دانشگاه آزاد اسلامی

واحد خمین

معماری کامپیوتری پیشرفته

دکتر محمد خلیلی درمنی

پائیز ۹۳

فهرست مطالب

عنوان **صفحه**

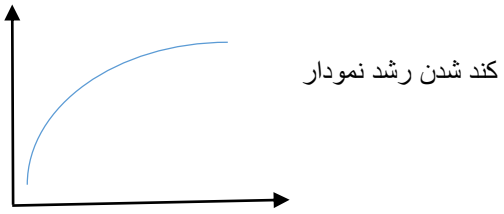
معماری کامپیوتر پیشرفته

مقدمه

حافظه ها و cpu در درون IC قرار می گیرند و تکنولوژی IC هم همیشه در حال پیشرفت است و برای داشتن حافظه بیشتر باید تعداد ترانزیستورهای داخل IC را افزایش دهیم اما اندازه IC ها از یک حد بیشتر نمی تواند باشد به همین دلیل اندازه ترانزیستور ها را کوچکتر و تعداد بیشتری IC در آن قرار می دهیم .با پیشرفت تکنولوژی اندازه ترانزیستورها کوچکتر و حافظه بزرگتر می شود . و بر این اساس شخصی به نام Moor یک قانون که به قانون طلایی Moor معروف است را بیان کرد طبق این قانون تعداد ترانزیستورهای درون IC هر ۱۸ ماه یک بار دو برابر می شود. به طور مثال اگر دریک IC ، ۱۰۰۰ ترانزیستور قرار داشته باشد ۱۸ ماه بعد ، ۲۰۰۰ ترانزیستور در IC قراردارند یعنی رشد این پیشرفت به صورت نمایی است . اما وجود دو مشکل باعث شد نمودار از یک حدی به بعد کند پیشرفت کند این مشکلات عبارتند از :

مشکلات :

۱. محدودیت سخت افزاری (فیزیکی) : وقتی اندازه ترانزیستو رها هر ۱۸ ماه یکبار کوچک تر شد تا به ۲۵ نانومتر رسید ولی دیگر نمی توان این سایز را کاهش داد چون ترانزیستور از یک اتم کوچکتر نمی تواند باشد .
۲. محدودیت اقتصادی : ساخت ترانزیستور برای شرکت ها هزینه دارد و بدون هزینه پیشرفتی حاصل نمی شود به همین دلیل این پیشرفت از هر ۱۸ ماه یکبار به ۲۴ ماه یکبار افزایش یافت .



ارائه راه حل مشکلات :

قبل از بیان راه حل این مشکلات مروری بر نحوه ساخت IC می نماییم . دو راه برای ساخت IC وجوددارد:

راه حل اول : IC ها را یک به یک رسم کرده و خروجه ها را با fabncation می دهیم.

اما کشیدن مدارات بزرگ سخت و مشکل است .

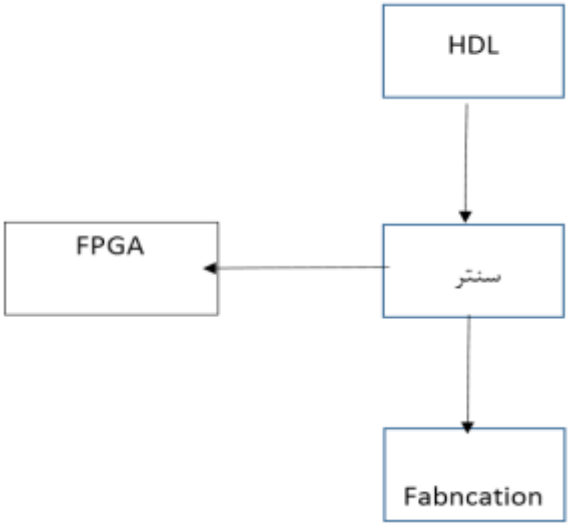
راه حل دوم : استفاده از زبان HDL (زبان توصیف سخت افزار)

دو زبان معروف برای توصیف سخت افزار وجود دارد :

۱- VHDL
۲- Veriling

با این زبان ها می توان سخت افزار را مدل کرد : به این صورت که مداری که داریم را با این زبان ها می نویسم بعد برنامه ها را به یک ابزار سنتر می دهیم و این ابزار برنامه را به مدار تبدیل کرده و سپس خروجی ان را به کارخانه می دهیم یعنی امروزه برای تولید سخت افزار نیز باید برنامه بنویسیم .

