فصل اول

روند توسعه معماری کامپیوتر

سازمان و معماری کامپیوتر Computer Organization and Architecture

در تشریح سیستم‌های کامپیوتری باید بین سازمان و معماری کامپیوتر تفاوت قایل شد. هر چند ارایه تعریف دقیق برای آن‌ها مشکل است، ولی در مورد موضوعات تحت پوشش آن‌ها توافق عام وجود دارد.

معماری کامپیوتر به صفاتی از سیستم اشاره می‌کند که قابل رویت به‌وسیله برنامه‌ساز می‌باشد. به عبارت دیگر، صفاتی که بر اجرای منطقی برنامه اثر مستقیم دارد. به عنوان مثال، از جمله صفات معماری کامپیوتر می‌توان به مجموعه دستورالعمل‌ها، تعداد بیت‌های اختصاص یافته به نمایش انواع داده‌ها، مکانیزیم ورودی و خروجی و تکنیک‌های نشانی‌دهی کلمات حافظه اشاره نمود.

سازمان کامپیوتر به واحدهای عملیاتی و اتصال بین آن‌ها اشاره می‌کند که در واقع مشخصات معماری را تحقق می‌بخشد. از جمله صفات در ارتباط با سازمان کامپیوتر، جزییات سخت‌افزاری است مانند سیگنال‌های کنترل، میانجی (Interface) بین کامپیوتر و دستگاه‌های جانبی، تکنولوژی مورد استفاده در حافظه، که این جزییات برای برنامه‌ساز ناپیدا و یا به عبارتی آشکار نیست.

برای روشن نمودن این تفاوت به مثال زیر توجه کنید. وجود یا عدم وجود دستورالعمل ضرب در فهرست دستور زبان ماشین یک کامپیوتر به معماری کامپیوتر مربوط می‌شود. ولی شامل نمودن سخت‌افزار مستقل ضرب‌کننده در داخل ALU، مربوط به سازمان کامپیوتر می‌باشد. تصمیم‌گیری بر سازمان کامپیوتر ممکن است مبتنی بر تکرار پردازش دستورالعمل ضرب، سرعت و اندازه واحد خاص سخت‌افزار ضرب انجام گیرد.

امروزه تفاوت بین معماری و سازمان کامپیوتر یک امر مهمی است. بسیاری از سازندگان کامپیوتر، خانواده‌ای از مدل‌های مختلف کامپیوتر ارایه داده‌اند که دارای معماری یکسان ولی سازمان متفاوت می‌باشند، در نتیجه مدل‌های مختلف در یک خانواده دارای قیمت و کارآیی متفاوت می‌باشند. هم‌چنین ممکن است معماری کامپیوترها طی سال‌ها پایدار و زنده بماند، ولی سازمان آنها با تغییر در تکنولوژی تغییر یابند. نمونه مثال از این دو پدیده مهم معماری IBM System / 370 است. این کامپیوتر اولین بار در سال 1970 ارایه داده شد و شامل مدل‌های متفاوت می‌باشد. استفاده کننده با نیاز نسبتاً کم می‌تواند مدل کند و ارزان را خریداری کند، ولی با افزایش نیازهایش، بدون صرف‌نظر از نرم‌افزاری که در گذشته تولید شده است، مدل سریع و گران را جایگزین سازد. این درس بیشتر در مورد سازمان کامپیوتر بحث خواهد کرد.

وظیفه و ساختار Function and Structure

کامپیوترهای معاصر، میلیون‌ها مولفه الکترونیکی در بر دارند. بنابراین توضیح و تشریح آن‌ها به صورت انفرادی مشکل است. برای توضیح دادن یک سیستم پیچیده کامپیوتر می‌توان از مفهوم سلسله مراتبی استفاده نمود که در آن مجموعه‌ای از مولفه‌ها به طور منطقی به یکدیگر متصل شده‌اند و هر کدام از مولفه‌ها نیز یک ساختار سلسله مراتبی خاص خود دارد. استفاده از مفهوم سلسله مراتبی هم برای طراحی و هم برای تشریح و توضیح مولفه‌ها مفید می‌باشد، زیرا طراح در هر لحظه از زمان تنها با یک سطح از سیستم سروکار خواهد داشت و در هر سطح طراح تنها با ساختار و وظیفه همان سطح در ارتباط خواهد بود.

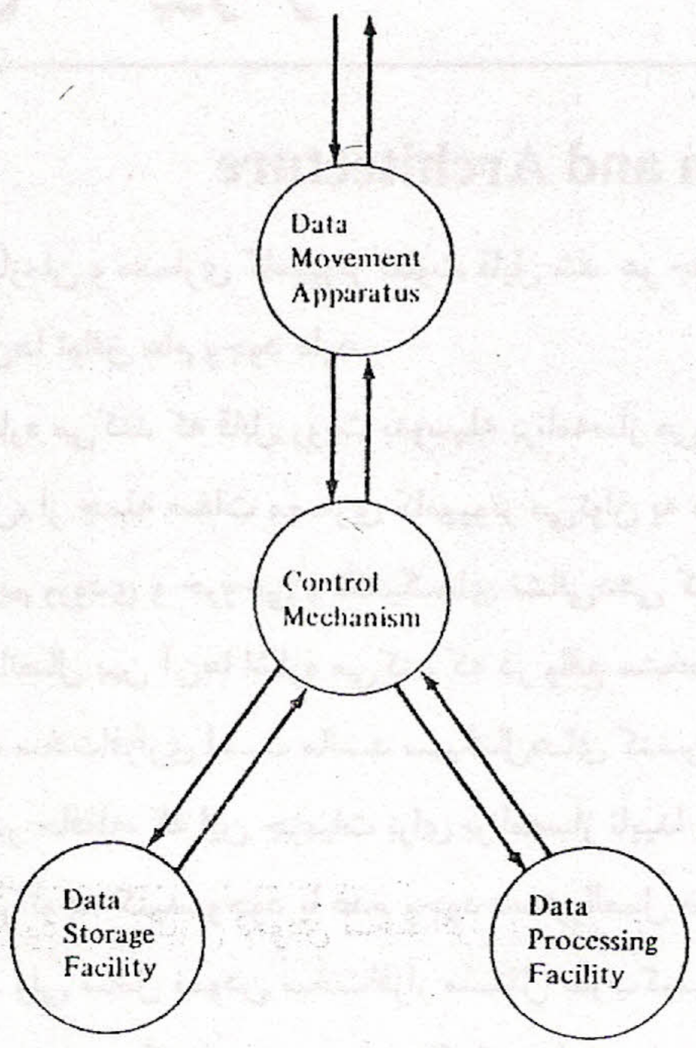
منظور از ساختار (Structure) نحوه اتصال منطقی مولفه‌ها به یکدیگر و برقراری ارتباط بین آن‌ها می‌باشد. ولی Function وظیفه یا عملی است که هر یک از مولفه‌ها به عنوان بخشی از ساختار از خود نشان می‌دهند.

ما در تشریح ساختار سیستم از روش Top-down استفاده خواهیم نمود.

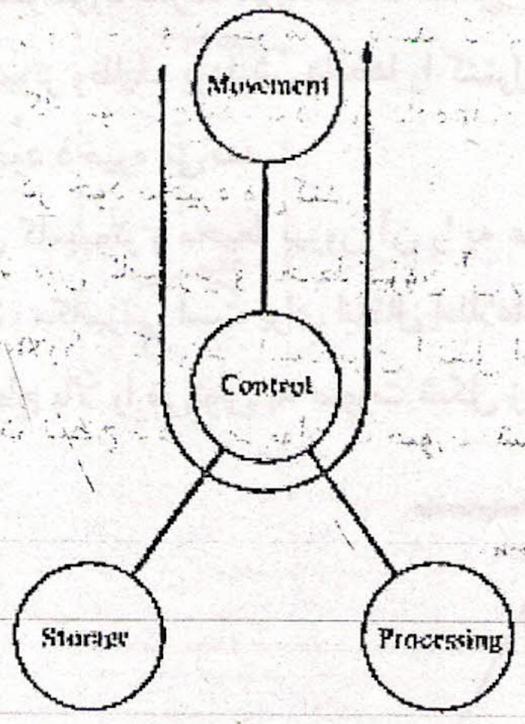
وظیفه یک کامپیوتر بسیار ساده است. شکل‌های زیر وظایف اصلی یک کامپیوتر را نشان می‌دهند که به چهار گروه Control , Data Movement , Data Storage , Data Processing تقسیم شده‌اند.

OPERATING ENVIRONMENT

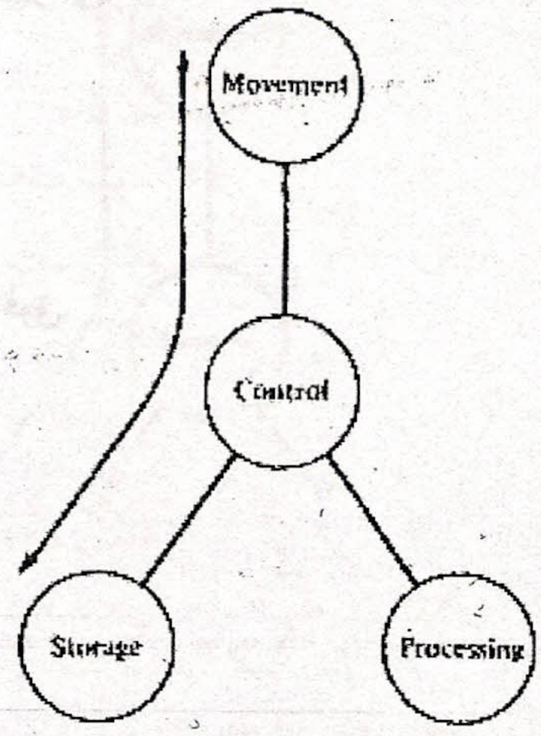
(Source and Destination of Data)



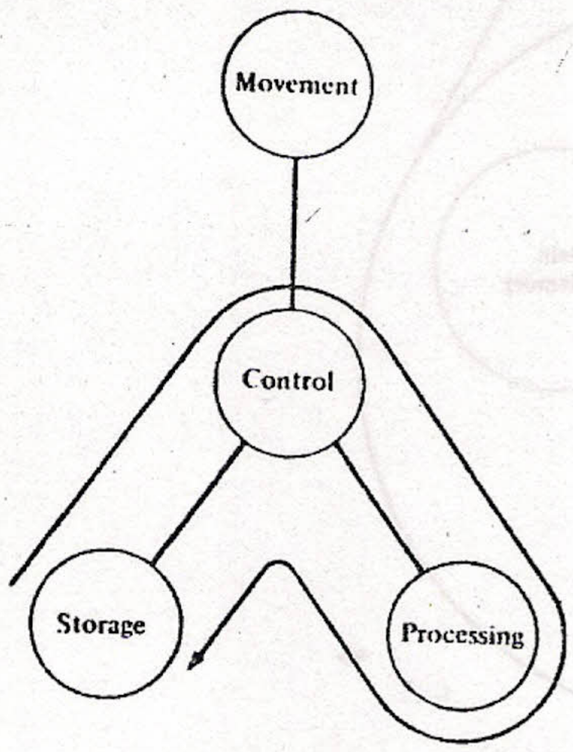
A Functional view of the computer



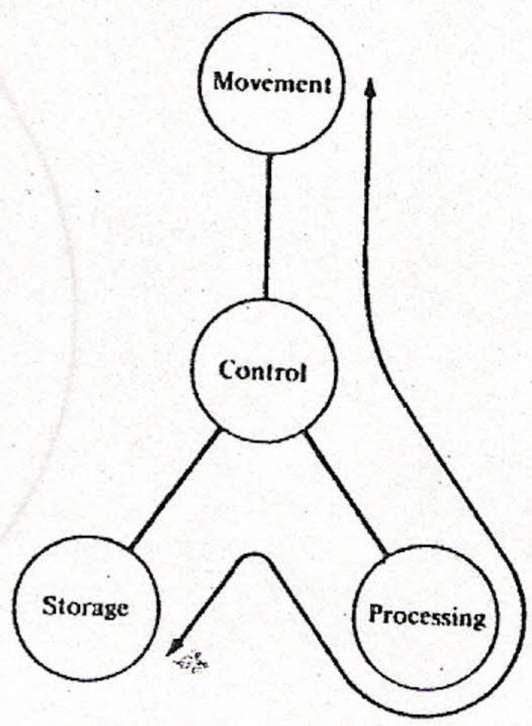
(b) Data Storage



(a) Data Movement



Data Processing
Possible computer operation
اطلاعات شامل در حافظه

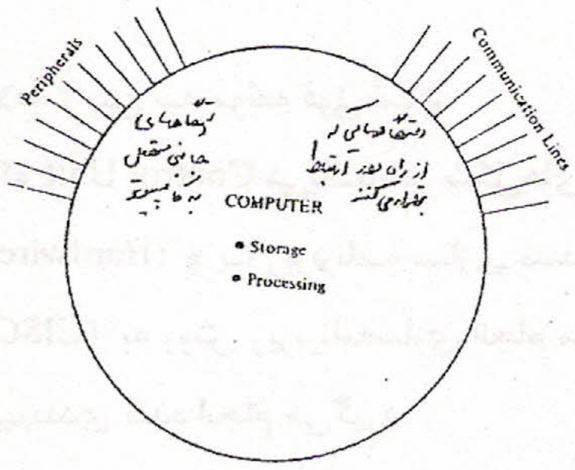


(c) Data Processing (d)

اطلاعات واقع در مسیر حافظه و محیط بیرونی

ساختار (Structure)

به‌طور کلی کامپیوتر وسیله‌ای است که به نحوی با محیط بیرونی خود ارتباط برقرار می‌کند. اتصال کامپیوتر به محیط بیرونی آن وسیله دستگاه‌های جانبی (Peripheral Devices) و خطوط ارتباطات (Communication) انجام می‌گیرد، که در شکل زیر نشان داده است.



The Computer

ساختار داخل کامپیوتر بیشتر مورد نظر ما می‌باشد. که شامل چهار مولفه اصلی زیر می‌باشد:

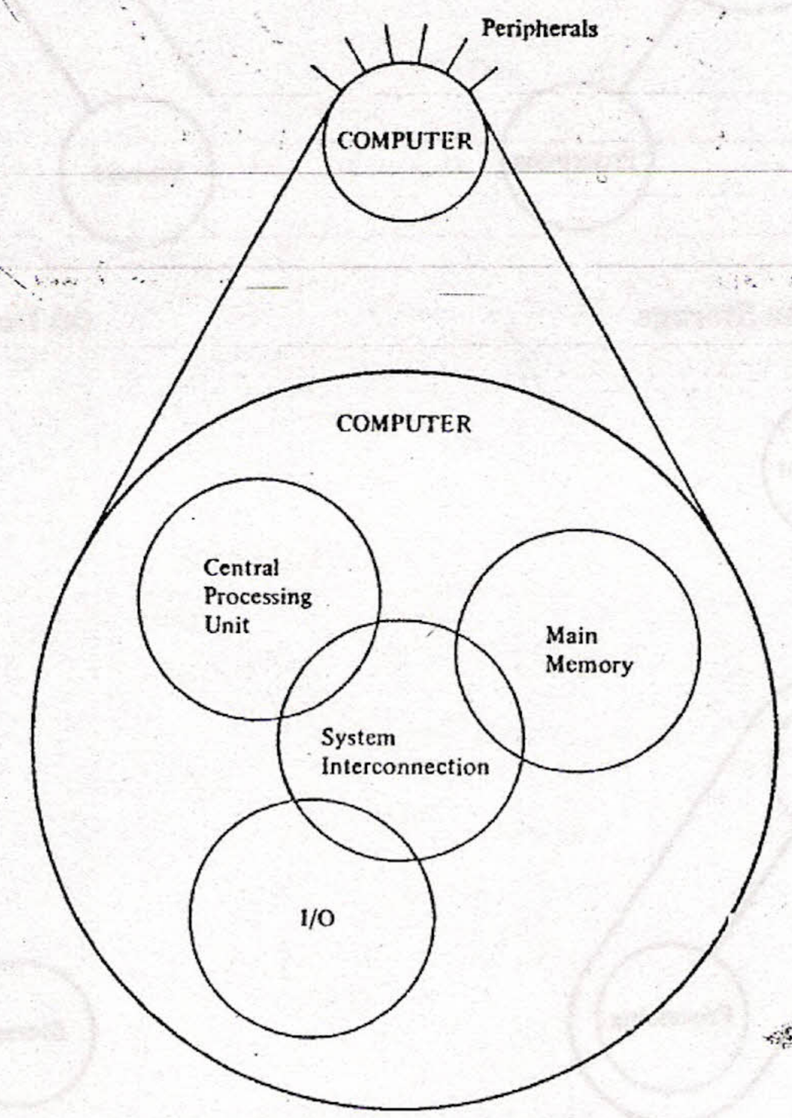
CPU : عملیات درون کامپیوتر وظایف پردازش داده‌ها را کنترل می‌کند.

Memory : داده‌ها را در خود ذخیره می‌کند.

I / O : انتقال اطلاعات بین کامپیوتر و محیط بیرون آن را به عهده دارد.

System Interconnection : مکانیزمی است برای انتقال اطلاعات بین سه مولفه فوق.

ساختار یک کامپیوتر در سطح بالا را می‌توان به صورت شکل زیر نشان داد.



The computer: top-level structure

در سیستم‌های کامپیوتری امروزه ممکن است از هر یک از مولفه‌های فوق، چندین کپی در داخل یک سیستم کامپیوتری وجود داشته باشد در کامپیوترهای قدیمی در داخل کامپیوتر تنها یک CPU وجود داشت که به Single Unit Processor معروف‌اند، ولی در سال‌های اخیر از چندین پردازنده در داخل یک سیستم استفاده می‌شود که به Multiple Processor مشهورند.

پیچیده‌ترین مولفه کامپیوتر CPU می‌باشد که مولفه‌های ساختاری آن عبارت‌اند از:

Control Unit : عملیات درون CPU و در واقع کامپیوتر را کنترل می‌کند.

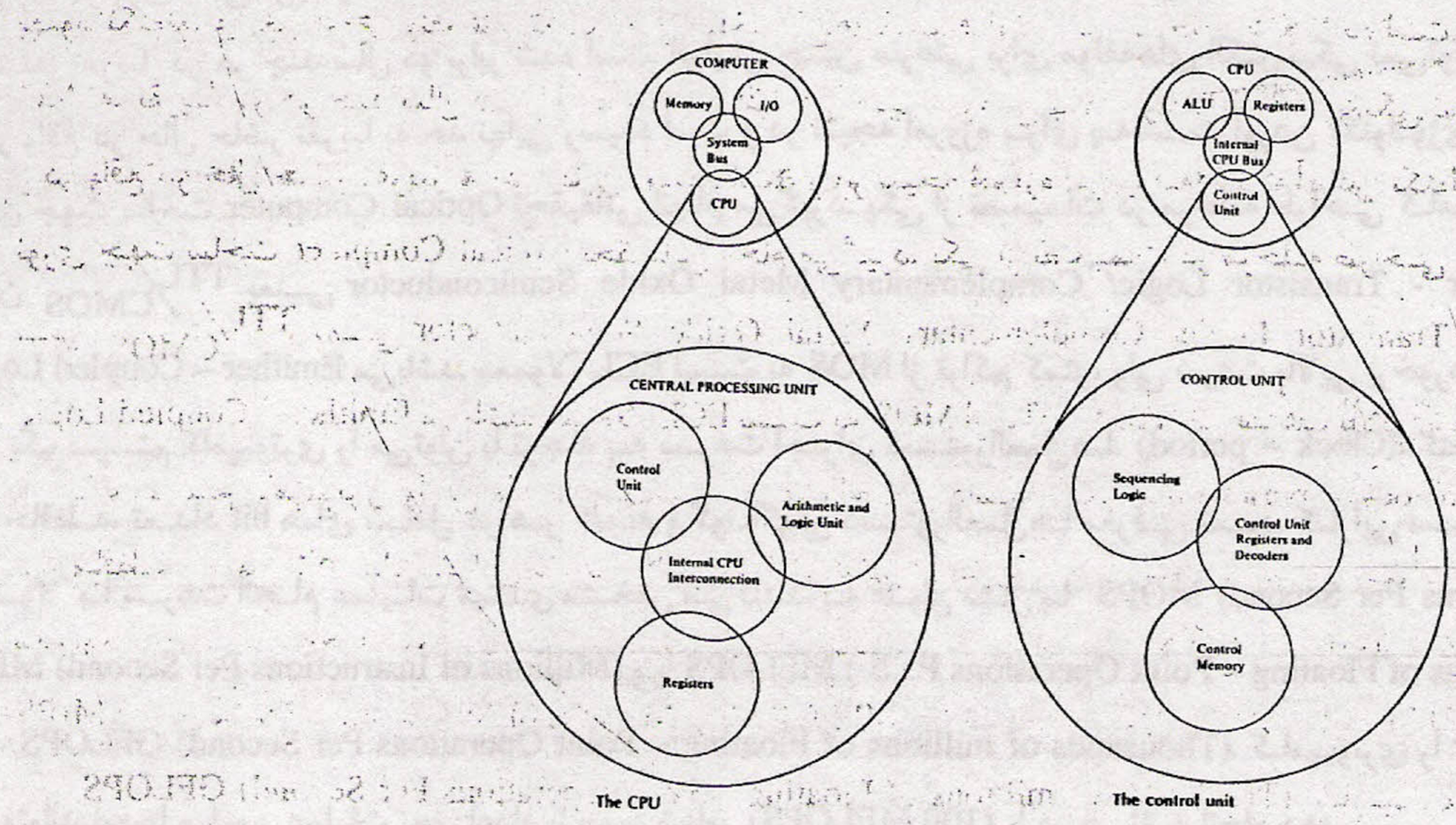
Arithmetic and Logic Unit (ALU) : وظیفه پردازش داده‌ها را انجام می‌دهد.

Registers : حافظه درون CPU را تدارک می‌بیند.

CPU Interconnection : مکانیزمی برای تبادل اطلاعات بین سه مولفه فوق است.

ساختار CPU و ساختار مهم‌ترین مولفه درون آن که Control Unit می‌باشد، در شکل‌های زیر نشان داده شده‌اند.

واحد کنترل معمولاً به صورت سیم‌بندی شده (Hardwired) و یا ریزبرنامه‌سازی شده (Microprogrammed) طراحی می‌شود. طراحی واحد کنترل بیشتر کامپیوترهای متداول (CISC) به روش ریزبرنامه‌سازی انجام می‌گرفت، ولی درکامپیوترهای (RISC) ابداً از ریزبرنامه‌سازی استفاده نمی‌شود، بلکه به صورت سیم‌بندی شده انجام می‌گیرد.



کامپیوترها با برنامه ذخیره شده The Stored Program Computer

سیستم کامپیوتری که در آن عملیات در binary رمزگذاری و در حافظه کامپیوتر ذخیره شده و سپس به ترتیب از قبل تعیین‌شده‌ای اجرا شوند Stored Program Computer و گاهی Von Numman Computer و یا Control – Flow Computer نامیده می‌شوند. این مفهوم حدود 100 سال قبل به وجود آمد و مبنای طراحی کامپیوترهای مدرن و پیچیده امروزی (Complex Instruction Set Computer , CISC) را بنا نهاد.

به هر حال، امروزه ساختار و محاسبه متفاوتی برای اجرای دستورالعمل‌های ذخیره شده وجود دارد، که در آن دنباله دستورالعمل‌ها به ترتیبی که در حافظه ذخیره شده‌اند، اجرا نمی‌شوند. چنین کامپیوترها را Data – Flow Computer می‌نامند و یا ممکن است که اصلاً نیازی به ذخیره شدن دستورالعمل‌ها در حافظه نباشد، چنین کامپیوترها را Neural Computer می‌نامند که احتمالاً در سال‌های آینده به بازار عرضه خواهند شد.

به‌طور خلاصه، مشخصات عملیاتی کامپیوتر با برنامه ذخیره شده را می‌توان به صورت زیر بیان نمود.

۱ـ تنها عملیات مبتدی مانند جمع و عملیات منطقی در آن انجام می‌شوند.

۲ـ برنامه‌ساز برای انجام محاسبه‌ای، مجموعه‌ای از عملیات را انتخاب می‌کند.

۳ـ عملیات به صورت رمزگذاری شده در حافظه ذخیره می‌شود.

۴ـ دستورهای ذخیره شده به صورت محض، ترتیبی اجرا می‌شوند (مگر در برنامه مشخص شده باشد).

۵ـ در همان حافظه‌ای که دستورالعمل‌ها ذخیره شده‌اند، ممکن است Data نیز ذخیره شود.

پیشرفت‌های ایجاد شده در کارایی کامپیوترها Improvements in Performance

از سال 1940 تاکنون، پیشرفت‌های ایجاد شده در بهبود کارایی کامپیوترها با برنامهٔ ذخیره شده را می‌توان در سه مورد زیر متمرکز نمود:

۱ـ Improvement in Technology ، بهبود در تکنولوژی

۲ـ Software Development ، ایجاد نرم‌افزار

۳ـ Architectural Enhancements ، فزونی (افزایش) در معماری

بهبود تکنولوژی یعنی نوع مولفه‌ها و تکنیک‌های ساخت آن‌ها باعث افزایش سرعت خیلی زیاد شده است. سرعت مولفه‌ها در سال‌های گذشته، تقریباً در هر چند سال دو برابر شده است. افزایش چنین سرعتی برای مولفه‌های الکترونیکی نمی‌تواند هم‌چنان ادامه یابد و در واقع در حال حاضر تقریبا به حد نهایی رسیده است و در نتیجه امروزه برای به‌دست آوردن تکنولوژی دیگر مانند تکنولوژی نوری جهت ساخت Optical Computer تحقیقاتی انجام می‌گیرد. یکی از تصمیمات در مرحله طراحی کامپیوتر، انتخاب تکنولوژی بین TTL/CMOS یعنی Transistor - Transistor Logic/ Complementary Metal Oxide Semiconductor و ECL یعنی Emither – Coupled Logic می‌باشد. معمولاً ECL نسبت به MOS از تراکم کمتر، ولی سرعت بالایی برخوردار است.

یک سیستم کامپیوتری را می‌توان با توجه به سرعت اجرای دستورالعمل‌ها، (Clock – period)، گنجایش و زمان دستیابی به حافظه، تعداد bit های شامل در هر کلمه و گوناگونی دستورالعمل‌ها معرفی نمود. کارایی سیستم کامپیوتری سریع، معمولاً با سرعت انجام عملیات مبتدی مشخص می‌کنند، به عنوان مثال با MOPS (Millions of Operations Per Second) ، MIPS (Millions of Instructions Per Second) و یا MFLOPS (Millions of Floating – Point Operations P . S) و یا حتی GFLOPS (Thousands of millions of Floating – Point Operations Per Second). کامپیوتری را Super Computer نامند، اگر بتواند صدها میلیون عملیات روی اعداد با ممیز شناور (100 MFLOPS) را در هر ثانیه انجام دهد.

ایجاد تکنیک‌ها و نرم‌افزار مانند انواع زبان‌های برنامه‌سازی سطح بالا و سیستم‌های عامل پیچیده، باعث شده است که در بیشتر سیستم‌های کامپیوتری، سخت‌افزار پشتیبان برای نرم‌افزار به‌وجود آید.

Architectural Developments

پیشرفت‌های ایجاد شده در معماری کامپیوترها با برنامه ذخیره شده را می‌توان در چهار گروه زیر دسته‌بندی کرد:

1. Those concerned with the processor.
2. Those concerned with the memory hierarchy.
3. Those around the processor memory interface.
4. Those involving use of multiple processor system.

Processor Function

مهم‌ترین وظیفه Processor (پردازنده) پردازش دستورالعمل‌ها است و آن را می‌توان به دو قسمت مستقل زیر تقسیم نمود:

۱ـ واکشی دستورالعمل Fetching an instruction

۲ـ اجرای دستورالعمل واکشی شده Executing the fetched instruction

ابتدا پردازنده با استفاده از محتوی PC یکی از دستورالعمل‌ها را از حافظه واکشی می‌کند و سپس دستورالعمل را رمزگشایی، Operandها را واکشی، عمل مشخص شده را انجام و نتیجه را در حافظه ذخیره می‌کند. به منظور افزایش کارایی پردازنده، می‌توان واکشی و اجرای دستورالعمل‌ها را به طور همزمان (Simultaneously) انجام داد. این روش Pipeline یا Overlap Fetch/Execute نامیده می‌شود که بعداً طی یک فصل مستقل از درس مطالعه خواهد شد.

طی سال‌های گذشته به منظور تامین نیازهای نرم‌افزار، مجموعه دستورالعمل‌های کامپیوتر پیچیده شده‌اند. فرض بر این است که عملیات پیچیده به‌وسیله سخت‌افزار سریع‌تر از نرم‌افزار انجام می‌گیرد. ولی ایجاد دستورالعمل‌های پیچیده، روی پردازش دستورالعمل‌های ساده اثر منفی دارد و در نتیجه باعث کند شدن پردازش دستورالعمل‌های ساده خواهد بود. بنابراین انتخاب مجموعه دستورالعمل‌های کامپیوتر، یکی از تصمیمات مهم در طراحی کامپیوتر به شمار می‌رود. امروزه عده‌ای بر این باروند که محدود نمودن دستورالعمل‌ها باعث افزایش کارایی سیستم می‌گردد و سیستم کامپیوتری با تعداد دستورالعمل‌های محدود (معمولاً کمتر از 128

دستورالعمل) RISC (Reduced Instruction Set Computer) نامیده می‌شود. از خصوصیات مهم کامپیوتری RISC می‌توان با استفاده از Pipline ساده، ثبات‌های زیاد و روش‌های نشانی‌دهی محدود اشاره نمود. ولی سیستم‌های کامپیوتری که سعی دارند بیشتر عملیات را به صورت سخت‌افزاری انجام دهند، معمولاً تا 300دستورالعمل دارند و CISC (Complex Instruction Set Computer) نامیده می‌شود.

Memory Hierarchy (حافظه سلسله مراتبی)

حافظه‌ای که بیرون پردازنده قرار دارد، شامل حافظه اصلی (Main memory) از نوع Random Access و حافظه ثانویه (Secondary memory) از نوع Non-random access می‌باشد. حافظه ثانویه معمولاً از نوع مغناطیسی شامل Magnetic Disk و Magnetic Tape می‌باشد. امروزه تکنولوژی دیسک نوری (Optical Disk) که گنجایش وسیع برای ذخیره نمودن اطلاعات دارد، سطح دیگری را به حافظه سلسله مراتبی افزوده است. حافظه اصلی از نوع نیمه هادی (Semiconductor) و نسبت به حافظه ثانویه سریع می‌باشد. استفاده موثر و آسان از حافظه سلسله مراتبی باعث حمایت از مفهوم MultiProgramming شده است.

