=pin,wr=pin,cd=port,ce=port,controlport=type,Dataport=Config graphLcd

mode=pin,mode=,fs

نوع (Type) که می‌توانید انواع ۱۲۴*۱۲۸،۱۲۸*۶۴،۱۲۸*۲۴۰ و۱۲۸*۲۴۰ باشد.برای LCD های نوع SED به طور مثال از 64SED*128 استفاده نمائید.

DATAPORT: مشخص کننده ی پورتی است که به عنوان ورودی داده LCD استفاده می‌شود. به طور مثال PORTA=DATAPORT که در این صورت پایه های $D_0 - D_7$ از LCD به ترتیب به پایه های PORTA.0-PORTA.7 متصل می‌شود.

CONTROLPORT: مشخص کننده ی پورتی است که از پایه های برای کنترل LCD استفاده می‌شود.

CE(CHIP ENABLE): شماره ی پایه ای است که برای فعال کردن چیپ موجود در LCD استفاده می‌شود.به طور مثال اگر PORTC=CONTROLPORT باشد،0=CE به معنای اتصال PORTC.0 به پایه CE از LCD می باشد.

CD(DATA/CODE): شماره ی پایه ای است که برای کنترل کردن پایه CD موجود در LCD استفاده می‌شود. به طور مثال PORTC=CONTROLPORT باشد،CD=1 به معنای اتصال PORTC.1 به پایه ی CD از LCD می باشد.

WR(WRITE): شماره پایه ای است که برای کنترل کردن پایه WR موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،WR=2 به معنای اتصال PORTC.2 به پایه ی WR از LCD می باشد.

RD(READ): شماره پایه ای است که برای کنترل کردن پایه RD موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،RD=3 به معنای اتصال PORTC.3 به پایه ی RD از LCD می باشد.

FS(FONT SELECT): شماره پایه ای است که برای کنترل کردن پایه FS موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،FS=4 به معنای اتصال PORTC.4 به پایه ی FS از LCD می باشد.

RESET: شماره پایه ای است که برای کنترل کردن پایه RESET موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،RESET=5 به معنای اتصال PORTC.5 به پایه ی RESET از LCD می باشد.

MODE: مشخص کننده ی تعداد ستون متنی LCD است که می‌تواند ۸ یا ۶ باشد.زمانی که از عدد ۶ استفاده می نمائید نهایتاً 6/PIXEL-X ستون متنی خواهید داشت و زمانی که از عدد ۸ استفاده نمائید نهایتاً 8/PIXEL-X ستون متنی خواهید داشت.

این گزینه همچنین مشخص کننده ی نوع فونت است.

دستورات کار با LCD گرافیکی:

دستور CLS: این دستور تمام صفحه نمایش LCD چه قسمت متنی و چه قسمت گرافیکی را پاک می‌کند .

دستور CLS GRAPH: این دستور فقط قسمت گرافیکی را پاک می‌کند .

دستور CLS TEXT: این دستور فقط قسمت متنی را پاک می‌کند .

دستور LCD: این دستور برای نوشتن متن بر روی LCD استفاده می‌شود. این دستور همانند دستور LCD،برای LCDهای ماتریسی عادی عمل می‌کنند .

دستور:PSET X,Y,COLOR: این دستور یک PIXEL را در مختصات (X,Y) به ازای COLOR=0 خاموش و به ازای COLOR=1 روشن می‌کند . Xاز 239-0 و Y از 127-0 می‌تواند تغییر کند.

دستور LOCATE ROW,COLUMN: این دستور مکان نما را در سطر (ROW) و ستون (COLUMN) مشخص شده قرار می‌دهد.ROW می‌تواند از 1 تا 16 تغییر کند.تغییرات COLUMN بستگی به انتخاب MODE دارد که می‌تواند از 1 تا 40 تغییر کند.

