کامپیوتر چیست؟

از ساختار تا کاربرد

بخش اول

دوستان اگر آموزش کمپیوتر برای همه برای شما مفید بوده لطفا آموزش کمپیوتر برای همه
را به دوستان خود معرفی کنید

عبدلکبیر MCTS MCITP
www.kabirict.com
www.facebook.com/kabirict
www.youtube.com/c/kabirict

این سایت به شما امکان می دهد تا بهترین و کمیاب ترین سافت ویر ها را بطور رایگان دانلود نماهید
سایت صرف بخاطر هموطنان عزیز به زبان فارسی طراحی نمودم تا سطح علم و دانش خودرا درعرصه علم ، تکنولوژی بالا ببرند

عنوان صفحه

مقدمه

چرا کامپیوتر؟

بنا به سنتی حسنه، در ابتدای هر درسی از بزرگان آن درس یاد کرده و تاریخچه‌ای از آن بیان می‌کند. این رسم علاوه بر آن که روند فکر کردن بشر نسبت به آن علم را روشن می‌کند، یک قدردانی از زحمات گذشتگان آن علم است –بماند که برخی آثار و فرهنگ‌ها که در گذشته برده شد، بدون بیان نام و نشان اصلی آن تنها به عده‌ای که آن علم را دزدیده بودند نسبت داده شد، حال خود قضاوت کنید که آیا این کمال اخلاق است؟- بنا بدین حکم این جا نیز ابتدا از بزرگان و کسانی که در این علم زحماتی کشیده‌اند نام برده می‌شود و سپس با بیان علم روز آن‌ها تکمیل می‌گردد. بدین سبب کمال امانت در ارائه‌ی مطالب گذشتگان رعایت شده و پس از آن روش‌های توسعه‌ی این علم بیان می‌گردد. امید است که در پایان فصول این بخش دید کلی از ساختار کامپیوت و روش‌های حل مسئله در ذهن شما ایجاد گردد.

نکته‌ی لازم به ذکر این که روش بیان مطالب در این فصول دقیقاً بر مبنای نحوه‌ی فکر کردن می‌باشد. از این رو عدم بیان برخی مطالب، و یا ارجاع و یا تمرین‌های ذکر شده بدان معنا ست که اگر به روش مشابه بیان شده فکر کنید می‌توانید خود نیز به همین نتایج برسید. در کل این توانایی که بتوانید برای هر مسئله روشی ابداع کنید در شما بالا خواهد رفت. بدین ترتیب در مسائل برنامه‌نویسی و الگوریتم‌نویسی نیز مهارت بیشتری پیدا خواهید کرد.
در کتب تاریخی ما، در بیان علم هندسه دیده می شود که به جای بیان جواب مسئله، روش فکر کردن و رسیدن به پاسخ مسئله مطرح شده است. این نحو بیان مطالب کمک می‌کند که هر نفر خود توانایی حل مسائل را پیدا کند و تنها به مسائل حل شده اکتفا نکند.

من تلاش خود را در تهیه‌ی بهینه‌ی این جزوه کرده ام، اما اگر احیاناً در مباحثی به خوبی توضیح داده نشده بود و یا این که مطالب گنگ بود نظر خود را برای من ارسال کنید تا هم پاسخ مناسب به شما داده شود و هم در نسخه-های بعدی این مشکل رفع شود.

فصل اول

تارخچه‌ی کامپیوتر

لغت کامپیوتر:

در زبان انگلیسی «کامپیوتر» به کسی می‌گفتند که محاسبات ریاضی را (بدون ابزارهای کمکی مکانیکی) انجام می‌داد. بر اساس واژه‌نامه ریشه‌یابی *Barnhart Concise* واژه کامپیوتر در سال ۱۶۴۶ به زبان انگلیسی وارد گردید که به معنی شخصی که محاسبه می‌کند بوده است و سپس از سال ۱۸۹۷ به ماشین‌های محاسبه مکانیکی گفته می‌شد. در هنگام جنگ جهانی دوم کامپیوتر به زنان نظامی انگلیسی و امریکایی که کارشان محاسبه مسیرهای شلیک توپ‌های بزرگ جنگی توسط ابزار مشابهی بود، اشاره می‌کرد.

در اوایل دهه ۵۰ میلادی هنوز اصطلاح ماشین حساب (*computing machines*) برای معرفی این ماشین‌ها به‌کار می‌رفت. پس از آن عبارت کوتاه‌تر کامپیوتر (*computer*) به‌جای آن به‌کار گرفته شد. ورود این ماشین به ایران در اوائل دهه ۱۳۴۰ بود و در فارسی از آن زمان به آن کامپیوتر می‌گفتند. واژه رایانه در دو دهه اخیر در فارسی رایج شده و به‌تدریج جای کامپیوتر را گرفت.

بیشتر بدانیم

برابر این واژه در زبان‌های دیگر حتماً همان واژه زبان انگلیسی نیست. در زبان فرانسوی واژه "*ordinateur*"، که معادل "سازمان‌ده" یا "ماشین مرتب‌ساز" می‌باشد به‌کار می‌رود. در اسپانیایی "*ordenador*" با معنایی مشابه استفاده می‌شود، همچنین در دیگر کشورهای اسپانیایی زبان *computadora* بصورت انگلیسی‌مآبانه‌ای ادا می‌شود. در پرتغالی واژه *computador* به‌کار می‌رود که از واژه *computar* گرفته شده و به معنای "محاسبه کردن" می‌باشد. در ایتالیایی واژه "*calcolatore*" که معنای ماشین حساب بکار می‌رود که بیشتر روی ویژگی حسابگری منطقی آن تاکید دارد. در سوئدی رایانه "*dator*" خوانده می‌شود که از "*data*" (داده‌ها) برگرفته شده است. به فنلاندی "*tietokone*" خوانده می‌شود که به معنی "ماشین اطلاعات" می‌باشد. اما در زبان ایسلندی توصیف شاعرانه‌تری بکار می‌رود، "*tölva*" که واژه‌ایست مرکب و به معنای "زن پیشگوی شمارشگر" می‌باشد. در چینی رایانه "*dian nao*" یا "مغز برقی" خوانده می‌شود. در انگلیسی واژه‌ها و تعابیر گوناگونی استفاده می‌شود، بعنوان مثال دستگاه داده‌پرداز ("*data processing machine*").

علم کامپیوتر:

دائرةالمعارف بریتانیا علم کامپیوتر را چنین تعریف می‌کند:

Computer Science: the study of computers, including their design architecture and their uses for computations, data processing, and systems control .

علم کامپیوتر: یادگیری کامپیوتر، شامل طراحی، معماری، و روش‌های استفاده از محاسبات، داده‌ها و هدایت سامانه (کنترل سیستم).

دانشنامه اینترنتی ویکی‌پدیا این تعریف را می‌دهد:

Computer science, or computing science, is the study of the theoretical foundations of information and computation and their implementation and application in computer systems.

علم کامپیوتر یا علم محاسبات یادگیری بنیان‌های نظری اطلاعات و محاسبات و پیاده‌سازی و کاربردی کردن آن‌ها در سامانه‌های کامپیوتری است.

در دهه‌ی ۱۳۵۰ اصطلاح «علوم کامپیوتر» در فارسی در برابر «*computer sciences*» رایج شد. این اصطلاح انگلیسی نام رشته‌ای تحصیلی در برخی دانشگاه‌های کشورهای انگلیسی زبان بود. همین رشته را در دیگر کشورهای اروپائی «انفورماتیک» می‌نامیدند. این واژه از دو جزء انفورماسیون (اطلاعات) و آتوماتیک (خودکار) درست شد که به معنی پردازش خودکار اطلاعات بود. اگر چه، واژه *informatics* درزبان انگلیسی نیز معنائی نزدیک به علم کامپیوتر ولی متفاوت با آن پیدا کرده ولی به هرحال چندان رایج نیست. لزوم نام‌گذاری و عرضه مطالب مختلفی که برای طراحی سیستم‌های کامپیوتری سخت‌افزاری و نرم‌افزاری لازم بود با رشد و گسترش کامپیوترها در دهه‌ی ۴۰ و ۵۰ میلادی احساس شده بود و در دهه‌ی ۶۰ نام کامپیوتر ساینس به عنوان رشته دانشگاهی رواج یافت و به ایران هم رسید.

بیشتر بدانیم:

چون تلاش‌ها برای ساختن دستگاهی که محاسبات را خودکار انجام بدهد عمدتاً به دست ریاضی‌دان‌ها و مهندسان برق انجام می‌گرفت، درس‌هائی که برای این رشته نو پا تعیین می‌شد از دو رشته ریاضی و برق می‌آمد. به تدریج درس‌های ویژه این رشته نیز شکل گرفت. فن برنامه‌سازی نیز زمینه‌ی جدیدی بود و نیاز به آموزش داشت. بنابراین رشته‌ای جدید ظهور کرد و احتیاج به نام داشت و بالاخره نام علم کامپیوتر بر آن ماند. در سال‌های اخیر اصطلاح رایانش رایج شده است ولی هنوز در فارسی به علم کامپیوتر علم رایانش نمی‌گویند (گویا رایانش را از رایاندن یا رایانیدن که ظاهراً در فارسی قدیم سابقه‌ای داشته است گرفته‌اند و رایانه به معنای کامپیوتر را هم از آن ساخته‌اند).

در مورد نام مناسب برای این رشته هنوز بحث ادامه دارد و دانشگاه های معتبر جهان نام‌های مختلفی برای دانشکده‌های این رشته انتخاب کرده‌اند. سه گفتاورد (نقل به معنی) از سه تن از بزرگان این رشته:

«ادسخر دایکسترا» دانشمند «هلندی»: علم کامپیوتر همان‌قدر به کامپیوتر مربوط است که جراحی به چاقو یا ستاره‌شناسی به تلسکوپ.

«دانلد کنوت»: علم کامپیوتر اساساً علم الگوریتم‌ها است و باید به آن الگوریتمیکس (*algorithmics*) یا علم محاسبات (*computing science*) گفت. اگر این حرف کنوت را بپذیریم نام فارسی این رشته علم رایانش می‌شود.

«فردریک بروکس»: علم کامپیوتر علم نیست. تمام رشته‌هائی که در نام خود کلمه علم دارند علم نیستند. علوم تربیتی، علوم اجتماعی، علوم نظامی... این‌ها علم نیستند. فیزیک و شیمی علم هستند.

رشته‌های دیگری که با نام مهندسی کامپیوتر یا فن‌آوری اطلاعات یا مهندسی نرم‌افزار مطرح هستند از یک نظر همه جزئی از رشته وسیع علم کامپیوترند. البته باید توجه کرد که این رابطه جزء و کل مانند مهندسی مکانیک است که از دید کلی بخشی از علم فیزیک است.

به طور کلی می توان گفت: بررسی علمی امور مربوط به طراحی و ساخت و استفاده از سیستم‌های کامپیوتری را علم کامپیوتر می‌نامند.

آی‌تی چیست؟

آی‌تی مخفف اینفورمیشن تکنولوژی (فناوری اطلاعات) است و بسیار از علم کامپیوتر وسیع‌تر (و مبهم‌تر) است. این اصطلاح در دهه‌ی ۱۹۹۰ جایگزین اصطلاحات: پردازش داده‌ها و سیستم‌های اطلاعات مدیریت شد که در دهه‌های ۱۹۷۰ و ۱۹۸۰ بسیار رایج بودند. آی‌تی معمولاً به تولید و پردازش و نگهداری و توزیع اطلاعات در موسسات بزرگ اشاره دارد.

اطلاعات:

اطلاع داده با معنی است. داده اعداد و ارقام و متن‌ها و تصویر ها هستند. اگر کاربری این اعداد داده‌ها را بکار بگیرد آنگاه این‌ها برای او ارزش اطلاعاتی دارند. مؤسسات مجموعه بزرگی از اطلاعات را بکار می‌گیرند. سیستم‌های کامپیوتری اطلاعات حسابداری و مالی و پرسنلی و فنی مؤسسات را تولید و پردازش می‌کنند. شبکه‌های کامپیوتری این اطلاعات را در درون موسسه یا بیرون از آن در دسترس کاربران قرار می‌دهد. مجموعه این عملیات در عرصه‌ی فناوری اطلاعات قرار دارد.

آی‌تی با علم کامپیوتر چه فرق دارد؟

البته در موارد زیادی با هم اشتراک دارند. اگر علم کامپیوتر را مشابه مهندسی مکانیک بگیریم، آی‌تی مشابه صنعت حمل و نقل است. در صنعت حمل و نقل، خودرو و راه‌آهن و هواپیما و کشتی داریم. همه این‌ها را مهندسان مکانیک طرح می‌کنند. در عین حال در صنعت حمل و نقل مسائل مربوط به مدیریت ناوگان و مدیریت ترافیک و تعیین استراتژی حمل و نقل در سطح شرکت و شهر و کشور مطرح است که ربط مستقیمی به مهندسی مکانیک ندارد.