گاهی اواقات هم می توان خروجی سنتز را به FPGA ها داد که FPGAیک Ic آماده است که برنامه را روی آن می ریزیم و همان IC مورد نظر ما می شود در این روش هزینه پایین است و برای تعداد کم مناسب است .



راه حل : بیان ایده های جدید به جای روش های قدیمی مثلا به جای ذخیره بیت و صفر ویک از Rbit استفاده کرد و یا از کامپیوتر های کوانتمی لستفاده کرد .ما برای هر اتم یک هسته و یک الکترون داریم که دور هسته می چرخند و در ان جهت چرخش الکترون در اتم را یک بیت در نظر می گیرند که به آن QBit می گویند به طور مثال از چپ به راست چرخیدن معادل عدد یک است .

مثال کامپیوتر کوانتمی : در رمز نگاری برای یک کامپیوتر معمولی صد سال طول می کشد تا رمز را تشخیص دهند ولی با کامپیوتر کرانتمی همین کار در یک ساعت انجام می شود و به جای اینکه روی ولتاژ کار کنند ر وی جریان کار می کنند .

همه این ایده ها سعی می کنند محدودیت ها را از بین ببرند .

تمرین در مورد کامپیوترهای کوانتومی توضیح دهید .

جواب : تاریخچه کامپیوتر کوانتومی:

نظریه کامپیوترکوانتومی ازسال ۱۹۸۲مطرح بوده است از زمانی که فیزیک دان مشهور و برنده جایزه نوبل «ریچارد فاینمن » برای نخستین بار ، پیشنهاد کرد که باید محاسبات از دنیای دیجیتال وارد دنیای جدیدی به نام کوانتوم شود،همچنین بیان کرد کامپیوتر کوانتومی چگونه ممکن است کار کند. این پیشنهاد تا اوایل سال۱۹۹۰مورد توجه جدی قرار نگرفت وبه صورت آکادمیک باقی ماند،البته در سال ۱۹۸۵،دویتش متوجه شد که اظهارات فاینمن ، می تواند تدریجاً به ساخت کامپیوتر کوانتومی منجر شودو مقاله ای را منتشر کرد مبنی بر اینکه اصولاً هر فرآیند فیزیکی را می توان به خوبی با کامپیوترهای کوانتومی مدل سازی کرد. بالاخره در ۱۹۹۴ « پیتر شور» نخستین گام را برای محقق کردن این آرزو برداشت. وقتی که بعضی از مشکلات کلیدی کامپیوتر های معمولی نشان داده شد ، کامپیوترهای کوانتومی در اصل می توانستند خارج از رونوشت های کلاسیکی خود محاسبات را انجام و اجرا نمایند یعنی کارایی بسیار بالاتری را نسبت به کامپیوترهای معمولی از خود نشان می داد ند. وی مقاله ای را منتشر نمود که حاوی روشی برای استفاده از کامپیوترهای کوانتومی در حل مشکل پیچیده ای در نظریه اعداد،به نام فاکتورگیری بود.او نشان داد که چگونه یک مجموعه از عملیات ریاضی که منحصراً برای کامپیوترهای کوانتومی طراحی شده اند ، می توانند چنین دستگاهی را به انجام فاکتورگیری از اعداد بیشماری با سرعت بالاتر از کامپیوتر های کلاسیک ، قادر سازد با این اختراع ، محاسبات کوانتومی از یک کنجکاوی به یک توجه جهانی تبدیل شد.

از آن موقع به بعد، گروه های تحقیقاتی در سرتاسر دنیا مسابقه ای را برای پیش قدم شدن در ساخت یک سیستم عملی آغاز نمودند. به این ترتیب ارتباط نوینی بین نظریه ی اطلاعات و فیزیک کوانتومی شروع به شکل گیری کرد که امروزه انرا محاسبات کوانتومی یا محاسبات نانو متری (Nano Computing) می نامیم. محاسبات کوانتومی مشکلات گذشته را برطرف می سازد و افق جدیدی را ایجاد می کند.