دستور NOBLINK/OFF BLINK/CURSOR ON: این دستور برای قسمت های متنی استفاده می‌شود. مکان نما می‌تواند در حالت های ON یا OFF و چشمک زدن (BLINK) یا چشمک نزدن (NOBLINK) باشد.

دستور COLORو (X_1وY_1)- LINE(X_0وY_0):

با این دستور از PIXEL اول با مختصات (X_0وY_0) به PIXEL دوم با مختصات (X_1وY_1) خطی با رنگ COLOR کشیده می‌شود. COLOR=0 خط را پاک کرده و به ازای COLOR=255 خطی با رنگ سیاه رسم خواهد شد.

دستورCIRCLE RADIUS,COLOR(,(X_0وY_0):

این دستور دایره ای به مختصات مرکزیت (X_0وY_0) و شعاع RADIUS و رنگ COLOR رسم خواهد کرد.COLOR=0 دایره را پاک کرده و به ازای COLOR=255 دایره با رنگ سیاه رسم خواهد شد.

دستور SHOWPIC X,Y ,LABLE: برای نمایش عکسی که در منوی TOOLS و قسمت GRAPHIC CONVERTER ذخیره کرده اید استفاده می‌شود. X مکان قرارگیری افقی وy مکان قرارگیری عمودی عکس را نشان می‌دهد.LABLE نام برچسبی است که اطلاعات عکس مورد نظر در آن قرار دارد.

برچسب FILE" $BGF".BGF" اشاره به فایل BGF و یا همان عکس مورد نظر که با فرمت BGF و با نام دلخواه FILE در کنار برنامه ی اصلی ذخیره شده است، دارد.

(KS108) SED Graphical Lcd display

نوع دیگری از LCD های گرافیکی به نام SED موجود می باشد که نسبت به نوع قبلی ارزانتر و کم سرعت تر هستند.این نوع LCDها دارای دو چیپ هستند و LCD به دو قسمت تقسیم شده است.

به طور مثال در مدل 64SED*LCD,128 به دو قسمت ۶۴*۶۴ تقسیم شده و بنابراین LCD دارای دو CHIP SELECT(CS) است.جهت کار با این نوع LCD بایستی از کتابخانه LBX.GLCDKS108 در فولدر LIB در مسیر نصب BASCOM استفاده نمود.در صورتی که پسوند آن LIB باشد،بایستی آن را به LBX تغییر دهید.

برای راه اندازی LCD گرافیکی از پیکربندی زیر استفاده می کنیم:

Config graphLcd =type,Dataport=port,controlport=port,ce=pin,ce2=pin,cd=pin,wr=pin,rd=pin,RESET=pin

,enable=pin

Type: که می‌توانید انواع 128*64SED و120*64SED و64SED باشد.

DATAPORT: مشخص کننده ی پورتی است که به عنوان ورودی داده LCD استفاده می‌شود. به طور مثال PORTA=DATAPORT که در این صورت پایه های $D_0 - D_7$ از LCD به ترتیب به پایه های 7.PORTA-0.PORTA متصل می‌شود.

CONTROLPORT: مشخص کننده ی پورتی است که از پایه های برای کنترل LCD استفاده می‌شود.

CE(CHIP ENABLE): شماره ی پایه ای است که برای فعال کردن چیپ موجود در LCD استفاده می‌شود.به طور مثال اگر PORTC=CONTROLPORT باشد،0=CE به معنای اتصال 0.PORTC به پایه CE از LCD می باشد.

:CE2(CHIP ENABLE 2)

شماره ی پایه ای است که برای فعال کردن چیپ دوم موجود در LCD استفاده می‌شود.به طور مثال اگر PORTC=CONTROLPORT باشد،1=CE2 به معنای اتصال 1.PORTC به پایه CE2 از LCD می باشد.

:CD(DATA/CODE)

شماره ی پایه ای است که برای کنترل کردن پایه CD موجود در LCD استفاده می‌شود. به طور مثال 2=CD به معنای اتصال 2.PORTC به پایه ی CD از LCD می باشد.