مهندسی نرم‌افزار

هدف مهندسی نرم‌افزار تولید برنامه‌های اطمینان‌پذیر با بودجه معین و در مدت مناسب است. بسیاری معتقدند که این امر با "مهندسی کردن" کار تولید نرم‌افزار حاصل می‌شود. یعنی می‌خواهند روش‌های مهندسی را که در رشته‌های دیگر مهندسی رایج است در کار تولید نرم‌افزار هم بکار بگیرند. به همین دلیل برخی مهندسی نرم‌افزار را جزو علم کامپیوتر نمی‌دانند و آن را در حیطه مهندسی قرار می‌دهند.

اصطلاح «مهندسی نرم‌افزار» در سال ۱۹۶۸ در کنفرانسی در شهر گارمیش در جنوب آلمان مطرح شد. در آن سال‌ها تولید نرم‌افزار کامپیوتر به حدی از بلوغ رسیده بود که اهمیت تولید نرم‌افزاراطمینان‌پذیری که در محدوده‌ی

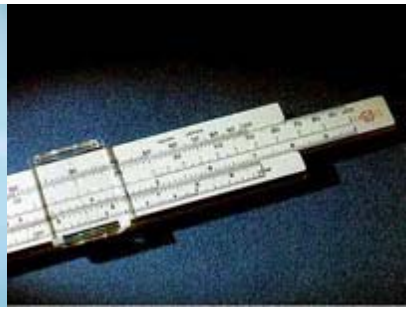
بودجه معین و در مدت معین تولید شود اهمیت بسیاری یافته بود. بسیاری از پروژه‌های نرم‌افزاری به دلیل نبود روش‌های مناسب کنترل پروژه با شکست روبرو شده بودند. اصطلاح بحران نرم‌افزار در آن سال‌ها بسیار رایج بود. یکی از هدف‌های مهندسی نرم‌افزار کاربرد روش‌های کنترل پروژه‌های مهندسی در کار تولید نرم‌افزار بود. در عین حال نرم‌افزار ویژگی‌های خود را دارد و نیازمند برخوردی متفاوت از رشته‌های رایج و کلاسیک مهندسی مانند مهندسی برق و مکانیک است.

از تحولات مهم در تاریخ مهندسی نرم‌افزار تشخیص این امر بود که هر پروژه نرم‌افزاری باید بر اساس مدلی از مراحل کار (یا به اصطلاح مدل چرخه زندگانی نرم‌افزار) انجام گیرد. معروف‌ترین این مدل‌ها به «مدل آبشاری» معروف است.

در بحث از مهندسی نرم‌افزار باید توجه کرد که این رشته با عنوان شغلی که گاه برای اشخاص بکار می‌رود تفاوت دارد. عنوان شغلی «مهندس نرم‌افزار» معمولاً به جای اصطلاح قدیمی‌تر «برنامه نویس» به کار می‌رود و دارنده آن عنوان ممکن است کارش امور مربوط به مهندسی نرم‌افزار باشد یا نباشد. برنامه نویسی بخشی از فعالیت مهندسی نرم‌افزار است ولی تمام آن نیست.

تاریخچه‌ی کامپیوتر:

با توجه به تعاریف بالا می‌توان هر وسیله‌ای که به انسان در جهت محاسبه کمک می‌کند را کامپیوتر نامید. اگر بپذیریم که بشر در ابتدا با دستان خود به شمارش می‌پرداخته (همانند آن که برای اندازه گیری از وجب و قدم استفاده می‌شد) و بر این مبنا سیستم اعداد ده‌دهی را اختراع کرده (اعداد طبیعی)، می‌توان دستان بشر را به عنوان اولین کامپیوتر در نظر گرفت. و بعد از آن چوب خط و تسبیح و بعدها چرتکه و خط کش محاسبه (که به آن‌ها کامپیوترهای آنالوگ نیز می‌گفتند).



خط کش محاسبه

چرتکه

چوب خط

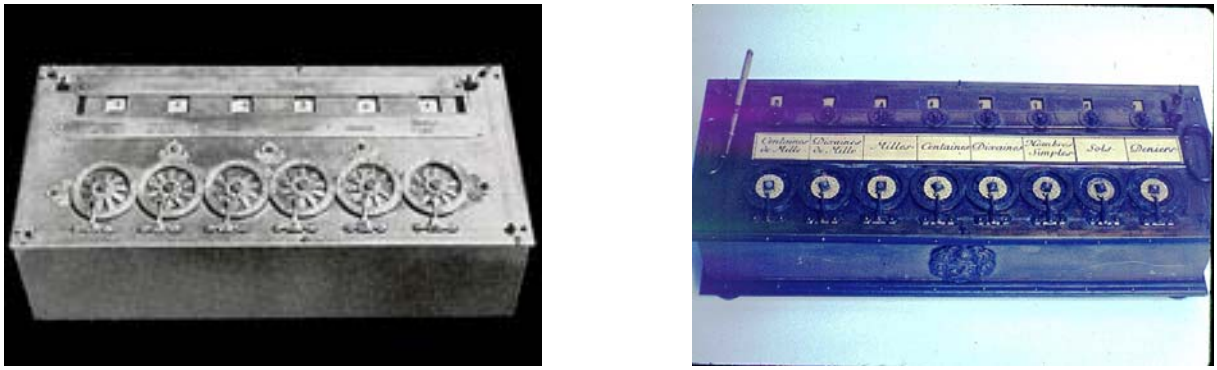
اما علم کامپیوتر که به نوعی بیان کننده‌ی روش استفاده از این ابزار و یا به عبارت دیگر روش محاسبه کردن است تاریخی دیگر دارد. حساب و محاسبه سابقه‌ای قدیم دارد و کهن‌ترین قسمت ریاضیات است. ارشمیدس، اقلیدس و دیگران روش‌های محاسباتی به نام خود دارند. خوارزمی روش‌های محاسباتی مهمی را برای حل معادلات در کتاب خود «الجبر والمقابله» نوشت که سال‌ها مورد استفاده بود. ترجمه کتاب او به لاتین نام او را به غرب برد و در زبان‌های غربی الگوریتم به معنای روش محاسبه جا افتاد.

صفحه ی اول نسخه‌ی قدیمی کتاب الجبر و المقابله از احمد خوارزمی

ماشین حساب‌های مکانیکی:

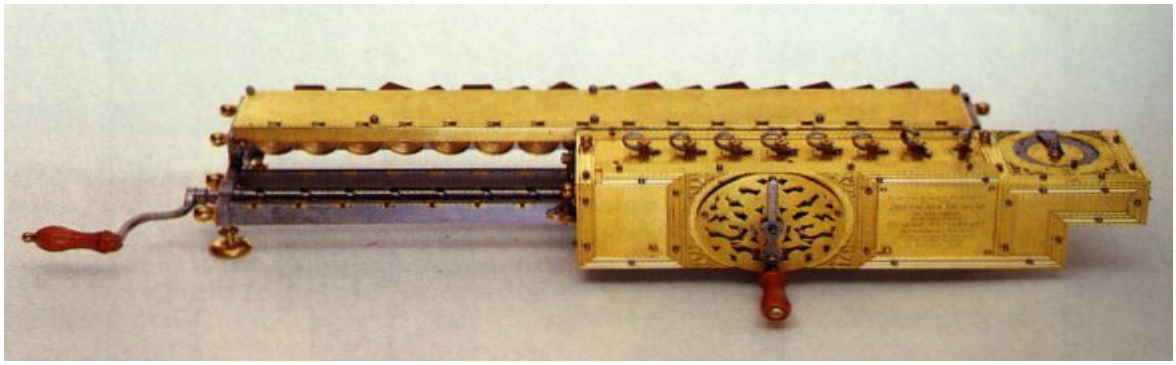
کوشش برای ساختن ماشین محاسبه و شمارش خودکار از قرن هفدهم آغاز شد –که اینجا یک سؤال پیش می‌آید که قبل از آن با آن همه پیشرفت مسلمانان در علم ستاره‌شناسی و محاسبات نجومی با دقت‌های بسیار بالا (محاسبه‌ی عدد π تا چندین رقم اعشار برای محاسبه‌ی محیط زمین) چگونه تاریخ ساخت چنین دستگاه‌هایی و حتی بسیاری علوم دیگر از قرن ۱۷میلادی می‌باشد و تاریخ در بازه‌ی قبل آن بسیار سخت قابل شناسایی است؟؟؟!!!- و این زمانی بود که گسترش علوم ستاره‌شناسی، دریانوردی، بازرگانی و پژوهش‌های فنی و علمی اهمیت بی‌سابقه‌ای به محاسبات عددی بخشیده بود. همزمان با توسعه دانش ریاضی، احتیاج بشر به محاسبات بیشتر گردید، چنانکه سبب اختراع وسایل مختلفی در این زمینه شد.

در سال ۱۶۴۲ پاسکال ریاضیدان فرانسوی دستگاهی را به نام ماشین جمع‌زن (*adding machine*) اختراع کرد. این ماشین مجموعه‌ای از چرخ‌دنده‌های کنار هم بود که چرخ اول نشان دهنده رقم یکان و چرخ‌های بعدی نشان دهنده رقم‌های دهگان، صدگان و ... بود. نکته مهم در ماشین اختراعی پاسکال این بود که می توانست بطور اتوماتیک ده بر یک (*cary*) را حساب کند لیکن تنها قادر به انجام عملیات جمع و تفریق بود. این ماشین کاملاً مکانیکی بود و اعداد به کمک وسیله ای نظیر صفحه شماره گیر تلفن وارد دستگاه می شد و عملیات به وسیله یک سری چرخ دنده و اهرم انجام می گرفت و نتیجه از دریچه مخصوصی قابل قرائت بود.



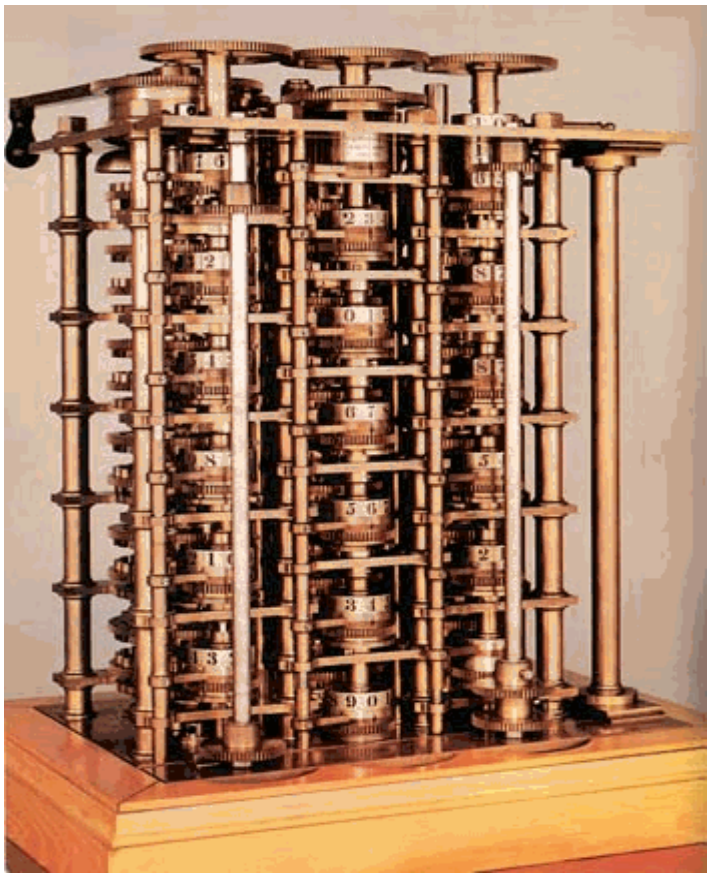
ماشین حساب پاسکال(*adding machine*)

مدتی بعد لایب نیتز(*Leibnitz*) ریاضیدان آلمانی موفق شد دستگاهی به نام ماشین محاسبه (*calculating machine*) بسازد که می توانست علاوه بر جمع و تفریق بر اساس روش پاسکال عملیات ضرب و تقسیم را نیز انجام دهد.