قدرت خارق العاده کامپیوتر کوانتومی در نتیجه وقوع پدیده ای موسوم به توازی کوانتومی، مکانیزمی که انجام و اجرای محاسبات حجیم، زیاد و مکرر را به طور همزمان مقدور می سازد. این یک مقابله سخت و نیرومند و فرق نمایان و بزرگ با کامپیوترهای کلاسیک است که قادرند تنها هر عملیات را فقط به صورت یک عمل در هر با ر و البته خیلی سریع انجام دهند.

چه تفاوتی میان یک کامپیوتر کوانتومی ویک کامپیوترکلاسیک وجود دارد؟

بین کامپیوتر های کوانتومی و کامپیوتر های کلاسیک تفاوت های اساسی وجود دارد:

۱. در کامپیوتر های کوانتومی به جای استفاده از ترانزیستورها و مدارهای رایانه ای معمولی از اتم ها و سایر ذرات ریزمانند نانو ذرات نیمه رسانا نقاط کوانتومی Quantum dots برای پردازش اطلاعات استفاده می کنند. یک اتم می تواند به عنوان یک بیت حافظه در رایانه عمل کند و جا به جایی اطلاعات از یک محل به محل دیگر نیز توسط نور امکان می پذیرد. ذخیره اطلاعات در کامپیوترهانیز به صورت سری هایی از بیت های با حالت های روشن و خاموش صورت می گیرد.

۲. در مقایسه این ۲نوع کامپیوتر می توان گفت،مسائلی که زمانی تصور می شد درکامپیوترهای کلاسیک غیر قابل حل است، درکامپیوتر های کوانتومی حل خواهد شدو شبیه سازی های صورت گرفته به واقعیت نزدیک تر می شود. حتی ابر کامپیوترها هم در برابر آنها رقیبی محسوب نخواهند شد. به عنوان مثال، به روز رسانی نرم افزار ،Email، بانک های آنلاین و تمام قلمرو رمز نگاری عمومی و امضاهای دیجیتال، فقط از دوروش رمز نگاری برای ایمن نگاه داشتن خود استفاده می کنند. RSA و ECC رمزنگاری منحنی بیضی دوروشی هستند که کشف رمز این روش ها، برای کامپیوترهای کلاسیک تا حد زیادی ناشدنی است. ولی یک کامپیوتر کوانتومی به اندازه کافی برای شکستن هردوی این کدها، قدرتمند است. ECC برای امضاهای دیجیتال استفاده می شود،که اطمینان می دهد یک پیغام واقعا توسط فرستنده مدعی، فرستاده شده است RSA. برای بیشتر سیستم های رمزنگاری کلید عمومی استفاده می شود، که در آن یک پیغام، با یک کلید عمومی مجاز کدگذاری می شود و باید با قوانین ریاضی مبتنی بر کلید سرّی رمزگشایی شود.

۳. کامپیوترهای کوانتومی از یک خاصیت دیگر هم سود می برند که آنها را از کامپیوترهای کلاسیک مستثنی می کند و آن انتقال از راه دور است.انتقال از راه دور موجب می شود، اطلاعات یک ذره به ذره دیگری منتقل شود. درنتیجه کامپیوترهای کوانتومی برای انتقال بیت در درون و بیرون ساختار خود، نیازمند سیم نیستند.

۴. تفاوت دیگر کامپیوترهای کوانتومی با کامپیوترهای کلاسیک این است که، اندازه ترانزیستورها هر سال کوچکتر می شود. وقتی اندازه ترانزیستورها به ابعاد اتمی نزدیک می شود، دیگر قوانین حاکم بر فیزیک کلاسیک بر رفتار اتم ها حاکم نیست. به طور مثال کسی نمی داند یک الکترون در زمان مشخصی، دقیقاً در کجا قرار دارد یا کسی نمی تواند به درستی تشخیص دهد که الکترون در یک سیم به کجا می رود. یعنی وقتی به ابعاد اتمی نزدیک می شویم، فیزیک کوانتومی رفتار اتم ها را توضیح می دهد و دیگر قوانین فیزیک کلاسیک کاربرد ندارد. در واقع این نوع کامپیوترها با استفاده از فناوری های میکروسکوپی ذره ها کار می کنند.