:RD(READ)

شماره پایه ای است که برای کنترل کردن پایه RD موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،3=RD به معنای اتصال 3.PORTC به پایه ی RD از LCD می باشد.

:RESET

شماره پایه ای است که برای کنترل کردن پایه RESET موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،4=RESET به معنای اتصال 4.PORTC به پایه ی RESET از LCD می باشد.

:ENABLE

شماره پایه ای است که برای کنترل کردن پایه ENABLE موجود در LCD استفاده می‌شود. به طور مثال اگر PORTC=CONTROLPORT باشد،5=ENABLE به معنای اتصال 5.PORTC به پایه ی ENABLE از LCD می باشد.

دستورات کار با SED LCD:

دستور CLS:

این دستور تمام صفحه نمایش LCD چه قسمت متنی و چه قسمت گرافیکی را پاک می‌کند .

دستور CLS GRAPH:

این دستور فقط قسمت گرافیکی را پاک می‌کند .

دستور CLS TEXT:

این دستور فقط قسمت متنی را پاک می‌کند .

دستور LCDAT:

این دستور برای نوشتن متن بر روی SED LCD استفاده می‌شود.

دستور:PSET X,Y,COLOR

این دستور یک PIXEL را در مختصات (X,Y) به ازای 0=COLOR خاموش و به ازای 1=COLOR روشن می‌کند . Xاز 239-0 و Y از 127-0 می‌تواند تغییر کند.

دستور LOCATE ROW,COLUMN:

این دستور مکان نما را در سطر (ROW) و ستون (COLUMN) مشخص شده قرار می‌دهد.ROW می‌تواند از 1 تا 16 تغییر کند.تغییرات COLUMN بستگی به انتخاب MODE دارد که می‌تواند از 1 تا 40 تغییر کند.

دستور NOBLINK/OFF BLINK/CURSOR ON:

این دستور برای قسمت های متنی استفاده می‌شود. مکان نما می‌تواند در حالت های ON یا OFF و چشمک زدن (BLINK) یا چشمک نزدن (NOBLINK) باشد.

دستور COLORو (X_1وY_1)- LINE(X_0وY_0):

با این دستور از PIXEL اول با مختصات (X_0وY_0) به PIXEL دوم با مختصات (X_1وY_1) خطی با رنگ COLOR کشیده می‌شود. 0=COLOR خط را پاک کرده و به ازای 255=COLOR خطی با رنگ سیاه رسم خواهد شد.

دستورCIRCLE RADIUS,COLOR(,(X_0وY_0):

این دستور دایره ای به مختصات مرکزیت (X_0وY_0) و شعاع RADIUS و رنگ COLOR رسم خواهد کرد.COLOR=0 دایره را پاک کرده و به ازای COLOR=255 دایره با رنگ سیاه رسم خواهد شد.

دستور SHOWPIC X,Y ,LABLE:

برای نمایش عکسی که در منوی TOOLS و قسمت GRAPHIC CONVERTER ذخیره کرده اید استفاده می‌شود. X مکان قرارگیری افقی وy مکان قرارگیری عمودی عکس را نشان می‌دهد.LABLE نام برچسبی است که اطلاعات عکس مورد نظر در آن قرار دارد.

برچسب FILE" $BGF".BGF:

اشاره به فایل BGF ویا همان عکس مورد نظر که با فرمت BGF وبا نام دلخواه FILE در کنار برنامه ی اصلی ذخیره شده است،دارد.