MultiProgramming عبارتی برای تشریح برنامه‌سازی سیستم است که در آن بخش‌هایی از برنامه‌های چندین User به طور هم‌زمان (Concurrenty) و به ترتیب اجرا می‌شوند. در صورت استفاده از حافظه سلسله مراتبی، چون گنجایش حافظه اصلی محدود است، باید روشی برای مدیریت حافظه وجود داشته باشد که بتواند به طور اتوماتیک نقل و انتقال برنامه‌ها و داده‌ها را بین حافظه اصلی و حافظه کمکی را انجام دهد، این مفهوم حافظه مجازی (Virtual Memory) نامیده می‌شود. در این روش بدون این‌که User برنامه خاصی جهت انتقال اطلاعات بنویسد، یک مکانیزم اتوماتیک انتقال اطلاعات را در اندازه‌های ثابت و یا متفاوت بین حافظه‌ها انجام می‌دهد. در این صورت نیازی به تغییر آدرس Operandها در داخل برنامه وجود نخواهد داشت، بلکه آدرس‌های تولید شده به‌وسیله پردازنده، به‌وسیله سخت‌افزاری خاص و به صورت پنهان از User به‌طور اتوماتیک به آدرس واقعی محل ذخیره شده در حافظه اصلی تبدیل می‌گردد.

Processor – Memory Interface

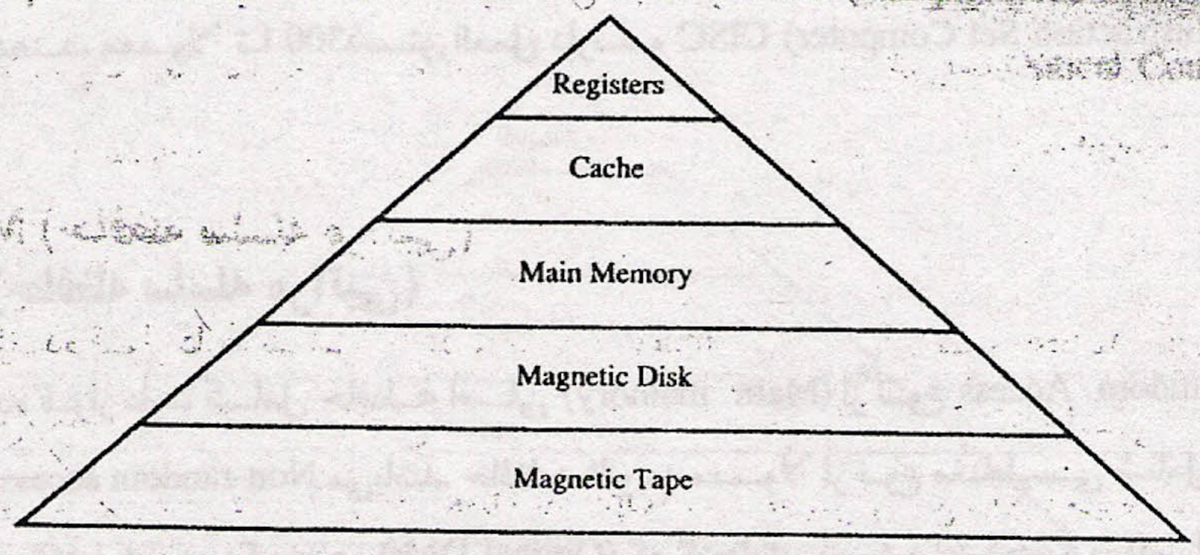
میانجی بین پردازنده و حافظه اصلی در ارتباط با دو امر زیر است:

۱ـ حمل دستورالعمل از حافظه به پردازنده (به هنگام واکشی دستورالعمل)

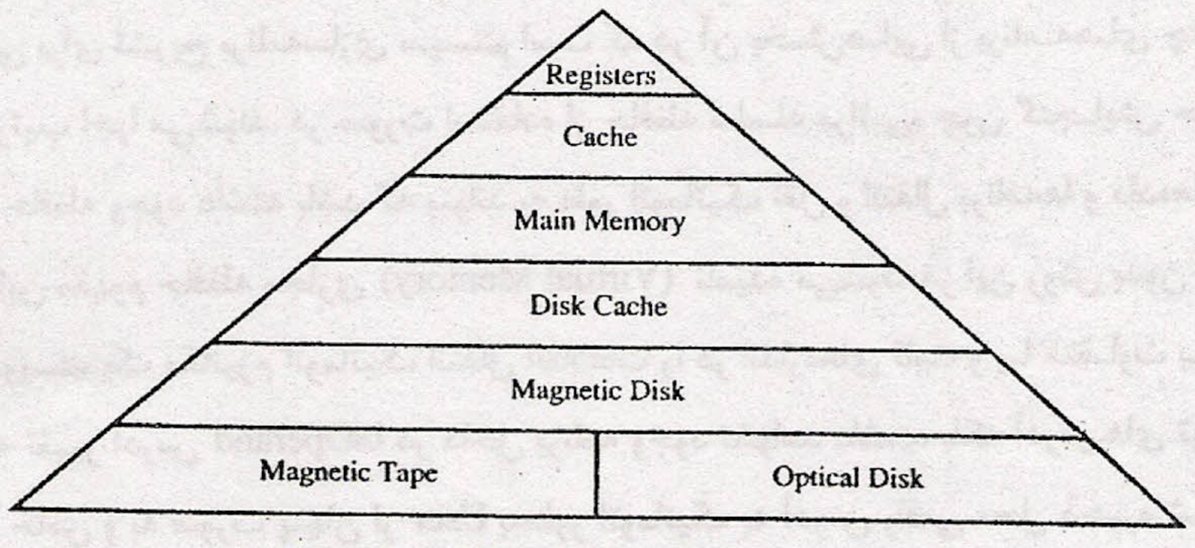
۲ـ حمل Data بین پردازنده و حافظه (به‌هنگام اجرای دستورالعمل)

باید سرعت انتقال اطلاعات به اندازه‌ای باشد که بتواند در مقصد هضم گردد. بنابراین به‌هنگام طراحی باید تطابق بین سرعت مبادله اطلاعات بین پردازنده و حافظه موردنظر قرار گیرد. در صورتی‌که Data Rate پردازنده و حافظه یکسان نباشند، سیستم با کندترین مولفه همگام خواهد شد و باید در حالت کلی از روش first – in –first – out استفاده شود تا تداخل در اطلاعات به‌وجود نیاید.

معمولاً سرعت پردازنده نسبت به حافظه بیشتر است. برای تطابق سرعت بین پردازنده و حافظه می‌توان یک حافظه سریع، ولی با گنجایش کم به نام Cache بین آن دو مطابق شکل زیر قرار داد.

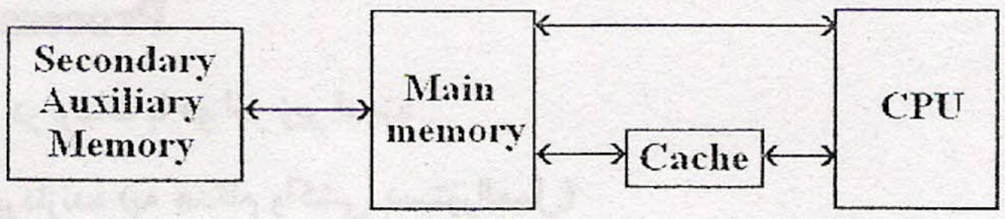


(a) Traditional Memory Hierarchy



(b) Contemporary Memory Hierarchy

The memory hierarchy

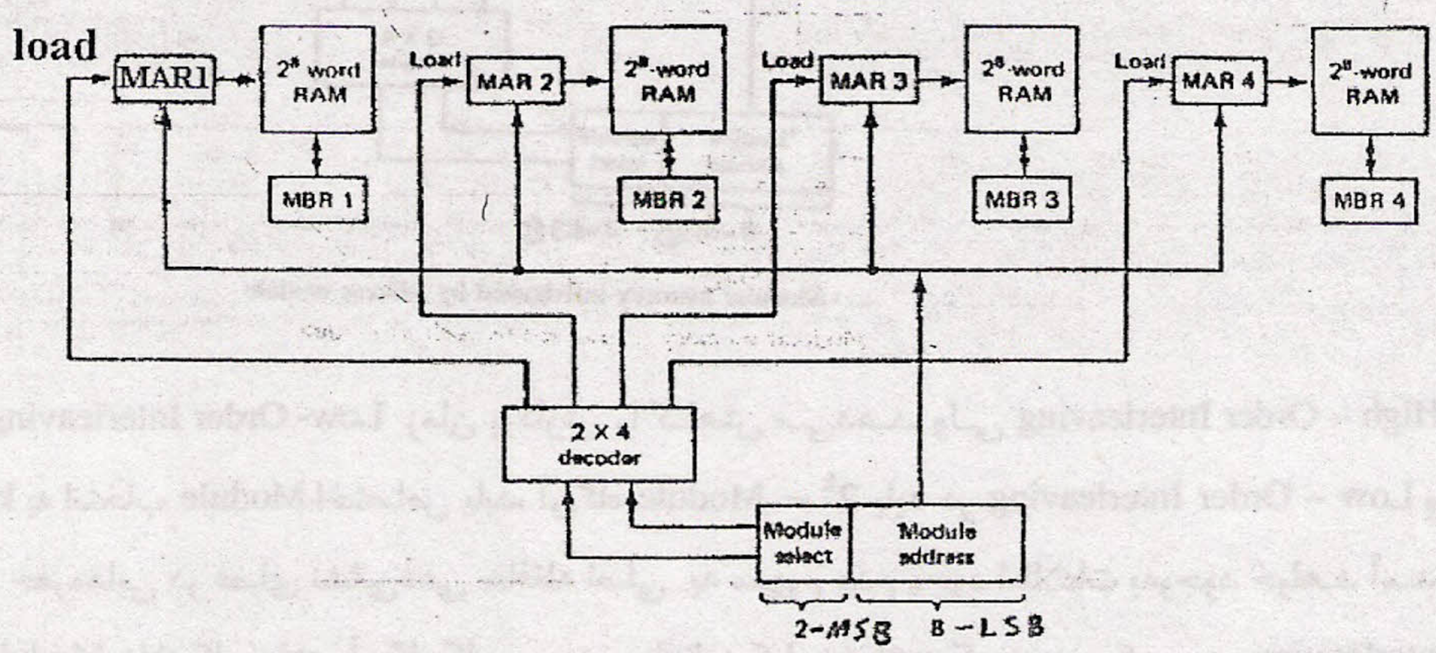


Interface of CPU, Cache, and Main Memory

دستورالعمل‌ها و داده‌های برنامه ابتدا به داخل Cache انتقال می‌یابد و سپس به وسیله پرازنده واکشی می‌شود. با فرض این‌که اطلاعات موجود در Cache معمولاً بیش از یک بار مورد استفاده قرار می‌گیرد (مانند Recursive Subroutine Call) در نتیجه سرعت کل سیستم افزایش می‌یابد.

روش دیگر برای تطابق سرعت مولفه‌های مبدا و مقصد ، استفاده از n کپی از مولفه‌های کند می‌باشد. مثلاً سرعت انتقال اطلاعات توسط CPU دو برابر حافظه باشد. می‌توان از دو Memory Module استفاده نمود و با ارسال آدرس یکسان به آن دو، امکان واکشی دو دستورالعمل را به طور هم‌زمان فراهم نمود. این روش در واقع طول کلمه حافظه را افزایش می‌دهد و به Linear Memory Addressing معروف است. نشانی‌دهی خطی حافظه در صورتی مفید خواهد بود که کلیه اطلاعات انتقال یافته واقعاً موردنیاز باشد و مورد استفاده قرار گیرد. این تکنیک بیشتر برای کاهش دادن زمان انتقال اطلاعات بین حافظه اصلی و Cache به کار می‌رود.

یکی دیگر از تکنیک‌های مهم در رابطه با معماری کامپیوتر استفاده از Modular Memory می‌باشد که در آن هر Module شامل MAR , MBR اختصاصی می‌باشد و در نتیجه چندین Read و یا Write به‌طور هم‌زمان انجام می‌گیرد. کاربرد اصلی این روش در Pipeline Processor و Array Processor می‌باشد. دو روش اصلی برای طراحی Modular Memory وجود دارد. در روش اول بیت‌های بالاترین رتبه برای انتخاب Module و بیت‌های پایین‌ترین رتبه برای انتخاب کلمه درون Module به کار می‌رود. شکل زیر حافظه‌ای با گنجایش 2^{10} – Word را نشان می‌دهد که به چهار Module هر کدام با گنجایش 2^{8} – Word تقسیم شده‌اند که به High – Order Interleaving معروف‌اند.

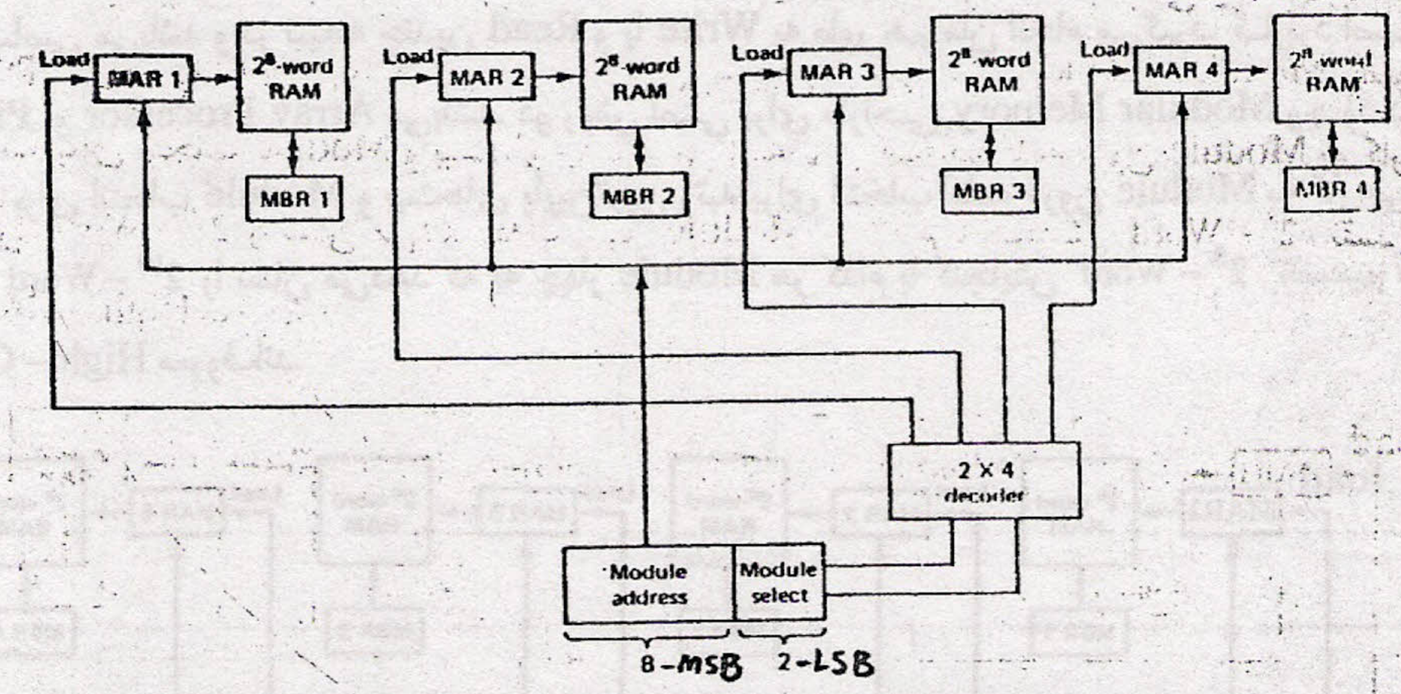


Modular memory by module address

High order interleaving

چون چهار Module وجود دارد، 2-MSB برای انتخاب Module و 8-LSB برای انتخاب کلمه‌ای درون Module به کار می‌رود. در این ساختار آدرس‌های متوالی به یک Module تعلق دارد، و چون دستورالعمل‌های برنامه معمولاً به طور متوالی پردازش می‌شوند، بنابراین تنها یک Module مشغول خواهد بود. در نتیجه دستگاه‌های ورودی و خروجی مجهز به DMA قادر خواهند بود با سایر Module ها ارتباط برقرار کنند. در روش دوم برای طراحی Modular Memory ، بیت‌ها با کمترین رتبه (LSB) برای انتخاب Module و بیت‌ها با بالاترین رتبه (MSB) برای انتخاب کلمه درون Module اختصاص می‌یابد. چنین حافظه‌ای Low – Order – Interleaving نامیده می‌شود. در این روش آدرس‌های متوالی بین Module های متوالی توزیع شده است و در نتیجه دستورالعمل‌های متوالی برنامه در کلمات Module های متفاوت قرار دارند. بنابراین چندین دستورالعمل را می‌توان به طور هم‌زمان واکشی نمود. این روش در صورت پردازش دستورالعمل‌ها به طور پشت سر هم می‌تواند باعث افزایش سرعت گردد. یعنی به عوض این‌که ابتدا دستورالعمل واکشی و سپس اجرا شود، می‌توان هم‌زمان با اجرای دستورالعمل جاری واکشی دستورالعمل بعدی را انجام داد. در صورتی‌که دستورالعمل جاری از نوع انشعاب شرطی باشد، ممکن است دستورالعمل متوالی با آن که از قبل واکشی شده است، اجرا نشود. به هر حال وقت اضافی برای این منظور تلف نخواهد شد.

شکل زیر ساختار Low – Order Interleaving با استفاده از مفروضات مثال قبل را نشان می‌دهد.



Modular memory interleaved by address module

هر چند استفاده از Low- Order Interleaving زمان پردازش را کاهش می‌دهد، ولی High – Order Interleaving انعطاف‌پذیری بیشتر دارد. اگر k-bit به انتخاب Module اختصاص یابد، آن‌گاه 2^k – Module باید در Low – Order Interleaving وجود داشته باشد و در غیر این صورت حفره‌هایی در فضای نشانی‌دهی حافظه اصلی به مفهوم عدم وجود اطلاعات به‌وجود خواهد آمد و هم‌چنین در صورتی که یکی از Module ها از کار بیفتد، آن‌گاه کل سیستم حافظه کار نخواهد کرد. بر عکس در High - Order Interleaving ضروری نیست که در ابتدای طراحی کلیه Module ها را در سیستم شامل نمود، بلکه تعداد Module های شامل ممکن است کمتر از 2^k باشد و در صورت نیاز به بسط حافظه، به آسانی در آینده می‌توان تعدادی Module به سیستم حافظه اضافه نمود و در صورت خرابی یکی از Module ها کل سیستم حافظه از کار نخواهد افتاد.

Multiple Processor System

یکی از روش‌های واضح برای افزایش سرعت کامپیوتر، شامل نمودن چندین پردازنده در داخل یک سیستم کامپیوتری است که بتواند به سبک منسجم روی یک Single Task کار کند. شاید بتوان چنین تصور کرد که اگر n پردازنده به طور مداوم روی یک مساله کار کند، سرعت نسبت به تک پردازنده n برابر افزایش یابد. ولی همیشه نمی‌توان یک مساله را چنان تقسیم‌بندی کرد که n پردازنده بتواند به طور هم‌زمان روی آن کار کنند و اگر چنین کاری ممکن باشد، به علت تبادل اطلاعات بین پردازنده‌ها هرگز سرعت n برابر افزایش نخواهد یافت. سیستم Multi Processor را می‌توان به کمک چندین پردازنده ایجاد نمود که از یک حافظه مشترک استفاده نمایند. نتیجه حاصل را Shared Memory Multiprocessor System می‌نامند. هر یک از پردازنده‌های ممکن است شامل حافظه محلی (Local memory) باشد، ولی به هر حال برای تبادل اطلاعات بین پردازنده‌ها به یک حافظه مشترک احتیاج دارند. به علت وجود اتصال حافظه و bus استفاده از این روش موقعی مناسب خواهد بود که تعداد پردازنده‌ها کم باشد.

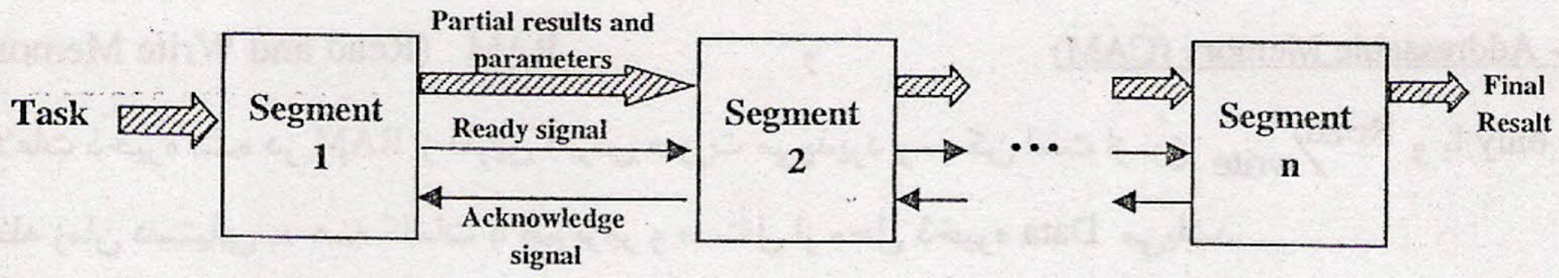
در روش دیگر، ممکن است بین پردازنده‌ها اتصال مستقیم ایجاد کرد که در آن پردازنده‌ها به طور مستقل کار کنند و مبادلهٔ نتایج بین پردازنده‌ها بدون استفاده از حافظه مشترک، مستقیماً توسط Message Passing انجام گیرد. افزایش تعداد پردازنده‌ها در Message Passing Multiprocessor به آسانی امکان‌پذیر است و مخصوصاً برای VLSI بسیار مناسب می‌باشد.

Pipelined System

Pipelining و Overlap در واقع تکنیک‌های یکسانی هستند که در آن یک وظیفه یا عملی به زیر وظایفی تقسیم شده‌اند که باید به ترتیب خاصی توسط واحدهای عملیاتی (Functional Units) که هر کدام Segment و یا Stage نامیده می‌شوند، انجام گیرند. Segment ها به طور Serial به یکدیگر متصل می‌شوند و به طور هم‌زمان هر یک جزیی از پردازش را انجام داده و نتیجه میانی را به Segment مجاور خود می‌فرستند. یعنی وقتی Segment ای روی زیر وظیفه i-ام کار می‌کند، Segment های مجاور با آن به طور هم‌زمان روی زیر وظیفه‌ای (i+1)ام و (i−1)ام کار خواهند کرد. (این روش بیشتر در خطوط تولید کارخانه‌ها به کار می‌رود)

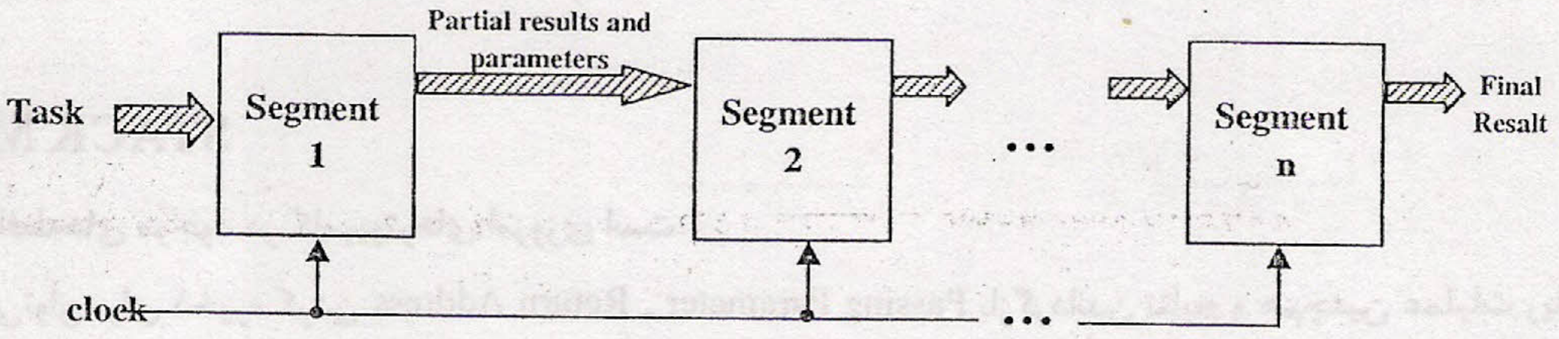
عبارت Pipelining برای تشریح و طراحی سیستمی به کار می‌رود که در آن محاسبهٔ خاصی به یک سری از مراحل تجزیه می‌شود. مثلاً Floating – Point Addition را می‌توان به مراحل مقایسهٔ نماها، همترازی مانتیس‌ها، جمع مانتیس‌ها و نرمال کردن مانتیس نتیجه که مراحل مستقل از یکدیگرند، تجزیه نمود. ولی عبارت Overlap به منظور تشریح طراحی سیستمی به کار می‌رود که در آن دو یا چند وظیفهٔ کاملاً مجزا به طور هم‌زمان انجام می‌گیرند، مثلاً عملیات Input / Output و CPU را می‌توان Overlap نمود.

انتقال Data بین Segment های سیستمی که به صورت pipeline طراحی شده است، به یکی از دو روش Asynchronous Method (غیر همگام) و Synchronous Method (همگام) انجام می‌گیرد که در شکل زیر نشان داده شده‌اند.



Asynchronous Method

در روش غیرهمگام بین هر دو Segment یک جفت Signal به نام Ready و Acknowledge وجود دارد. در صورتی‌که Segment ای زیر وظیفه خود را کامل کند، سیگنال Ready را فعال می‌کند تا بدین‌وسیله آمادگی خود را جهت ارسال نتیجه میانی به Segment مجاور اعلام دارد و Segment مجاور پس از دریافت نتیجهٔ میانی، سیگنال Acknowledge را فعال می‌کند.



Synchronous Method

Synchronous Method

در روش همگام که در شکل فوق نشان داده شده است، یک سیگنال تنظیم وقت (Timimng Signal) که متناسب با کندترین Segment می‌باشد، سبب می‌شود که خروجی کلیهٔ Segment ها به طور هم‌زمان به Segment مجاور با آن انتقال یابد.

روش غیرهمگام انعطاف‌پذیری بیشتری از نظر نوع Operand ورودی و تاخیر Segment ها از خود نشان می‌دهد و بیشتر در کامپیوترهای Data-Flow به کار می‌رود. اما در بیشتر کامپیوترهای متداول از روش همگام استفاده می‌شود و در آن به منظور جلوگیری از تداخل نتایج میانی بین Segment ها، Latch قرار می‌دهند.

حافظه‌های موجود در کامپیوتر

حافظه را می‌توان به دو گروه اصلی Main Memory و Auxiliary Memory تقسیم کرد.

حافظه اصلی در واقع منبع مرکزی یک سیستم کامپیوتری است که باید به طور پویا به برنامه‌های استفاده‌کنندگان اختصاص یابد.

پس حافظه اصلی دستورالعمل‌ها و داده‌هایی را در خود ذخیره می‌کند که توسط CPU پردازش می‌شود و در نتیجه توسط CPU مستقیماً قابل دسترسی است.

سرعت پردازش برنامه‌ها به سرعت حافظه اصلی بستگی دارد. اگر گنجایش حافظه اصلی زیاد باشد، سرعت پردازش اطلاعات افزایش می‌یابد، زیرا اکثر اطلاعات موردنیاز در آن موجود خواهد بود و نیازی به ارجاع به حافظه کمکی نیست ولی باید توجه داشت، حافظه‌ای با گنجایش زیاد نسبت به حافظه‌ای با گنجایش کم، کند عمل می‌کند.

با توجه به شکل سازمان حافظه RAM متوجه می‌شویم برای دستیابی به کلمات حافظه باید محتوای MAR رمزگشایی شود و Decoder هر چند مدار یک سطحی است، ولی تاخیر دارد و از طرف دیگر حافظه‌های با گنجایش زیاد با اتصال Chip ها ایجاد می‌گردد. بنابراین به منظور از بین بردن fan – in باید Decoder ها در چند سطح به صورت درختی به هم وصل شوند، در نتیجه زمان دستیابی به اطلاعات طولانی خواهد بود.

با توجه به مکانیسم به کار رفته برای ذخیره و بازیابی اطلاعات، حافظه‌های اصلی به دو گروه زیر تقسیم می‌شوند:

RAM (Read and Write Memory) و Content – Addressable Memory (CAM)

دستیابی به اطلاعات ذخیره شده در RAM از طریق آدرس صورت می‌پذیرد و ممکن است از نوع $\text{Read}/_{\text{write}}$ و یا Read only باشد.

در این نوع حافظه زمان دستیابی به همه کلمات با هم برابر و مستقل از محل ذخیره Data می‌باشد.

در حافظه‌های CAM دستیابی به اطلاعات به عوض آدرس توسط خود محتوا صورت می‌گیرد.

برنامه‌ها و داده‌هایی که بلافاصله توسط کامپیوتر موردنیاز نیستند، در حافظه ارزان و کند و به‌نام Auxiliary Memory یا Secondary Memory ذخیره می‌شوند. این نوع حافظه‌ها از نظر ذخیره و بازیابی اطلاعات به دو گروه زیر تقسیم می‌شوند:

Sequential Access Memory (SAM) , Direct Access Memory (DAM)

مثال برای SAM : Magnetic Tape

مثال برای DAM : Magnetic , Drum

STACK MEMORY

یکی دیگر از حافظه‌های موجود در کامپیوترهای امروزی است.

از Stack می‌توان برای ذخیره کردن Passing Parameter , Return Address بازگرداندن نتایج و هم‌چنین عملیات ریاضی استفاده نمود. Stack را می‌توان هم به صورت نرم‌افزاری و هم سخت‌افزاری ایجاد کرد.

در برخی کامپیوترها Stack قسمت اصلی حافظه را تشکیل می‌دهد که به آن‌ها Stack Computer می‌گویند.

دو نمونه بزرگ آن، کامپیوتر Burroughs (B5500 , B6500 , B 7500) و HP 3000 است.

Stack معمولاً ساختار LIFO دارد، ورود و خروجی اطلاعات از یک طرف به‌نام Top - of- Stack انجام می‌گیرد.

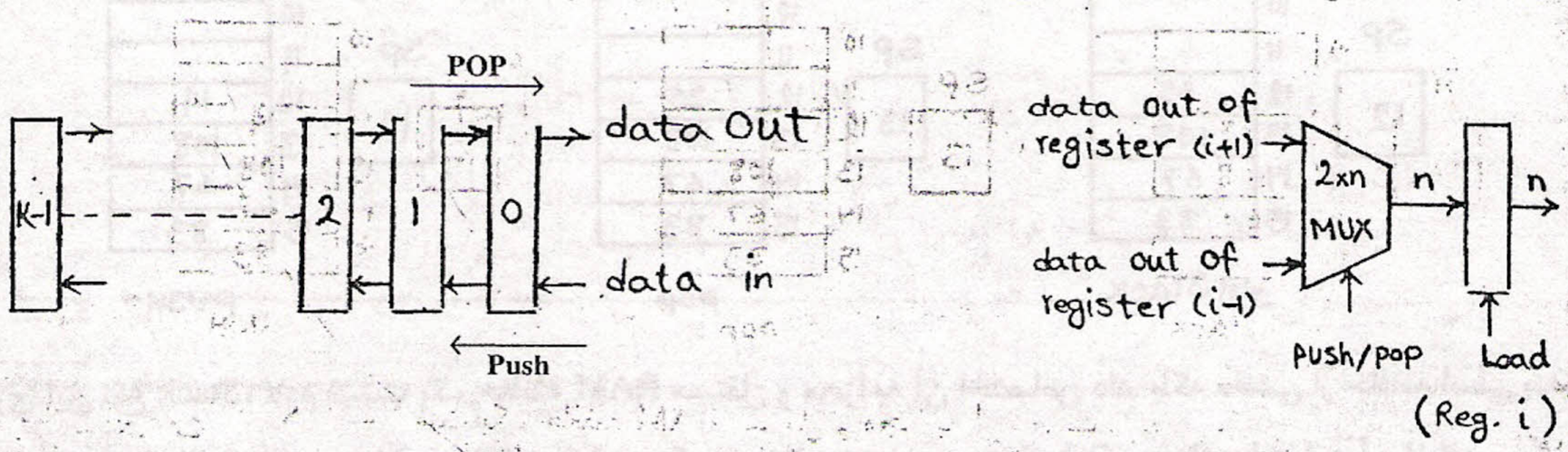
ورود و خروج اطلاعات به ترتیب به‌وسیله Pop , Push انجام می‌گیرد.