۵. همان طورکه می دانیم دریک کامپیوتر کوانتومی نسبت به کامپیوترهای کلاسیک، اصول حاکم تغییر نموده اند. نه تنها، یک بیت کوانتومی، موسوم به کیوبیت می تواند درحالت های صفرویک کلاسیک وجود داشته باشد بلکه همچنین می توانددرحالت برهم نهی قرارداشته باشد.هرگاه هر کیوبیت دریک کامپیوترکوانتومی درحالت برهم نهی واقع شده باشد، آنگاه کامپیوتررامی توان درهرحالت ممکنی مجسم کرد که آن کیوبیت هامی توانند ازخودنشان دهند. درواقع کامپیوتر های کوانتومی می تواند در یک زمان چندین حالت داشته باشد و این امکان راایجادمی کند که میلیون ها بار سریع تر و قدرتمند تر از ابرکامپیوتر های فعلی کار کند. چند حالت پذیری کیوبیت ها همان دلیلی است که باعث می شود کامپیوتر های کوانتومی ذاتاً از پردازش موازی بهره ببرند. پردازش موازی امکان کار کردن بر روی میلیون ها محاسبه در یک لحظه را به این کامپیوتر ها می دهد در حالی که کامپیوتر شخصی شما فقط یک محاسبه در لحظه انجام می دهد.

نحوه برقراری ارتباط درکامپیوترهای کوانتومی

با توجه به ماهیت ساختار کامپیوترهای کوانتومی، روش برقراری ارتباط آنها کاملاً متفاوت با کامپیوترهای امروزی است. بدین صورت که پالس های رادیویی نقش صفحه کلید را دارند،که به وسیله آن اطلاعات وارد کامپیوتر می شود و دستگاه تشدید مغناطیسی که شبیه به دستگاه MRI بیمارستان است، نقش صفحه نمایش را ایفا می کند و با ارائه تصویر مغناطیسی از توده مولکولها ، کامپیوتر توده محاسبات را به ما می دهد.

از سوی دیگر باید تلاش کرد ترانزیستورهایی از جنس مورد نظر ساخت، زیرا ترانزیستورها،عامل تقویت ولتاژ در مدارهای الکترونیکی هستند و قدرت تقویت کنندگی آنها موجب افزایش سرعت کامپیوتر ها است.

این ترانزیستورها تاثیر مهمی در تولید کامپیوترهای آینده دارند و در صورتی که در ابتدا یا انتهای ساختار آنها،ترکیبی با دیگر نیمه هادی ها به خصوص طلا ایجاد شود یا حتی روی پوسته آنها نیمه هادی مهمی چون «روی» قرار داده شود،گام مهمی برای تولید قدرتمندترین ترانزیستورها برداشته ایم.

در واقع زمانی که این نیمه هادی های ترکیب شده،به یک باطری متصل می شوند و الکتریسیته دریافت می کنند، همچون یک ترانزیستور عمل کرده و موجب تقویت ولتاژ در مدار و همچنین موجب افزایش سرعت کامپیوتر می شوند. از آنجایی که جریان ورودی به این نیمه هادی ها قابل کنترل است، جریان خروجی از

آنها هم قابل کنترل است. این ترکیب ها این قابلیت را دارند که در ساخت نقاط کوانتومی مورد استفاده در کامپیوترهای نسل آینده استفاده شوند.

این نقاط کوانتومی در واقع کریستال هایی از نوع نیمه هادی هستند،که قابلیت ذخیره کردن الکترون ها در آنها فوق العاده بالاست.این نقاط کوانتومی بهترین مکان برای ذخیره سازی اطلاعات در کامپیوترهای پیشرفته هستند. از سوی دیگر در صورتی که بتوان نقاط کوانتومی را با یکدیگر پیوند داد، می توان آنها را به اندازه تنها چند سانتی متر مربع در ساختار سخت افزاری کامپیوترها جای داد. به این ترتیب هر سانتی متر مربع از درایورهای ما می توانند صدها گیگا بایت از اطلاعات را در خود ذخیره کنند.