مثال:

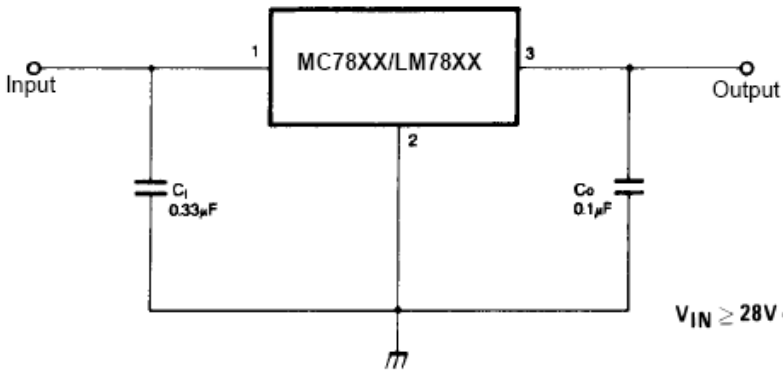
با استفاده از LCD گرافیکی و دستورات مربوط به آن برنامه ای بنویسید که در مرکز LCD دایره ای کشیده شود.پس از زمان اندکی همینطور دوایر دیگری با شعاع های بزرگتر به دور آن کشیده شود و هنگامی که آخرین دایره کشیده شد صفحه LCD پاک شده و این کار ادامه پیدا کند.

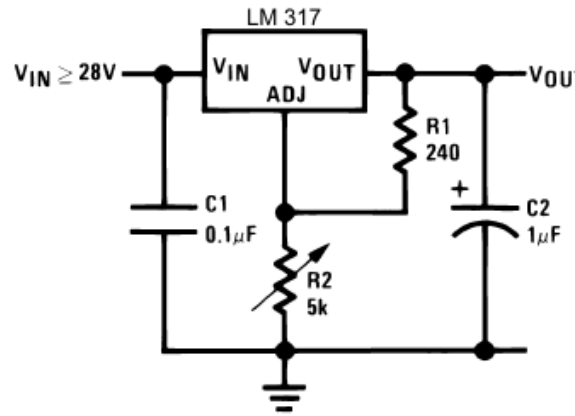
رگولاتور ولتاژ

رگولاتورها المانهای الکترونیکی هستند که جهت تامین یک ولتاژ مستقل از بار و ولتاژ ورودی به کار می روند که علیرغم تغییرات ولتاژ ورودی و تغییرات در بار(جریان خروجی) همواره دارای ولتاژ ثابتی می‌باشند(البته با در نظر گرفتن محدوده تغییرات تعریف شده توسط کمپانی سازنده). از این رو المانهای بسیار مفیدی در مدارهای الکترونیکی جهت تغذیه IC ها و دیگر مدارهای مجتمع می‌باشند. رگولاتورها بسته به ولتاژ و جریان مورد نیاز دارای تنوع زیادی می‌باشند. ازجمله ساده ترین انواع رگولاتورها، رگولاتورهای خطی (Linear Regulators)، هستند که به صورت گسترده مورد استفاده قرار می گیرند. به عنوان مثال داریم:

LM317(ADJ) LF33(3. 3Volt)،LM7912(-12Volt)،LM7905(-5Volt)،LM7812(12Volt)،LM7805(5Volt)، نمونه های متداول نوع خطی در بازار هستند. از مشکلات این نوع بازدهی کم آنها و در نتیجه دفع انرژی به صورت گرما می‌باشد و در جریان های بالا بایستی حتما از Heat sink استفاده گردد. اما در جریانهای پایین بسیار مناسب می‌باشند. از جمله مزایای آنها قیمت مناسب (حدود ۲۰۰ تومان) و نویز پایین آنهاست.

در شکلهای زیر نمونه ای از این رگولاتورها را می بینید.