هم‌چنین دو flag جهت پر و خالی شدن Stack وجود دارد.

«پیاده‌سازی سخت‌افزاری»

1) Register – based Implementation:

I ـ یک Stack با گنجایش k کلمه n بیتی را می‌توان با اتصال k ثبات n بیتی یا قابلیت انتقال Parallel ایجاد کرد.



در صورت فعال شدن سیگنال PUSH، ثبات i - ام محتوای خود را به ثبات (i+1)- ام انتقال می‌دهد و ثبات شماره صفر داده ورودی را در خود ذخیره می‌کند.

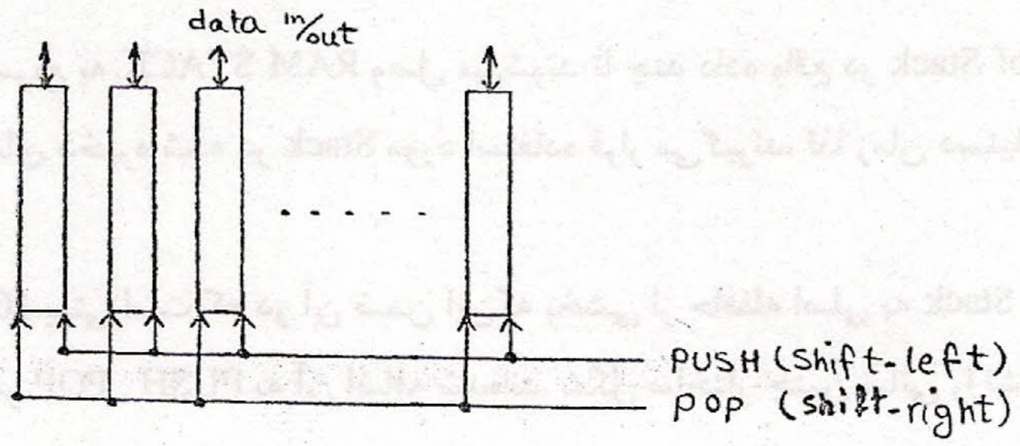
در صورت فعال شدن سیگنال POP ثبات i - ام محتوای خود را در ثبات (i-1)- ام ذخیره می‌کند و محتوای ثبات صفر به عنوان دادهٔ خروجی خارج می‌شود.

مدار کنترلی متناظر با هر ثبات در شکل نشان داده شده است.

برای نشان دادن حالت پر و خالی Stack از یک Counter و دو flag استفاده می‌شود که در شکل ذکر نشده است.

II ـ برای پیاده‌سازی Stack با گنجایش k کلمه n بیتی می‌توان از n ثبات k بیتی یا قابلیت تغییر مکان به چپ و راست نیز استفاده نمود.

شکل ساده این نوع Stack در زیر دیده می‌شود:

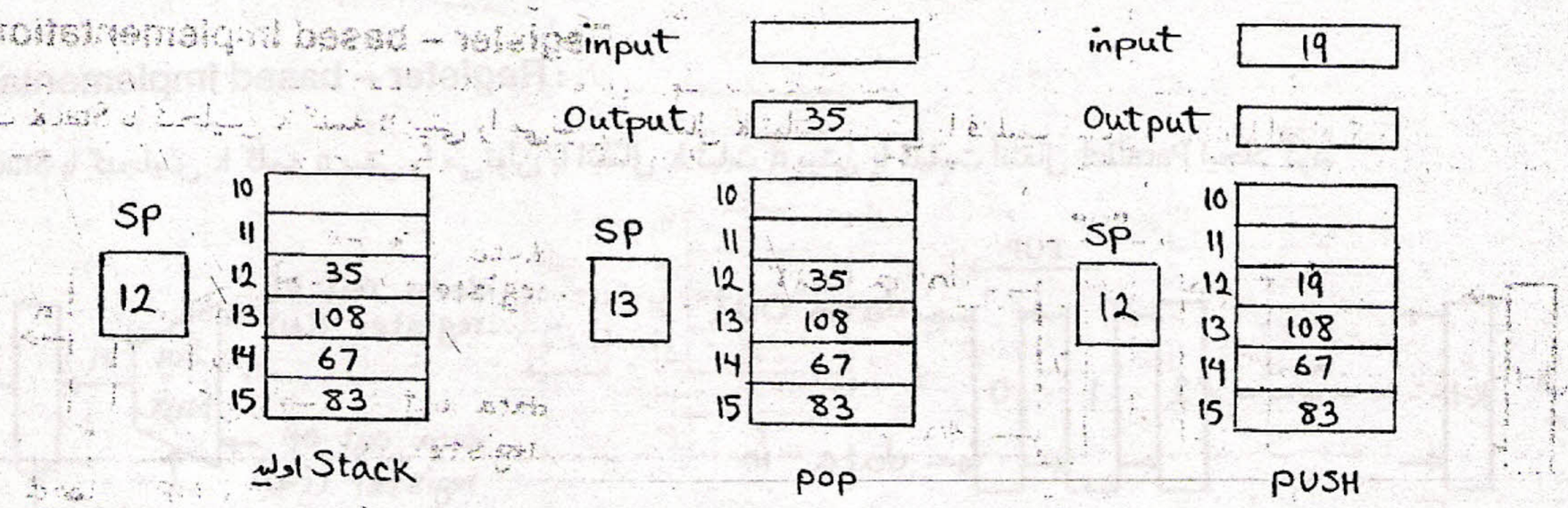


هر چند این دو نوع Stack بسیار سریع عمل می‌کنند، ولی به خاطر قیمت بالای آن می‌توان از حافظه‌های RAM نیز جهت پیاده‌سازی Stack استفاده نمود.

2) RAM – based Implementation:

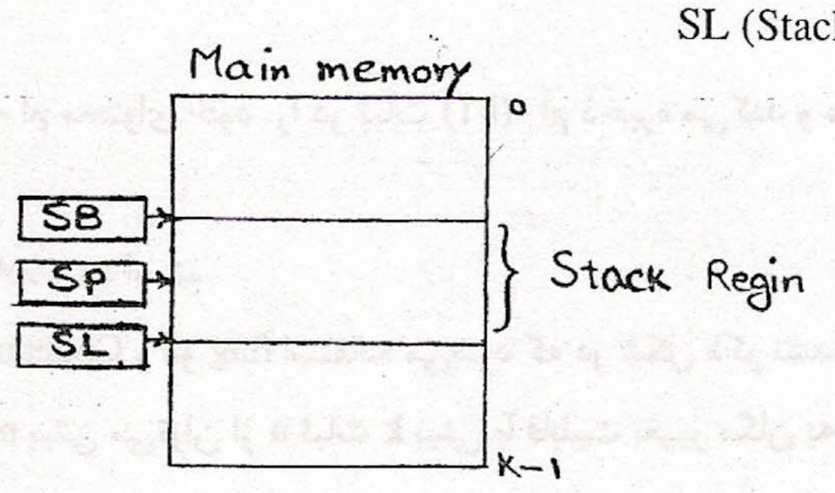
بر خلاف دو مورد قبل، در این روش به هنگام PUSH , POP نیازی به انتقال محتوای کلمات به کلمات مجاور نیست، بلکه با شامل نمودن ثباتی تحت نام SP (Stack Pointer) می‌توان آدرس Top-of- Stack را در آن ذخیره کرد. لذا به هنگام ذخیره و یا بازیابی Data از Stack می‌توان با نمو یا کاهش SP، دستیابی به کلمات Stack را انجام داد.

مثال زیر این روش را نشان می‌دهد:



برای پیاده‌سازی این نوع Stack، لازم نیست یک حافظه RAM مستقل و مجزا به آن اختصاص داد. بلکه بخشی از حافظه اصلی به‌نام Stack – Region به آن اضافه می‌شود و علاوه بر SP دو ثبات دیگر به منظور تشخیص Under flow , Overflow به آن اضافه می‌گردد که عبارت‌اند از:

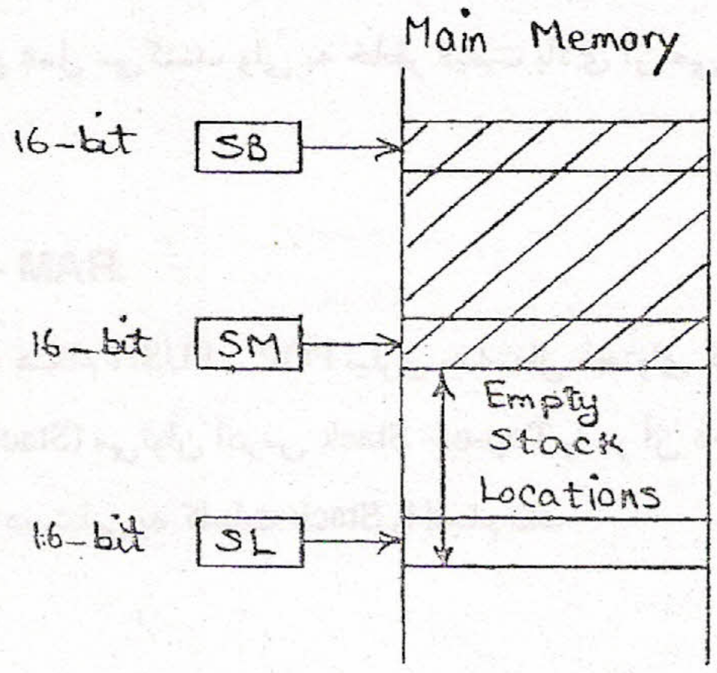
SL (Stack Limit) , SB (Stack Base)



HP- STACK :

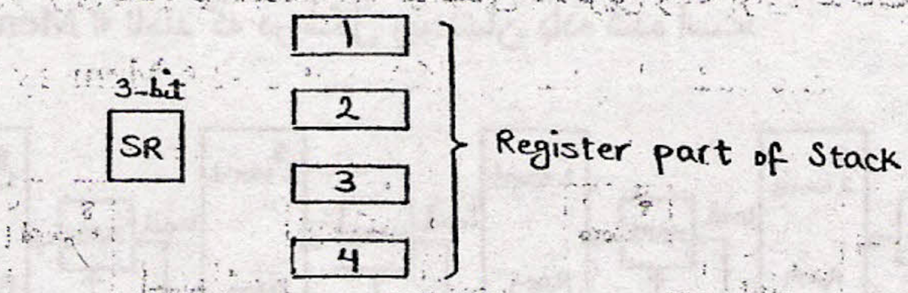
در برخی کامپیوترها چندین ثبات سریع به RAM STACK وصل می‌شوند تا چند داده واقع در Top of Stack را در خود ذخیره کنند. چون این داده‌ها بیش از سایر داده‌های ذخیره شده در Stack مورد استفاده قرار می‌گیرند، لذا زمان دستیابی به منظور ذخیره و بازیابی اطلاعات در Stack کاهش می‌یابد.

HP 3000 یک مینی کامپیوتر 16 بیتی است که در آن ضمن این‌که بخشی از حافظه اصلی به Stack اختصاص یافته است، 4 ثبات دیگر نیز به منظور افزایش سرعت در PUSH , POP به آن اضافه شده‌اند. شکل ساختار چنین ثباتی را نشان می‌دهد.



ثبات SM به بالاترین کلمه ذخیره شده در Stack اشاره می‌کند، ولی الزاماً Top – of – Stack واقعی نیست، زیرا ممکن است تا 4 کلمه از اطلاعات واقع در Stack – Region در ثبات‌های اختصاصی Stack وجود داشته باشد.

SR ثباتی است 3 بیتی که نشان می‌دهد 0 , 1 , 2 , 3 یا 4 کلمه از اطلاعات ذخیره شده در Stack در ثبات‌های اختصاصی قرار دارد.



ثبات SP به مفهوم Stack – Pointer می‌باشد، ولی سخت‌افزار خاصی طراحی شده است که با ارزیابی SP با مقادیر ثبات‌های SM و SR ارجاع صحیح به Data را انجام دهد.

وقتی برنامه از Stack استفاده نمی‌کند، سخت‌افزار موردنظر اطلاعات موجود در Stack را به ثبات‌های متصل به آن انتقال داده و مقادیر ثبات‌های متصل به آن را به روز در می‌آورد و در نتیجه محتوای ثبات‌های SM و SR را تغییر می‌دهد.

Modular Memory

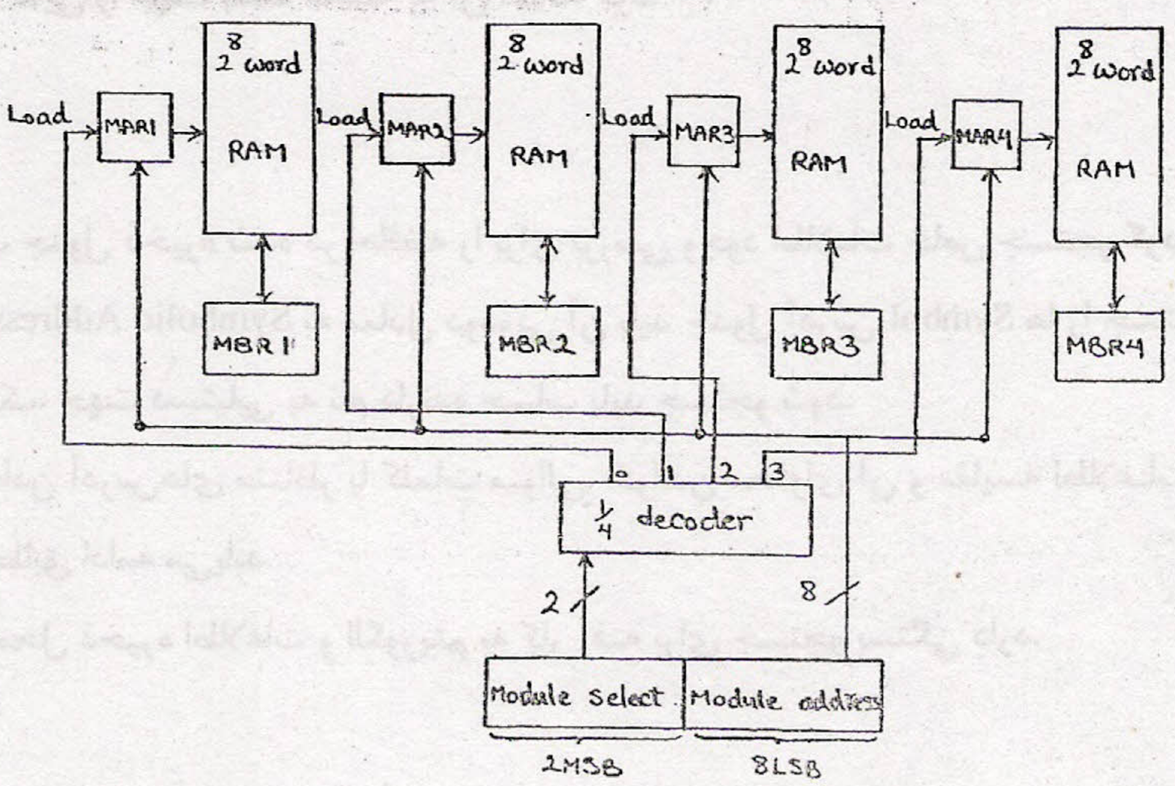
کامپیوترها برای پردازش برنامه ابتدا دستورالعمل‌ها را از حافظه واکشی و داده را از حافظه دریافت یا به آن ارسال می‌دارند.

به منظور کاهش زمان دستیابی به حافظه و در نتیجه افزایش سرعت پردازش برنامه‌ها باید بتوان بیش از یک Read و Write را هم‌زمان انجام داد. در صورت استفاده از یک RAM، انجام چندین Read و چندین Write به طور هم‌زمان ممکن نخواهد بود. زیرا به هنگام Read یا Write، خطوط Address و Data در طول Memory cycle به طور مداوم بایستی آدرس و داده متناظر با کلمه حافظه را در بر داشته باشد. برای رفع این محدودیت می‌توان RAM را به چندین Module تقسیم نمود که هر کدام MAR , MBR مختص به خود داشته باشند. در این صورت Module ها قادر خواهند بود چندین Read و یا Write را هم‌زمان انجام دهند. کاربرد اصلی این نوع حافظه‌ها در Pipeline – Processor و Array - Processor ها می‌باشد.

دو روش اصلی برای طراحی چنین حافظه‌هایی وجود دارد :

روش اول: بیت‌های بالاترین رتبه برای انتخاب Module و بیت‌های پایین‌ترین رتبه برای انتخاب کلمه درون آن به‌کار می‌رود.

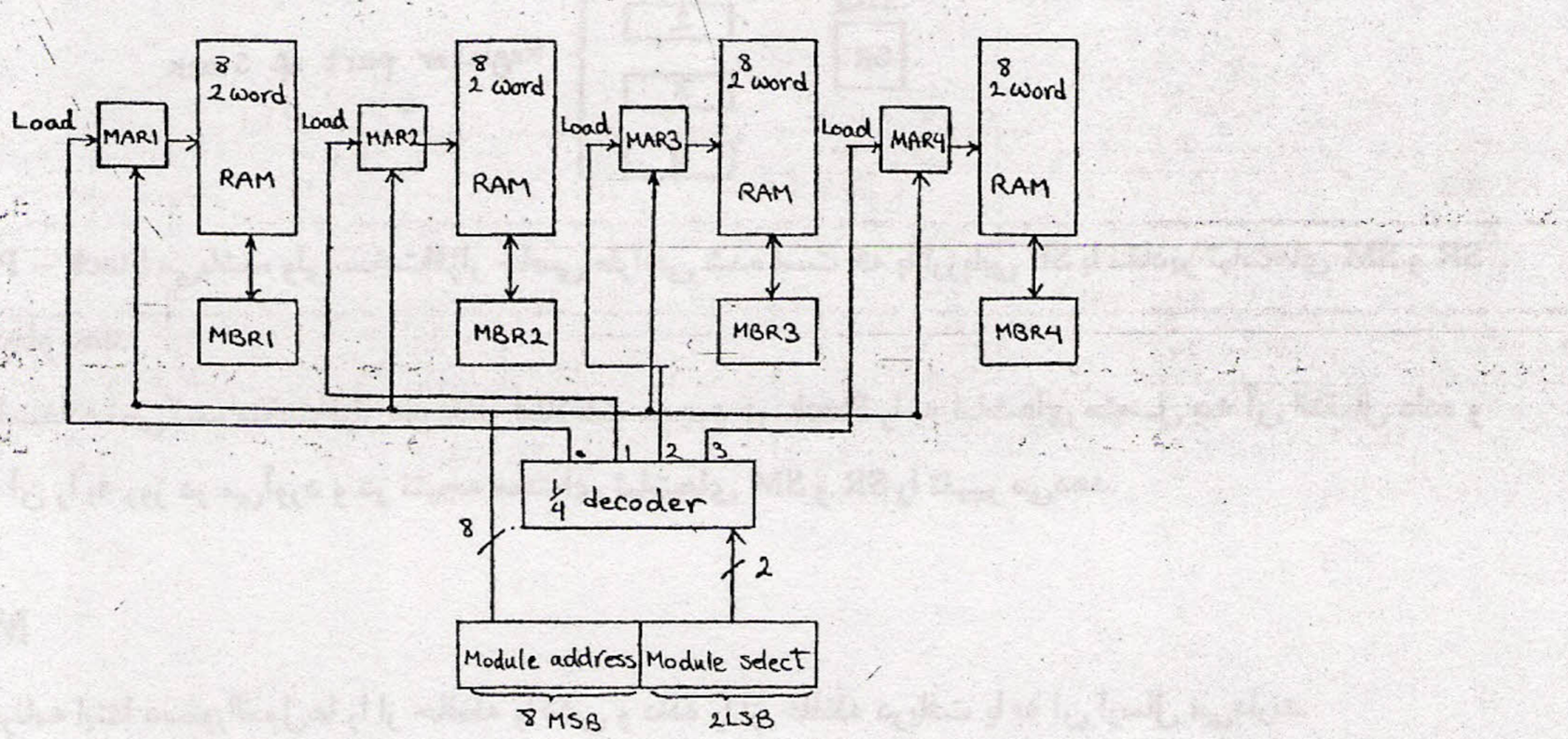
شکل زیر حافظه‌ای با گنجایش 2^{10} word را نشان می‌دهد که به چهار Module با گنجایش 2^8 word تقسیم شده است.



در این ساختار کلمات متوالی حافظه در Module یکسان قرار دارند و چون دستورات برنامه معمولاً به طور متوالی پردازش می‌شوند، تنها یک Module برای پردازش برنامه مشغول خواهد بود، در نتیجه دستگاه‌های ورودی و خروجی مجهز به DMA قادرند با سایر Moduleها ارتباط برقرار کنند.

روش دوم: بیت‌های پایین‌ترین رتبه برای انتخاب Module و بیت‌های بالاترین رتبه برای انتخاب کلمه درون آن به‌کار می‌رود.

چنین حافظه‌ای را «Memory Interleaving » نامند که در شکل زیر نشان داده شده است:



در این روش کلمات متناظر با دستورات متوالی در Module های متوالی و متفاوت قرار دارند و بنابراین چندین دستورالعمل را می‌توان هم‌زمان واکشی نمود. این روش در صورت پردازش دستورالعمل‌ها به طور متوالی باعث افزایش سرعت می‌شود، زیرا هم‌زمان با اجرای دستورالعمل جاری می‌توان دستور متوالی را واکشی نمود.

در صورتی‌که دستورالعمل جاری از نوع انشعاب شرطی باشد ممکن است دستوری که هم‌زمان با اجرای آن واکشی شده است، اجرا نشود. به هر حال زمان اضافی برای این منظور تلف نخواهد شد.

هر چند استفاده ازMemory Interleaving به هر حال زمان پردازش را کاهش می‌دهد، ولی روش اول انعطاف‌پذیری بیشتری دارد، مثلاً اگر k- bit برای انتخاب Module اختصاص یافته باشد باید 2^k – Module در ساختار Memory Interleaving وجود داشته باشد، در غیر این صورت حفره‌هایی به مفهوم عدم وجود اطلاعات در فضای نشانی‌دهی حافظه اصلی وجود خواهد داشت.

بر عکس در روش اول تعداد دلخواهی از Module ها را می‌توان در حافظه شامل نمود و نیازی به شامل نمودن همه آن‌ها نیست.

در آینده می‌توان به آسانی Module هایی را جهت بسط حافظه به آن اضافه کرد.

Associative Memory

در پردازش اطلاعات گاهی لازم است جدول ذخیره شده در حافظه را برای بررسی وجود اطلاعات خاص جستجو کرد.

مثلاً برنامه اسمبلر برای تبدیل Symbolic Address به معادل دودویی آن باید جدول آدرس Symbol ها را جستجو کند و یا فایل متناظر با شماره حساب مشتریان بانک، جهت دستیابی به نام دارنده حساب باید جستجو شود.

معمولاً جستجوی یک فایل با دادن آدرس‌های متناظر با کلمات متوالی، خواندن محتوای آن و مقایسه اطلاعات به طور متوالی انجام می‌شود و این عمل تا مرحله تطابق ادامه می‌یابد.

تعداد دستیابی به کلمات حافظه به محل ذخیره اطلاعات و الگوریتم به کار رفته برای جستجو بستگی دارد.

زمان جستجو را می‌توان به عوض دادن آدرس به حافظه، با ارایه خود محتوا کاهش داد. حافظه‌ای که دستیابی به اطلاعات آن بر اساس ارایه محتوا انجام گیرد Associative Memory و یا Content Addressable Memory (CAM) نامیده می‌شود.

در این نوع حافظه‌ها، دستیابی به کلمات هم‌زمان و به‌طور موازی انجام می‌شود.

به هنگام Write در چنین حافظه‌ای نیاز به دادن آدرس نیست، بلکه حافظه قادر است به طور اتوماتیک کلمه خالی را پیدا کند و Data را در آن ذخیره نماید.

به هنگام Read کل محتوی یا بخشی از Data به حافظه ارایه می‌گردد و حافظه تمام کلماتی را که با Data ارایه داده شده تطابق داشته باشد، علامت‌گذاری کرده و برای خواندن آماده می‌کند.

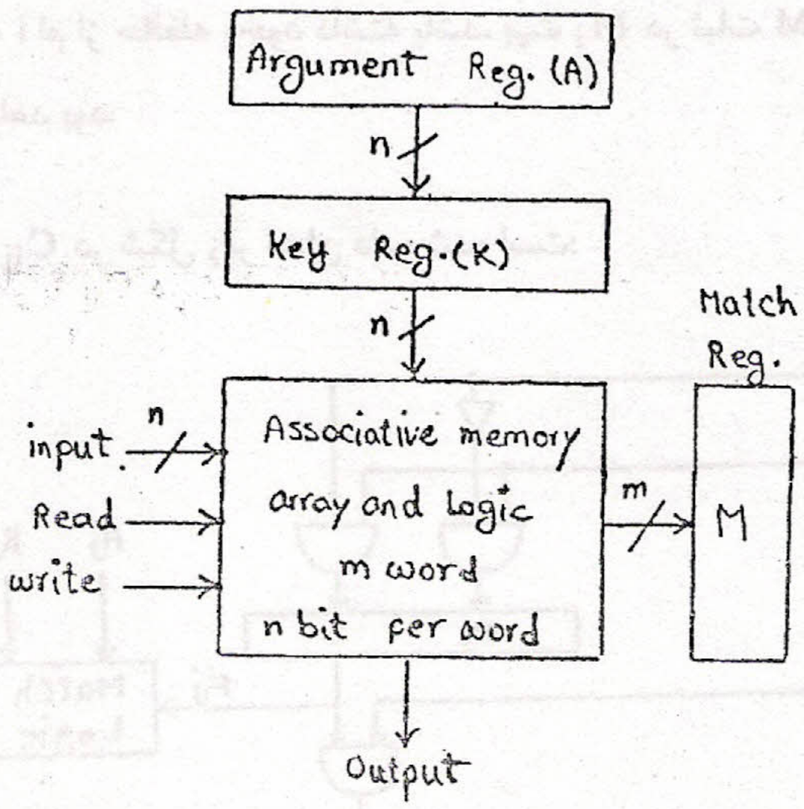
به دلیل سازمان خاصی که حافظه انجمنی دارد، استفاده از این حافظه تنها هنگامی مناسب خواهد بود که عمل جستجو به طور موازی و با مشارکت Data انجام گیرد. حافظه انجمنی بسیار گران‌تر از حافظه RAM است، زیرا علاوه بر قابلیت خواندن و نوشتن، مدار منطقی خاصی را در بر دارد که می‌تواند محتوای خود را با Data ارایه داده شده مقایسه کند و وجود تطابق را تشخیص دهد.

به همین دلیل از این حافظه تنها وقتی استفاده می‌شود که زمان جستجو بحرانی بوده و باید بسیار کوتاه باشد.

«Hardware Organization»

نمودار بلوکی حافظه انجمنی در شکل صفحه بعدی نشان داده شده است.

هر یک از کلمات حافظه به طور موازی با محتوای ثبات A مقایسه می‌شود و FF نظیر کلماتی که با محتوای A تطابق دارند، در ثبات M برابر 1 می‌شوند و سپس عمل خواندن محتوای کلماتی که FF آن‌ها Set می‌باشد، با دستیابی پشت سر هم انجام می‌گیرد. ثبات K به منظور Mask نمودن محتوای ثبات A به کار می‌رود. اگر تمامی بیت‌های ثبات K برابر 1 باشد، کلیه بیت‌های ثبات A با محتوای کلمات حافظه به طور هم‌زمان مقایسه می‌شوند و در غیر این‌صورت، تنها محتوای ناحیه‌ای از ثبات A که بیت‌های نظیر آن در ثبات K مساوی با 1 باشد، در مقایسه شرکت خواهند کرد. بنابراین نقش ثبات K، مشخص نمودن قسمتی از اطلاعات است که در عمل جستجو باید شرکت کند.

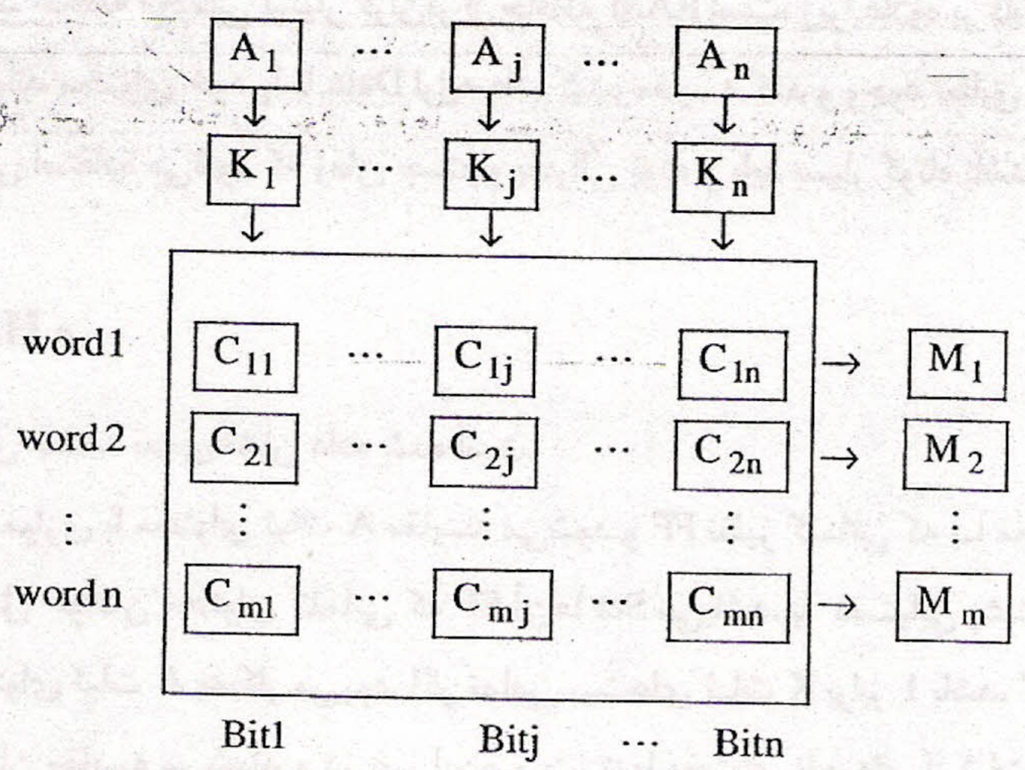


برای نشان دادن مفهوم فوق به مثال عددی زیر توجه کنید:

A	(101)	111100	
K	111	000000	
word1	100	111100	no match
word2	(101)	000001	match

چون فقط 3 بیت بالاترین رتبه ثبات K مساوی 1 می‌باشد، بنابراین 3 بیت بالاترین رتبه ثبات A با 3 بیت بالاترین رتبه در همه کلمات حافظه انجمنی به‌طور موازی و هم‌زمان مقایسه می‌شود، پس فقط FF نظیر کلمه دوم در ثبات M، 1 خواهد شد.