ماشین محاسبه‌ی لایب نیتز (*calculating machine*)

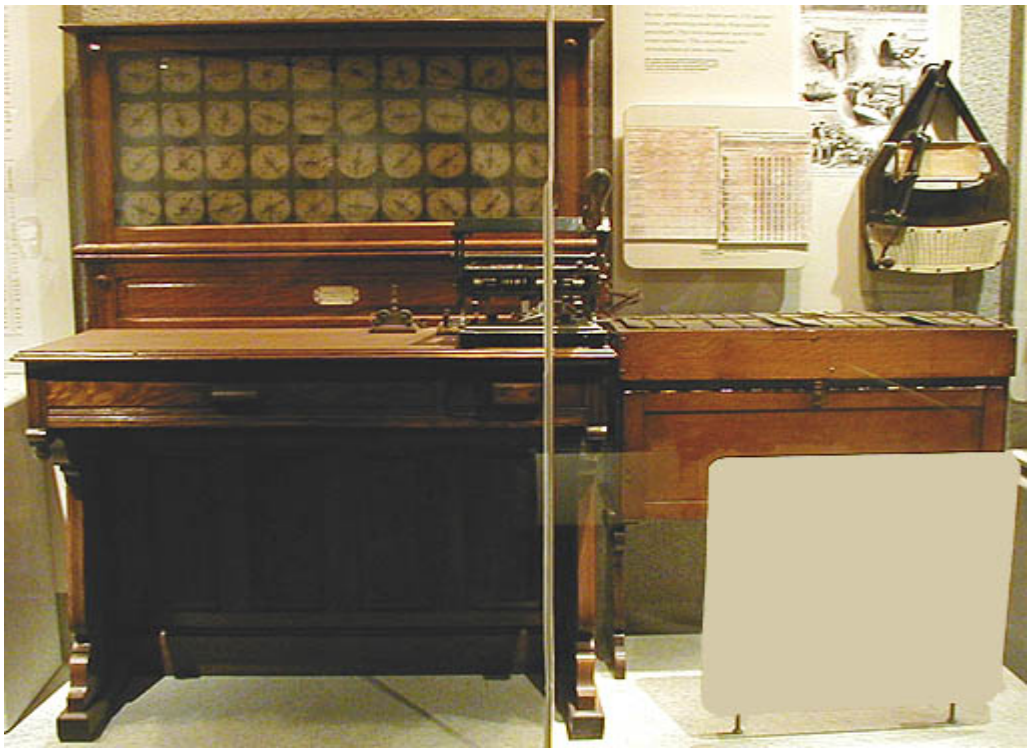
فکر ساختن ماشینی که بطور خودکار و با برنامه ای کار کند برای اولین بار در سال ۱۸۲۲ توسط چارلز بابیج (*Charles Babbage*) مطرح شد. داستان از این قرار است که بابیج به اتفاق یکی از همکارانش مشغول محاسبات نجومی بود که خستگی عجیبی بر او مستولی شد. در این لحظه گفت: «خدایا، ای کاش این محاسبات با ماشین محاسبه‌گر نیروی بخار انجام می شد.» و این فکر سرآغاز ساختن ماشین تفاضلی (*difference engine*) بود. او سال‌های زیادی از عمرش را صرف ساختن این ماشین کرد که در اثر محدودیتهای فنی و مهندسی آن دوره، نتوانست آن را تکمیل کند. طرح‌های تهیه شده توسط بابیج بسیار پیشرفته تر از زمان او بود و بعدها بسیاری از طرحها و افکار وی در ساختن کامپیوترهای اولیه مورد استفاده قرار گرفت و به همین دلیل وی را پدر کامپیوتر لقب داده اند.



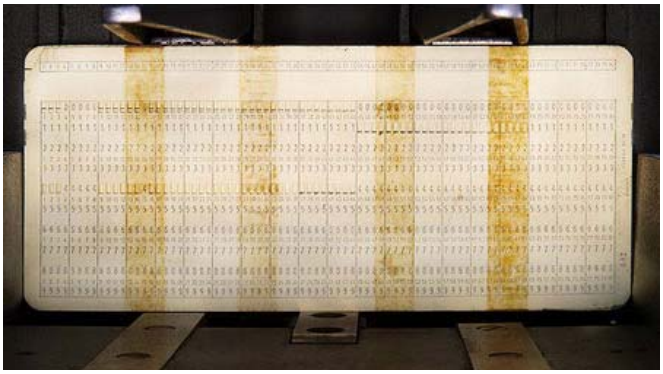
در اواخر قرن نوزدهم، دفتر آمار ایالات متحده آمریکا برای استخراج نتایج سرشماری سال ۱۸۸۰ با مشکلات بزرگی مواجه شد، چرا که دریافت استخراج و نتیجه گیری از این اطلاعات به زمان بسیاری نیاز دارد و بالاخره با سعی و کوشش و صرف وقت زیاد نتایج این سرشماری در سال ۱۸۸۷ به پایان رسید (با استفاده از ابزار آن زمان).

اما همین که فهمیده شد برای سرشماری بعدی به ۱۰ سال وقت نیاز هست فکر اختراع وسیله ای که بتواند این کار را در زمان کوتاهتری انجام دهد تقویت گردید. در این موقع هرمان هالریث (*Herman hollerith*) که با دفتر آمار آمریکا همکاری داشت، اصول جدیدی را برای ضبط، طبقه بندی و جدول بندی اطلاعات به طریق مکانیکی عرضه داشت که ۸ بار سریع تر از روش دستی بود و ماشین جدول‌بندی (*tabulating machine*) نامیده می‌شد. بر اساس این روش برای نمایش هر یک از مطالب و اطلاعات مورد نظر لازم بود در نقطه بخصوصی از نوار کاغذی یک سوراخ

منگنه شود و ماشین دیگری می توانست به کمک مدارهای الکتریکی سوراخهای مذکور را حس کند و ضمن عبور از داخل دستگاه جدولی از مطالب ضبط شده تهیه نماید. برودی نوار جای خود را به کارت های مقوایی داد که اطلاعات هر خانواده روی آن منگنه می شد. (به کار بردن کارت های مقوایی سوراخ شده اولین بار توسط ژاکارد فرانسوی برای کنترل ماشین های بافندگی خودکار و دادن طرح پارچه به کار رفته بود) در اینجا اصطلاح تک‌کار (*unit_record*) برای ماشین‌هایی که امور کارتی را انجام می دهند، پذیرفته شد و معمول گردید.



در سال ۱۹۸۰ اولین سری ماشین های دسته بندی و تفکیک کارت های منگنه شده، توسط هالریث اختراع گردید، به کمک همین ماشین ها نتیجه سرشماری سال ۱۸۹۰، در عرض ۲/۵ سال یعنی حدود یک سوم زمان نتیجه گیری سرشماری قبل، آماده گردید. در سال ۱۸۹۶ هالریث، شرکت ماشین جدول‌بندی را تأسیس کرد و بعد ها با ۱۰ شرکت دیگر ادغام شد و شرکت *IBM* به وجود آمد. افزایش و گسترش فعالیت‌های اقتصادی و سرانجام یافتن انقلاب صنعتی و پیشرفت و توسعه همه جانبه تمدن بشری ایجاب می کرد که وسیله ای سریعتر از دستگاههای تک کار به وجود آید.



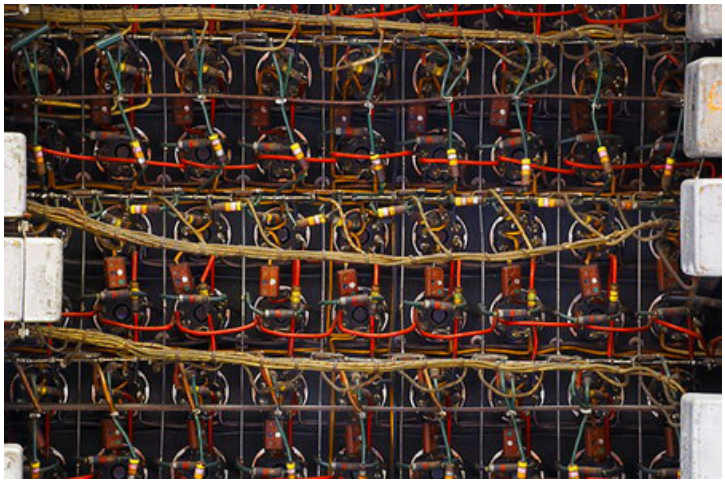
Puncher card، طرح ۰۷۷، سال تولید: ۱۹۳۷، رشد یافته توسط *IBM* برای کاربردهای اجتماعی

اولین ماشین الکترو مکانیکی به وسیله هاوارد ایکن (*Howard Aiken*) در دانشگاه هاروارد و با کمک مالی و فنی شرکت *IBM* ساخته شد. ساختن این ماشین ۵ سال طول کشید و در سال ۱۹۴۴ کامل گردید. این ماشین می‌توانست عملیات جمع، ضرب، تقسیم، تفریق و محاسبه لگاریتم و توان‌های مختلف و همچنین محاسبه توابع مثلثاتی مانند سینوس و کسینوس را انجام دهد. این ماشین بطور مخفف آ، اس، سی، سی (*[ASCC]Automatic Sequence Controlled Calculator*) یا مارک یک (*Mark 1*) نامیده می‌شد. ماشین مزبور عمل ضرب را در مدت ۳ ثانیه انجام می داد و این سرعت هنوز مطلوب نبود.

ماشین‌های الکترونیکی:

در سال ۱۹۴۶ اولین کامپیوتری که از لامپ خلاء استفاده می کرد بنام ENIAC توسط J.Eckert و J.Mauchly به وجود آمد.

ENIAC Tubes، سال تولید: ۱۹۴۶، رشد یافته توسط آمریکا در پایان جنگ جهانی دوم جهت داشتن برتری فنی نسبت به دشمن قیمت: ۵۰۰۰۰۰ دلار



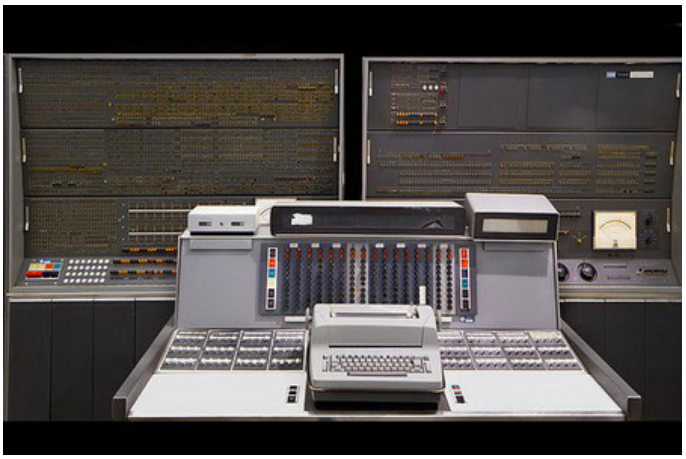
ENIAC Wiring، سال تولید: ۱۹۴۷، این کامپیوتر می توانست ۵۰۰۰ عمل را در یک ثانیه انجام دهد ولی مشکل بزرگش نداشتن هارد دیسک بود که برای هر کار جدید نیاز به جایگزینی بود

SAGE، سال تولید:۱۹۵۴، این کامپیوتر دارای وزنی ۳۰۰ تونی بود و فضای بسیار زیاد را اشغال می کرد! این کامپیوتر بیشتر برای پدافند هوایی ساخته شد تا توانایی کنترل هواپیما ها به صورت ممتد وجود داشته باشد.

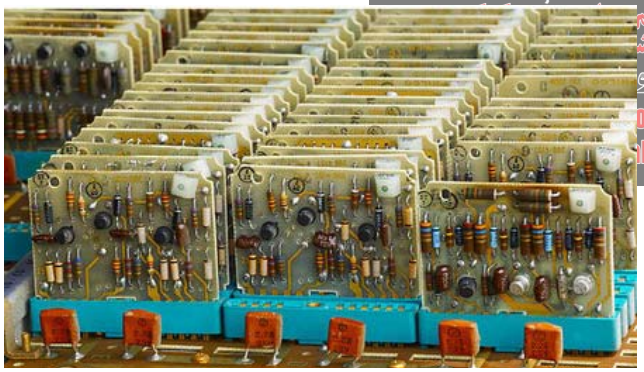
Johnniac، سال تولید:۱۹۵۴، قیمت:۴۷۰۰۰۰ دلار، حافظه:۴ کیلوبایت (تعجب نکنید درست تایپ شده کیلوبایت، رم‌های کنونی تا ۲۰۰۰۰۰۰ کیلوبایت و هارددیسک‌های امروزی تا ۱۰۰۰۰۰۰۰۰۰کیلوبایت را هم می‌توانند ذخیره کنند!)