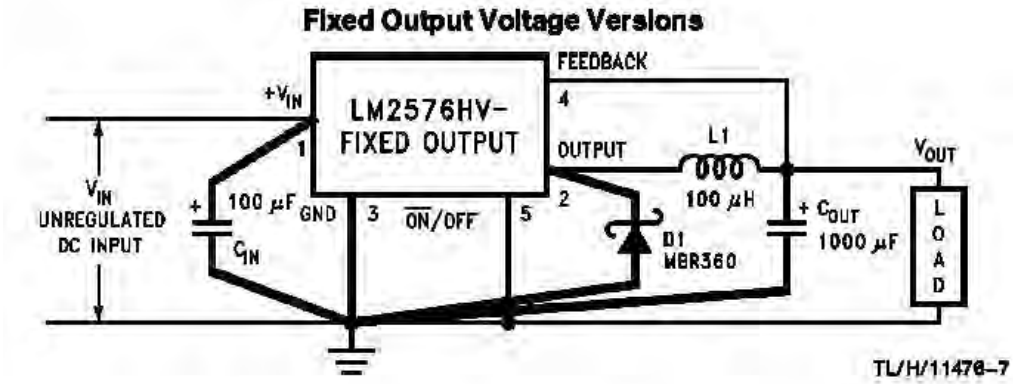


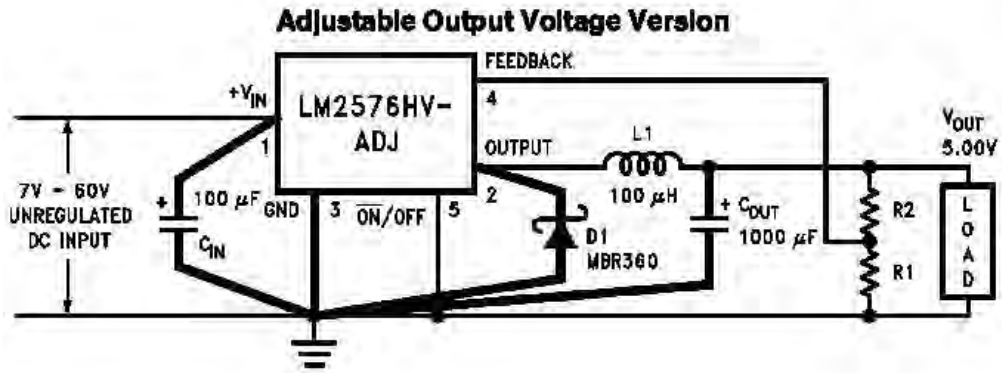


نوع دیگر رگولاتورها، رگولاتورهای سویچینگ هستند. با ظهور منابع سویچینگ تحولی در منابع تغذیه بوجود آمد و بازدهی این مدارها چندین برابر شد. در اینجا قصد بحث در مورد منابع تغذیه سویچینگ و طراحی آنها را نداریم. از انواع رگولاتورهای

سویچینگ می‌توان سری LM2576 که ۳ آمپری است وLM2575 که ۱ آمپری است را می‌توان نام برد. این IC در دو نمونه HV و معمولی در دسترس می‌باشد که در نوع HV ماکزیمم ورودی تا ۶۰ ولت و در نوع معمولی تا ۴۰ ولت می‌باشد.

این IC در ولتاژهای ۱۵و۱۲و۵و۳و۳/۳ و همینطور ADJ (قابل تنظیم ۱/۵تا ۵/۵) و با قیمتی در حدود ۱۰۰۰ تومان در دسترس می‌باشد. در شکل شماتیک مدار نمونه آن را می بینید.





برای LM2576 جریان ۳ آمپر تضمین شده می‌باشد از جمله مزایای این رگولاتورها جریان خروجی بالا، بازدهی بالا تا ۸۸٪، ولتاژ ورودی بالا ۷ تا ۴۰ ولت و در ورژن HV تا ۸۰ ولت میباشد.