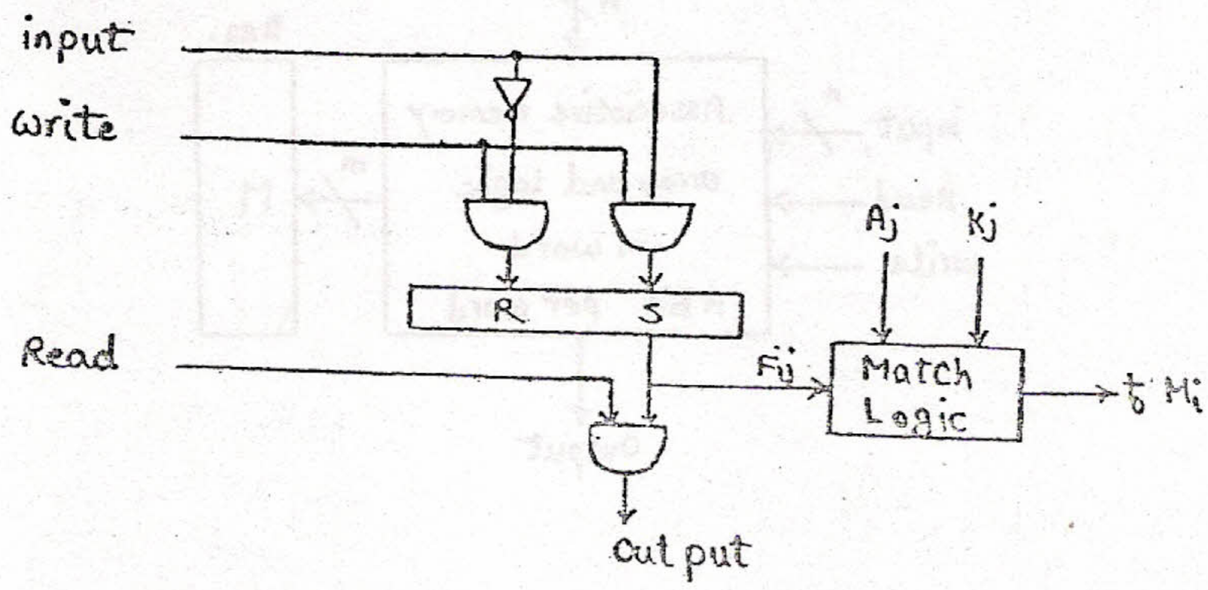
رابطه بین آرایه حافظه و ثبات‌های جانبی آن در شکل زیر نشان داده شده است:



اگر $k_j=1$ باشد، A_j با کلیه بیت‌های هم رتبه خود که در ستون j ام قرار دارند به‌طور هم‌زمان مقایسه می‌شوند و این عمل برای $j=1,2,....,n$ نیز انجام خواهد گرفت.

اگر تطابق بین محتوای ثبات A و کلمه i ام از حافظه وجود داشته باشد، بیت M_i در ثبات M مساوی 1 و اگر یک یا چند بیت از آن‌ها یکسان نباشد، بیت M_i مساوی 0 خواهد بود.

سازمان داخلی یک Binary cell متناظر با C_{ij} در شکل زیر نشان داده شده است:



ابتدا بیت K_j را در نظر نمی‌گیریم. با توجه به رابطه $(j=1,2,...,n): x_j = A_j F_{ij} + A'_j F'_{ij}$

اگر $A_j = F_{ij}$ باشد $\Rightarrow x_j = 1$ اگر $A_j \neq F_{ij} \Rightarrow x_j = 0$

بنابراین برای این‌که محتوای کلمه i ام با محتوای ثبات A یکسان باشد، باید داشته باشیم:

$$M_i = x_1 x_2 ... x_n$$

حال بیت K_j را در نظر می‌گیریم، باید k_j را در منطق مقایسه شامل کنیم، در نتیجه:

اگر $k_j = 0 \Rightarrow$ نیاز به مقایسه A_j با F_{ij} نخواهد بود و باید Match Logic در خروجی مقدار یک تولید کند.

اگر $k_j = 1 \Rightarrow$ نیاز به مقایسه A_j با F_{ij} خواهد بود و در نتیجه ابتدا Match Logic مقایسه را انجام دهد. در صورتی‌که یکسان باشند، خروجی یک و یکسان نباشند، مقدار صفر در خروجی تولید کند. در نتیجه رابطه بولی به صورت زیر خواهد بود:

$$x_j + k'_j = \begin{cases} x_j & k_j = 1 \\ 1 & k_j = 0 \end{cases}$$

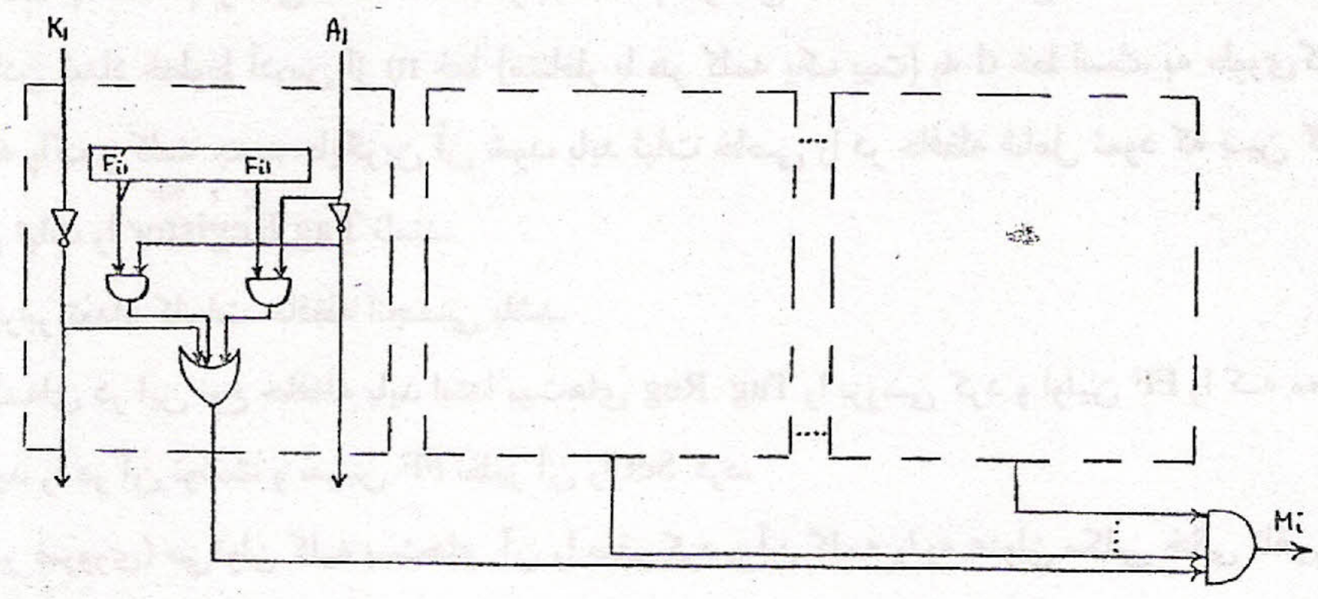
با در نظر گرفتن رابطه فوق، Match Logic متناظر با کلمه i ام را می‌توان با تابع بولی زیر نشان داد:

$$M_i = (x_1 + k'_1)(x_2 + k'_2)...(x_n + k'_n)$$

یا

$$M_j = \prod_{j=1}^{n} (A_j F_{ij} + A'_j F'_{ij} + K'_j) \qquad \text{for} \quad j = 1, 2, ..., n$$

مدار منطقی لازم برای تطابق یک کلمه در شکل زیر نشان داده شده است:



توجه کنید که Not –gate متناظر با k_j, A_j در هر ستون تنها یک بار ظاهر خواهد شد، ولی خروجی آن به طور مشترک در کلیه B.C های یک ستون استفاده می‌شود. حال اگر $M_i = 1$ باشد، بین محتوای ثبات A و کلمه i ام از حافظه انجمنی، تطابق وجود خواهد داشت و اگر $M_i = 0$ باشد، عدم تطابق را نشان خواهد داد.

اگر همه بیت‌های ثبات K مساوی صفر باشد، بدون توجه به محتوای ثبات A خروجی‌های M_i مساوی با 1 و در نتیجه کلیه FF های ثبات M با 1 پر خواهند شد و باید از این وضع در عملکرد عادی حافظه جلوگیری کرد.

« Read Operation»

اگر بیش از یک کلمه با محتوای ثبات A تطابق داشته باشد، بیش از یک بیت در ثبات M مساوی با 1 خواهد بود، در این صورت لازم است بیت‌های ثبات M را یک به یک بررسی نمود. و اگر بیتی در ثبات M مساوی 1 باشد، خواندن محتوای آن به توالی و با فعال کردن Read Signal به هر کلمه انجام می‌گیرد.

در برخی کاربردها، جدول ذخیره شده در حافظه انجمنی فاقد کلمات با متحوای یکسان است، در این صورت تنها یک کلمه از حافظه با محتوای ثبات A تطابق خواهد داشت، در نتیجه خروجی M_i به عوض وصل شدن به ثبات M، می‌تواند مستقیماً به Read Signal کلمه متناظرش وصل شود و محتوای کلمه تطابق یافته را در خروجی حافظه ظاهر کرد که در این صورت به Read Signal خاصی نیاز نخواهد بود، به علاوه اگر کلمه‌ای با محتوای صفر در حافظه انجمنی وجود نداشته باشد، خروجی متناظر با کلیه خطوط صفر نشان‌دهنده عدم تطابق خواهد بود.

« Write Operation»

حافظه انجمنی علاوه بر قابلیت خواندن و نوشتن اطلاعات، باید عمل جستجو را نیز در بر داشته باشد.

شیوه نوشتن در حافظه‌های انجمنی با توجه به کاربرد آن متفاوت است. محتوای کلیه کلمات حافظه انجمنی قبل از انجام جستجو باید به اطلاعات جدید تغییر یابد، در این صورت عمل نوشتن را می‌توان با دادن آدرس متوالی و با استفاده از رمزگشایی همانند RAM انجام داد.

این روش سبب خواهد شد، حافظه به هنگام نوشتن مانند RAM و به هنگام خواندن مانند CAM عمل کند.

مزیت این روش، کاهش دادن تعداد خطوط آدرس از m خط (متناظر با هر کلمه یک بیت) به d خط است، به طوری‌که $m = 2^d$ اگر کلمهٔ غیر ضروری باید از حافظه پاک و کلمه جدید جایگزین آن شود، باید ثبات خاصی را در حافظه شامل نمود که بین کلمات فعال و غیر فعال تفاوت قایل شود. این ثبات را Tag Register نامند.

تعداد FF های Tag Reg باید برابر تعداد کلمات حافظه انجمنی باشد.

به منظور ذخیره نمودن کلمه‌ای در این نوع حافظه باید ابتدا بیت‌های Tag Reg را بررسی کرد و اولین FF را که محتوایش صفر است، انتخاب کرد و Data جدید را در آن نوشت و سپس FF نظیر آن را Set کرد.

به منظور ذخیره یک کلمه (غیر ضروری) می‌توان کلیه بیت‌های آن را صفر کرد و آن کلمه را به عنوان مکانی خالی تلقی کرد.

به علاوه کلماتی را که بیت نظیر آن‌ها در Tag Reg مساوی صفر است، باید به کمک Key-Register و Mask نمود تا در مقایسه فقط کلمات فعال شرکت کنند.

CACHE MEMORY (حافظه نهان): در فصل حافظه‌ها به‌طور مفصل مورد مطالعه قرار خواهد گرفت.

RISC & CISC Computers

در معماری کامپیوتر یکی از جنبه‌های مهم طراحی، مجموعهٔ دستورالعمل‌های پردازنده آن می‌باشد. در کامپیوترهای قدیمی به منظور کاهش سخت‌افزار، مجموعه دستورالعمل‌های ماشین، ساده و محدود بودند. با پیشرفتی که در تولید ICها ایجاد شد، قیمت سخت‌افزار کاهش یافت در نتیجه تعداد دستورالعمل‌های کامپیوترها بیشتر شد و هم‌چنین پیچیدگی دستورالعمل‌ها افزایش یافت، به طوری که اغلب کامپیوترهای رایج امروزی حدود 100 تا 300 دستورالعمل دارند و از انواع Data Type و Addressing Mode استفاده می‌کنند.

تمایل به داشتن سخت‌افزار پیچیده در کامپیوترهای رایج به سبب ایجاد تسهیلاتی در تبدیل برنامه به زبان سطح بالا به برنامه معادل زبان ماشین، زبان‌های کاربردی و کوشش در ایجاد ماشین‌هایی است که توابع را از پیاده‌سازی نرم‌افزاری به پیاده‌سازی سخت‌افزاری سوق دهند.

کامپیوتر با تعداد زیاد از چنین دستورالعمل‌ها را Complex Instruction Set Computer (CISC) گویند.

جدول زیر مشخصات برخی کامپیوترها بر حسب تعداد دستورالعمل‌ها، طول دستورالعمل‌ها، روش‌های نشانی‌دهی، تعداد ثبات‌ها، گنجایش حافظه کنترل و Cache را نشان می‌دهد:

مشخصات	CISC			RICS	
	IBM 370/168	VAX 11/780	Intel 80486	Motorola 88000	MIPS R 4000
سال تولید	1973	1978	1989	1988	1991
تعداد دستورالعمل‌ها	208	303	255	51	94
طول دستورالعمل‌ها (bytes)	2-6	2-57	1-11	4	32
تعداد ثبات‌های همه منظوره	16	16	8	32	32
گنجایش حافظه کنترل (K bits)	420	480	246	-	-
گنجایش حافظه Cache (Kbytes)	64	64	8	16	128

در اوایل سال 1980، تعدادی از طراحان کامپیوتر پیشنهاد کردند که کامپیوترها از مجموعه دستورالعمل‌های کم با ساختمان ساده استفاده کنند. این کامپیوترها ارجاع به حافظه را کاهش می‌دهند و اکثراً با استفاده از ثبات‌های درون CPU، دستورالعمل‌ها را با سرعت زیاد اجرا می‌کنند و معروف به کامپیوترهای Reduced Instruction Set Computer (RISC) هستند.

مشخصات CISC :

در طراحی مجموعه دستورالعمل‌های کامپیوتر نه تنها به ساختمان زبان ماشین بلکه به ملزومات زبان سطح بالای برنامه‌سازی نیز باید توجه کرد.

تبدیل برنامه از سطح بالا به زبان ماشین به وسیله کامپایلر انجام می‌گیرد و تمایل به تولید دستورالعمل‌های پیچیده به منظور آسان نمودن این تبدیل می‌باشد.

وظیفه کامپایلر تولید مجموعه دستورالعمل‌های زبان ماشین برای هر Statement می‌باشد.

مهم‌ترین هدف در معماری کامپیوترهای CISC مبتنی بر این است که هر یک Statement به یکInstruction تبدیل شود.

نمونه‌هایی از این معماری، کامپیوترهای IBM , VAX می‌باشد.

مشخصه دیگر معماری CISC وجود دستورالعمل‌ها با طول‌های متفاوت است.

مثلاً دستورالعمل‌هایی که اپراندهای خود را مستقیماً از ثبات‌ها دریافت می‌کنند، ممکن است فقط شامل 2 بایت، ولی دستورالعمل‌هایی که ارجاع به حافظه دارند، ممکن است به 5 یا 6 بایت نیاز داشته باشند، در نتیجه جا دادن دستورالعمل‌ها با طول و فرمت گوناگون در کلمات حافظه به رمزگشایی پیچیده‌ای نیاز دارد.

در کامپیوترهای CISC بیشتر دستورالعمل‌ها اپراندهای خود را از حافظه دریافت می‌کنند.

مثلاً دستورالعمل Add ممکن است یکی از اپرندهای خود را با Index Addressing و دیگری را با Direct Addressing دریافت کند و نتیجه را نیز با روش نشانی‌دهی خاصی در حافظه ذخیره کند.
در نتیجه این دستورالعمل نیاز به 3 ارجاع به حافظه دارد.

هر چند در کامپیوترهای CISC دستورالعمل‌هایی وجود دارد که اپرندهای خود را مستقیماً از ثبات‌های درون CPU دریافت می‌کنند، ولی وجود انواع عملیات و روش‌های نشانی‌دهی مختلف کار کامپایلر را در تبدیل برنامه به زبان سطح بالا آسان می‌کند. به هر حال استفاده از انواع متعدد دستورالعمل‌ها و روش‌های نشانی‌دهی، سخت‌افزار بیشتری نیاز دارد و در نتیجه سرعت عملیات را پایین می‌آورد.

به طور خلاصه مشخصات معماری کامپیوترهای CISC را می‌توان به صورت زیر فهرست نمود:

۱) وجود تعداد زیاد دستورالعمل، معمولاً بین 100 تا 300 دستورالعمل

۲) وجود برخی از دستورالعمل‌ها در طول اجرای برنامه که وظیفه خاصی را انجام می‌دهند و کمتر مورد پردازش قرار می‌گیرند.

۳) استفاده از روش‌های نشانی‌دهی متعدد، معمولاً بین 5 تا 20 روش

۴) وجود فرمت گوناگون برای دستورالعمل‌ها با طول متفاوت

۵) وجود تعداد زیادی از دستورالعمل‌ها که ارجاع به حافظه را نیاز دارند.

جدول زیر برخی مشخصات زبان‌های سطح بالا را به هنگام عملیات نشان می‌دهد:

	Dynamic Occurence		Machine Instruction Weighted		Memmory Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
Assign	45	38	13	13	14	15
Loop	5	3	42	32	33	26
Call	15	12	31	32	44	45
If	29	43	11	21	7	13
Goto	-	3	-	-	-	-
Other	6	1	3	1	2	1

مشخصات RISC :

معماری کامپیوترهای RISC بیشتر سعی می‌کند با ساده کردن مجموعه دستورالعمل‌ها، زمان اجرا دستورالعمل‌ها را کاهش دهد و جنبه‌های اصلی آن عبارت‌اند از:

۱) نسبتاً از تعداد دستورالعمل‌های کمتری استفاده می‌کنند.

۲) نسبتاً از تعداد روش‌های نشانی‌دهی کمتری استفاده می‌کنند.

۳) ارجاع به حافظه فقط به دستورالعمل‌های LOAD و STORE محدود می‌شود.

۴) کلیه عملیات روی اپرندهای موجود در ثبات‌های درون CPU انجام می‌شود.

۵) طول دستورالعمل‌ها ثابت، در نتیجه رمزگشایی آن‌ها آسان می‌شود.

۶) بیشتر دستورالعمل‌ها در یک Clock Cycle انجام می‌گیرد (با استفاده از روش pipeline مناسب).

۷) در طراحی واحد کنترل به عوض ریزبرنامه‌سازی، از روش سیم‌بندی شده استفاده می‌شود.

چون در کامپیوترهای RISC بیشتر دستورالعمل‌ها از نوع Register-to-Register می‌باشد، در نتیجه اجـرا آن‌هـا در یـک Clock Cycle ممکن خواهد بود که این عمل را می‌توان با Overlap کردن مراحل Fetch ، Decode و Execute دو یا سه دستورالعمل و با اسـتفاده از رویه Pipeline ممکن ساخت.

سایر مشخصات معماری کامپیوتر RISC عبارت‌اند از:

۱) وجود تعداد زیادی ثبات درون CPU

۲) استفاده از Overlapped Register Windows برای افزایش سرعت در Procedure call و Return

۳) استفاده مناسب از Pipeline

۴) پشتیبانی Compiler برای ترجمه کارای برنامه نوشته شده به زبان‌های سطح بالا به برنامه معادل به زبان ماشین

۱) وجود تعداد زیادی ثبات در ذخیره نمودن نتایج میانی بسیار مفید خواهد بود، زیرا بر خلاف کامپیوترهای CISC که در آن به هنگام Return & Procedure call انتقال پارامترها و نتایج، به کمک کلمات حافظه انجام می‌شود، در کامپیوترهای RISC می‌توان این انتقـال اطلاعات بین Procedure ها را مستقیماً به‌وسیله ثبات‌ها انجام داد و از ارجاع به حافظه جلوگیری نمود.

2) « Overlapped Register Windows »

پنجره‌های ثبات همپوشان

در زبان‌های برنامه‌سازی سطح بالا Procedure Call & Return به طور مکرر انجام می‌گیرد و به هنگـام ترجمـه بـه زبـان ماشـین یـک Procedure Call به دنباله‌ای از دستورات زبان ماشین تبدیل می‌گردد. این دستورات در ذخیره نمودن مقادیر ثبات‌ها، انتقال پارامترهـا و فعال نمودن Subroutineها به کار می‌رود.

بعد از بازگشت از رویه مجدداً برنامه مقادیر قبلی ثبات‌ها را ذخیره می‌کند و نتایج را به برنامه فراخوانده انتقال می‌دهد.

این چنین ذخیره‌سازی و انتقال پارامترها و نتایج حاصل، مدت زمان بیشتری را به خود اختصاص می‌دهد که برخی کامپیوترهـا بـه این منظور از Memory Stack استفاده می‌کنند. بنابراین ارجاع مکرر به منظور push و pop به حافظه ایجاد می‌گردد که باعـث اتـلاف وقت بیشتری خواهد بود.

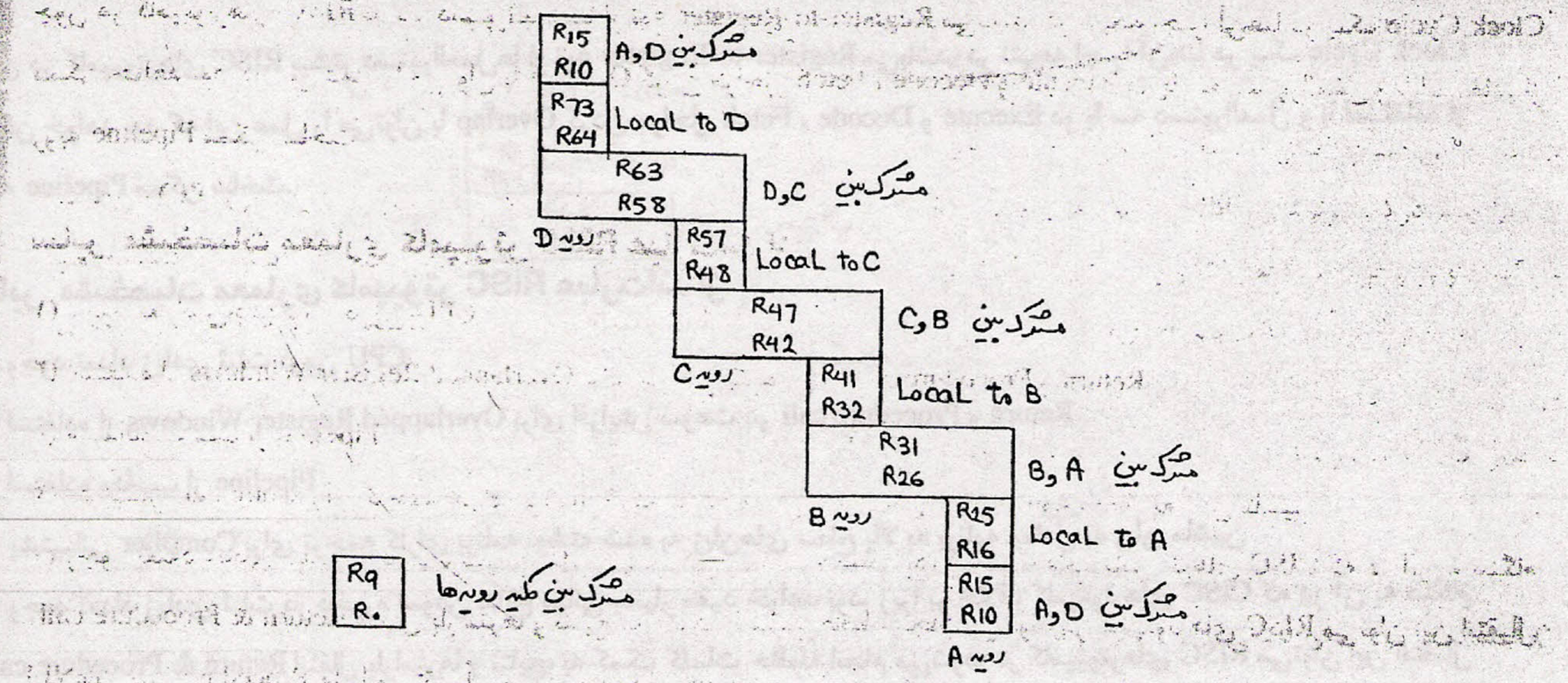
در کامپیوترهای RISC از Overlapped Register Window استفاده می‌شود که می‌تواند انتقال پارامترها را مسـتقیماً انجـام دهـد و از ذخیره‌سازی و دوباره برگرداندن محتوای ثبات‌ها جلوگیری کند.

هر call به رویه با نمو دادن Pointer یک Reg.window را فعال می‌کند.

و هر Return به رویه با کاهش دادن Pointer یک Reg.window قبلی را فعال می‌کند.

Windowهای متناظر با رویه‌های مجاور شامل ثبات‌های مشترکی هستند که به کمک آن‌ها پارامترها و نتایج بین رویـه‌هـا، مبادلـه می‌شود که این مفهوم را می‌توان به صورت شکل زیر نشان داد.

فرض کنید، سیستم شامل 74 ثبات است که ثبات‌های (9 تا 0) را Global (ثبات‌های عام) نامند و پارامترهای مشـترک بـین کلیـه رویه‌ها را در خود ذخیره می‌کند. 64 ثبات باقیمانده بین 4 Window تقسیم شده‌اند که به رویه‌های A , B , C و D اختصاص یافته‌اند. هر Reg . window شامل 10 ثبات Local (محلی) و 2 مجموعه 6 ثباتی مشترک بین Windowهای مجاور است.



ثبات‌های Local برای ذخیره نمودن متغیرها از نوع Local و ثبات‌های Common برای تعویض پارامترها و نتایج حاصل بین رویه‌های مجاور به کار می‌روند. بنابراین ثبات‌های مشترک سبب می‌شوند Data بدون جابه‌جایی واقعی توسط Window های مجاور مورد استفاده قرار گیرد.

در هر لحظه تنها یک Window فعال است و ثبات‌های سطح بالای رویه فراخواننده با ثبات‌های سطح پایین رویه فراخوانده شده با هم Overlap می‌شوند، بنابراین پارامترها به طور اتوماتیک بین رویه‌ها انتقال می‌یابد.

به عنوان مثال:

فرض کنید رویه A رویه B را فراخواند. ثبات‌های 31 تا 26 بین آن دو مشترک‌اند، در نتیجه رویه A پارامترهای لازم برای B را در این ثبات ذخیره می‌کند. رویه B از ثبات‌های 41 تا 32 برای ذخیره نمودن متغیرهای Local استفاده می‌کند.

حال اگر رویه B بخواهد رویه C را فراخواند، B نیز پارامترها را توسط ثبات‌های 47 تا 42 به رویهٔ C انتقال می‌دهد.

به هنگام بازگشت از رویه B، نتایج حاصل در ثبات‌های 31 تا 26 ذخیره می‌شود و با کاهش یافتن Pointer کنترل به Window متناظر با رویه A بر می‌گردد. توجه داشته باشید که ثبات‌های 15 تا 10 بین رویه‌های A , D مشترکند و در نتیجه Windows ها یک سازمان حلقوی شکل تشکیل می‌دهند. با توجه به شکل فوق متوجه می‌شویم که هر رویه فعال 32 ثبات شامل:

10 Global Reg , 10 Local Reg , 6 High Overlapping Reg , 6 Low Overlapping Reg

در اختیار خواهد داشت.

در حالت کلی سازماندهی Register ها را می‌توان با رابطه زیر بیان نمود:

تعداد ثبات‌های مشترک بین دو Window مجاور =C ، تعداد ثبات‌های G = Global

تعداد Window ها =W ، تعداد ثبات‌های L = Local

تعداد ثبات‌های موجود در هر Window به صورت زیر محاسبه می‌شود:

Window size = L+2C+G

و تعداد کل ثبات‌های موردنیاز پردازنده عبارت است از:

Register File = (L+C) W+G

برای سیستم قبل داریم: Register File = (10+6) 4+10=74 و Window Size = 10+ 2(6) + 10 = 32

«بررسی چند نوع کامپیوتر RISC»

Berkeley RISC1 اولین پروژه برای نشان دادن مزایای معماری RISC در دانشگاه Berkeley صورت گرفت.

RICS I شامل یک "32 bit CPU" به صورت مدار مجتمع است.

آدرس آن 32 بیتی بوده و داده‌های آن ممکن است 16،8 و یا 32 بیتی باشند. کلاً شامل 31 دستورالعمل 32 بیتی است.

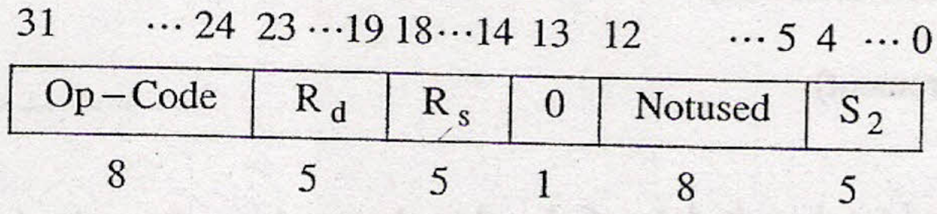
روش‌های نشانی‌دهی آن عبارت‌اند از:

Register addressing , Immediate , Relative to PC (برای دستورات انشعاب شرطی)

Register File آن شامل 138 ثبات است، شامل 8 Window , 10 Golobal (هر کدام شامل 32 ثبات) و سازمان آن مشابه سازمانی است که قبلاً توضیح داده شد.

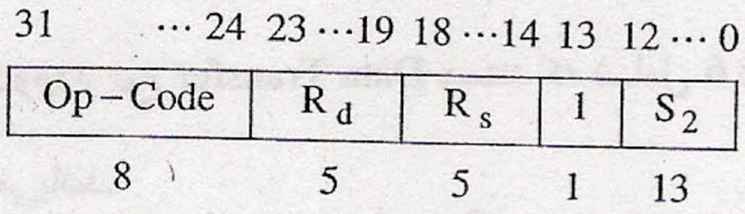
چون در هر لحظه تنها یک Window فعال خواهد بود، بنابراین آدرس ثبات‌ها را می‌توان با شامل نمودن یک ناحیه 5 بیتی در فرمت دستورالعمل مشخص نمود.