WISC، سال تولید: ۱۹۵۵، رشد یافته توسط دانشگاه *Viskonsin*. این کامپیوتر قادر بود ۴ تساوی را به طور همزمان انجام دهد (حل کند) که یک پدیده بی همتا در آن زمان بود، قیمت ۵۰۰۰۰ دلار، حافظه: ۱کیلوبایت

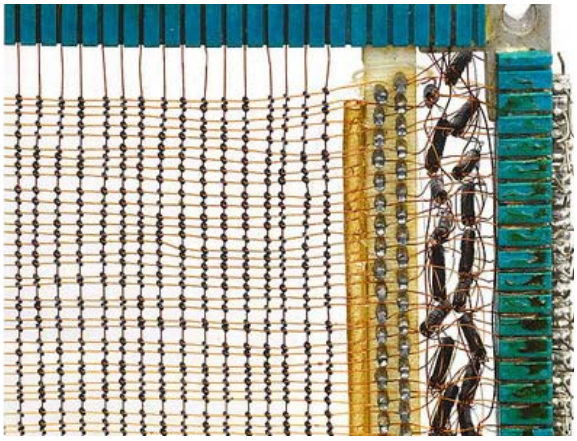
NEAC 2203، سال تولید: ۱۹۶۰



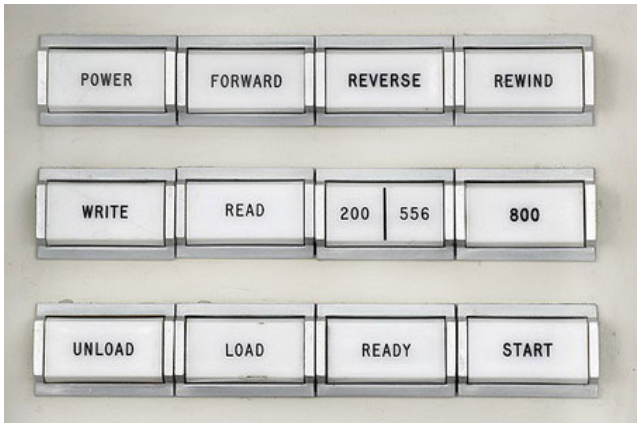
IBM 7030، سال تولید: ۱۹۶۱، این کامپیوتر اولین تلاش *IBM* برای ساخت یک ابرکامپیوتر بود



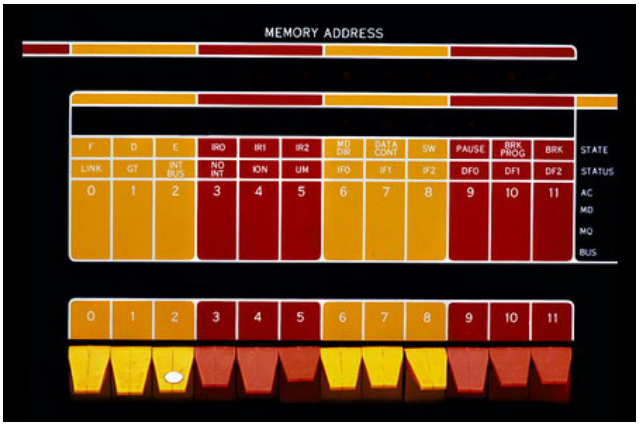
Philco 212، سال تولید: ۱۹۶۲، قیمت:۱۸۰۰۰۰۰ دلار، حافظه: ۶۴ کیلوبایت



CDC 160A Button Array، سال تولید: ۱۹۶۲



PDP-8، سال تولید: ۱۹۶۵، اولین کامپیوتر کوچک که در فروش هم موفق بود و با قیمتی مناسب (۱۸۰۰۰) دلار به فروش می رسید



Supercomputer CDC 6600، قوی ترین کامپیوتر تا سال ۱۹۶۹

Neiman Marcus Kitchen Computer، سال تولید: ۱۹۶۹

Apple 1، سال تولید: ۱۹۷۶، قبل از ارائه *Apple II* از آن استقبال خوبی شد

Computer keyboard Commodore 64، سال تولید: ۱۹۸۲، بیشتر از ۳۰ میلیون فروش

Bowels of super computer Cray-3، سال تولید: ۱۹۹۳

به طور اجمالی: فکر برای ساخت ماشین‌های محاسبه کننده اساس تشکیل فکر برای ساخت کامپیوترها بود. اولین کامپیوتر (به معنای محاسبه‌ی اتوماتیک) را می توان به ماشین پاسکال نسبت داد، بعد از آن به نسل اول کامپیوترها یعنی لامپ خلاء می‌رسیم، بعد از آن استفاده از ترانزیستورها به عنوان نسل دوم. سپس در نسل سوم استفاده از مدارهای مجتمع می‌باشد. استفاده از مدارهای بسیار مجتمع نیز به عنوان نسل چهارم بیان می‌گردد. نسل پنجم و ششم را نسل‌های هوشمند کامپیوترها نامند.

فصل دوم

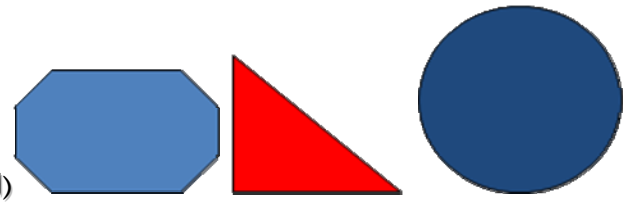
مبانی کامپیوتر

اساس کار دستگاه‌های دیجیتال:

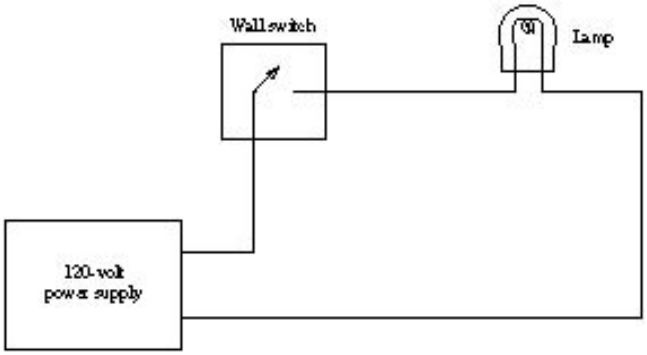
با پیشرفت علم کامپیوتر، ساخت دستگاه‌های محاسباتی با استفاده از مدارهای دیجیتالی، ترانزیستور، گیت‌ها و مدارهای مجتمع(*IC*) به عنوان اساس کار کامپیوترهای جدید مورد استفاده قرار گرفت. از این رو آشنایی با اساس کار این دستگاه‌ها خالی از لطف نیست.

در این فصل ابتدا مشکلاتی که بشر با آن-ها روبه روست به عنوان مسئله طرح می‌گردد و سپس راه‌حل کامپیوتری آن ارائه می‌شود. برای شروع ابتدا نیاز ذخیره کردن داده‌ها را بیان می‌شود:

چه اطلاعاتی در کامپیوتر باید ذخیره شود، و یا با آن ها کار کرد؟

- *Numbers* (اعداد)
  - *Signed: -10, 5, 20, -123, …* (علامت‌دار)
  - *Unsigned: 10, 120, 2…* (بی‌علامت)
  - *Integers: 1, 2, 3, …* (صحیح)
  - *Floating point:* 2×10^{-12}, 6.23×10^{4} , … (اعشاری)
  - *Complex: 2+j1, 4.25+j6.23, …* (مختلط)
  - *Rational: 2.3456, 10.239, …* (حقیقی)
- *Text* (متن)
  - *Characters: a, b, c, …* (کاراکتر)
  - *Strings: "In the name of Allah", "Allah", …* (رشته)
- *Images:* (عکس)
  - *Pixels:* (نقطه‌ای) 
  - *Shapes:* (اشکال) 
- *Sound* (صدا)
- *Logical* (منطقی)
  - *True: "3>2"* (درست)
  - *False: "6-3=5"* (غلط)
- *Instructions* (دستورالعمل)
  - *Add two numbers, multiply 3 by 2, …*

حال باید ببینیم چگونه می‌توان این اطلاعات را در کامپیوتر ثبت کرد. راه‌حل ارائه شده استفاده از کلیدهای الکترونیکی است که در دو وضعیت باز و بسته قابل بررسی است. در این روش اگر کلیدی بسته باشد (یعنی جریان وجود داشته باشد) مقدار یک و اگر کلید باز باشد (یعنی جریان صفر ولت) مقدار صفر منظور می‌گردد.
برای درک بهتر مدار زیر را (که با کلید مکانیکی نمایش داده شده‌است) در نظر بگیرید:



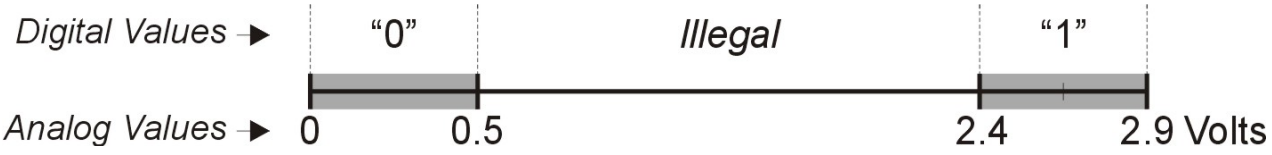
جواب سؤال زیر مقدار را مشخص می‌کند:

آیا لامپ روشن است؟ اگر بله = *True* = ۱ = ۲/۹ ولت اگر نه = *False* = ۰ = ۰ ولت

آیا کلید بسته است؟ اگر بله = *True* = ۱ = ۲/۹ ولت اگر نه = *False* = ۰ = ۰ ولت

در نهایت می‌توان گفت: اگر کلید=۱ آنگاه لامپ = ۱ در غیر این صورت (یعنی اگر کلید=۰) لامپ=۰.
لامپ را به عنوان خروجی در نظر می‌گیریم، کلید را به عنوان ورودی، چون با تغییر ورودی خروجی تغییر می‌کند (البته در مواردی ممکن است با تغییر ورودی خروجی تغییر نکند، اما همواره می‌توان گفت خروجی وابسته به ورودی است).

البته مقادیر منطقی (گسسته، دیجیتال) را معادل یک بازه از ولتاژهای پیوسته (آنالوگ) می گیریم به طور مثال:



تذکر: تفاوت مدارهای دیجیتال با مدارهای آنالوگ استفاده از همین مقادیر صفر و یک است. شکل بالا توضیح دهنده‌ی این موضوع است. مقادیر آنالوگ پیوسته هستند، ولی مقادیر دیجیتال گسسته اند.

ملاحظه می شود که با این روش ما توانایی ایجاد اعداد صفر و یک را دارا هستیم و با کمی محاسبه در ریاضیات می‌خواهیم تمامی داده‌های بیان شده را در کامپیوتر ذخیره کنیم.

ابتدا فرض کنید که روش ثبت اعداد در کامپیوتر را می‌دانیم (که به وسیله‌ی اعمال ریاضی و ذخیره‌ی صفر و یک‌ها امکان پذیر است)، حال به ثبت دیگر داده‌ها می‌پردازیم:

۱. متون (*Text*):

ابتدا کاراکترها را ثبت می‌کنیم. برای این کار کافی است به هر کاراکتر یک عدد نسبت دهیم و چون عدد را می‌توانیم ذخیره کنیم، مشکل حل می‌شود. جدول کد *ASCII* جدولی است جهت هماهنگ کردن این کد ها که در زیر جدول آن آمده است.

برای ثبت متون کافی است از یک کد غیر کاراکتری برای نشان دادن پایان متن استفاده کنیم. این کد همان *0* یا *null* می‌باشد که به منزله‌ی اتمام متن است.

00	*nul*	*10*	*dle*	*20*	*sp*	*30*	*0*	*40*	*@*	*50*	*P*	*60*	*`*	*70*	*p*
01	*soh*	*11*	*dc1*	*21*	*!*	*31*	*1*	*41*	*A*	*51*	*Q*	*61*	*a*	*71*	*q*
02	*stx*	*12*	*dc2*	*22*	*"*	*32*	*2*	*42*	*B*	*52*	*R*	*62*	*b*	*72*	*r*
03	*etx*	*13*	*dc3*	*23*	*#*	*33*	*3*	*43*	*C*	*53*	*S*	*63*	*c*	*73*	*s*
04	*eot*	*14*	*dc4*	*24*	*$*	*34*	*4*	*44*	*D*	*54*	*T*	*64*	*d*	*74*	*t*
05	*enq*	*15*	*nak*	*25*	*%*	*35*	*5*	*45*	*E*	*55*	*U*	*65*	*e*	*75*	*u*
06	*ack*	*16*	*syn*	*26*	*&*	*36*	*6*	*46*	*F*	*56*	*V*	*66*	*f*	*76*	*v*
07	*bel*	*17*	*etb*	*27*	*'*	*37*	*7*	*47*	*G*	*57*	*W*	*67*	*g*	*77*	*w*
08	*bs*	*18*	*can*	*28*	*(*	*38*	*8*	*48*	*H*	*58*	*X*	*68*	*h*	*78*	*x*
09	*ht*	*19*	*em*	*29*	*)*	*39*	*9*	*49*	*I*	*59*	*Y*	*69*	*i*	*79*	*y*
0a	*nl*	*1a*	*sub*	*2a*	***	*3a*	*:*	*4a*	*J*	*5a*	*Z*	*6a*	*j*	*7a*	*z*
0b	*vt*	*1b*	*esc*	*2b*	*+*	*3b*	*;*	*4b*	*K*	*5b*	*[*	*6b*	*k*	*7b*	*{*
0c	*np*	*1c*	*fs*	*2c*	*,*	*3c*	*<*	*4c*	*L*	*5c*	**	*6c*	*l*	*7c*	*|*
0d	*cr*	*1d*	*gs*	*2d*	*-*	*3d*	*=*	*4d*	*M*	*5d*	*]*	*6d*	*m*	*7d*	*}*
0e	*so*	*1e*	*rs*	*2e*	*.*	*3e*	*>*	*4e*	*N*	*5e*	*^*	*6e*	*n*	*7e*	*~*
0f	*si*	*1f*	*us*	*2f*	*/*	*3f*	*?*	*4f*	*O*	*5f*	*_*	*6f*	*o*	*7f*	*del*

۲. عکس‌ها (*Image*):

کافی است به هر رنگ یک عدد نسبت دهیم و به هر نقطه دو مختصه (x,y) که مجموعه ای از این سه عدد یک عکس را مشخص می‌کند. در سیستم‌های جدید به جای نسبت دادن یک عدد به رنگ هر نقطه، ۳ عدد را به رنگ هر نقطه نسبت می‌دهند (*RGB = Red,Green,Blue*)؛ و در مجموع برای هر نقطه ۵ عدد ذخیره می‌گردد.