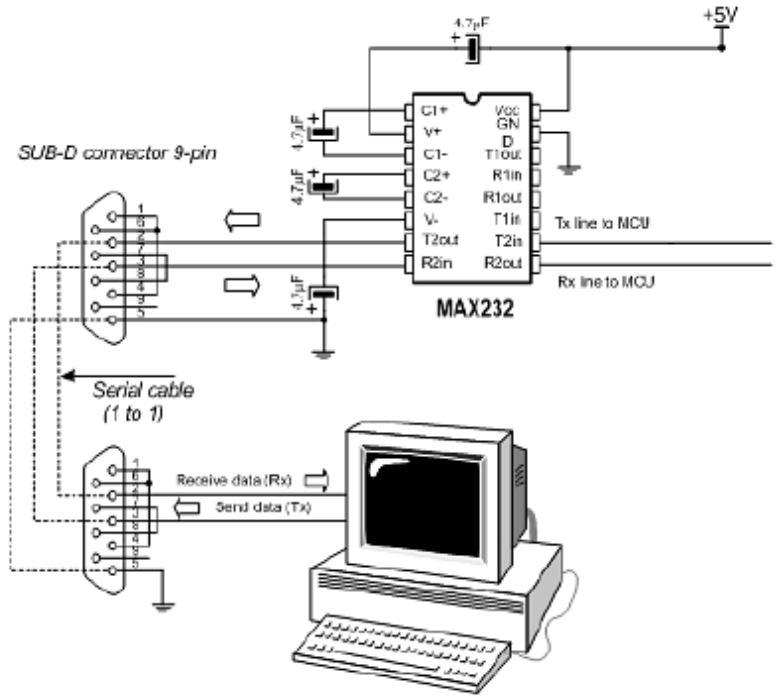
استاندارد RS-232

RS-232 یکی از استانداردهای پرکاربرد در کامپیوترهای شخصی و کاربردهای صنعتی است که هم ارتباط سریال سنکرون و هم آسنکرون را پشتیبانی کرده و بصورت Full Duplex عمل می‌کند. کامپیوترهای شخصی تنها ارتباط آسنکرون را پشتیبانی می‌کنند و از طریق چیپ UART موجود در برد اصلی، اطلاعات را از حالت موازی به سریال و یا از سریال به موازی تبدیل کرده و با تنظیمات زمانی آنرا از طریق پورت سریال ارسال و یا دریافت می‌کنند.

پورت سریال یک کانکتور ۹ پین است که چون این استاندارد از ابتدا برای ارتباط با مودم طراحی شده بود، دارای پینهای Handshaking و وضعیت نیز هست. اما نوع خاصی از ارتباط با RS-232 است که به نام Null-Modem بوده و فقط شامل پینهای ارسال و دریافت است که برای ارتباط با غیرمودم استفاده می‌شود. بنابراین تنها به دو پین RX و TX (و البته زمین) نیاز است.

Pin		عملکرد
۱	Carrier Detect	آیا مودم به یک خط تلفن متصل است ؟
۲	Receive Data	کامپیوتر اطلاعات ارسال شده توسط مودم را دریافت می نماید.
۳	Transmit Data	کامپیوتر اطلاعاتی را برای مودم ارسال می دارد.
٤	Data Terminal Ready	کامپیوتر به مودم آمادگی خود را برای ارتباط اعلام می دارد.
٥	Signal Ground	زمین سیگنال
٦	Data Set Ready	مودم آمادگی خود را برای ارتباط به کامپیوتر اعلام می دارد.
۷	Request To Send	کامپیوتر از مودم در رابطه با ارسال اطلاعات سوال می نماید.
۸	Clear To Send	مودم به کامپیوتر اعلام می نماید که می تواند اطلاعاتی را ارسال دارد.
۹	Ring Indicator	زنگ تلفن تشخیص داده خواهد شد.

در استاندارد RS-232 سطح ولتاژ ۳+ تا ۱۲+ نمایانگر وضعیت Space یا صفر منطقی و بازه ی ۳- تا ۱۲- ولت نمایشگر وضعیت Mark یا یک منطقی است.


+5V
SUB-D connector 9-pin
MAX232
Tx line to MCU
Rx line to MCU
Serial cable
(1 to 1)
Receive data (Rx)
Send data (Tx)