دستورالعمل‌های RISC1 شامل 3 فرمت زیر می‌باشد:



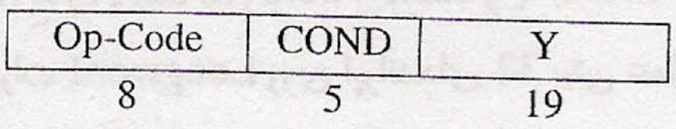
a) Register mode

(S_2 مشخص کننده آدرس یک ثبات از Window)



b) Register Immediate mode

(S_2 مشخص کننده یک اپراند)



c) PC Relative mode (y=displacement)

هر دستورالعمل شامل 32 بیت است که در آن 7 بیت از ناحیه opcode نوع عمل و بیت هشتم مشخص کننده تغییر یا عدم تغییر Status bit در ALU خواهد بود. Reg – To – Reg

در دستورالعمل‌های نوع Reg-to-Reg ناحیه 5 بیتی R_d مشخص کننده محل ذخیره نتایج حاصل از اجرای دستورالعمل است و داده‌هایی که عمل روی آن‌ها انجام می‌گیرد، توسط نواحی R_S و S_2 مشخص می‌شود.

اگر بیت شماره 13، صفر باشد، 5 بیت واقع در سمت راست ناحیه S_2 یکی از ثبات‌های منبع را مشخص می‌کند.

اگر بیت شماره 13 یک باشد، ناحیه 13 بیتی S_2 نشانگر یک 13- bit constant خواهد بود که ممکن است علامت‌دار یا بدون علامت باشد.

بنابراین در دو فرمت اول از دستورالعمل‌ها که هر کدام شامل 13 آدرس است، اپراند دوم ممکن است محتوای یکی از 32 ثبات و یا داده‌ای به صورت Immediate باشد.

در دستورالعمل‌هایی که نیاز به ارجاع به حافظه دارند، R_S یک آدرس 32 بیتی واقع در ثبات و S_2 نشان‌دهندهٔ offset است و محتوای ثبات R_0 همیشه صفر خواهد بود.

در فرمت سوم از دستورالعمل‌ها، 3 ناحیه آخر با هم ترکیب شده و یک 19 – bit Relative address تولید می‌کنند که در دستورالعمل‌های Jump , Call به کار می‌رود.

ناحیه COND در دستورالعمل‌های Jump، به منظور بررسی وجود یا عدم وجود یکی از 16 شرط شامل شده است.

31 دستورالعمل RISC1 در 3 گروه دسته‌بندی شده‌اند:

گروه اول: عملیات Shift , Logic , Arithmetic را انجام می‌دهند:

مثلاً به کمک دستورالعمل ADD از گروه اول می‌توان عملیات گوناگونی انجام داد.

ADD	R_{22}, R_{21}, R_{23}	$R_{23} \leftarrow R_{22} + R_{21}$
ADD	$R_{22}, \neq 150, R_{23}$	$R_{23} \leftarrow R_{22} + 150$
ADD	$R_0 \neq 150, R_{22}$	$R_{22} \leftarrow 150$ (Load Immediate)
ADD	$R_{22} \neq 1, R_{22}$	$R_{22} + R_{22} + 1$ (Increment)

با استفاده از ثبات R_0 می‌توان محتوای ثباتی با یک مقدار ثابت را به ثبات دیگری منتقل کرد:

(Move)

ADD	R_0, R_{21}, R_{22}	$R_{22} \leftarrow R_{21}$

گروه دوم: از نوع Data Transfer هستند که شامل 6 دستورالعمل Load , 3 دستورالعمل Store و 2 دستورالعمل جهت انتقال PSW می‌باشد.

ثبات PSW شامل PC , Status bit , Pointer به Window ها و سایر اطلاعات مربوط به حالت CPU می‌باشد.

دستورالعمل‌های Load , Store به منظور انتقال Data بین ثبات‌های درون CPU و حافظه به کار می‌روند و ممکن است از نوع Short – word (16 bit) , byte (8bit) و یا Long – word (32bit) باشد.

دستورات Longword روی اپراندهای 32 بیتی عمل می‌کنند و به کمک آن‌ها می‌توان غیر از 3 نوع نشانی‌دهی ذکر شده، برخی روش‌های دیگر را نیز ایجاد کرد.

LDL	$(R_{22})\#150, R_5$	$R_5 \leftarrow M[R_{22}] + 150$

در این دستورالعمل برای تولید نشانی موثر از روش نشانی‌دهی Register Plus Displacement استفاده شده است.

LDL	$(R_{22})\#0, R_5$	$R_5 \leftarrow M[R_{22}]$

در این دستورالعمل با استفاده از مقدار ثابت صفر از روش نشانی‌دهی Register Indirect استفاده شده است.

LDL	$(R_0)\#500, R_5$	$R_5 \leftarrow M[500]$

در این دستورالعمل برای تولید محتوای ثابت R_0 از روش نشانی‌دهی Direct Addressing استفاده شده است.

گروه سوم: از نوع Program Control است که به منظور توالی‌دهی دستورالعمل‌های برنامه به کار می‌رود و شامل« 2 دستورالعمل Jump , دستورالعمل Call » است.

JMP	$COND, S_2(R_S)$	$PC \leftarrow R_S + S_2$

در این دستورالعمل از روش نشانی‌دهی Index Plus Displacement استفاده شده است.

JMPR COND, y $PC \leftarrow PC + y$

در این دستورالعمل از روش نشانی‌دهی Relative to PC استفاده شده است.

دستورالعمل‌های Call , Return از یک ثبات 3-bit Current Window Pointer (CWP) استفاده می‌کنند که در آن محتوای این ثبات به Register Window فعال اشاره می‌کند.

با هر Call , CWP یک واحد کاهش یافته و با هر Retuurn یک واحد اضافه می‌شود.

CALL $R_d, S_2(R_S)$ $R_d \leftarrow PC$, RET R_d, S_2 $PC \leftarrow R_d + S_2$

$PC \leftarrow R_S + S_2$ $CWP \leftarrow CWP + 1$

$CWP \leftarrow CWP - 1$ $CWP \leftarrow CWP + 1$

تمرین: برای هر یک از عملیات زیر مثالی از دستورالعمل RISC1 بنویسید:

a) Decrement a register
b) Complement a register
c) Negate
d) Clear a register to 0
e) Divide a signed number by 4
f) No Operation

تمرین: مشخصات 3 کامپیوتر که از مفهوم Register Window استفاده می‌کنند، در جدول زیر داده شده است.

برای هر یک، Window Size و تعداد کل ثبات‌های موجود در هر یک از کامپیوترها را محاسبه کنید:

	Computer 1	Computer 2	Computer 3
Global register	10	8	16
Local register	10	8	16
Common register	6	8	16
Number of windows	8	4	16

3) « RISC Pipelining »

استفاده از پردازش لوله‌ای معمولاً برای افزایش کارآیی CPU می‌باشد.

در معماری کامپیوترهای RISC اکثر دستورالعمل‌ها از نوع Reg-to-Reg می‌باشند، در نتیجه چرخه دستورالعمل شامل دو مرحله زیر می‌باشد:

I : Instruction Fetch

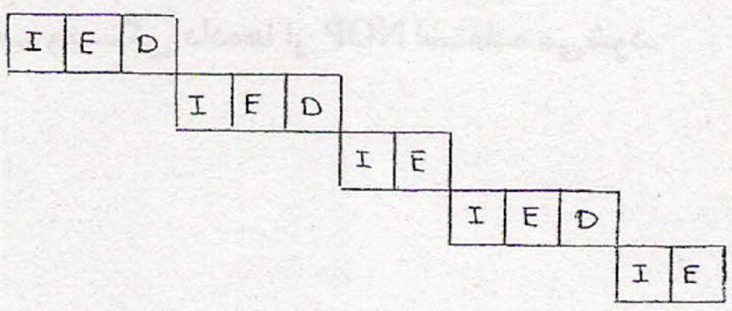
E : Execute (به مفهوم انجام عملیات در درون ALU با ورودی و خروجی متناظر با ثبات‌ها)

برای عملیات Load و Store می‌توان از 3 مرحله زیر استفاده نمود:

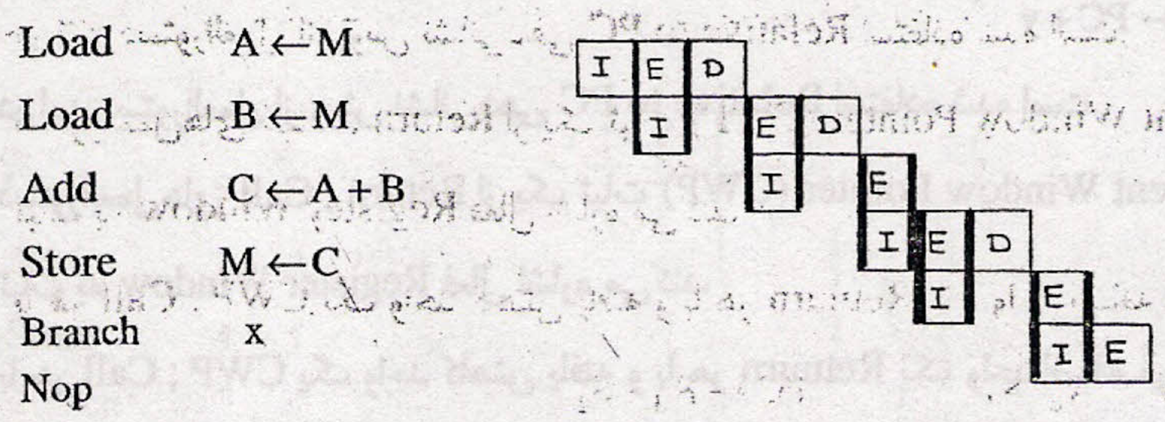
I : Instruction Fetch E: Execute D: (به مفهوم عملیات در ارتباط بین ثبات‌ها و کلمات حافظه)

شکل زیر زمان‌بندی مجموعه دستورالعمل‌هایی را نشان می‌دهد که بدون استفاده از Pipeline به طور Serial انجام می‌گیرد:

Load $A \leftarrow M$
Load $B \leftarrow M$
Add $C \leftarrow A + B$
Store $M \leftarrow C$
Branch x



با استفاده از تنها « 2-Segment Pipleline » که در آن مراحل I و E به طور هم‌زمان انجام می‌گیرد، می‌توان کارآیی سیستم را بالا بـرد و در نتیجه نمودار زمان‌بندی شده آن را به صورت شکل زیر نشان داد:



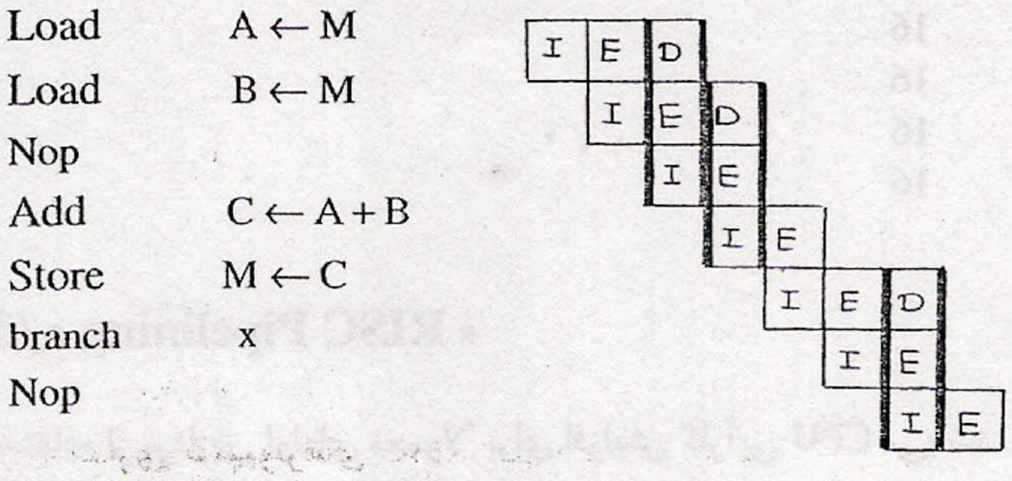
با این روش می‌توان سرعت پردازش دستورالعمل‌ها را تا حدود 2 برابر افزایش داد، ولی دو مشکل اساسی از رسیدن به حـداکثر سـرعت جلوگیری می‌کند:

مشکل اول در صورت استفاده از Single – Port Memory روی خواهد داد، چون در هر لحظه تنها یک دستیابی به حافظه وجـود دارد، در نتیجه در برخی از دستورالعمل‌ها باید از الحاق حالت Wait استفاده کرد.

مشکل دوم در ارتباط با اجرای دستورالعمل‌های انشعاب است که در ترتیب اجرای دستورالعمل‌ها وقفه ایجاد می‌کند.

برای رفع این مشکل باید از الحاق دستور NOP توسط کامپایلر یا اسمبلر استفاده نمود.

پردازش لوله‌ای را می‌توان با ممکن ساختن 2 دستیابی به حافظه در هر مرحله سرعت بیشتری بخشید و در نتیجه مراحل پـردازش دستورالعمل را می‌تـوان بـا اسـتفاده از « 3-Segment Pipeline » سـرعت بیشـتری بخشـید کـه در ایـن صـورت مراحـل پـردازش 3 دستورالعمل را می‌توان به صورت Overlap انجام داد که در شکل زیر نشان داده شده است:



مشکل دیگر در پردازش لوله‌ای، در ارتباط با وابستگی Data مـی‌باشـد. اگـر دسـتورالعملی بـه اپرانـدی نیـاز داشـته باشـد کـه توسـط دستورالعمل قبل از خود تعیین شود، در این صورت تاخیری لازم خواهد بود که مجدداً این تـاخیر را مـی‌تـوان توسـط NOP ایجـاد نمود. پردازش لوله‌ای فوق در صورتی بهتر کار می‌کند که تاخیر Segment ها یکسان باشد.

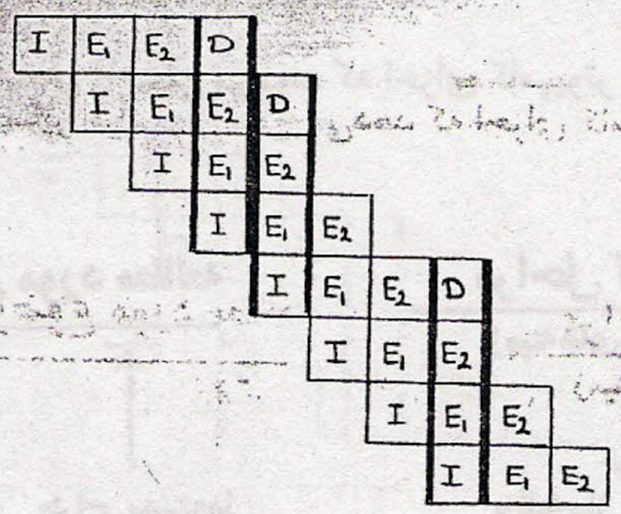
چون مرحله E در ارتباط با انجام عملیات در درون ALU می‌باشد و معمولاً زمان بیشتری را به خود اختصاص می‌دهد، می‌توان E را به دو مرحله زیر تقسیم نمود:

E_1 : Register File Read , E_2 = ALU Operation and Register Write

به دلیل ساده بودن مراحل دستورالعمل‌ها در معماری RISC می‌توان از « 4-Segment Pipeline » استفاده کرد که مجدداً برای تامین تاخیر متناسب با دستورات انشعاب وابستگی داده‌ها از NOP استفاده می‌شود.

به زمان‌بندی مثال زیر توجه کنید:

Load	A ← M
Load	B ← M
Nop	
Add	C ← A + B
Store	M ← C
Branch	x
Nop	
Nop	



« Optimizing of pipeline »

به دلیل طبیعت ساده بودن دستورالعمل‌های RISC می‌توان از پردازش لوله‌ای به راحتی استفاده نمود.

به طوری‌که قبلاً ذکر شد، تاخیر متناظر با وابستگی داده‌ها و دستورات انشعاب مانع از بهره‌برداری صددرصد از توان عملیاتی پردازنده لوله‌ای است. برای جبران این تاخیرها می‌توان از تکنیک «دوباره سازماندهی کدها» استفاده نمود.

ابتدا دستور انشعاب را در نظر می‌گیریم. به طوری‌که در درس‌های قبل اشاره شد، Delayed Branch روش مناسبی است برای بهبود کارآیی پردازنده لوله‌ای که در آن عمل انشعاب پس از پردازش دستور متوالی انجام می‌گیرد.

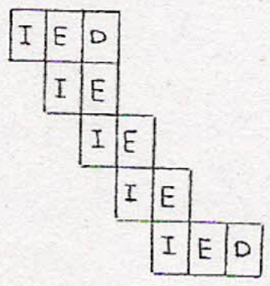
این رویه ناآشنا را می‌توان توسط جدول زیر روشن ساخت:

Address	Normal branch		Delayed branch		Optimized delay branch	
100	Load	x, A	Load	x, A	Load	x,A
101	ADD	1 , A	ADD	1 , A	JUMP	105
102	JUMP	105	JUMP	106	ADD	1 , A
103	ADD	A , B	NOP		ADD	A , B
104	SUB	C , B	ADD	A , B	SUB	C , B
105	STORE	A , Z	SUB	C , B	STORE	A , Z
106			STORE	A , Z		

در ستون اول از چپ یک قطعه برنامه معمولی را می‌بینیم که بعد از دستور واقع در آدرس 102 باید دستور واقع در آدرس 105 اجرا شود. به منظور تنظیم پردازنده لوله‌ای در ستون دوم، یک دستورالعمل NOP الحاق شده است، ولی برای افزایش کارآیی پردازنده در ستون سوم، جای دستورالعمل‌های واقع در آدرس 101 و 102 عوض شده است که شکل زیر اثر این تعویق را به خوبی نشان می‌دهد.

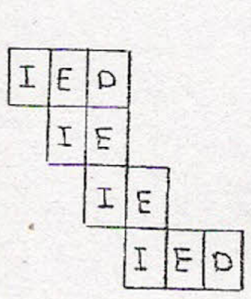
روش Inserted NOP

100	Load	x, A
101	ADD	1 , A
102	Jump	106
103	NOP	
106	STORE	A, Z



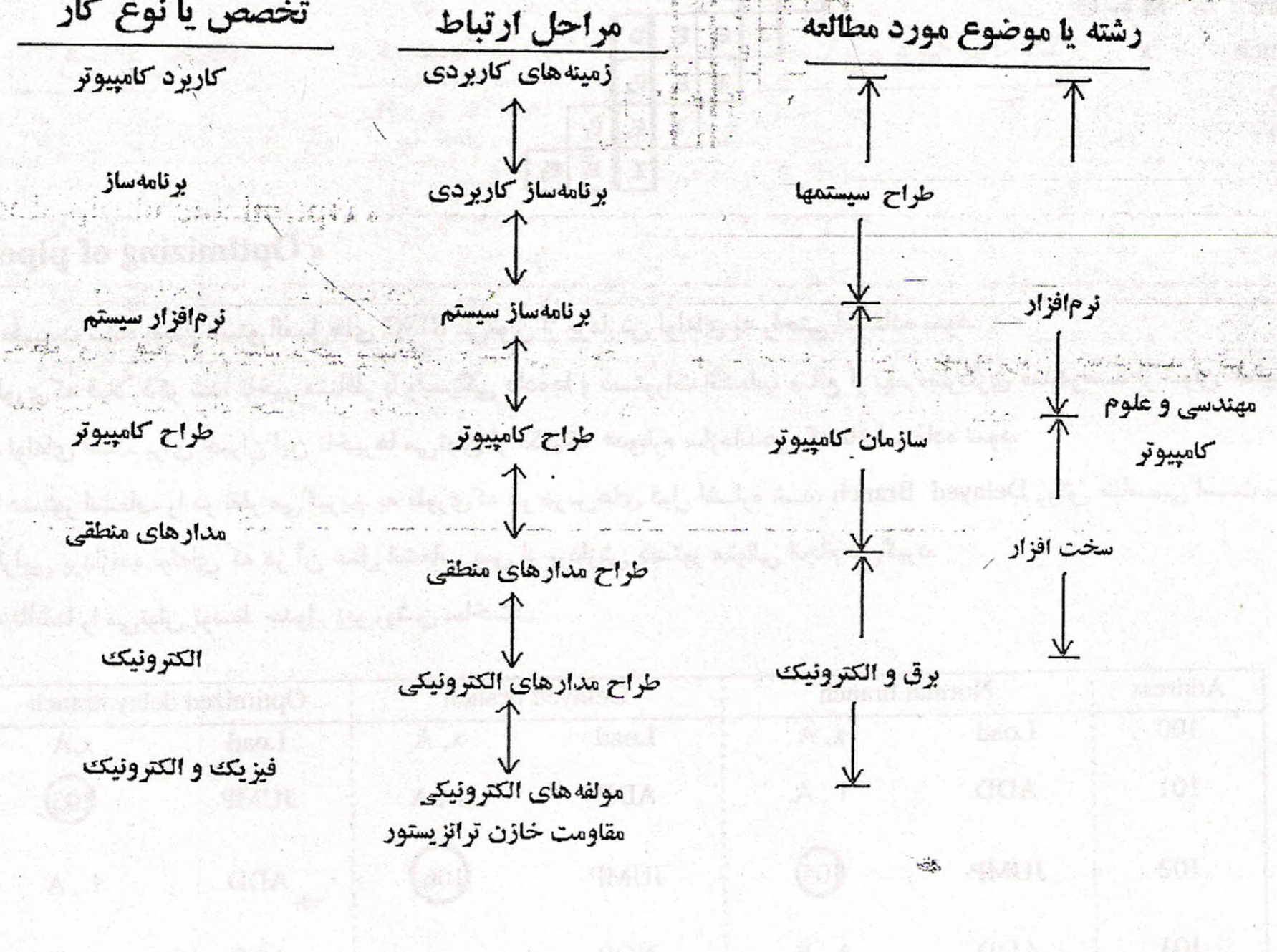
روش Reversed Inst

100	Load	x, A
101	Jump	105
102	ADD	1, A
105	STORE	A, Z



سازمان کامپیوتر بخشی است از مهندسی و علوم کامپیوتر که روش‌های ساختن سیستم‌های کامپیوتری را به‌وسیله عناصر تشکیل‌دهنده آن بررسی می‌کند.

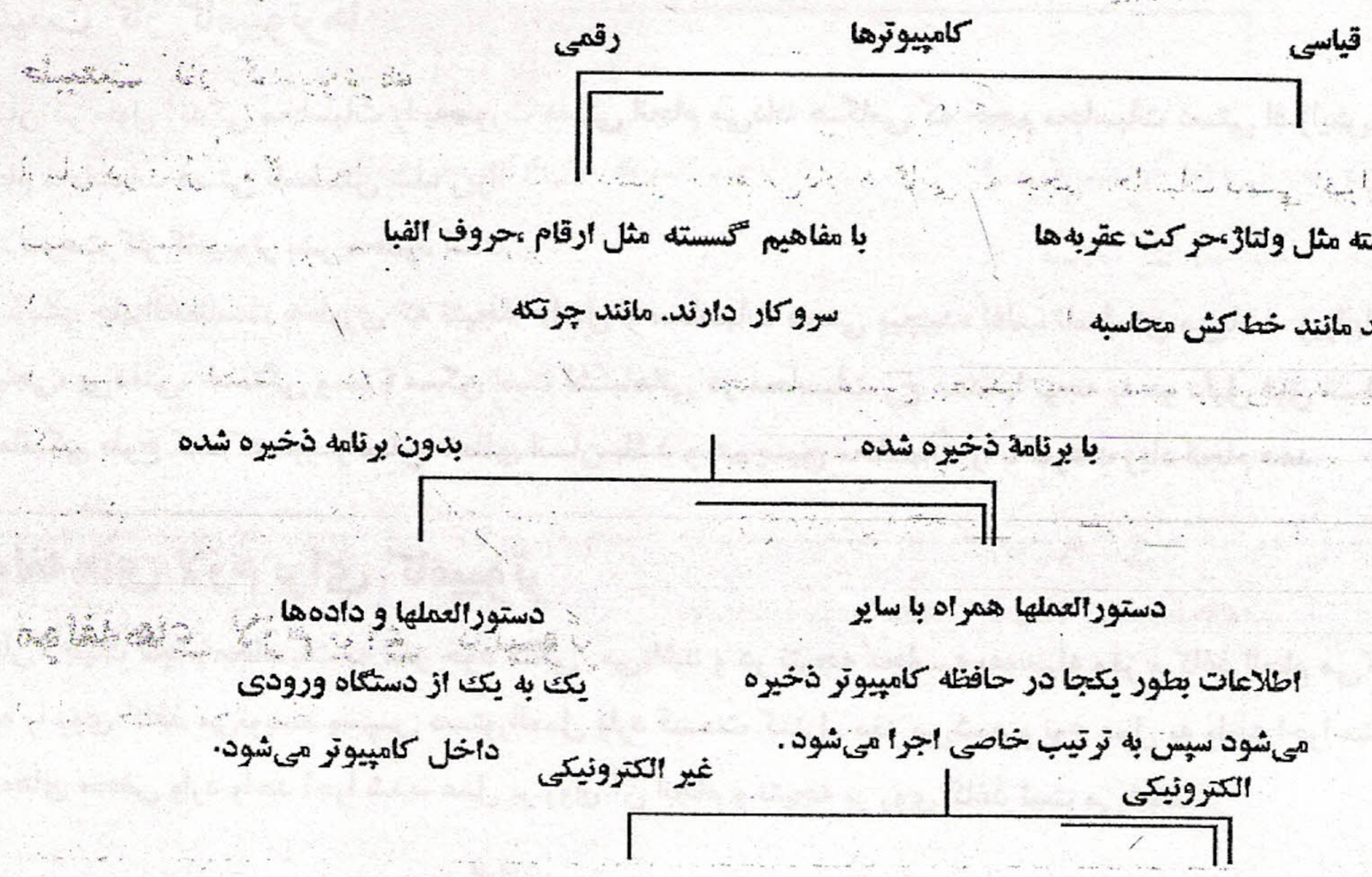
شکل زیر رابطه طراح کامپیوتر با سایر افرادی را نشان می‌دهد که اجزای کامپیوتر را به زمینه‌های کاربردی آن ارتباط می‌دهند.



طبقه‌بندی کامپیوترها

آن‌چه که با شنیدن کامپیوتر در نظر اکثر ما مجسم می‌شود، نوع به‌خصوصی از کامپیوتر است که به صورت کامل " کامپیوتر رقمی با برنامه ذخیره شده همه منظوره الکترونیکی" نامیده می‌شود.

شکل زیر یک روش تقسیم‌بندی برای کامپیوترها را نشان می‌دهد.



سازمان کامپیوتر

طبیعت کار کامپیوترها

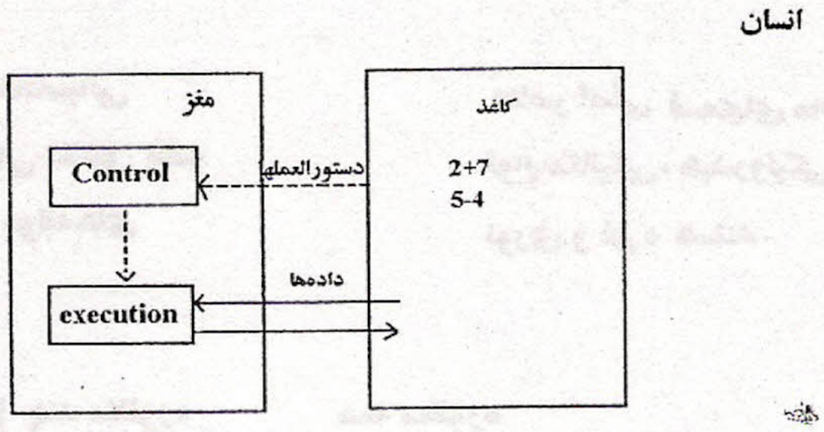
انسان در طول زندگی محاسبات را به‌صورت دستی انجام می‌داد. هنگامی که حجم محاسبات دستی افزایش یافت و نتیجه حاصل از انجام محاسبات دستی نامطمئن شد، زیرا:

۱ ـ سرعت کار کامپیوتر بشر محدود است.

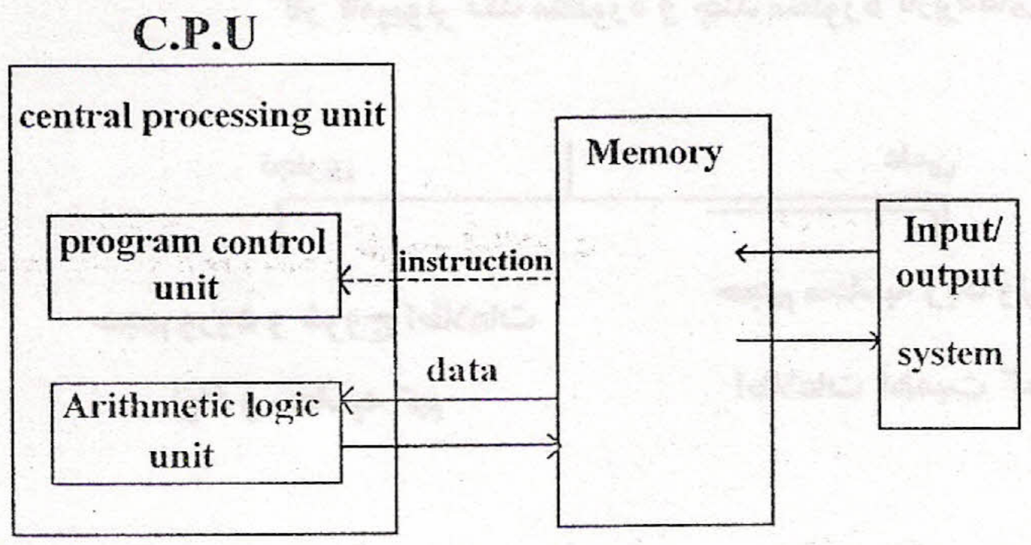
۲ ـ بشر جایزالخطاست به‌طوری که نتیجهٔ حاصل از محاسبات دستی پیچیده اغلب نامطمئن می‌باشد. زیرا با توجه به منابع خطای بشر (گیجی، بی‌دقتی، خستگی و.....) ممکن است اشتباهاتی در محاسبات رخ دهد، با توجه به دو دلیل فوق انسان به فکر افتاد که ماشین اتوماتیکی طرح کند که دور از منابع خطای انسان باشد و هم‌چنین محاسبات را با سرعت زیاد انجام دهد.

مولفه‌های لازم برای کامپیوتر

انسان جهت انجام محاسبات به مغز خود متکی می‌باشد و در نتیجه محاسبه به‌وسیله مغز و کاغذ انجام می‌گیرد، زیرا الگوریتم‌ها و داده را روی کاغذ می‌نویسد وسپس دستورالعمل وارد قسمت کنترل مغز می‌شود و نوع عمل به واجد اجرا مشخص می‌گردد، آن‌گاه داده‌های محض وارد واحد اجرا شده، عمل بر روی آن انجام و نتیجه بر روی کاغذ ثبت می‌شود.