۳. صداها (*Sound*):

برای این مورد می‌بایست صدا را به فرکانس تبدیل کرده و عدد آن را ثبت کنیم.

۴. عبارات منطقی (*Logical*):

بدیهی است که با دو مقدار ۰ و ۱ قابل ثبت است.

۵. دستورالعمل (*Instructions*):

با دادن یک کد (*up code*) می‌توان آن‌ها را از هم متمایز کرد.

با توجه به موارد بالا کافی است که روش ذخیره‌ی اعداد در کامپیوتر را بیابیم تا بتوانیم بقیه‌ی اطلاعات را در آن ثبت کنیم.

در این جا با دانش به این موضوع که ما می‌توانیم دو مقدار صفر و یک را به سیستم بفهمانیم به دنبال روش‌هایی می‌گردیم که اعداد دیگر را نیز در آن ذخیره کنیم.

کمی ریاضیات:

دسیمال و باینری:

دستگاه اعدادی که با آن آشنا هستیم دستگاه ده‌دهی است. برخی می گویند علت رایج شدن این دستگاه استفاده از انگشتان دست برای شمارش بوده است. ما در زندگی روزمره با دستگاه اعداد دیگری نیز آشنا هستیم و آن دستگاه اعداد شصت‌شصتی و بیست‌وچهاری است که برای محاسبه‌ی ساعت از آن استفاده می‌کنیم. به طور مثال می‌گوییم ساعت *10:21′:22″* و برای آن که زمانی دیگر نسبت به آن را مشخص کنیم گوییم *43′:45″* دیگر. نکته‌ی قابل توجه آن که دیگر از جمع به صورت معمول استفاده نمی‌کنیم و هر گاه به عدد ۶۰ می‌رسیم ۰ گذاشته و ۱ واحد به دیگری اضافه می‌کنیم.

carry ده بر یک *01 : 01′ : 00″*

10:21′:22″

\+ *00:43′:45″*

11:05′:07″

ملاحظه می‌شود که در جمع ابتدا ۶۷=۴۵+۲۲ و چون ۶۷ از ۶۰ کوچکتر نیست ۷ را نوشته و یک واحد به دقیقه اضافه می‌کنیم، سپس ۶۵ = ۴۳+۲۱+۱ و باز هم چون ۶۵ از ۶۰ کوچکتر نیست ۵ را نوشته و یک واحد بر بعدی اضافه می‌کنیم. در قسمت ساعت شاهد دستگاه اعداد ۲۴تایی هستیم. بدان معنی که هرگاه به عدد ۲۴ برسیم یک واحد به روز اضافه کرده و باقیمانده‌ی عدد بر ۲۴ را می‌نویسیم. در ادامه بیشتر به بررسی دستگاه اعداد می‌پردازیم و جمع آن را مورد بررسی قرار می‌دهیم.

حال خالی از لطف نیست که جمع معمولی (در دستگاه ده‌دهی) را نیز در اینجا مورد بررسی قرار دهیم:

0 1 0 1 0

5:0:4:2:5

\+ *0:5:8:6:9*

5:6:2:9:4

باز هم مشاهده می شود که همان قاعده برقرار است. ۱۴=۹+۵ و چون ۱۴ از ۱۰ بزرگتر است ۴ نوشته شده و ۱ واحد به بعدی اضافه گردیده است. و همچنین است در بقیه‌ی رقم ها.

دستگاه ده‌دهی (دسیمال) همان دستگاهی است که به طور معمول از آن استفاده می‌کنیم و اعداد را در مبنای ده به‌کار می‌بریم. در این دستگاه هر رقم کمتر از ۱۰ می‌باشد. به طور مثال می‌توان به صورت زیر چنین نوشت:

$(456)_{10} = 4\times10^2+5\times10^1+6\times10^0$ *Decimal Number*

تذکر: عدد زیر پرانتز به معنای مبنای آن عدد است و هرگاه که گذاشته نشود در ریاضیات به معنای مبنای ۱۰ است.

در دستگاه دودویی همانند دستگاه دهدهی، هر رقم کمتر از عدد ۲ می‌باشد. نحوه‌ی محاسبه‌ی آن هم در مثال زیر نمایش داده شده است.

$(101001)_2=1\times2^5+0\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0$ *Binary Number*

و برای اعداد در دستگاه شصت‌شصتی داریم:

$(21^{\circ}28')_{60} = 21\times60^1 + 28\times60^0$ که به عبارتی همان مقدار ثانیه در کل است در مبنای ۱۰

در حالت کلی می‌توان هر عدد را با این روش در مبنای هر عدد دیگر نمایش داد. که در حالت کلی می‌توان نوشت:

$(a_na_{n-1}\dots a_2a_1a_0)_m = a_n\times m^n+a_{n-1}\times m^{n-1}+\dots+a_2\times m^2+a_1\times m^1+a_0\times m^0 = (b_tb_{t-1}\dots b_2b_1b_0)_{10}$

$a_i<m \quad (a_i = 0,1,2,3,\dots,m-1)\ ,\ i = 0,1,2,3,\dots,n-1,n$

$b_i<10 \quad (b_i = 0,1,2,3,\dots,9)\ ,\ i = 0,1,2,3,\dots,t-1,t$

ملاحظه می‌شود که با روش ارائه شده به راحتی می‌توان اعداد مختلف را با صفر و یک بیان نمود و بنابر این قابلیت ذخیره‌سازی در کامپیوتر را دارا خواهند شد.

کمی بیشتر در رابطه با دستگاه اعداد در کامپیوتر:

به هر یک از صفر یا یک‌هایی که در کامپیوتر ذخیره می‌شود یک بیت (*Bit*) گویند. به عبارت بهتر هربیت همان رقم‌های (*digits*) عدد در مبنای ۲ است. به طور مثال : $(۱۱۱۰)_۲ = (۱۴)_{۱۰}$ که در مبنای ۱۰ دارای ۲ رقم است و در مبنای ۲ دارای ۴ رقم و به این عدد ۴رقمی یک عدد ۴بیتی می‌گوییم.

نکته: گاهی تعداد ارقام با تعداد بیت متفاوت است. مثلاً: $(۰۰۱۱)_۲ = (۳)_{۱۰}$ که در آن عدد باینری دارای ۴ بیت و ۲ رقم است. صفر پشت عدد خوانده نمی‌شود، اما در کامپیوتر به معنای خانه‌ی حافظه است.

نکته: از آنجایی که هر بیت بسیار واحد کوچکی برای بیان حافظه است از واحدهای دیگری نیز برای بیان حافظه بهره می‌گیرند. به هر ۸ بیت ۱ بایت (*Byte*) می‌گویند. و به هر $(\approx۱۰^۳)$ $۲^{۱۰}$ بایت ۱ کیلوبایت (*KB*) –این علامت را با علامت *Kb* به معنای کیلو بیت اشتباه نگیرید- می‌گویند. و به همین ترتیب: به هر $(\approx۱۰^۳)$ $۲^{۱۰}$ کیلو بایت ۱ مگابایت (*MB*) و به هر $(\approx۱۰^۳)$ $۲^{۱۰}$ مگابایت ۱ گیگابایت (*GB*) گویند.

عملگرهای دیجیتالی چیست؟ *(Logical Operations)*

عملگرهایی که اینجا مورد بررسی قرار می‌دهیم سه عملگر پرکاربرد در برنامه نویسی است: و (*AND*)، یا (*OR*)، نه (*NOT*)

این عملگرها بین حداکثر دو ورودی عملیات خود را انجام می‌دهند. برای درک بهتر به مثال‌های زیر توجه کنید:

$AND(0,1) = (0) . (1) = 0$

$OR(1,1) = (1) + (1) = 1$

$NOT(1) = (1)' = 1 = 0$

همان طور که می‌بینید NOT تنها یک ورودی می‌گیرد و OR و AND دو ورودی می‌گیرند. این ورودی‌ها می‌توانند یک بیت و یا بیش از یک بیت باشد. اگر ورودی بیش از یک بیت باشد، به ترتیب هر بیت وارد شده و خروجی آن‌ها نیز به ترتیب نوشته می‌شود. به عبارت دیگر:

$O(a_n a_{n-1} \dots a_1 a_0) = O(a_n)\, O(a_{n-1}) \dots O(a_1)\, O(a_0)$

$O(a_n a_{n-1} \dots a_1 a_0, b_n b_{n-1} \dots b_1 b_0) = O(a_n, b_n)\, O(a_{n-1}, b_{n-1}) \dots O(a_1, b_1)\, O(a_0, b_0)$

تذکر: O از ابتدای لغت *Operator* هر یک از عملگرهای معرفی شده می‌تواند باشد.

به عنوان مثال:

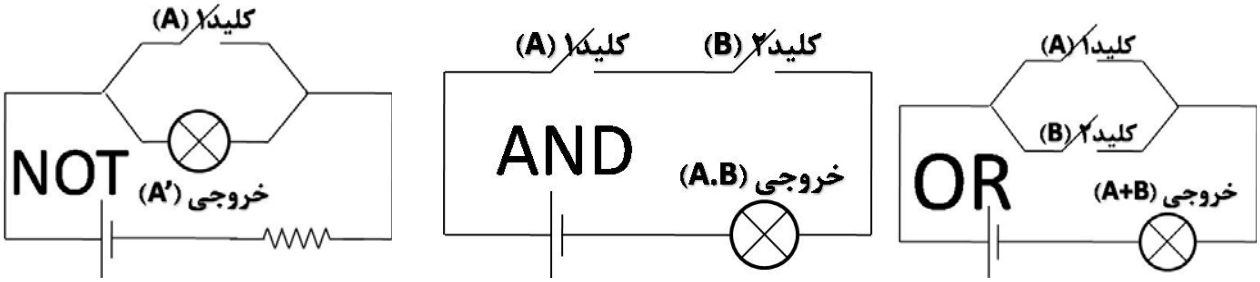
$AND(0100101,111) = AND(0100101,0000111) = AND(0,0)\, AND(1,0)\, AND(0,0)\, AND(0,0)$
$AND(1,1)\, AND(0,1)\, AND(1,1) = 0000101$

نکته: اگر در عملگرهای با دو ورودی تعداد ارقام یکی از ورودی‌ها کمتر از تعداد ارقام ورودی دیگر بود با اضافه کردن صفر در پشت آن می‌توان ارقام آن ها را مساوی کرد.

از این رو کافی است بدانیم که این عملگرهای بین ۲ یا ۱ بیت چگونه خروجی را تغییر می‌دهند.

NOT		AND			OR		
A	A'	A	B	$A.B$	A	B	$A+B$
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1
		1	0	0	1	0	1
		1	1	1	1	1	1

به هر یک از این جداول «جدول درستی» می‌گویند. از آنجایی که این جداول بر مبنای صفر ویک چیده شده، می‌توان به راحتی با کلیدهای الکترونیکی مداری ساخت که جداول بالا را پاسخ دهد. یعنی به ازای هریک از ورودی‌ها، خروجی مورد نظر تولید شود. ابتدا همانند اولین مداری که با آن مقادیر صفر و یک توضیح داده شد مداری رسم می‌کنیم که پاسخ‌های مورد نظر در هر جدول را ایجاد کند.



ملاحظه می‌شود که اگر کلیدها را با توجه به جداول (۰ = کلید باز، ۱ = کلید بسته) قرار دهیم، لامپ خروجی مورد نظر را تولید می‌کند (۰ = خاموش، ۱ = روشن). کلیدهای استفاده شده در اینجا کلیدهای مکانیکی است. اما ما نیاز به ایجاد کلیدهای الکترونیکی داریم، تا بتوانیم به طور اتوماتیک این کار را انجام دهیم.

ترکیب عملگرها:

عملگرها می‌توانند در کنار هم قرار گرفته و عملگر جدیدی را ایجاد کنند. به عنوان مثال ترکیب دو عملگر *NOT* و عملگر *AND* عملگر *NAND* و ترکیب دو عملگر *NOT* و *OR* عملگر *NOR* را ایجاد می‌کند. حال می‌خواهیم جدو درستی این عملگرها را رسم کنیم. می دانیم:

$$NAND(A,B) = NOT(AND(A,B))$$
$$NOR(A,B) = NOT(OR(A,B))$$

به وضوح مشخص است که همان جداول *OR* و *AND* با *NOT*کردن خرووجی‌ها جدول درستی این دو مدار را تشکیل می‌دهد.