تا چند سال دیگر، کامپیوترهای کوانتومی از داخل آزمایشگاه های تحقیقاتی دانشمندان علوم رایانه، فیزیک و ریاضی دانان بیرون خواهند آمد و به صورت کاربردی و عملی مورد استفاده قرار خواهند گرفت. آن دسته از مسائل که با محاسبات پیچیده ی خود، کامپیوترهای جبری امروز را به ستوه می آورند، توسط کامپیوترهای کوانتومی به آسانی حل خواهد شد.

محاسبات کوانتومی

هدف محاسبات کوانتومی یافتن روش هایی برای طراحی مجدد ادوات شناخته شده ی محاسبات (مانند گیت ها و ترانزیستورها) به گونه ای است که بتوانند تحت اثرات کوانتومی، که در محدوده ی ابعاد نانومتری و کوچکتر بروز می کنند کار کنند. ورود به دنیای محاسبات کوانتومی نیازمند دو پیش زمینه مهم است، نخست باید اصول اساسی و برخی تعابیر مهم مکانیک کوانتومی را به طور دقیق بررسی کرد سپس مفهوم اطلاعات در فیزیک نیز، چه به صورت کلاسیک و چه در معنای جدیدکوانتومی آن باید درک شود. بنابراین محاسبات کوانتومی را به عنوان یک زمینه و روش جدید و بسیار کارآمد مطرح می کنند. هر سیستم محاسباتی دارای یک پایه اطلاعاتی است که نماینده ی کوچکترین میزان اطلاعات قابل نمایش، چه پردازش شده و چه خام است. همان طورکه درقسمت قبل نیزگفتیم در محاسبات کلاسیک، این واحد ساختاری را بیت می نامیم که گزیده واژه « عدد دو دویی » است زیرا می تواند تنها یکی از دو رقم مجاز صفر و یک را در خود نگه دارد به عبارت دیگر هر یک از ارقام یاد شده در محاسبات کلاسیک، کوچکترین میزان اطلاعات قابل نمایش محسوب می شوند. پس سیستم هایی هم که برای این مدل وجود دارند باید بتوانند به نوعی این مفهوم را عرضه کنند.ودر محاسبات کوانتومی هم چنین پایه ای معرفی می شود ،که آنرا (QUBIT) یا بیت کوانتومی می نامیم. اما این تعریف کیوبیت نیست و باید آنرا همراه با مفهوم و نمونه های واقعی و

فیزیکی درک کرد. در ضمن فراموش نمی کنیم که کیوبیت ها سیستم هایی فیزیکی هستند، نـه مفـاهیمی انتزاعی و اگر از ریاضیات هم برای توصیف آنها کمک می گیریم تنها بدلیل ماهیت کوانتومی آنها است.

در فیزیک کلاسیک برای نگه داری یک بیت از حالت یک سیستم فیزیکی استفاده می شود. در سیستم های کلاسیکی اولیه (کامپیوترهای مکانیکی) از موقعیت مکـانی دندانـه هـای چنـد چـرخ دنـده بـرای نمـایش اطلاعات استفاده می شد. از زمانیکه حساب دودویی برای محاسبات پیشنهاد شد، سیسـتم هـای دو حـالتی انتخابهای ممکن برای محاسبات عملی شدند. به این معنی که تنها کافی بود تـا سیسـتمی دو حالـت یـا دو پیکربندی مشخص، متمایز و بدون تغییر داشته باشد تا بتوان از آن برای این منظور استفاده کرد. به همـین جهت، از بین تمام کاندیداها، سیستم های الکتریکی و الکترونیکی برای این کار انتخاب شدند. به این شکل، هر بیت، یک مدار الکتریکی است که یا در آن جریان وجود دارد یا ندارد.

هر بیت کوانتومی یا کیوبیت عبارتست از یک سیستم دو دویی که می تواند دو حالت مجزا داشته باشـد. بـه عبارت فنی تر، کیو بیت یک سیستم دو بعدی کوانتومی با دو پایه به شکل $|0>$ و $|1>$ است. البته نمـایش پایه ها یکتا نیست، به این دلیل که بر خلاف محاسبات کلاسیک در محاسـبات کوانتـومی از چنـد سیسـتم کوانتومی به جای یک سیستم ارجح استفاده می کنیم.