مولفه‌های اصلی کامپیوتر نیز مشابه به مولفه‌های لازم برای محاسبه دستی توسط انسان می‌باشد.



تفاوت اصلی بین ماشین و انسان در نحوه نمایش اطلاعات می‌باشد، بنابراین برای تبدیل اطلاعات از زبان ماشین به زبان دنیای خارج آن و بالعکس وسیله‌ای لازم است که I/OSystem نامیده می‌شود و در نتیجه مولفه‌های لازم برای کامپیوتر عبارت‌اند از:

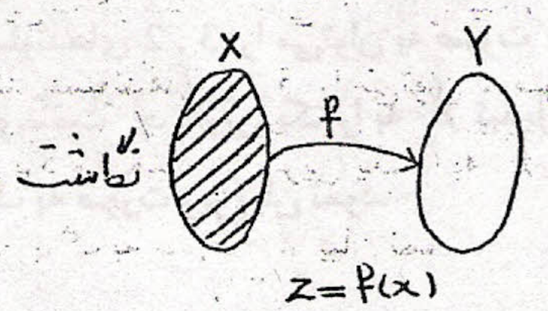
۱ ـ Processor : که قادر به تفسیر و اجرای دستورالعمل‌ها می‌باشد.

۲ـ Memory : برای ذخیره نمودن برنامه‌ها و داده‌ها.

۳ ـ I/O System : وسیله‌ای برای تبادل اطلاعات بین حافظه و پردازنده و بین کامپیوتر و دنیای خارج آن.

یک مدل انتزاعی برای کامپیوتر:

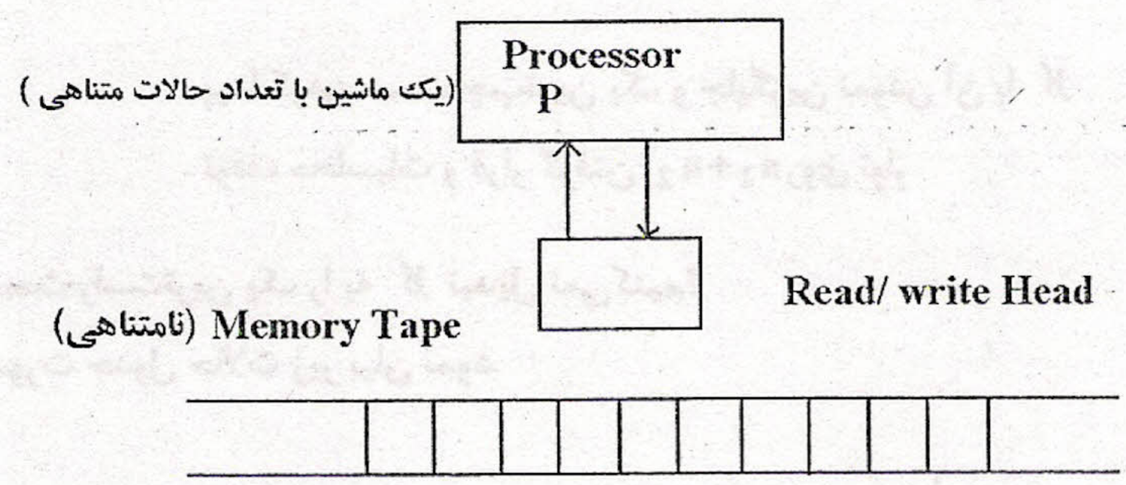
وظیفه کامپیوتر محاسبه است و محاسبه را می‌توان به عنوان ارزیابی تابعی مانند $f(x)=z$ تلقی نمود که در آن x ورودی و z خروجی و f پردازشی است که روی x انجام و نتیجه z را تولید می‌کند.



برای این‌که بتوان تابع f را به‌وسیله کامپیوتر ارزیابی نمود باید ارزیابی طی مراحل متناهی $f_1, f_2, ..., f_n$ انجام گیرد.

$$y_1=f_1(x) \quad , \quad y_2=f_2(y_1),....,f_n(y_{n-1})=z$$

بنابراین چنین ماشینی باید محاسبه را طی مراحل متناهی انجام دهد . یک مدل انتزاعی به‌وسیلهٔ ریاضیدان انگلیسی به نام آلن تورینگ (Alen Turing) معرفی شده است که مولفه‌های لازم برای کامپیوتر را در بر دارد و می‌تواند محاسبه را طی مراحل متناهی انجام دهد.



R/W Head قادر است علامت خانه‌ای را که در مقابل آن قرار گرفته است تشخیص دهد، علامت جدید را جایگزین علامت قبلی کند، یک خانه به‌طرف چپ یا راست تغییر مکان یابد. ماشین تورینگ شامل مجموعه‌ای از دستورالعمل‌ها می‌باشد که فرمت آن به صورت زیر است:

$$S_h \ t_i \ O_j \ S_k$$

به مفهوم این‌که اگر پردازندهٔ P در حالت فعلی S_h باشد و نوک خواندن و نوشتن روی خانه‌ای با علامت t_i قرار گرفته باشد، باید Operation یا عمل O_j انجام گیرد و پردازنده به حالت جدید S_k تغییر یابد.

O_j ممکن است یکی از چهار عمل زیر باشد :

۱ - $O_j=t_j$ به معنی این‌که علامت t_j جایگزین علامت قدیم t_i شود.

۲- $O_j=R$ یعنی R\W Head یک خانه به راست تغییر مکان یابد.

۳- $O_j=L$ یعنی R\W Head یک خانه به چپ تغییر مکان یابد.

۴- $O_j=H$ پردازنده محاسبه را متوقف می‌کند.

ارزیابی تابع $z=f(x)$ به‌وسیلهٔ ماشین تورینگ به ترتیب زیر انجام می‌گیرد:

ابتدا ورودی x در قسمت خالی نوار قرار می‌گیرد، آن‌گاه ماشین تورینگ شروع به‌کار می‌کند و طی مراحل متناهی $f_1, f_2, ...$ تا f_n محاسبه را انجام در پایان عمل n ام ماشین متوقف می‌شود و نتیجه حاصل از محاسبه یعنی z روی نوار ذخیره می‌گردد.

مثال: ماشین تورینگی طرح کنید که حاصل جمع دو عدد طبیعی را تولید کند.

با فرض این‌که هر عدد طبیعی n را به‌وسیلهٔ n تا 1 نشان دهیم و از ✗ برای جداسازی استفاده کنیم.

آن‌گاه مثلاً عملوندهای 2 ، 3 را می‌توان به صورت ...✗11✗111✗...✗✗ نشان داد. برای پیدا کردن حاصل جمع باید ✗ بین دو عدد را از بین برد و سمت چپ‌ترین یک را به ✗ تبدیل نمود. این الگوریتم جمع را می‌توان به‌وسیله مجموعه‌ای از دستورالعمل‌های ماشین تورینگ به صورت زیر بیان نمود.

instruction				
S_0	✗	R	S_0	حرکت R/W Head به طرف راست عدد n_1
S_0	1	R	S_1	
S_1	1	R	S_1	
S_1	✗	1	S_2	جایگزین نمودن 1 با ✗ بین دو عدد n_1 و n_2
S_2	1	L	S_3	حرکت R/W Head به طرف چپ n_1
S_3	1	L	S_3	
S_3	✗	R	S_4	
S_4	1	✗	S_5	پیدا کردن سمت چپ‌ترین یک و جایگزین نمودن آن با ✗
S_5	✗	H	S_0	توقف محاسبات و قرار گرفتن n_1+n_2 روی نوار

تمرین: در الگوریتم جمع چرا سمت راست‌ترین یک را به ✗ تبدیل نمی‌کنیم؟

قطعه برنامه قبلی را می‌توان به‌صورت جدول حالات زیر بیان نمود.

حالت \ علامت	b	1
S_0	S_0 /R	S_1 /R
S_1	S_1 /I	S_1 /R
S_2	don't care	S_3 /R
S_3	S_4 /L	S_3 /L
S_4	—	S_5 /b
S_5	S_o /H	—

Desing Methodology (اسلوب‌شناسی طراحی)

کامپیوتر رقمی یک سیستم است. سیستم مجموعه‌ای است از اجزا که با اتصال منطقی به یکدیگر یک شیئی جدید با وظیفه و هدف کاملاً مشخص و معین را به‌وجود می‌آورد. ولی مورد توجه ما سیستم پردازش اطلاعات می‌باشد که وظیفه آن تبدیل ورودی A به خروجی B می‌باشد و این تبدیل را می‌توان به‌وسیله تابع f نشان داد.

ورودی محاسبات خروجی

$$f : A \rightarrow B$$

$$\forall a \in \underbrace{A}_{\text{برنامه‌ها و داده‌ها}} \qquad \exists b \in \underbrace{B}_{\text{نتایج حاصل از اعمال برنامه روی داده}} \qquad \ni b = f(a)$$

$$b \leftarrow f(a)$$

مدل‌سازی سیستم (System Modeling)

بهترین روش برای مدل‌سازی کامپیوتر استفاده از گراف می‌باشد.

گراف $G = < V \text{ و } E >$ شامل دو مجموعه است که در آن V مجموعه غیر تهی از گره‌ها و E مجموعه‌ای از یال‌ها می‌باشد.

↓ مجموعه غیرتهی گره‌ها node ↓ مجموعه لبه‌ها edge

به‌طوری که $E \subseteq V \times V$ به معنی این‌که اگر $< v_1 , v_2 > \in E$ باشد، آن‌گاه گره v_1 به‌وسیله یک خط جهت‌دار به گره v_2 متصل می‌شود.

معمولاً گره‌ها را با دایره کوچک و یال‌ها را با خط جهت‌دار نشان می‌هند، گراف حاصل را نمودار (Diagram) گویند.

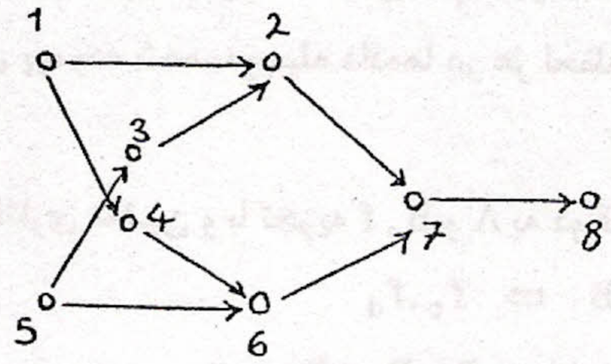
سیستم رقمی معمولاً شامل دو جز است:

۱ ـ مجموعه‌ای از اجزا پردازش C

۲ـ مجموعه‌ای از سیگنال‌های S که وظیفه انتقال اطلاعات بین اجزا را به عهده دارد.

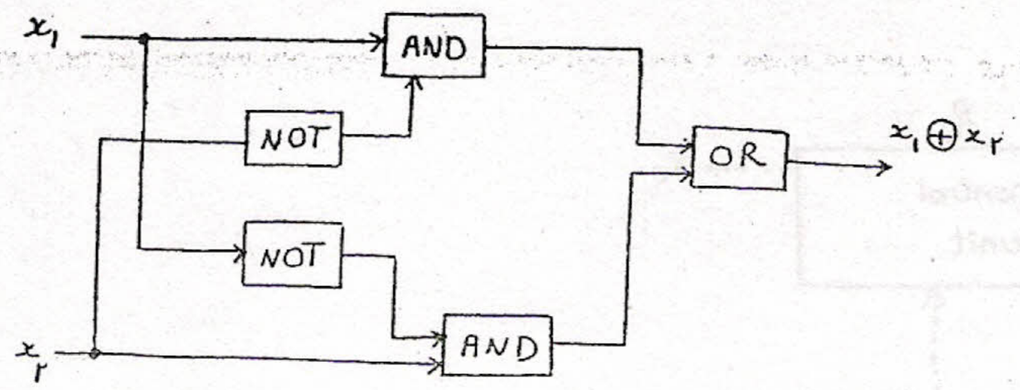
یک روش برای توصیف ساختاری استفاده از گراف است.

توصیف ساختاری: تنها مولفه‌ها را نامبرده و نحوه اتصال منطقی آن‌ها را مشخص می‌کند. ولی اطلاعی در مورد رفتار سیستم به ما نمی‌دهد. مثلاً گراف زیر ساختار یک سیستم را نشان می‌دهد، که شامل 8 گره و 9 یال می‌باشد.



شکل (۱) توصیف ساختاری

حال اگر به جای گرههای انتزاعی، مولفه‌های واقعی مانند Gateها و به‌جای یال‌های انتزاعی، سیگنال‌ها را قرار دهیم، نمودار بلوکی (Block Diagram) زیر به‌دست می‌آید که رفتار یک X – OR Gate را از خود نشان می‌دهد.



شکل (۲) نمودار بلوکی X-OR) Gate) که هم توصیف رفتاری و هم توصیف ساختاری را در بر دارد.

بنابراین توصیف ساختاری تنها اجزا تشکیل‌دهنده سیستم را نام برده و اتصال منطقی بین آن‌ها را معین می‌کند. ولی توصیف رفتاری به شخص امکان می‌دهد که برای هر ورودی مانند a بتوان خروجی $f(a)=b$ را معین کند. (f را به‌عنوان رفتار سیستم در نظر می‌گیریم.)

برای نمایش رفتار مدارهای ترکیبی می‌توان از جدول ارزش (Truth Table) استفاده کرد.

ورودی		b=f(a)
x_1	x_2	$\overbrace{x_1 \oplus x_2}$
0	0	0
0	1	1
1	0	1
1	1	0

شکل (۳) توصیف رفتاری

حال با در دست داشتن رفتار و مولفه‌ها بایدمولفه‌ها را چنان به‌طور منطقی به هم متصل نمود که ساختار سیستم را با حداقل هزینه ایجاد کند که یک روش برای این منظور استفاده از نمودار بلوکی (شکل 2) است که هم توصیف ساختاری و هم توصیف رفتاری را دربر دارد.

Data and Control

هر سیستم رقمی به دو جز تقسیم می‌شود:

۱ ـ واحد پردازش داده‌ها (D . P . U) و (Data Processing Unit)

۲ ـ واحد کنترل (C . U .) و (Control Unit)

مجموعه اجزا و مسیرهای پیموده شده به‌وسیله داده را واحد پردازش داده‌ها گویند و سایر قسمت‌ها واحد کنترل نامیده می‌شود. عملیات انجام داده شده در واحد پردازش و مسیرهای پیموده شده به‌وسیله داده‌ها در هر لحظه از زمان به‌وسیله واحد کنترل مشخص می‌گردد.

تمایز Data و Control را می‌توان با استفاده از تابع رفتاری ماشین و با تجزیه f , B و A به دو قسمت کنترل و داده نشان داد.

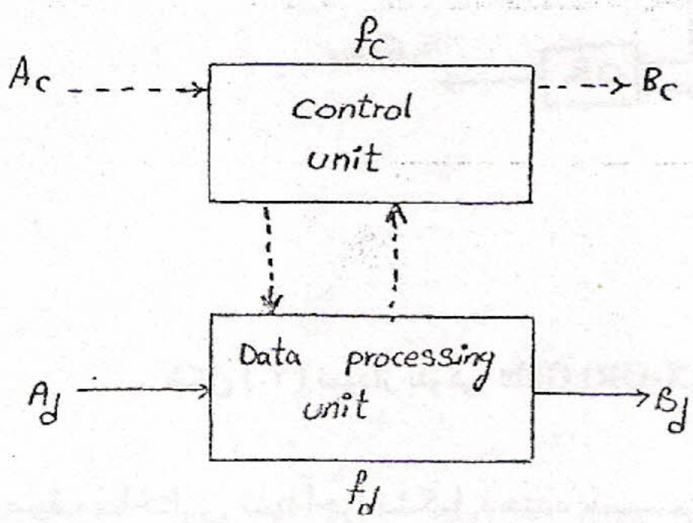
$$f:A \to B \quad \Rightarrow \quad f_c , f_d$$

$$A \subseteq A_c \times A_d \ , \ B \subseteq B_c \times B_d$$

$$f_c : A_c \to B_c$$

$$f_d : A_d \times A_c \to B_d$$

در صورتی‌که دو تابع کنترل و داده را به‌صورت جداگانه پیاده‌سازی (Implement) کنیم، نمودار بلوکی زیر به‌دست می‌آید.



و احد کنترل را می‌توان به دو طریق طراحی نمود:

اگر f_c ثابت و دایمی باشد، واحد کنترل را سیستم‌بندی شده (Hardwired) گویند.

اگر f_c به کمک ذخیره نمودن اطلاعات کنترل در حافظه انجام گیرد، واحد کنترل را ریزبرنامه‌سازی شده (Microprogrammed) گویند.

روش‌های توصیفی : اجزای سیستم در نمودار بلوکی به صورت مستطیل نشان داده می‌شود که در آن ساختار داخلی مشخص نیست، ولی رفتار آن به‌طور ناقص یا کامل مشخص است. اجزا یا مولفه‌های سیستم در واقع ماشین با تعداد حالت‌های متناهی است که در آن ورودی A به دو جز تقسیم می‌شود.

۱ ـ سیگنال‌های ورودی اصلی X که به‌وسیله منابع خارجی فراهم می‌شود.

۲ـ حالات درونی Y که اطلاعات ذخیره شدهٔ درونی را تشکیل می‌دهد.

هم‌چنین خروجی B به دو جز تقسیم می‌شود :

۱ـ سیگنال‌های خروجی اصلی Z

۲ ـ حالات جدید درونی Y

$$f: X \times Y \rightarrow Y \times Z$$

$$\forall \langle x,y \rangle \in X \times Y \quad \exists \langle y,z \rangle \in Y \times Z \qquad \ni \langle y,z \rangle = f \langle x,y \rangle$$

و این تابع را می‌توان به‌وسیلهٔ State Table نیز نشان داد که در آن سطرها نشان‌دهنده حالات و ستون‌ها نشان‌دهنده ورودی‌ها می‌باشد.

روش دیگر برای نشان دادن رفتار سیستم استفاده از State Transition می‌باشد. یا معادلاتی به صورت زیر می‌باشد.

$$f(x_0, y_0) = (z_1, y_1)$$
$$f(z_1, y_1) = (z_2, y_2)$$
$$\vdots$$
$$f(z_{n-1}, y_{n-1}) = (z_n, y_n)$$

هر یک از معادلات فوق یک حکم (Statement) نامیده می‌شود و احکام بر دو نوعند :

۱ ـ اخباری : بیانی از آن‌چه که ماشین انجام می‌دهد. (نرم‌افزار)

۲ـ دستوری : بیانی از آن‌چه ماشین باید انجام دهد. (سخت افزار)

مجموعه‌ای از احکام دستوری را که شبیه یک برنامه است. (Register Tranfere Language (RTL گویند که یک زبان برای طراحی سخت‌افزار کامپیوتر می‌باشد.

سطوح طراحی : طراحی کامپیوترهای رقمی در سه سطح زیر انجام می‌شود :

Desing level	Components	Information Unit	Time Unit
Processor (LSI , VLSI)	CPU , I/O Devices I/O Processor و Memories	Block of Data برنامه‌ها و داده‌ها	$10^{-3}-10^{3}$ S
Register (MSI)	Register , Combinational ckt Simple Sequential ckt	Word	$10^{-9}-10^{-6}$ S
Gate (SSI)	Logical Gates , FFs	Bit	$10^{-10}-10^{-8}$ S

طراحی سلسله مراتبی « Hierarchical Design »

در Top - down design می‌توان طراحی سیستم پیچیده مانند کامپیوتر را به صورت زیر انجام داد:

۱ ـ ساختار سیستم را در سطح پردازنده‌ها (Process level) مشخص می‌کنیم.

۲ ـ ساختار هر یک از مولفه‌های مرحله اول را در سطح ثبات‌ها (Register level) مشخص می‌کنیم.

۳ ـ ساختار هر یک از مولفه‌های مرحله دوم را در سطح گیت‌ها (Gate level) مشخص می‌کنیم.

Register Level

واحد اطلاعاتی مجموعه‌ای است از bit ها که Word نامیده می‌شود و مسیری که اطلاعات طی می‌کند BUS (گذرگاه) نامیده می‌شود.

مولفه‌های لازم برای طراحی در Register Level عبارت‌اند از:

a. Combinational Components

1. Word Gates
2. Multiplexer تسهیم‌کننده
3. Decoder and Encoder رمزگشا و رمزگذار
4 . General - purpose Logic Array آرایه منطقی همه منظوره
5 . Arithmetic Element (Adder, comparator ...)

b . Sequential Components

1.Register (به انضمام ثبات‌های تغییر مکان)
2. Counter

Word Gate

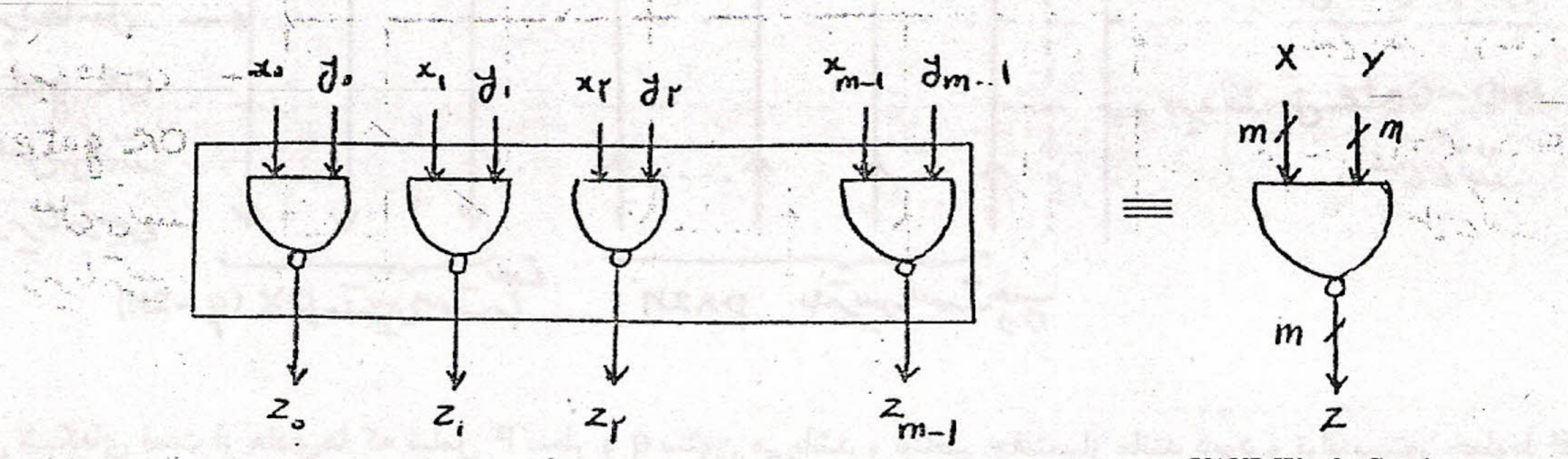
فرض کنید $X=(x_0, x_1, ..., x_{m-1})$ و $Y=(y_0, y_1, ..., y_{m-1})$ دو کلمه m بیتی و

$f \in \{ AND, OR, NOT, NAND, NOR, XOR \}$ باشد، آن‌گاه $Z=f(X,Y)$ اگر $z_i = f(x_i, y_i)$ برای $i = 0, 1, ..., (m-1)$

و این تعمیم جبر بول دو مقداری $B=\{0,1\}$ از تابع $f: B^n \to B$ به تابع $f: (B^m)^n \to B^m$ می‌باشد.

مثلاً $Z=\overline{XY}$ نشان‌دهندهٔ یک NAND Date با ورودی و خروجی m - bit خواهد بود.

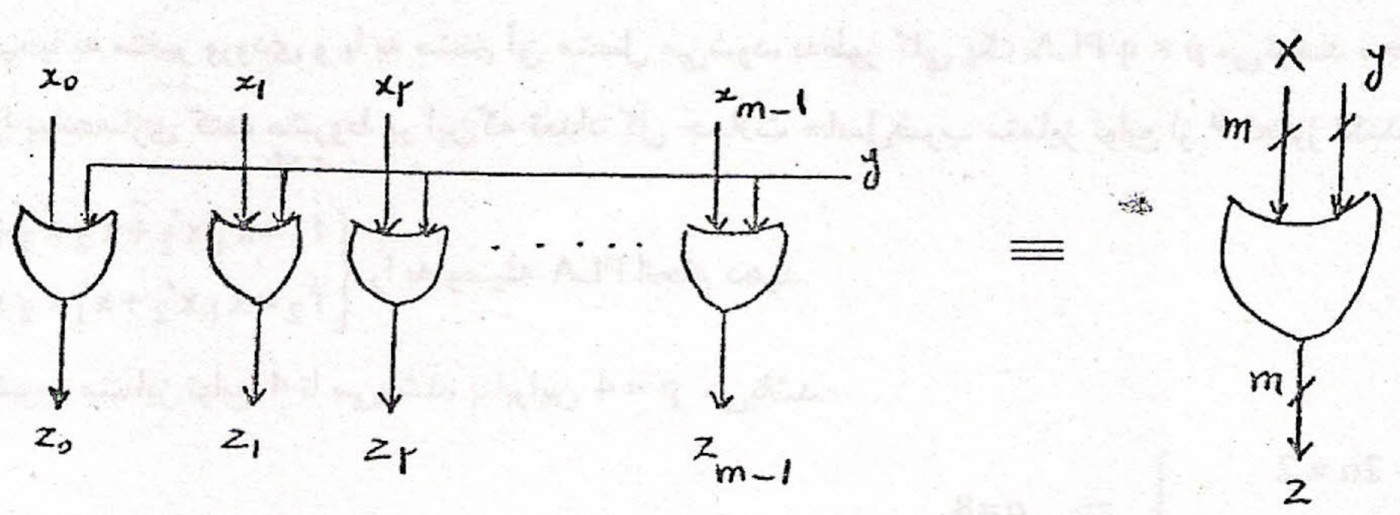
$$(z_0, z_1, ..., z_{m-1}) = (\overline{x_0 y_0}, \overline{x_1, y_1}, ..., \overline{x_{m-1} y_{m-1}})$$



(NAND Word - Gate)

Y ممکن است تک بیتی باشد و در این‌صورت به‌عنوان مثال داریم:

$$X + Y = (x_0 + y, x_1 + y, ..., x_{m-1} + y)$$



(OR Word - Gate)

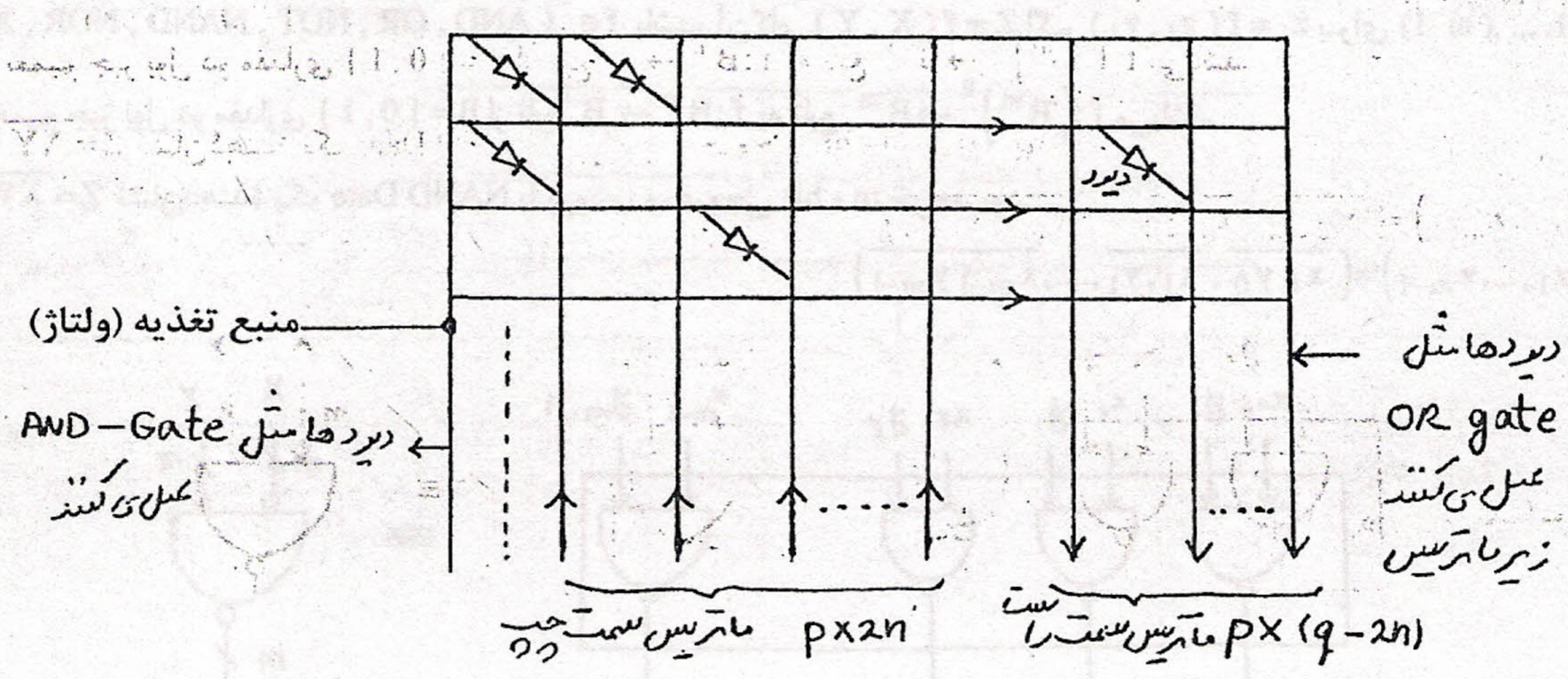
بقیه مولفه‌ها و کاربرد آن‌ها را در درس مدار منطقی مطالعه نموده‌ایم.

Logic Array Elements

در طراحی واحد کنترل و برخی دیگر از کاربردها مجموعهٔ توابعی که باید انجام گیرند تصادفی هستند که دسته‌بندی آن‌ها آسان نخواهد بود و در چنین حالتی از آرایه‌های منطقی برای پیاده‌سازی توابع ترکیبی تصادفی استفاده می‌شود که این مولفه‌ها در دو شرط زیر صدق می‌کند.

۱ـ ساختمان داخلی آن‌ها یکنواخت هستند، زیرا با استفاده از تکنولوژی ساخت مدارهای مجتمع می‌توان آنها را به‌صورت انبوه تولید نمود. (Mass Production)

۲ـ ساختمان داخلی آن به هنگام ساخت چنان در نظر گرفته می‌شود که در آینده بتوان تغییراتی در آن به‌وجود آورد و توابع موردنظر را انجام داد. یکی از این مولفه‌ها معروف به آرایه منطقی برنامه‌پذیر یا Programmable Logic Array (PLA) می‌باشد که می‌تواند مجموعه توابع ترکیبی را به‌صورت مدار دو سطحی متناظر با مجموع حاصل‌ضرب‌ها تولید کند.



این ماتریس شبکه‌ای است از هادی‌ها که شامل P سطر و q ستون می‌باشد و عناصر جفت‌ساز مانند دیود و ترانزیستور خطوط افقی و عمودی را به یکدیگر متصل می‌کند.