NAND			*NOR*		
A	*B*	*A.B*	*A*	*B*	*A+B*
0	*0*	*1*	*0*	*0*	*1*
0	*1*	*1*	*0*	*1*	*0*
1	*0*	*1*	*1*	*0*	*0*
1	*1*	*0*	*1*	*1*	*0*

به عکس هم می‌توان گفت یعنی:

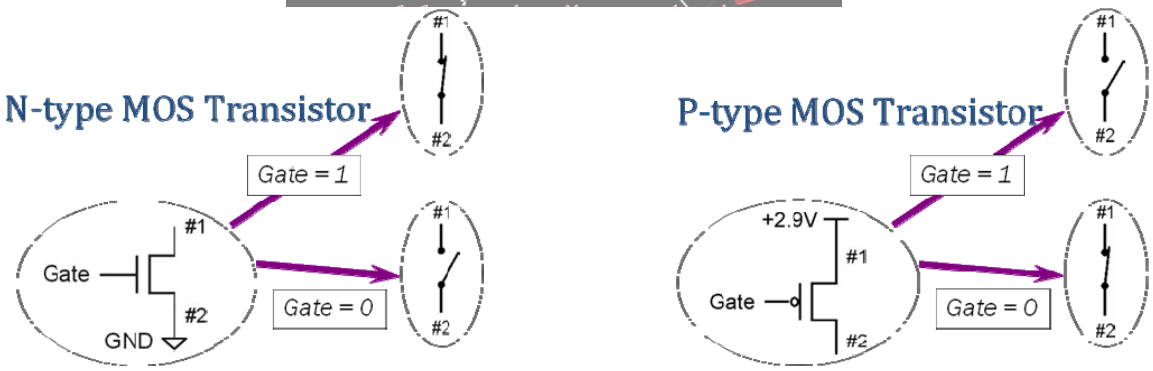
$$AND(A,B) = NOT(NAND(A,B))$$
$$OR(A,B) = NOT(NOR(A,B))$$

بسته به آن که طراحی کدام یک راحت‌تر است ابتدا یکی را طراحی کرده، سپس با قرار دادن یک *NOT* دومی نیز طراحی می‌شود.

تمرین: مداری طراحی کنید که جدول درستی بالا را پاسخ دهد.

کلیدهای الکترونیکی:

کلیدهای الکترونیکی دیجیتال همان ترانزیستورها هستند. برای آشنایی با کار آن‌ها به شکل‌های زیر توجه کنید.



این دو نوع کلید (ترانزیستور نوع $\mathcal{N}$ و ترانزیستور نوع $\mathcal{P}$) که تکنولوژی آن به MOS (مخفف *Metal Oxide Semiconductor*) مشهور است، با استفاده از نیمه هادی‌ها طراحی می‌شود.

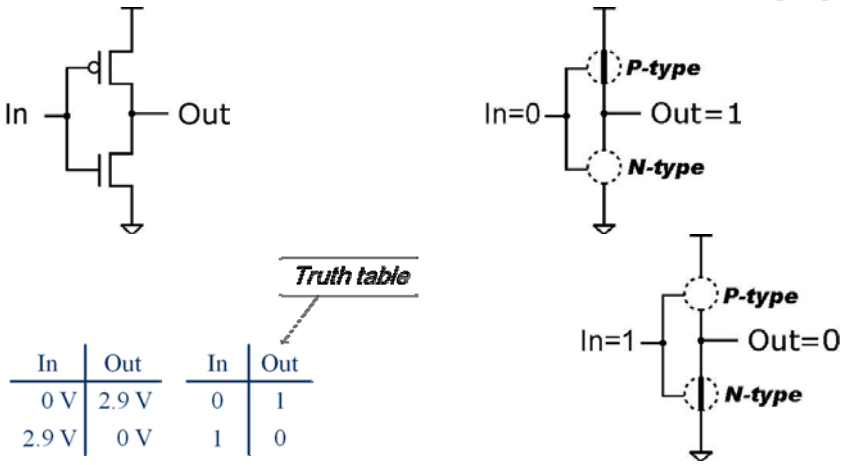
شکل به صورت گویا عملکرد این دو نوع کلید را توضیح داده است. مقدار *Gate* همان ولتاژهای بیان شده می‌باشد که به صورت منطقی نوشته شده‌است.

ریزپردازنده‌های (*Microprocessors*) امروزی دارای میلیون‌ها ترانزیستور هستند. به عنوان مثال:

- *Intel Pentium II : 7 million*
- *Intel Pentium III : 28 million*
- *Intel Pentium IIII: 54 million*

از اینجا می‌توان به اهمیت کار این کلیدها پی برد. همانگونه که در تاریخچه بیان شد، ترانزیستورها تحولی بزرگ در صنعت سخت افزاری کامپیوتر به وجود آوردند، مدارهای بسایر بزرگ را، بسیار کوچک کردند. امروزه در مدارهای-مجتمع نیز از ترانزیستورها استفاده می‌شود، که به صورت فشرده در یک قطعه جاسازی می‌شود. همین CPUهای بیان شده ابعادی در حدود ۵ سانتیمتر در ۵ سانتیمتر دارند(با این تعداد ترانزیستور).

با این کلیدهای الکترونیکی نیز می‌توان مدارهای NOT ،OR ،AND را طراحی کرد. در اینجا تنها مدار NOT را طراحی کرده و باقی به عنوان تمرین مطرح می‌شود.



Truth table

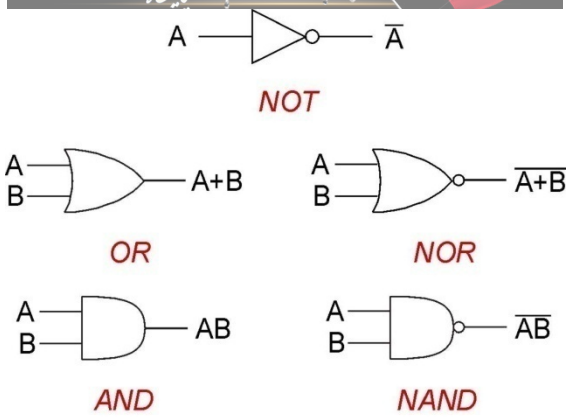
In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0

تمرین: گیت‌های AND و OR را با ترانزیستورها طراحی کنید.

گیت‌ها:

برای ایجاد مدارهای کاربردی تر مدارهای گفته شده را به عنوان پایه در نظر گرفته و مدارهای پیچیده‌تر را طراحی می‌کنند. این مدارهای پایه (مدار عملگرهای گفته شده) را گیت می‌نامند. تا کنون گیت‌های زیر را به کاربرده‌ایم.



یادآوری: گیت $NAND = NOT(AND(A,B))$ به عبارت بهتر جدول درستی AND همان جدول درستی $NAND$ است به شرطی که در خروجی به جای ۰ها ۱ و به جای ۱ ۰ گذاریم.

یادآوری: گیت $NOR = NOT(OR(A,B))$ به عبارت بهتر جدول درستی OR همان جدول درستی NOR است به شرطی که در خروجی به جای ۱ها ۰ و به جای ۰، ۱ گذاریم.

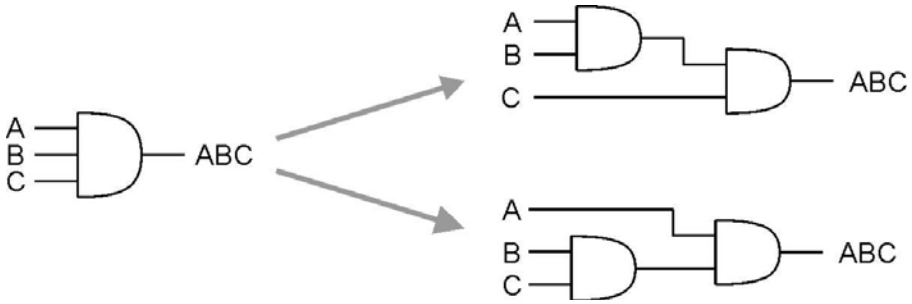
با استفاده از این گیت‌ها می‌توان به ازای هر ورودی خروجی دلخواه را ایجاد کرد. برای شروع می‌خواهیم ورودی گیت AND را بیشتر از ۲ پارامتر کنیم. بدین معنا که ۲ پارامتر ورودی را ANDکند و جواب آن را با پارامتر سوم AND کند و سپس جواب را با پارامتر چهارم AND کند و الی آخر. جدول درستی ۳ پارامتره‌ی این مدار به صورت زیر است:

A	B	C	$O=A.B$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

حال برای رسم مدار این جدول از مدار AND بهره می گیریم، زیرا با توجه به تعریف بالا داریم:

$$AND(a_0,a_1,a_2,\ldots,a_{n-1},a_n) = AND(AND(\ldots(AND(AND(a_0,a_1),a_2),\ldots),a_{n-1}),a_n)$$

پس به تعداد ورودی‌ها منهای یک AND باید انجام پذیرد. از این رو به همین تعداد مدار AND نیازمندیم و می‌توان شکل زیر را در جواب این جدول رسم کرد.



این شکل تعمیم یافته‌ی مدار AND است. به طور مشابه می‌توان برای دیگر مدارها نیز عمل کرد. و همچنین به طور مشابه برای تعداد بالاتر ورودی‌ها نیز می‌توان عمل کرد.

نکته: به جز گیت NOT همه‌ی گیت‌ها می‌توانند بیشمار ورودی داشته باشند، اما همه‌ی گیت‌ها (از جمله NOT) تنها یک خروجی دارند. برای رسم مدارهای پیچیده از تعمیم یافته‌ی مدارها نیز بهره می‌گیریم.

اما برای رسم هرجدول درستی دیگر می‌توان قواعدی وضع کرد. به همین جهت به مدارهایی که تاکنون داشته‌ایم به گونه‌ای دیگر می‌نگریم:

مدار AND تنها در صورتی خروجی‌اش یک می‌باشد که تمام ورودی‌های آن یک باشد.

در مدار AND می‌توان ورودی‌های یک را در نظر نگرفت.

در مدار AND یک ورودی صفر کافی است برای خروجی صفر.

مدار OR تنها در صورتی خروجی‌اش صفر است که تمام ورودی‌های آن صفر باشد.

در مدار OR می توان ورودی‌های صفر را در نظر نگرفت.

در مدار OR یک ورودی یک کافی است برای خروجی یک.

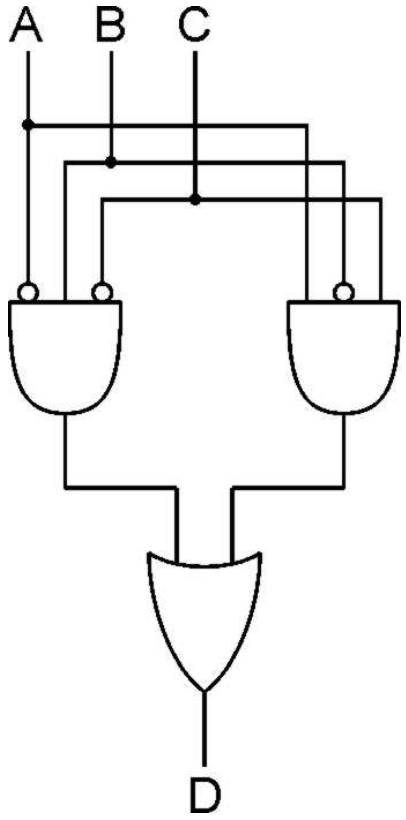
مدار NOT می‌تواند ورودی و یا خروجی را $Invert$ کند. یعنی اگر در ورودی یک گیت مدار NOT بگذاریم، به جای یک صفر و به جای صفر یک وارد گیت می‌شود. و اگر در خروجی NOT قرار گیرد، خروجی نیز $Invert$ می‌شود.

یک روش ساده برای رسم جداول درستی آن است که از خاصیت مدار AND استفاده کنیم. چون مدار AND تنها در یک حالت دارای یک می‌باشد، با توجه به جدول درستی، یک‌های جدول را پیدا می‌کنیم و به ازاء هر کدام گیت AND و NOT را به گونه‌ای قرار می‌دهیم که تنها در حالت مفروض خروجی این گیت یک شود. بعد از آن کافی است ورودی‌های مشترک را به هم وصل کرده و خروجی‌ها را به یک مدار OR وصل کنیم. بدین ترتیب هر یک از گیت‌ها که یک شود خروجی مدار OR -که همان خروجی کل مدار رسم شده می‌باشد- نیز یک می‌شود و در بقیه‌ی حالات خروجی صفر است.

به عنوان مثال مدار جدول درستی روبهرو را رسم می‌کنیم:

A	B	C	D
0	0	0	0
0	0	1	0

0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



تذکر: برای راحتی در نمایش به جای آن که گیت NOT رسم شود از یک حباب (○) استفاده می‌شود.

نکته: مدار طراحی شده برای جدول درستی انحصاری نبوده و می‌تواند بینهایت حالت داشته باشد.

تمرین: روشی بیابید که با گیت OR به جای استفاده از گیت AND بتوان جدول درستی بالا را به مدار معادلش رساند.

(راهنمایی: از این خاصیت مدار OR بهره بگیرید که تنها در یک حالت صفر می‌شود، و آن حالتی است که تمامی ورودی‌های گیت صفر باشد.)