انتخاب ایده ال برای نمایش کیوبیت استفاده از مفهوم اسپین است که معمولا اتم هیدروژن برای آن به کـار می رود،چون دریک اتم هیدروژن هم پروتون وهم الکترون ،دارای اسپین می باشد. در اندازه گیـری اسـپین یک الکترون ، احتمال بدست آمدن دو نتیجه وجود دارد: یا اسپین رو به بالاست که آنرا با نشان می دهند و معادل $|0>$ است و یا رو پایین است که آن را با نشان می دهیم و معادل با $|1>$ است .بـالا یـا پـایین بـودن جهت اسپین در یک اندازه گیری از آنجا ناشی می شود که اگر اسپین اندازه گیری شده در جهـت محـوری باشد که اندازه گیری را در جهت آن انجام داده ایم، آنرا بالا و اگر در خلاف جهت این محور باشد آنرا پـائین می نامیم. شاید بتوان گفت مهم ترین تفاوت بیت و کیوبیت در این دانست که بیت کلاسیک فقط می تواند در یکی از دو حالت ممکن خود قرار داشته باشد در حالیکه بیت کوانتومی می تواند به طور بالقوه در بیش از دو حالت وجود داشته باشد. تفاوت دیگر در اینجاست که هرگاه بخواهیم می توانیم مقدار یک بیت را تعیین کنیم اما اینکار را در مورد یک کیوبیت نمی توان انجام داد.

به زبان کوانتومی یک کیوبیت را با عبارت نشان می دهیم. البته اندازه گیر ی یک کیوبیت حتما یکی از دو نتیجه ممکن را بدست می دهد. از سوی دیگر اندازه گیری روی سیستم های کوانتومی حالت اصلی آنها را تغییر می دهد. کیوبیت در حالت کلی در یک حالت برهم نهاده از دو پایه ممکن قرار دارد.

اما در اثر اندازه گیری حتما به یکی از پایه ها برگشت می کند. به این ترتیب هر کیوبیت، بیش از اندازه گیری شدن می تواند اطلاعات ز یادی را در خود داشته باشد.بر اساس اصل برهم نهی((superposition، هر سیستم کوانتومی که بیش از یک حالت قابل دسترس دارد، می تواند به طور همزمان در یک ترکیب خاص از آن حالت ها هم قرار داشته باشد. در اصطلاح می گوئیم که سیستم کوانتومی علاوه بر حالت های ناب یک یا چند حالت آمیخته یا بر هم نهیده (blend or superposed) نیز دارد. پس اگر یک ساختار حافظه ای n کیوبیتی داشته باشیم، طبق این اصل، این تعداد می توانند در ۲ nپیکربندی متمایز وجود داشته باشند. به این ترتیب یک کامپیوتر کوانتومی این امکان را می یابد که مانند یک کامپیوتر موازی کلاسیک بسیار پر قدرت عمل کند که در یک لحظه روی چندین مسیر اطلاعاتی پردازش می کند. البته مشاهده و متمایز کردن تک تک این محاسبه گرهای کوانتومی غیر ممکن است. چون کامپیوتر کوانتومی با تعداد بسیار زیادی مسیر محاسباتی کار می کند، می توان کاری کرد که این محاسبات با هم تداخل یا بر هم تاثیر هم داشته باشند. به عبارتی، محاسباتی که به طور موازی با هم انجام می شوند طبق اصل تداخل می توانند اثر هم را تقویت یا تضعیف کنند. در نتیجه محاسبه ای شبکه ای بوجود می آید که نوعی خاصیت جمعی از تمام محاسبات را نشان می دهد. خاصیت بسیار شگفت انگیز در مکانیک کوانتومی خاصیت در هم تافتگی است. اگر دو یا چند کیوبیت را در بر هم کنش با هم قرار دهیم، می توانند برای مدتی در یک حالت کوانتومی مشترک قرار بگیرندبه طوریکه نتوان آن حالت را به شکل حاصلضربی از حالت های جدا ازهم اولیه نشان داد.حالت این واحدهای اطلاعاتی راگنگ یا نادقیق(fuzzy) می نامیم.

این است که یک جفت کیوبیت درهم پیچیده روی یکدیگر درهم تافتگیentanglementیک نتیجه مهم تاثیرهمزمانی را می گذارند که به فاصله آن ها ازیکدیگر وماده ای که این فاصله را پرمی کند بستگی ندارد.