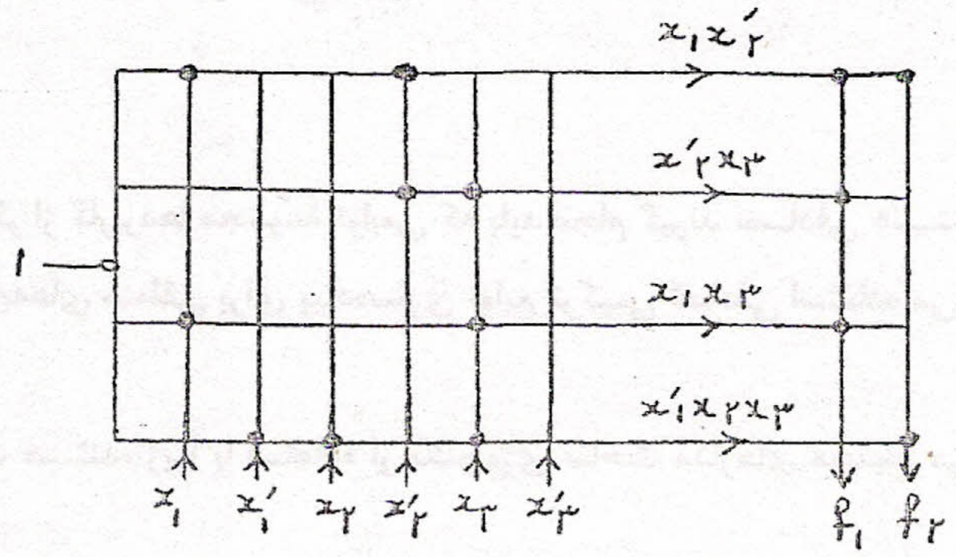
هر خط افقی یک جمله حاصل‌ضرب تولید می‌کند که به‌عنوان ورودی به زیر ماتریس ($P \times (q - 2n)$ مورد استفاده قرار می‌گیرد و هر خط عمودی در زیر ماتریس سمت راست یک تابع به صورت مجموع حاصل‌ضرب‌ها تولید می‌کند و هر یک از خطوط عمودی در زیر ماتریس سمت چپ یا به متغیر ورودی و یا به متمم آن متصل می‌شود، به‌طور کلی یک $p \times q$ PLA می‌تواند مجموعه‌ای از ($q - 2n$ تابع n متغیری را پیاده‌سازی کند، مشروط بر این‌که تعداد کل جملات حاصل‌ضرب متمایز توابع از P تجاوز نکند.

مثال: توابع $\begin{cases} f_1 = x_1 x'_2 + x'_2 x_3 + x_1 x_3 \\ f_2 = x_1 x'_2 + x'_1 x_2 x_3 \end{cases}$ را به وسیله PLA انجام دهید.

تعداد جملات حاصل‌ضرب متمایز توابع 4 تا می‌باشد، بنابراین $p = 4$ می‌باشد.

تعداد توابع $q - 2n = 2$
تعداد متغیرها $n = 3$
$\Rightarrow \quad q = 8$

بنابراین به یک 4×8 PLA نیاز داریم .



شمارنده‌ها Counters

شمارنده در اصل ثباتی است که به محض دریافت سیگنال شمارش (Count Signal) دنباله‌ای از حالات شمار درونی S_0 , S_1 ... تا S_{n-1} را طی می‌کند و پس از دریافت سیگنال شمارش n ام به حالت اولیه S_0 باز می‌گردد.

Modulo - n counter

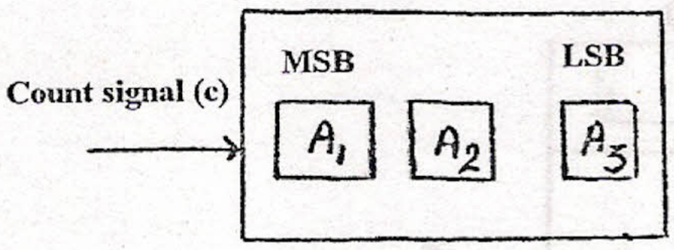
$S_{i+1} = S_i + 1 \pmod n$

با توجه به کدهای به‌کار رفته برای نمایش اعداد و Timing Mode (که منظور ممکن است Synchronous , Asynch) شمارنده‌ها در انواع مختلف موجودند که چند نمونه از متداول‌ترین آن‌ها را مطالعه می‌کنیم.

۱ ـ Binary Counter شمارنده دودویی

ساده‌ترین شمارنده‌ای است که عمل شمارش را در سیستم اعداد باینری انجام می‌دهد، مثلاً Modulo - n binary Counter دنباله‌ای از اعداد 0, 1 ... , n -1 را می‌شمارد و پس از دریافت سیگنال شمارش n ام، به حالت اولیه صفر باز می‌گردد.

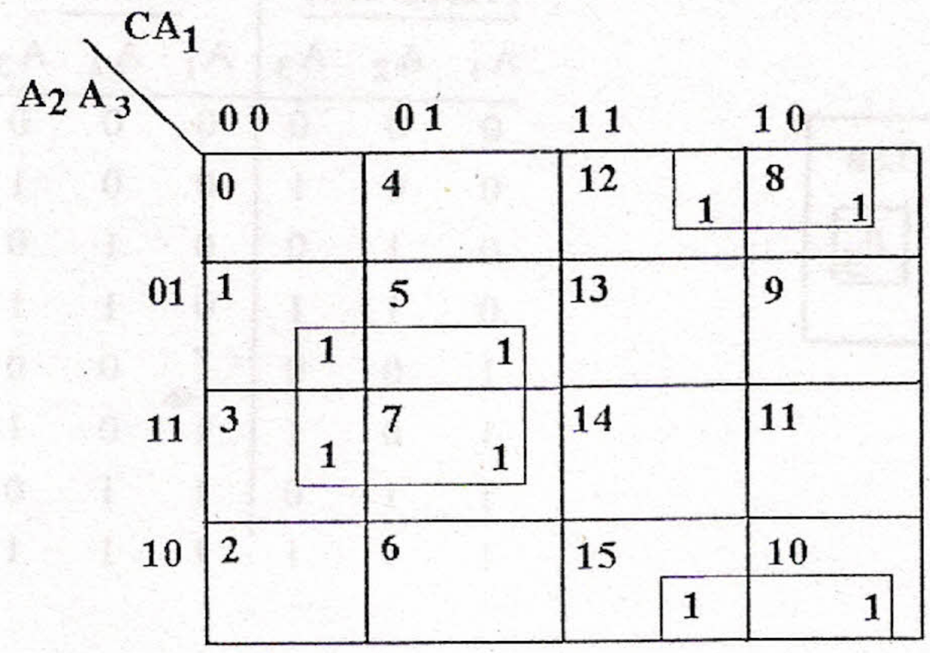
مثال: با استفاده از D - FF یک Modulo - 8 Binary طرح کنید.



Present state			Next state C=0			Next state C=1		
A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	1	0	0
1	0	0	1	0	0	1	0	1
1	0	1	1	0	1	1	1	0
1	1	0	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0

با توجه به جدول حالات متوجه می‌شویم که شمارنده در صورتی عمل شمارش را انجام می‌دهد که $C = 1$ باشد و در صورتی که $C = 0$ باشد، حالت درونی شمارنده بدون تغییر باقی خواهد ماند. از این رو برای ساده نمودن عمل طراحی می‌توان C را از جدول حذف نمود، ولی به‌خاطر داشت که آن را با ورودی هر یک از FF ها AND نمود که این موضوع برای T - FF , JK - FF که از نوع جهشی هستند، صحیح است ولی برای D - FF و SR - FF نمی‌توان سیگنال شمارش C را از جدول حذف نمود. بنابراین جدول شامل 16 سطر خواهد بود.

input	P.S.			N.S.			input s to FFS			min term
C	A_1	A_2	A_3	A_1	A_2	A_3	D_1	D_2	D_3	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1	1
0	0	1	0	0	1	0	0	1	0	2
0	0	1	1	0	1	1	0	1	1	3
0	1	0	0	1	0	0	1	0	0	4
0	1	0	1	1	0	1	1	0	0	5
0	1	1	0	1	1	0	1	1	1	6
0	1	1	1	1	1	1	1	1	0	7
1	0	0	0	0	0	1	0	0	1	8
1	0	0	1	0	1	0	0	1	0	9
1	0	1	0	0	1	1	0	1	1	10
1	0	1	1	1	0	0	1	0	0	11
1	1	0	0	1	0	1	1	0	1	12
1	1	0	1	1	1	0	1	1	0	13
1	1	1	0	1	1	1	1	1	1	14
1	1	1	1	0	0	0	0	0	0	15

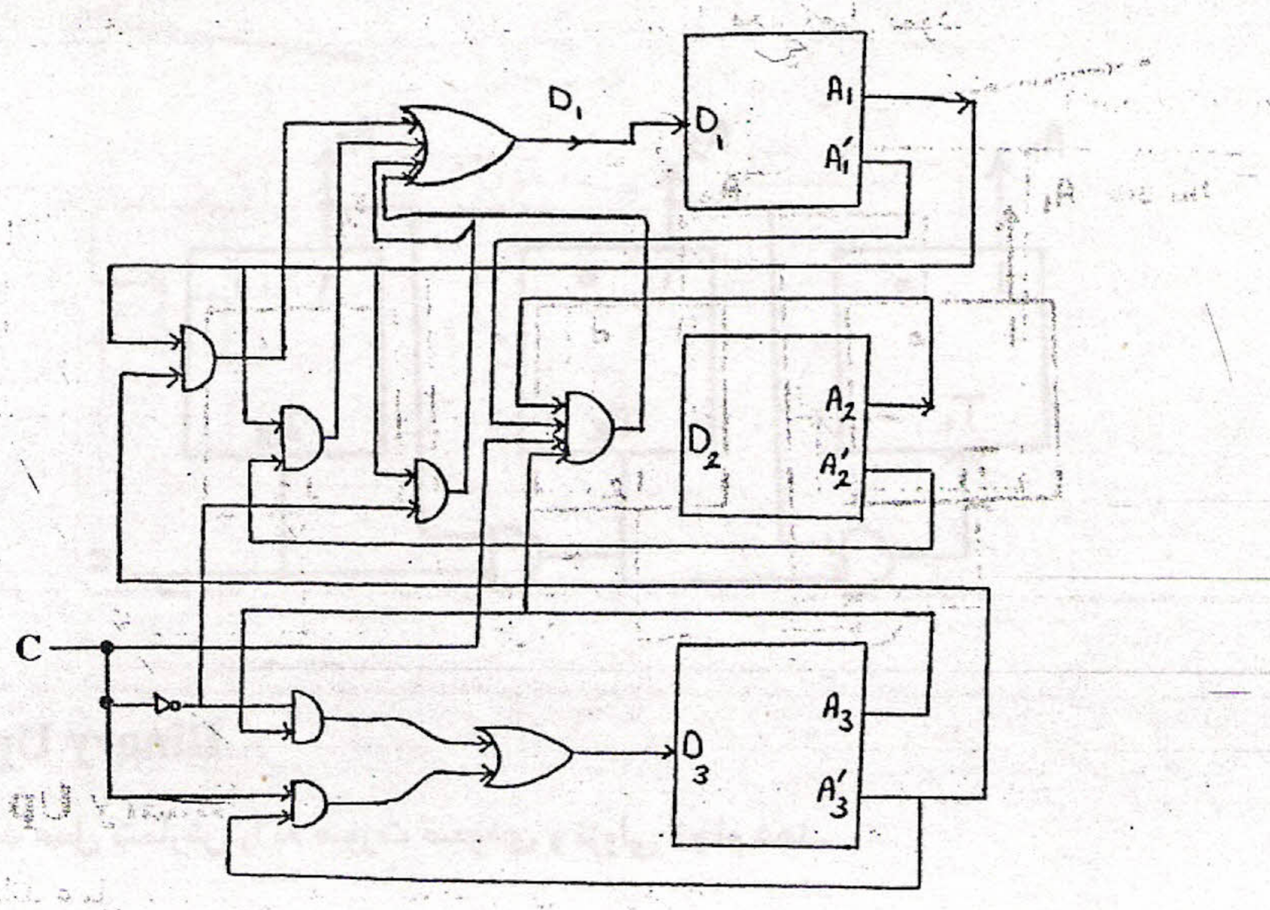


برای D_3

$D_3 = C'A_3 + CA'_3$

$D_1 = C'A_1 + A_1A_2 + A_1A'_3 + CA'_1$

$D_2 = C'A_2 + A_2A'_3 + CA'_2A_3$



Decade Counter

شمارنده‌ای است که عمل شمارش را در سیستم اعداد دهدهی انجام می‌دهد، مثلاً شمارنده ارقام دهدهی (Decade of Decimal Counter) از صفر تا 9 می‌شمارد و پس از دریافت سیگنال دهم به صفر باز می‌گردد و یک رقم نقلی به مرحله بالاتر تولید می‌کند. عمل شمارش ممکن است در NBCD انجام گیرد.

تمرین: با استفاده از T – FF یک Decade Counter طرح کنید.

مثال: با استفاده از T - FF یک شمارنده صعودی با پیمانه 8 طرح کنید.

P.S			N.S.		
A_1	A_2	A_3	A_1	A_2	A_3
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

شرایط شمارش صعودی

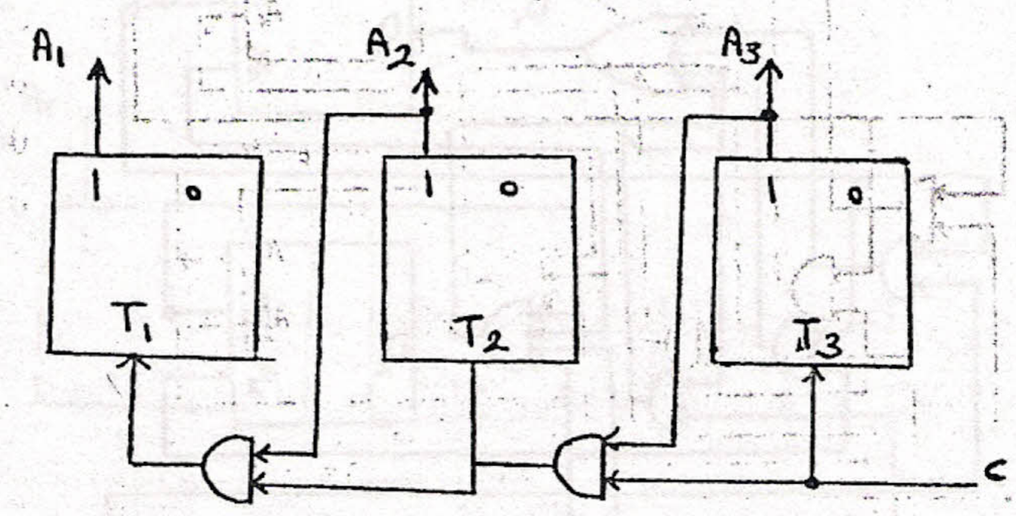
۱ ـ فلیپ فلاپ A_3، با دریافت هر سیگنال شمارش تغییر حالت می‌دهد.

۲ـ فلیپ فلاپ A_2 وقتی تغییر حالت می‌دهد که FF مرتبه پایین‌تر، یعنی A_3 در حالت 1 باشد.

۳ـ فلیپ فلاپ A_1 وقتی تغییر حالت می‌دهد که هر دو FF مرتبه پایین‌تر، یعنی A_3 , A_2 هر دو درحالت 1 باشد.

در این شیوه از طراحی می‌توان سیگنال شمارش را از جدول حذف نمود و در نتیجه بر خلاف مثال قبلی جدول حالات از 16 سطر به 8 سطر تقلیل می‌یابد.

و به راحتی می‌توان پیمانه آن را افزایش داد و یا چند IC از این شمارنده را به‌هم وصل نمود.



Binary Up - Down Counter

شمارنده‌ای است که می‌تواند عمل شمارش را به صورت صعودی و نزولی انجام دهد.

مثال: با استفاده از T - FF یک شمارنده دودویی صعودی و نزولی با پیمانه 8 طرح کنید.

با فرض این‌که شمارنده در هر Clock Pulse یا سیگنال شمارش صعودی (Up - Count) و یا سیگنال شمارش نزولی (Down - Count) دریافت کند، یعنی در هر Clock Period شمارنده یا به‌طور صعودی یا به طور نزولی عمل شمارش را انجام می‌دهد. طراحی شمارنده صعودی را در مثال قبل انجام دادیم، حال ابتدا طراحی شمارنده نزولی را با بررسی State table زیر انجام می‌دهیم :

P.S.			N.S.		
A_1	A_2	A_3	A_1	A_2	A_3
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0
0	0	0	1	1	1

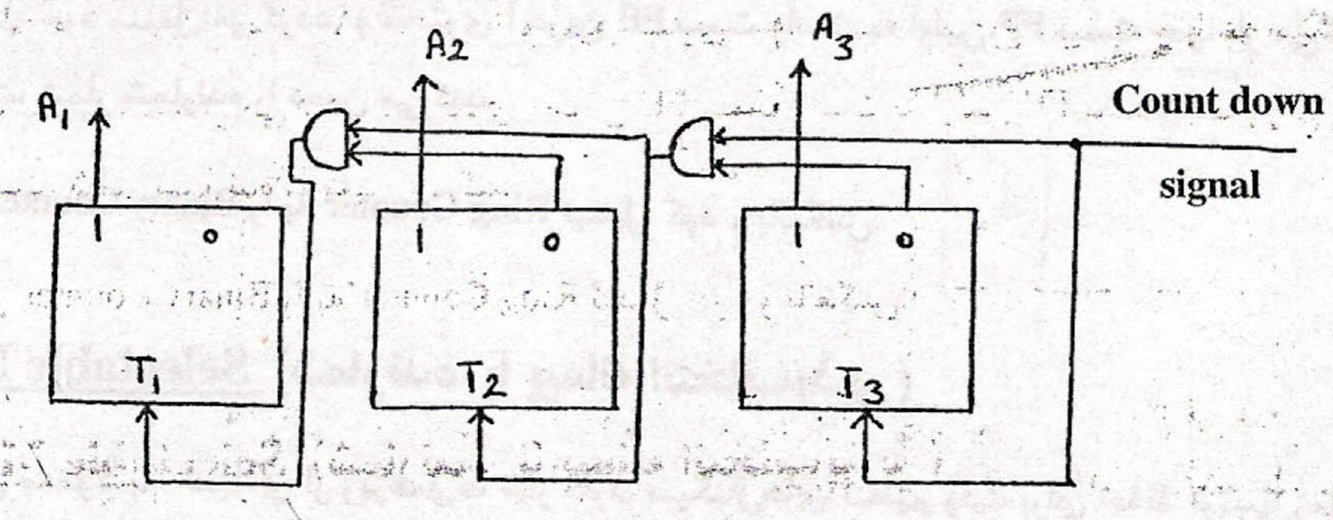
با بررسی جدول حالات (State table) متوجه می‌شویم.

شرایط شمارش نزولی

۱) A_3 با هر سیگنال شمارش نزولی تغییر حالت می‌دهد.

۲) A_2 وقتی تغییر حالت می دهد که FF مرتبه پایین‌تر، یعنی A_3 در حالت 0 باشد.

۳) A_1 وقتی تغییر حالت می‌دهد که FF های مرتبه پایین‌تر، یعنی هم A_3 و هم A_2 در حالت 0 باشند.



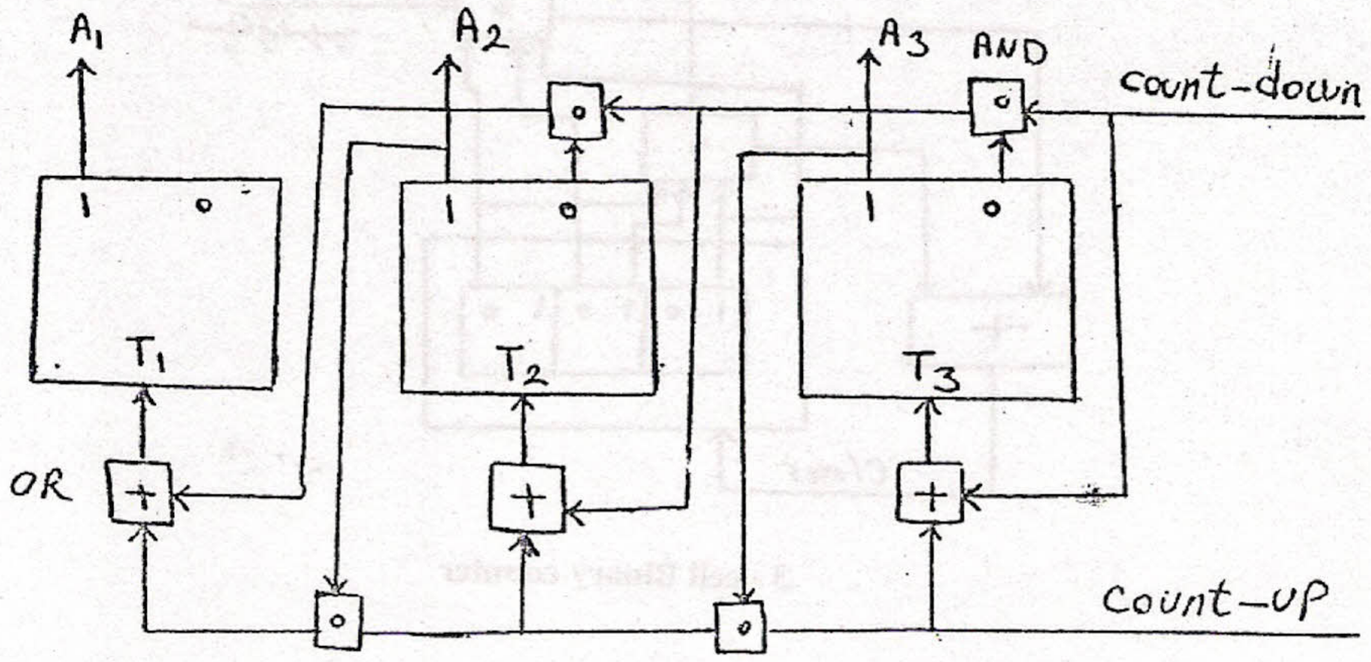
Modulo - 8 Binary down - counter

بنابراین اگر سیگنال‌های شمارش صعودی و نزولی مجزا باشند، عبارت ورودی FF ها را می‌توان به روش زیر به‌دست آورد:

۱ ـ عبارت ورودی FF ها را برای شمارش صعودی به‌دست می‌آوریم.

۲ ـ عبارت ورودی FF ها را برای شمارش نزولی تعیین می‌کنیم.

۳ ـ دو شرط را با هم OR منطقی نموده و به ورودی FF ها وصل می‌کنیم.

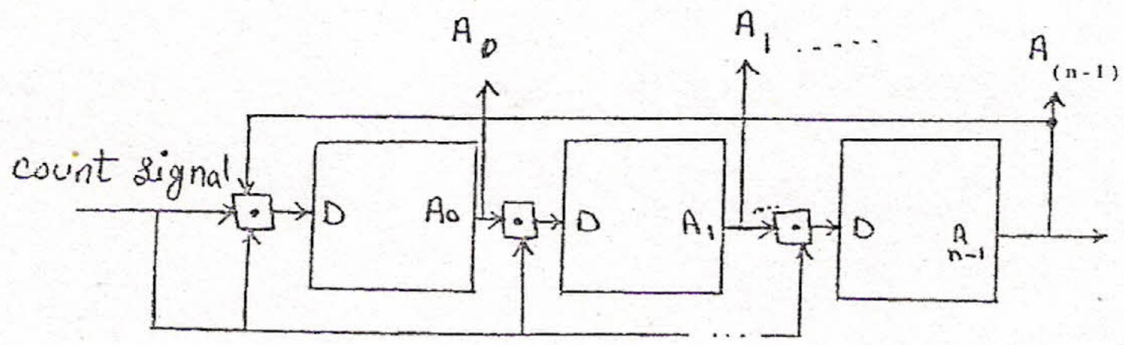


3 - cell up - down Binary count

Ring Counter شمارنده حلقوی

این شمارنده به مدار منطقی اضافی نیاز ندارد و در واقع همان ثبات تغییر مکان می‌باشد.

شکل زیر یک n - cell Ring Counter را نشان می‌دهد که با استفاده از D - FF طرح شده است.

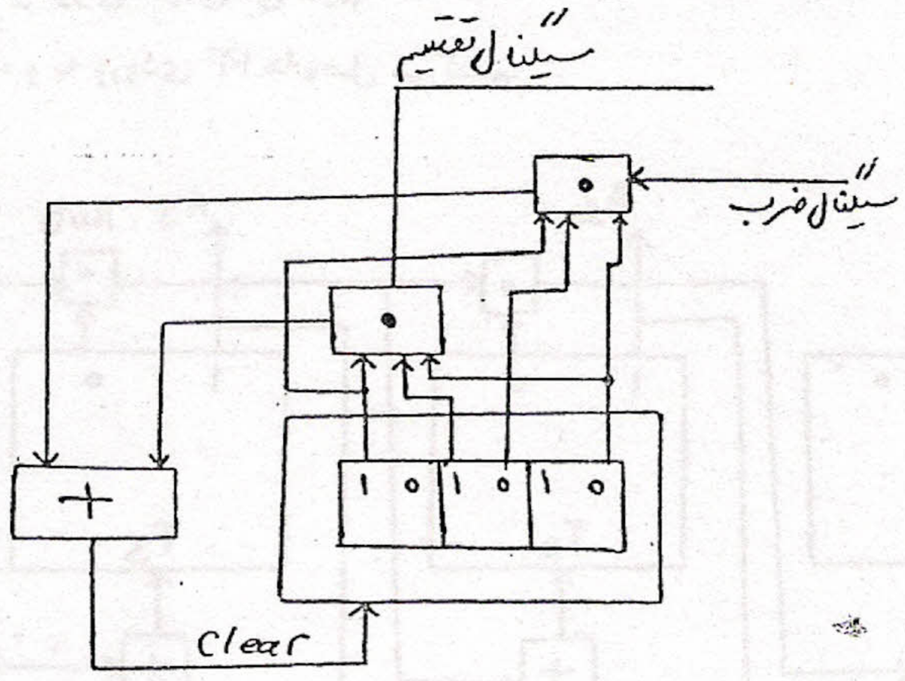


عمل شمارش با حالت اولیه که در آن فقط A_0 مساوی 1 و بقیه مساوی صفر هستند، شروع می‌شود و با هر سیگنال شمارش محتوی هر FF به FF سمت راست خود منتقل می‌گردد. و محتوی آخرین FF سمت راست به اولین FF سمت چپ باز می‌گردد. موضع FF ای که شامل حالت یک است، شمار شمارنده را معین می‌کند.

تمرین: چگونه می‌توان Binary Counter را به Ring Counter تبدیل کرد و بالعکس.

Selectable Modulo Counter (شمارنده با پیمانه انتخاب‌پذیر)

عملیات منطقی یا حسابی معمولاً به تعدادی از ریزعمل‌ها نیاز دارد. سیگنال‌های تنظیم وقت برای حفظ ترتیب زمان انجام هر یک از ریزعمل‌ها را می‌توان به کمک Counter ها به‌وجود آورد و در نتیجه واحد کنترل به تعداد زیادی شمارنده نیاز خواهد داشت. برای رفع این مشکل می‌توان فقط یک شمارنده با پیمانه انتخاب‌پذیر طرح نمود که جایگزین کلیه شمارنده‌ها گردد. مثلاً فرض کنید عمل ضرب و تقسیم دو عدد به ترتیب به 4 و 6 سیگنال تنظیم وقت نیاز داشته باشد، برای تولید این سیگنال‌های تنظیم وقت می‌توان از طرح زیر استفاده نمود.

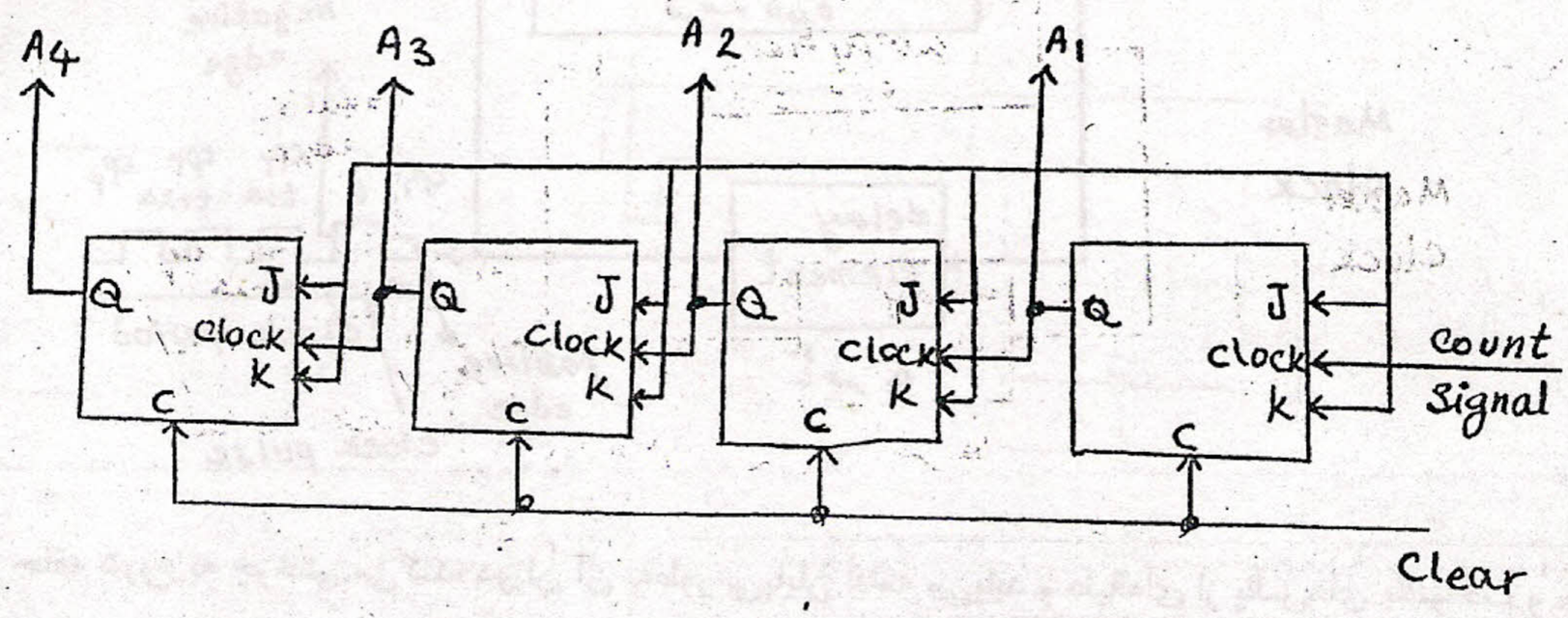


3 - cell Binary counter

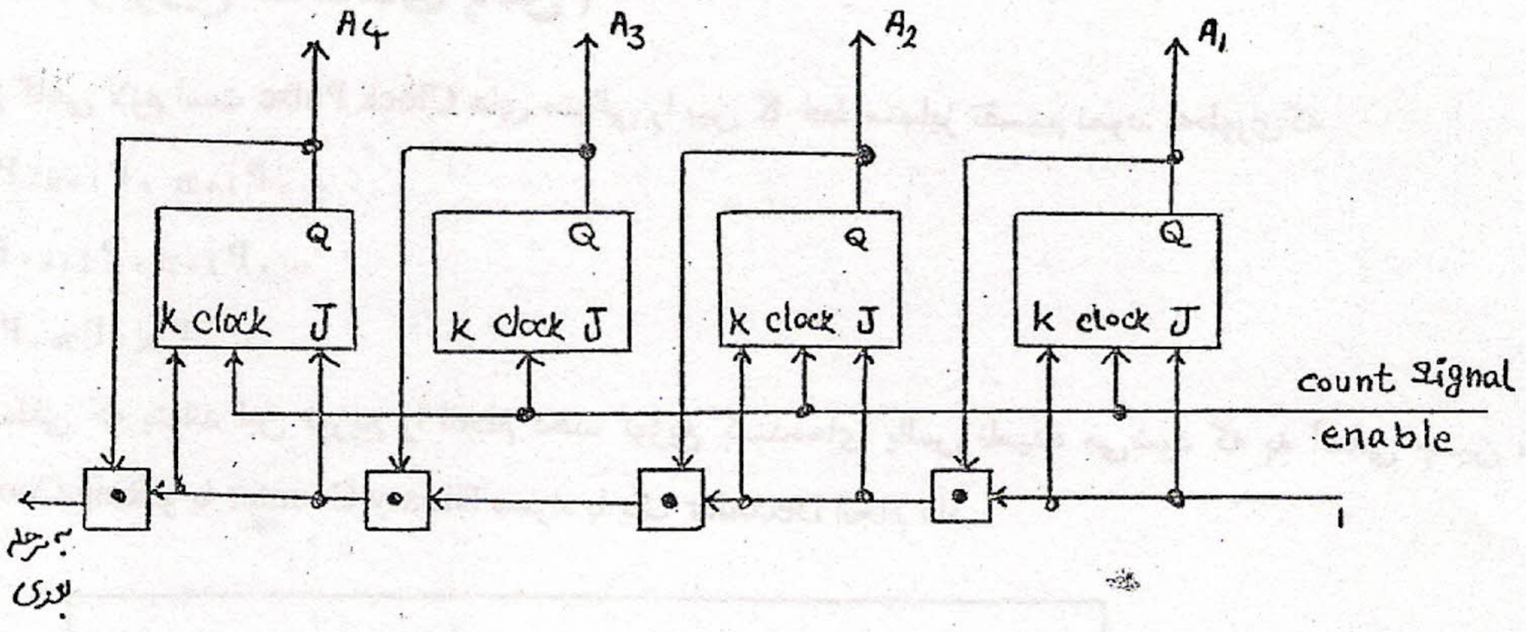
شمارنده مواج Ripple Counter

در این شمارنده تغییر حالت همه FF بجز FF واقع در LSB به‌وسیله خروجی FF ماقبل آن تعیین می‌شود. به عبارت دیگر سیگنال شمارش به عنوان Clock به اولین FF متصل می‌شود و عمل شمارش به‌طور غیرهمگام (Asynchronous) انجام می‌گیرد. سرعت آن به تعداد FF بستگی دارد و مزیت آن به این است که بدون نیاز به مدار منطقی اضافی، بسط داده شود و در واقع ثباتی از نوع Serial – Input parallel – output می‌باشد.