تمرین: روشی بیابید که با گیت NOR به جای استفاده از گیت AND بتوان جدول درستی بالا را به مدار معادلش رساند.

ساختمان طراحی کامپیوتر:

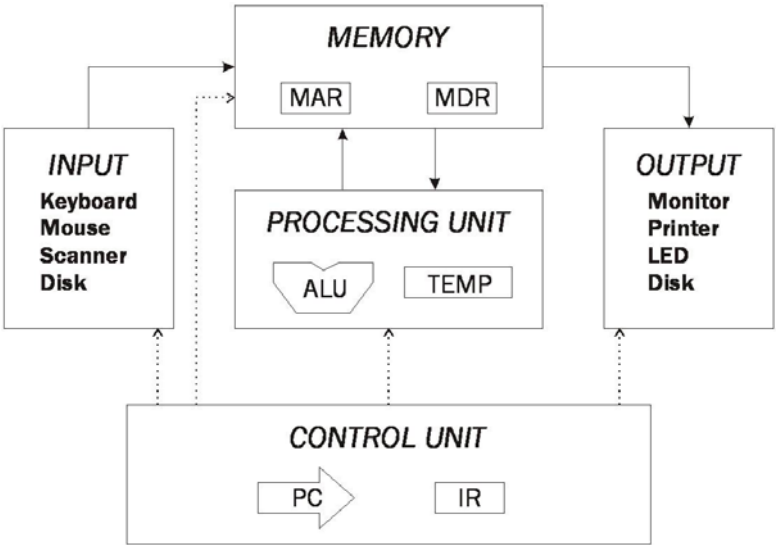
حال که کمی با اساس کار کامپیوتر آشنا شدیم و معنای استفاده از اعداد باینری (سامانه‌ی دودویی) را فهمیدیم، سؤال این است که چگونه می‌توان این مدارها را در کنار هم قرار داد و یک کامپیوتر با کاربری‌های عملیاتی تر ایجاد کرد.

گیریم ما دانستیم که می‌توان مداری داشت که با دادن ورودی‌های صفر و یک و دادن دو عدد باینری مجموع باینری آن را به ما بازگرداند، اما این کار که سرعت کار ما را کمتر می‌کند. از این رو سامانه‌ای نیاز است تا این مدارها را کنترل کند و در نهایت جواب دسیمال و معمولی را به ما باز گرداند. از طرفی می‌بایست کار با این دستگاه به قدری آسان باشد که افراد غیر حرفه‌ای نیز در استفاده از آن مشکلی نداشته باشند.

از این رو برای این اتصال یک «مدل» طراحی می‌شود. این مدل طرز قرارگیری عناصر مختلف در کنار هم را نشان می‌دهد.

مدل وون نومن:

در تاریخچه‌ی کامپیوتر با سیر تکامل کامپیوترها آشنا شدید. دیدیم که پیشرفت کامپیوترها در ابتدا جهت بالا بردن سرعت محاسبات بوده است. وون‌نومن (*John von Neumann*) در سال ۱۹۴۵ میلادی گزارشی منتشر کرد درباره‌ی سامانه‌های قابل برنامه‌سازی؛ این گزارش به مدل وون‌نومن مشهرو است، که کلیات آن در شکل زیر نمایش داده شده است:



Input: ورودی

Output: خروجی

Memory: حافظه

Processing Unit: واحد پردازنده

Control Unit: واحد کنترل کننده

فصل سوم

اصول برنامه نویسی

حل مسائل در کامپیوتر:

همانگونه که می‌دانید امروزه کامپیوترها تنها برای انجام محاسبات ساده‌ی ریاضی مورد استفاده قرار نمی‌گیرند. کاربری کامپیوترها در مدل‌سازی‌ها، امروزه بسیار زیاد است. اما چگونه می‌توان این مدل‌ها را به کامپیوتر فهماند؟ و چگونه می‌توان پاسخ‌های مختلف و دلخواه را به دست آورد؟ در برنامه‌نویسی با دادن یک سری ورودی می‌توان یک سری خروجی را دریافت کرد. نحوه‌ی ارتباط بین ورودی‌ها و خروجی‌ها کار برنامه نویس است. از این رو برنامه نویس باید هم به ورودی‌ها تسلط کامل داشته باشد، هم به خروجی‌ها، و هم به توابعی که می‌توانند ارتباطی بین این دو برقرار کنند. به عنوان مثال: می‌خواهیم دو عدد را با عملیات جمع، در هم ضرب کنیم. برای این مسئله می‌بایست به ورودی که دو عدد با خاصیت‌هایی (اعشاری، منفی، مثبت، نماد علمی،...) و خروجی (که خاصیت ضرب است) تسلط داشت. و از همه مهمتر به ارتباط بین عملیات جمع و ضرب! و چگونگی به کارگیری آن‌ها.

شروع برنامه نویسی با طرح مسئله است. مثلاً: برنامه‌ای بنویسید که *sin* یک عدد را حساب کند، برنامه‌ای بنویسید که مانده‌ی حساب بانکی را مشخص کند، برنامه‌ای بنویسید که شطرنج را شبیه سازی کند، برنامه‌ای برای مدیریت کارت سوخت بنویسید، بازی *fifa* طراحی کنید و ... .

همانگونه که می بنید مسئله می‌تواند از هر نوعی باشد، آن چیزی که مهم است پیدا کردن ورودی و خروجی و رابطه‌ی بین آن دو است.

نکته: در بعضی موارد ممکن است راه حل ما با امکاناتمان هم‌سو نباشد در این موارد امکانات نیز باید در نظر گرفته شود.

مثلاً وقتی که ما امکان ضرب دو عدد را نداریم، نباید با دنبال استفاده از این خاصیت باشیم.

الگوریتم نویسی:

الگوریتم روش حل مسئله است. در الگوریتم نویسی ابتدا فعالیت هایی که لازم است را مشخص میکنیم، سپس آن-ها را ساده و ساده‌تر می‌کنیم تا با توجه به امکاناتمان آن مسئله را حل کنیم. یک الگوریتم دارای یک شروع است، اما اگر به پایان نرسد آن را الگوریتم نمی‌دانیم. یک الگوریتم هنکامی الگوریتم است که شروع و پایان داشته باشد. اما ممکن است الگوریتم ما کار مورد نظر ما را انجام ندهد، در این صورت باید اشکالات آن را پیدا و حل کرد.

مراحل کلی الگوریتم نویسی در مثالی مطرح می‌گردد. تعمیم این حالت برای دیگر مسائل بر عهده‌ی شماست.

صورت مسئله: می‌خواهیم تعداد تکرارهای یک کاراکتر که کاربر به ما می‌دهد را در یک متن بیابیم و نتیجه را روی صفحه‌ی نمایش چاپ کنیم. مثلاً می‌خواهیم تعداد تکرار حرف *u* را در عبارت زیر بیابیم:

“We wish to count the number of occurrences of a character in a file. The character in question is to be input from the keyboard; the result is to be displayed on the monitor.”

ورودی‌های ما: متن بالا و حرف *u* می‌باشد. و خروجی ما یک عدد صحیح است که تعداد تکرارهای حرف *u* در متن می‌باشد. ارتباط بین ورودی وخروجی، پیدا کردن حرف *u* در متن و شمردن آن است. متن ممکن است تغییر کند، بنابراین یک نام کلی برای آن انتخاب می‌کنیم تا همه‌ی این متن‌ها را پوشش دهد. به عنوان مثال *string* و

همچنین حرف *u* لزوماً ثابت نمی‌باشد و با توجه به ورودی صفحه کلید تغییر می‌کند. برای این ورودی نیز نامی برمی‌گزینیم، مثلاً *input*.

حال صورت مسئله به این شکل در می‌آید: می‌خواهیم تعداد تکرار *input* را در *string* بشمریم که در آن *input* از صفحه‌کلید گرفته می‌شود و *string* متن مورد نظر است (*input* یک کاراکتر است و قابل ذخیره و *string* نیز یک متن است و قابل ذخیره).

قبل از شروع به حل برخی سؤالات را جهت فهم بهتر مسئله باید بپرسیم. مثلاً آیا حروف کوچک و بزرگ در تفاوتند؟ متن مورد نظر در کجا قرار دارد؟ کی می‌فهمیم که به پایان متن رسیده‌ایم یا نه؟ چگونه می‌توان مقدار خروجی را چاپ کرد؟

همانطور که می‌بینید برخی از این سؤالات می‌بایست از سمت مطرح کننده‌ی سؤال پاسخ داده شود. اگر امکان پرسش نبود، یک حالت را در نظر گرفته و سپس آن را برای کاربر می‌نویسیم. اما برخی دیگر از سمت خود برنامه نویس باید پاسخ داده شود. مانند این که چگونه می‌توان روی صفحه کلمه‌ای را نشان داد.

پس از روشن شدن مسئله، حل آن را در پیش می‌گیریم. بدین جهت می‌بایست مسئله به جزءهای ساده‌تر تقسیم گردد. و آن جزءها نیز به جزءهای ساده‌تر تقسیم گردد، تا آنجا این کار را ادامه می‌دهیم تا با توجه به امکاناتمان (زبان برنامه نویسی مورد استفاده) بتوانیم آن جزء را پیاده کنیم.

برای این که بتوانیم مسائل را جزئی تر کنیم، سه روش وجود دارد. (ترتیبی، شرطی، حلقه)

۱. ترتیبی:

- در این جا به ترتیب هر یک از اعمال را می‌نویسیم و با پیکان جهت حرکت را مشخص می‌کنیم. به عنوان مثال: ورودی را بگیر ← فایل را بگیر ← ورودی را در فایل بیاب ← ...
- نوشتن ترتیبی در بیان توابع کاربرد زیادی دارد. به طور مثال: کار۱(متغیر۱، متغیر۲،...) ← کار۲(متغیر۱، متغییر۲،...) ← ...

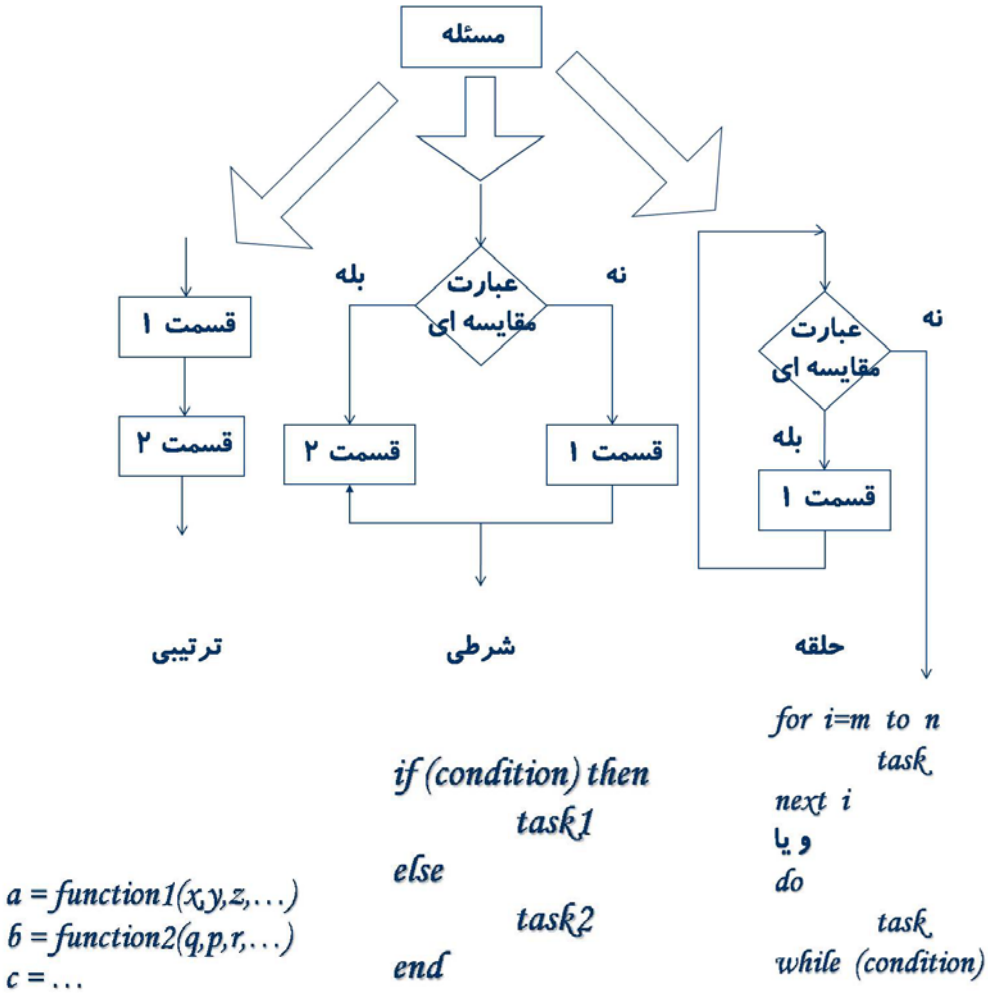
۲. شرطی:

- عبارت شرطی عبارتی است که در آن از «اگر» استفاده شود. در جواب «بله» یک مسیر را می‌رود و در جواب «نه» مسیر دیگری را می‌رود. به طور مثال: اگر ورودی برابر کاراکتر بود برو جلو در غیر این صورت برگرد.
- نگارش شرط‌ها با نوشتن «اگر(شرط) (کار۱) درغیراینصورت (کار۲) پایان» می‌باشد.