مثال: با استفاده از JK - FF یک Ripple Counter modulo - 16 طرح کنید.



بر خلاف شمارنده‌های غیرهمگام در Synchronous Counter، شمارنده‌های همگام سیگنال شمارش به‌طور مشترک به ورودی Clock همه FF متصل است.



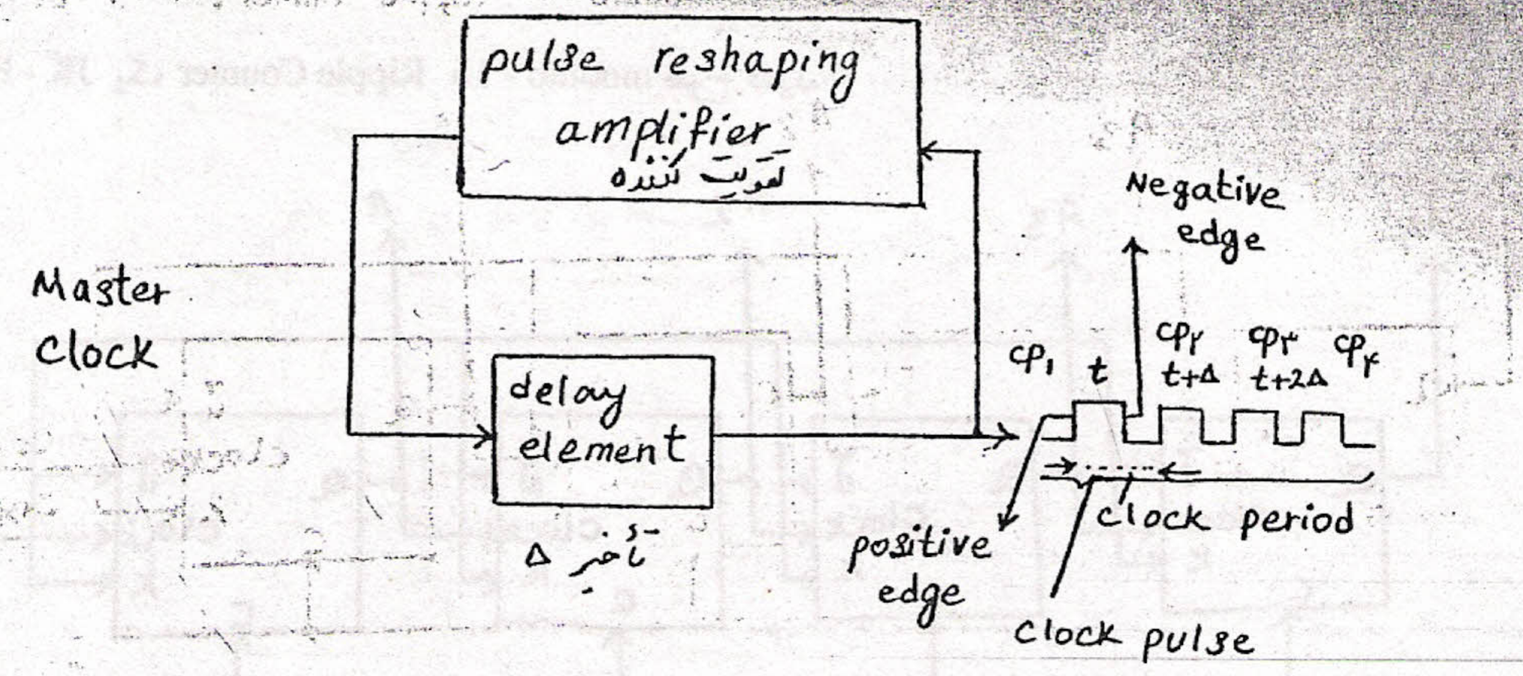
کاربرد شمارنده‌ها

۱ـ برای ذخیره نمودن حالات درونی واحد کنترل : مثلاً اگر PC آدرس دستورالعمل جاری را داشته باشد. با نمو دادن محتوی آن می‌توان آدرس دستورالعمل متوالی را تولید نمود.

۲ـ برای تولید سیگنال تنظیم وقت، به کمک سیگنال‌های تنظیم وقت، می‌توان ترتیب زمان انجام زیرعمل‌های لازم برای انجام هر ریزعمل را تولید نمود.

« مکانیزم کنترل همگام »

یکی از وظایف مهم واحد کنترل فراهم نمودن سیگنال‌های تنظیم وقت (Timing Signals) برای مولفه‌های کامپیوتر می‌باشد، این سیگنال‌های تنظیم وقت توالی لازم برای انجام زیرعمل‌هایی را مشخص می‌کند که برای انجام یک Operation (عمل) موردنیاز می‌باشد و این سیگنال‌های تنظیم وقت را می‌توان به کمک Master Clock تولید نمود. Clock در واقع نوسانگری است که مدار آن به صورت زیر می‌باشد.



وقتی سیگنال در حلقه شروع به چرخش می‌کند، دوران آن به‌طور بی‌پایان ادامه می‌یابد و دنباله‌ای از پالس‌های یکنواخت و متساوی الفاصله و بی‌پایان از نظر زمانی تولید می‌شود.

Pluse Distributor (توزیع کننده‌های پالس)

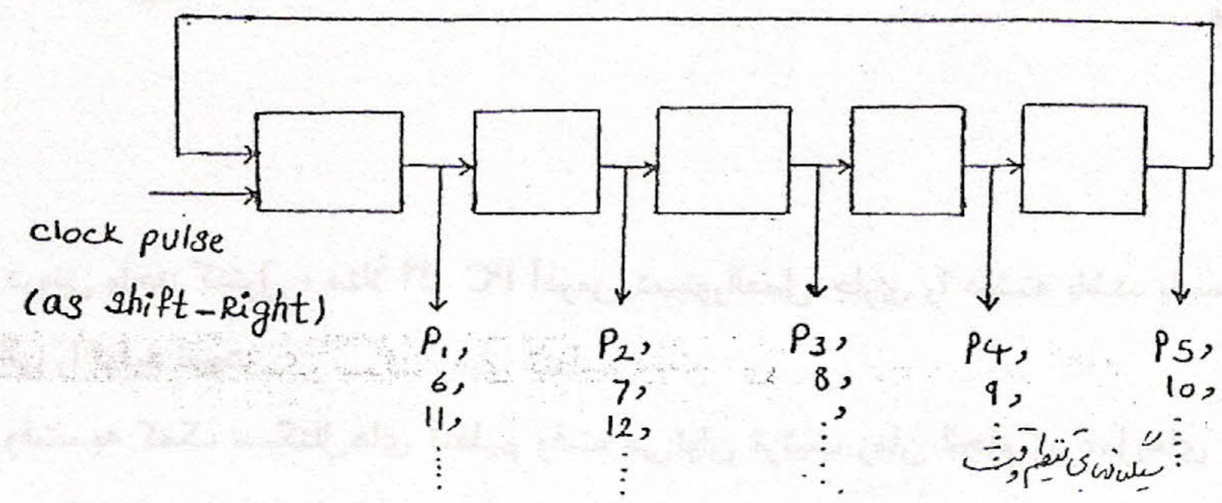
در کامپیوترهای رقمی گاهی لازم است Clock Pulse های متوالی را بین K خط متمایز تقسیم نمود. به‌طوری‌که

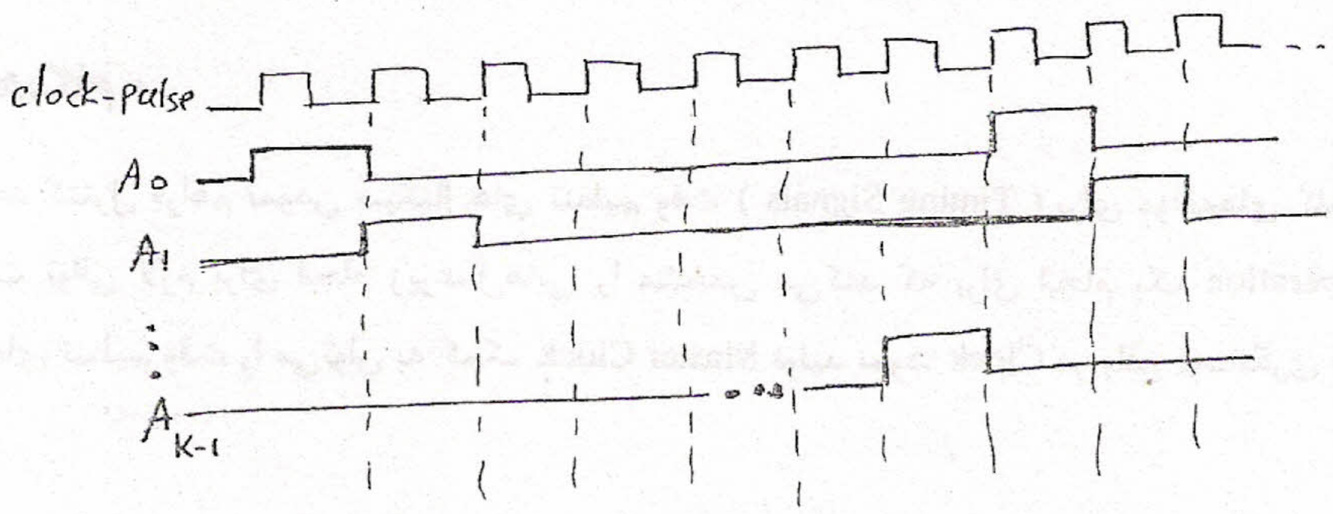
خط اول پالس‌های P_1 , P_{1+k} , P_{1+2k} , ...

خط دوم پالس‌های P_2 , P_{2+k} , P_{2+2k} , ...

خط K ام پالس‌های P_k , P_{2k} , P_{3k} , ...

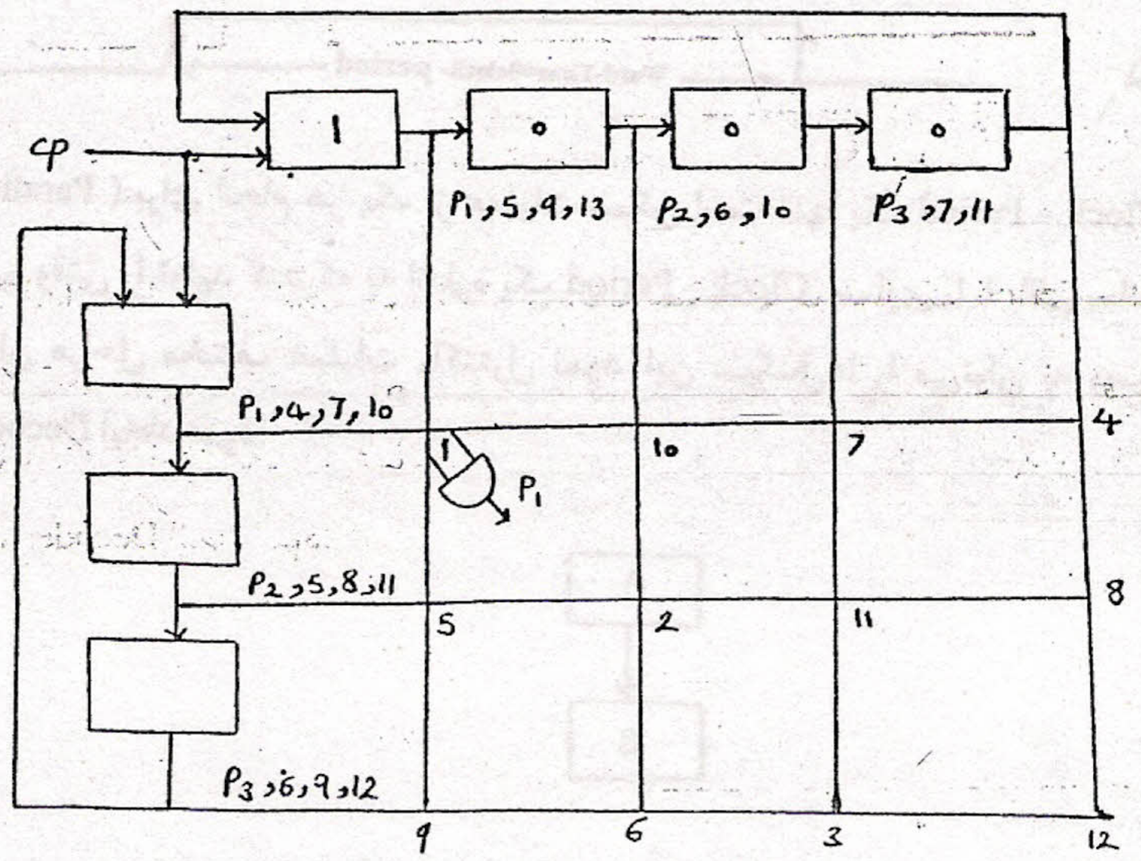
دریافت کنند. مدار منطقی که بتواند این توزیع را انجام دهد، توزیع کننده‌های پالس نامیده می‌شود که به آسانی چنین مداری را می‌توان به‌وسیله Ring Counter و یا Binary Counter همراه با یک Decoder انجام داد.





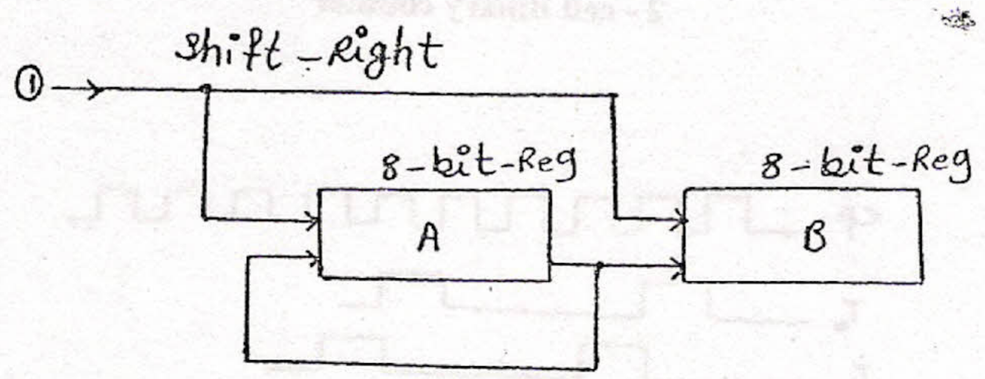
اگر تعداد سیگنال‌های تنظیم وقت یعنی K، عدد بزرگی باشد، می‌توان از دو Ring Counter با چرخه تقریباً $\sqrt{K}$ استفاده نمود. مثلاً اگر K = 12 باشد، می‌توان از دو Ring Counter با طول چرخه 3 و 4 برای تولید سیگنال‌های تنظیم وقت استفاده نمود.

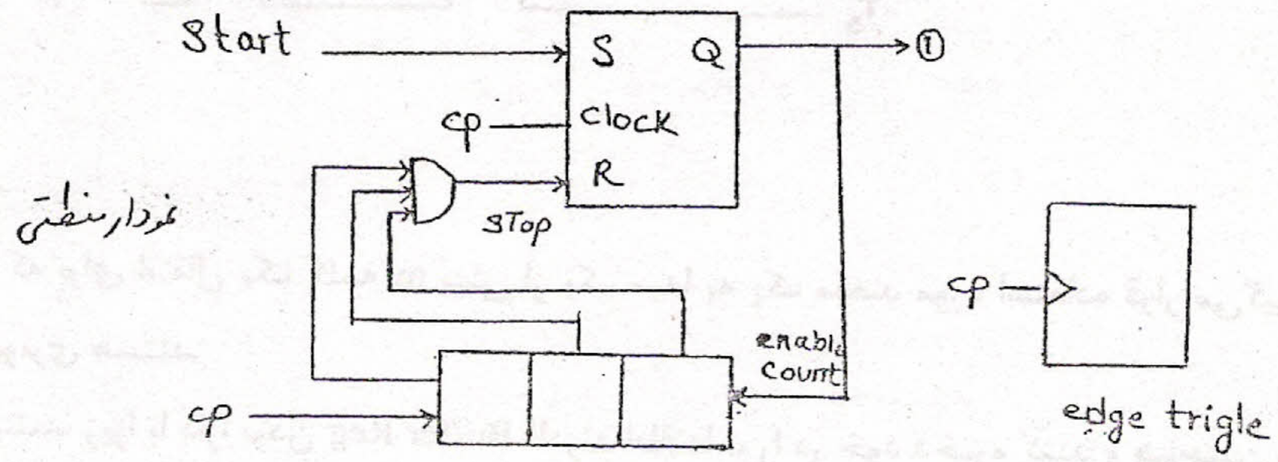
هر کدام از این سیگنال‌ها را می‌توان به خط توانا ساز یک ثبات وصل نمود تا ترتیب زمان انجام زیرعمل‌های لازم را کنترل کند.



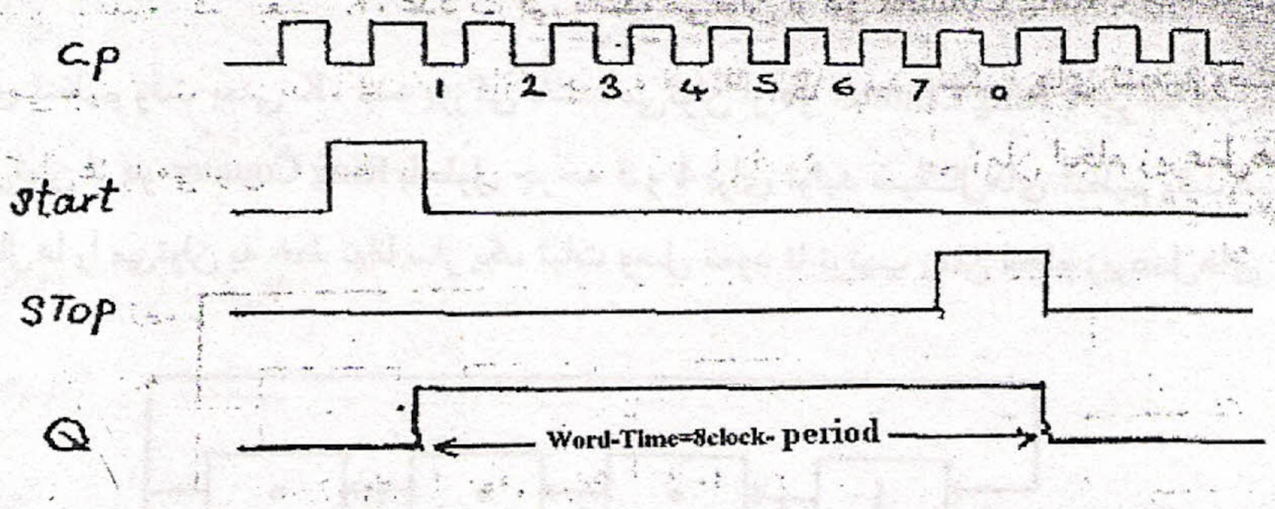
Word - Time Generator

در Serial Operation گاهی لازم است واحد کنترل سیگنالی تنظیم وقتی تولید کند که به تعداد پالس‌های مساوی با تعداد بیت‌های یک کلمه یک باقی بماند. چنین سیگنالی را Word Time گویند که به آسانی می‌توان به‌وسیلهٔ یک Counter تولید نمود.

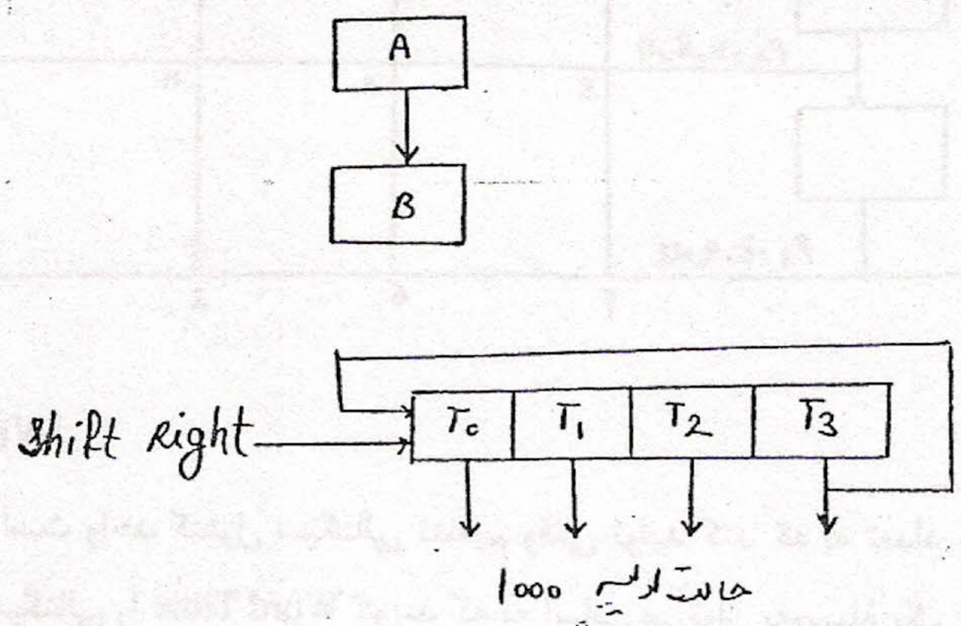




3 - cell Binary counter

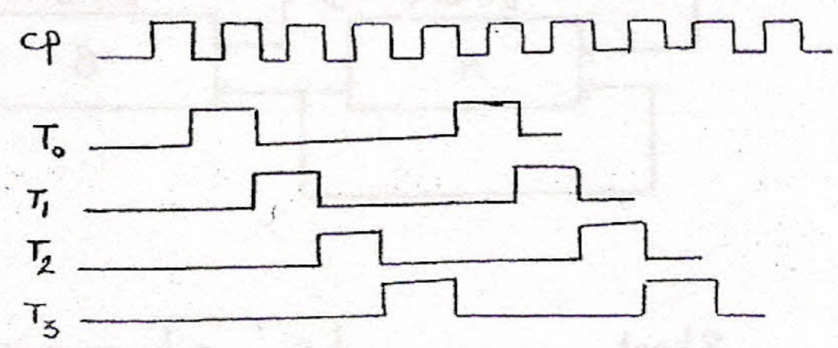


در عملیات موازی (Parallel Operation) برای انجام هر یک از عملیات ممکن است تنها یک Clock - Period کافی باشد، بنابراین واحد کنترل باید سیگنال‌های تنظیم وقتی را تولید کند که به اندازه یک Clock - Period مساوی با 1 باقی بماند. چنین سیگنال‌هایی باید متمایز از یکدیگر باشد تا بتوان مراحل مختلف عملیات راکنترل نمود. این سیگنال‌ها را می‌توان به وسیله Ring Counter یا Binary Counter همراه با یک Decoder ایجاد نمود.



2 - cell Binary counter

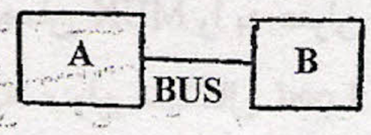
نمودار تنظیم وقت



BUS (گذرگاه)

مجموعه‌ای است از سیم‌های رابط که برای انتقال یک کلمه m بیتی از یک مبدا به یک مقصد مورد استفاده قرار می‌گیرند و معمولاً مبداها و مقصدهای ثبات‌های کامپیوتری هستند.

گذرگاه‌ها دارای خاصیت ثبات‌ها هستند. زیرا با دارا بودن Buffer Reg قادرند اطلاعات را در خود ذخیره کنند و هم‌چنین مانند ثبات‌ها آدرس‌پذیر هستند.



C_1:Reg A → Data BUS ← سیگنال کنترل

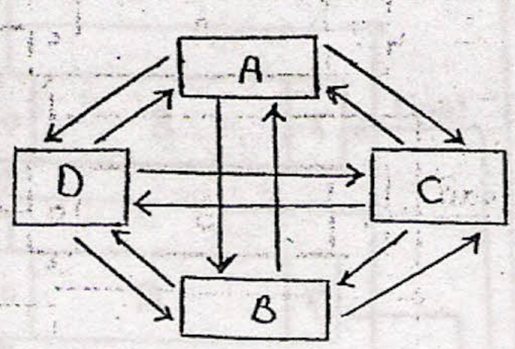
C_2:Data BUS → Reg B

سیستم گذرگاه ممکن است، یکسو یا دوسو باشد.

BUS ها به دو گروه تقسیم می‌شود :

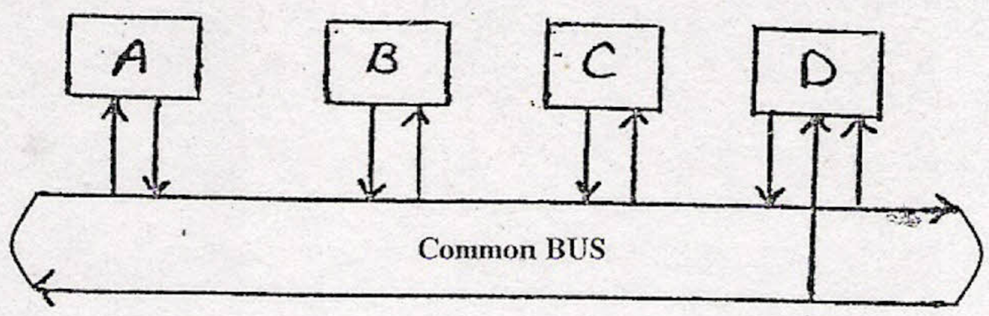
۱ ـ گذرگاه اختصاصی (Dedicated BUS)

در گذرگاه اختصاصی، به ازای هر یک از منابع و مقصدها یک گذرگاه اختصاصی یکسو وجود دارد و به طور کلی اگر n واحد وجود داشته باشد، برای انتقال اطلاعات بین این n واحد به (n – 1) n گذرگاه اختصاصی نیاز خواهیم داشت.



۲ ـ گذرگاه مشترک (Common BUS)

هر چند گذرگاه‌ها هیچ Functionای از خود نشان نمی‌دهند، ولی با توجه به گیرنده‌ها، فرستنده‌ها سیم‌های رابط، مدارهای راه‌گزینی، تقویت‌کننده‌ها و هم‌چنین تعداد Pin های موجود در IC هزینه‌ای را در بر دارد.



به‌دلیل این هزینه‌ها معمولاً از Common BUS (گذرگاه مشترک) استفاده می‌کنند که می‌تواند هر یک از منابع را به هر یک از مقصدها متصل کند.

در گذرگاه‌ها اختصاصی انتقال هم‌زمان می‌تواند صورت بگیرد، ولی در گذرگاه مشترک نمی‌تواند. مدار کنترل‌کننده اختصاصی ساده است، ولی مدار کنترل گذرگاه مشترک پیچیده است.

در گذرگاه اختصاصی برای هر یک از واحدها یک تسهیم‌کننده لازم است. ولی در گذرگاه مشترک به تعداد بیت‌های کلمه به MUX نیاز داریم.

دو ریزعمل در گذرگاه اختصاصی ممکن است در Clock Period - 1 انجام می‌گیرد، ولی همان دو ریز عمل در گذرگاه مشترک ممکن است در Clock Period – 2 انجام گیرد.

سیستم گذرگاه را می‌توان به کمک MUX و Decorder طرح نمود.

MUX برای انتخاب یکی از منابع و Decorder برای انتخاب یکی از مقصدها به‌کار می‌رود.

مثال: برای انتقال اطلاعات بین 4 ثبات 8 بیتی A, B, C و D، یک Commaon BUS طرح کنید.

بیت‌هایی را که دارای رتبه یکسان هستند به ورودی یک MUX متصل می‌کنیم، خروجی همان MUX را به عنوان ورودی محرک به همه FFهایی که دارای رتبه یکسان هستند وصل می‌کنیم. خروجی Decorder را نیز به عنوان سیگنال Load به Reg ها وصل می‌کنیم.

بنابراین MUX به منظور انتخاب منبع و Decorder برای انتخاب مقصد به‌کار می‌رود.