۳. حلقه:

- گاهی لازم است کاری را چند بار تکرار کنیم، در این مواق از حلقه استفاده می‌کنیم. به طور مثال: کاراکترها را ببین «تا وقتی که» به پایان متن برسی.
- در همان مورد بالا اگر یکی از دو کاری که انجام می‌دهیم بازگشت و یا پرش به وظایف قبلی باشد یک حلقه تشکیل می‌دهیم. حلقه را با دو عبارت نمایش می‌دهند «*n*بار "کار۱" را انجام بده» و یا «"کار ۱" را انجام بده تا وقتی که "شرط" برقرار است».

برای آن که موارد بالا بهتر قابل درک باشد، آن‌ها را به صورت نمودار نشان می‌دهند. اما برای نگارش آن‌ها نیز از عباراتی استفاده می‌کنند تا بتوانند برنامه را به صورت نوشته نیز بیان کنند(که در بالا اشاره شد). به این نوشته‌ها شبه‌کد می‌گویند و با آن راه حل مسئله را بسیار نزدیک به زبان برنامه نویسی می‌توان بیان کرد.



نکته: در حالت ترتیبی هر یک از توابع (*function*) یک مقدار خروجی دارد که در پارامتر پشت مساوی ذخیره می‌شود.

تعریف: هر متغیر یک حرف یا یک کلمه‌ی پیوسته است که مقداری را می‌تواند در خود ذخیره کند. مثلاً: *string*، *input*، *a*، *b*، *InputValue*، *value_input*، *input1*، و هر عبارتی که با استفاده از حروف و اعداد و _ تشکیل شود. برای آن که پارامترها با حروفی که در برنامه به کار می‌رود اشتباه نشود حروفی که می‌خواهیم چاپ کنیم و یا عین عبارتی را بخواهیم بیان کنیم داخل " ویا ' قرار می دهیم. به طور مثال:

string = "We wish to count the number of occurrences of a character in a file."

و یا:

string = 'We wish to count the number of occurrences of a character in a file.'

نکته: برای نمایش آن که مقداری داخل یک متغیر می‌ریزیم از = استفاده می‌کنیم (در بعضی دیگر از نگارش ها برای این منظور از← استفاده می‌کنند). بدین معنا که مقدار سمت راست در پارامتر سمت چپ ذخیره می‌شود.

نکته‌ی مهم این که ابتدا تمامی عملیات‌های سمت راست مساوی انجام می‌شود و بعد مقدار نهایی در پارامتر سمت چپ ذخیر می‌شود. به عنوان مثال:

$a = a+2+b$

در اینجا ابتدا مقدار a که قبل از این خط در a ذخیر شده با مقدار b و عدد 2 جمع می‌شوند و مجدداً در a ریخته می‌شود.

نکته: در حلقه عبارت $next\ i$ معادل $i = i+1$ می‌باشد (به جای i هر پارامتر دیگری می‌توان گذاشت). یعنی به مقدار i یک واحد افزوده می‌شود. بنابراین حلقه‌ی for را می‌توان به صورت زیر نیز بیان کرد:

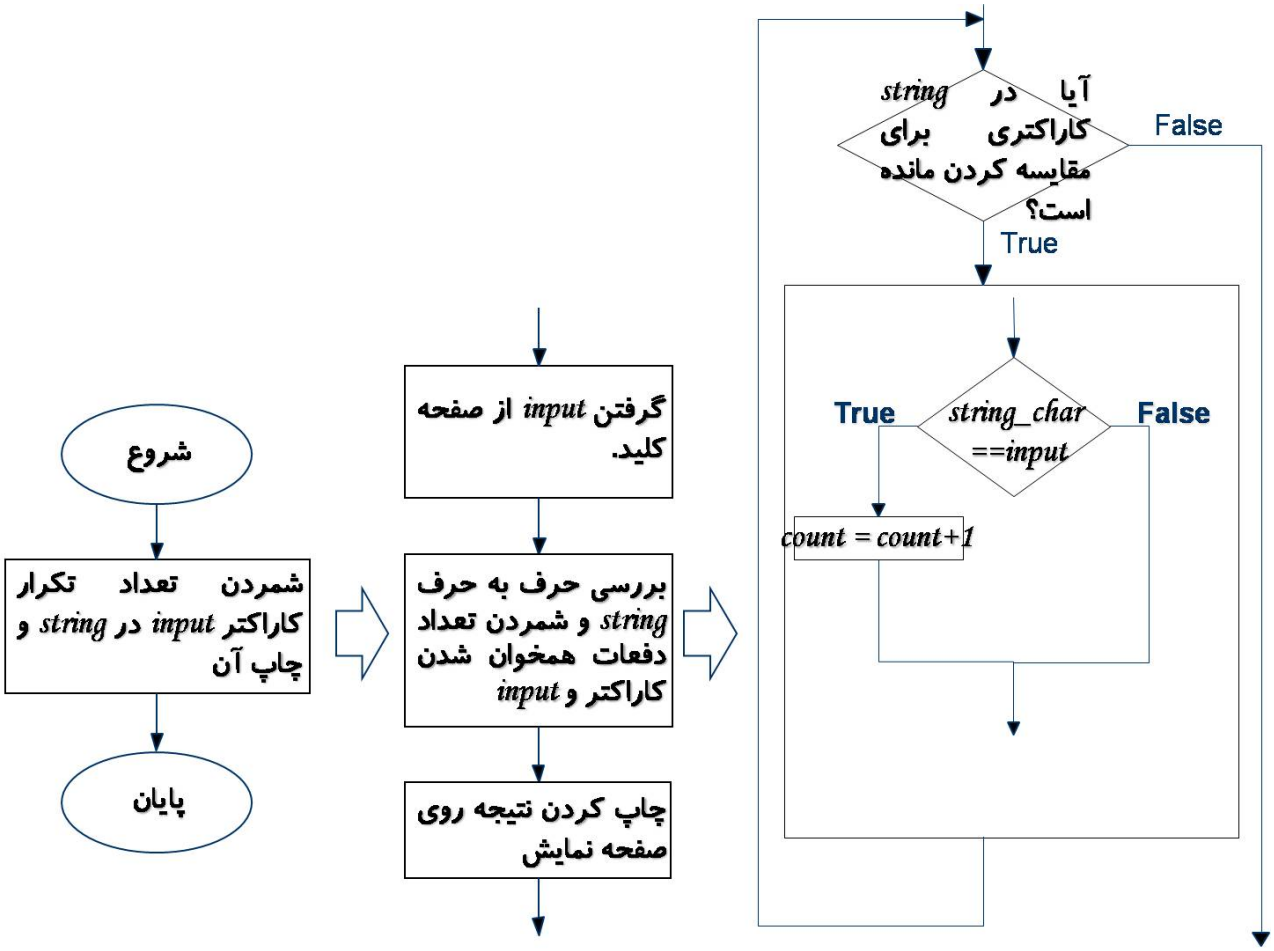
```
i = m
do
    task
    i=i+1
while(i =<n)
```

یادآوری:

کوچکتر مساوی<=	کوچکتر <
بزرگتر مساوی=>	بزرگتر >
نامساوی<> یا =!	
مساوی= یا ==	

یادآوری بالا جهت مقایسه کردن بین پارامترها کاربرد دارد. هر شرط (*condition*) از این پارامترها بهره می‌گیرد.

ادامه‌ی راه‌حل:



می‌توان نمودار بالا را بدین شکل نیز نوشت:

```
begin
input = read()              //یعنی از ورودی یک کاراکتر بخوان و در طرف دوم تساوی ذخیره کن
string = 'We wish to count the number of occurrences of a character in a file.  The character in
        question is to be input from the keyboard; the result is to be displayed on the monitor.'
count = count_char (string,input)      //باید این تابع برای برنامه تعریف شود، چون قبلاً تعریف نشده
write(count)                //یعنی روی صفحه مقدار داخل پرانتز را چاپ کن
end
```

حال می‌بایست تابع *count_char* را نیز تعریف کنیم:

```
count(string,input)
        begin
        count=0
        i=1
        do
                string_cahr = get_char_string(string,i)     //این تابع که یکی یکی کاراکترهای متن را
                if (string_char == input)                   //می‌دهد نیز باید تعریف شود
                        count = count+1                     //اما متناسب با هر زبانی به گونه‌ای می‌باشد
                end
                i=i+1
        while(string_char != null)
        return count   //یعنی پارامتری که مساوی با این تابع می باشد مقدار بازگشت شده را می‌گیرد
        end
```

نکته: در شرط استفاده از *else* اختیاری است.

نکته: قبلاً بیان شد که پایان متن یک کاراکتر با کد ۰ می‌گذارند. در برخی زبان‌ها در ابتدای هر متن طول آن را مشخص می‌کنند. برای روشن کردن این موضوع در کنار شبه‌کد خود این نکات باید ذکر شود.

می‌توان ادعا کرد که مسئله حل شده است. برای تابع پایانی هم مثلاً در زبان *C* می توان چنین عمل کرد:

```
get_char_string(string,i)
{
        return string[i-1]
}
```

نکته: خروجی هر تابع (*return value*) مقداری است که به سمت چپ مقدارده (=) اختصاص می‌یابد. مثلاً:

```
C = get_char_string('computer',5)
```

در اینجا مقدار *C* حرف *'u'* می‌باشد که همان حرف پنجم این عبارت (*computer*) است.

خطاها و رفع آن:

خطاها همان اشکالات برنامه می‌باشد. مثلاً برنامه‌ای که نوشته‌ایم به درستی عمل نمی‌کند. در مثال گفته شده، ممکن است مقدار خروجی یک واحد بیشتر از تعداد مورد نظر باشد و یا این که اصلاً خروجی همواره یک عدد باشد. برخی دیگر از خطاها این است که به ازاء ورودی‌های مختلف دچار اشکال شود، در مثال بالا اگر *string* مقداری نداشته باشد چنین مشکلی پیش خواهد آمد. برای حل این مشکلات ابتدا باید دید مشکلی هست یا نه؟ اگر هست در کجای برنامه قرار دارد؟برای این منظور برنامه‌ی خود را خط به خط بررسی و اجرا می‌کنیم و در هر جا که مشکلی دیده شد آن را حل می‌کنیم. به طور کلی مراحل زیر برای مطمئن کردن برنامه مورد نیاز است:

Errors & Tracing:

برای یافتن خطاهای برنامه بایستی ابتدا خود برنامه را با ورودی‌های مختلف مورد بررسی قرار دهیم.

نکته: در اینجا باید دقت کنیم که حالات خاص حتماً چک شود، مثلاً در مثال قبل ورودی به جای یک کاراکتر یک متن باشد.

از انواع خطاها می‌توان به «خطاهای نوشتاری، خطاهای منطقی، خطاهای ورودی» اشاره کرد. خطاهای نوشتاری را معمولاً خود زبان برنامه نویسی مشخص می‌کند. اما در دو مورد بعدی خود برنامه‌نویس باید دقت داشته باشد.

در *tracing* (ردیابی) شما به جای کامپیوتر شروع به اجرای برنامه می‌کنید. برای آن که بتواند به درستی برنامه را ردیابی کنید، می‌بایست جدولی از پارامترها تشکیل دهید و در هر قسمت مقادیر آن‌ها را بنویسید. برای یک‌پارچه شدن و اشتباه کمتر بهتر از در برخی از خطوط حساس علامت گذاشته و هر وقت به آن علامت رسیدید تمامی مقادیر را در جدول یادداشت کنید.

خط به خط اجرا کردن برنامه به شما این امکان را می‌دهد تا در هر لحظه متغیرهای خود را بررسی کنید و در هر جا با مقداری که تصور می‌کردید متفاوت شد، آنجا را به عنوان محل وقوع خطا در نظر بگیرید.

بسیار مهم است که در مرحله‌ی ردیابی بتوانید به خوبی خطاهای منطقی را بیابید، بعد از آن با دادن ورودی‌های غیرعادی و نامناسب بازهم برنامه را چک می‌کنیم. اگر در جایی دچار مشکل شدیم، می‌بایست این حالات خاص را نیز مشخص کنیم.

Debugging:

در این مرحله با توجه به خطاهایی که در مرحله‌ی قبل پیدا شد، شروع به رفع تک تک آن‌ها می‌کنیم. اگر خطا در دادن خروجی بود، اشکال را پیدا کرده و پس از رفع آن دوباره مرحله‌ی قبل را تکرار می‌کنیم. اگر اشکال از ورودی-ها باشد می‌توان با عبارات شرطی حالات خاص ورودی را از برنامه دور ساخت.