فصل اول

روش های عددی متفاوت

RNS 1-1(Residue Number System) : سیستم اعداد مانده ای

کامپیوترها امروزه در مبنای ۲ کار می کنند اما در گذشته مبنای کامپیوترها ۱۰ بوده است در حالت کلی هرچه مبنا بالاتر باشد نیاز به هوش بیشتری می باشد .

همه اعمال در کامپیوتر قابل تبدیل به جمع هستند اما در جمع کردن دو عدد باینری وقتی اعداد طولانی می شوند تاخیر انتشار Carry به وجود می آید و راه حل این مشکل RNS می باشدکه سرعت پردازنده را بالا می برد . در این روش ما مبحث پیمانه ها را داریم .

قانون پیمانه ها : پیمانه ها باید نسبت به هم اول باشند (پیمانه ها دو به دو نسبت به هم اول باشند یعنی بزرگترین مقسوم علیه مشترکشان یک باشد).

اعداد اول : ۲و۳و۵و۷و۱۱و۱۳و.....

بزرگترین عددی که در پیمانه(۹و۸و۷) می توان نشان داد ۹*۸*۷ می باشد .

در پیمانه (۵و۴و۳) بزرگترین عدد قابل نمایش ۵*۴*۳=۶۰ می باشد

می خواهیم اعداد را در این پیمانه نمایش بدهیم مثلا عدد ۱۲

12=(2,0,0)

که در آن 2=12 mod 5 و 0=12 mod 4 و 0= 12 mod 3 می باشد و رقم نقلی نداریم و جواب باید از پیمانه کوچکتر باشد .

14=(4,2,2)

26=(1,2,2)

که در ان ۱ به این صورت محاسبه می شود : جمع ۲ و ۴ عدد ۶ می شود ولی چون پیمانه ۵ است و ۶ بزرگتر از ۵ است پس 6-5=1 می شود چون 6 mod 5 =1

نمایش اعداد در پیمانه باعث می شود اعداد کوچکتر شوند مثلا در پیمانه فوق بزرگترین عددی که می توانیم داشته باشیم ۴ است و ۵ هم نمی توانیم داشته باشیم و در ضمن می توانیم محاسبات را موازی انجام دهیم . پس می توان گفت اعداد را به پیمانه می بریم (به حالت RNS) و عمل جمع را انجام داده و دوباره به حالت اولیه می گردانیم

(5,4,3)		
(1,1,0)	+	21
(3,2,0)		18
(4,3,0)		39

(2,1,2)		17
(2,1,2)	+	17
(4,2,1)		34

در این روش یعنی RNS:

۱-Carry نداریم

۲-اجرای موازی و محاسبات ساده تر

(.,1,2)	*	5
(1,3,2)		11
(0,3,1)		55

(3,2,0)		17
(0,1,2)	-	17
(, ,-2)		34

برای تبدیل به اندازه پیمانه می توانیم اضافه کنیم 0+3 =3 , -2+3=1

13=(3,1,1)

ضرب و تفریق انجام می شود تقسیم مشکل است .

در تفریق اگر جواب منفی شد به اندازه پیمانه به آن اضافه می کنیم .

اعداد در این پیمانه هذا و در این بازه ها منحصر به فرد هستند .

۱-۱-۱ اثبات درستی عمل جمع در RNS:

فرضیات :

(m_2,m_1,m_0)

$K=(k_2,k_1,k_0)$ $\begin{cases} k=r_2m_2+k_2 \\ K=r_1m_1+k_1 \\ K=r_0m_0+k_0 \end{cases}$

$X=(x_2,x_1,x_0)$ $\begin{cases} x=L_2m_2+x_2 \\ X=L_1m_1+x_1 \\ X=L_0m_0+x_0 \end{cases}$

حکم قضیه :

$K+x=(k_2+x_2 \bmod m_2,\ k_1+x_1 \bmod m_1,\ k_0+x_0 \bmod m_0)$

اثبات حکم قضیه :

$K+x=r_2m_2+k_2+L_2m_2+x2=(r_2+L_2)m_2+k_2+x_2$

اگر k_2+x_2 از m_2 کوچکتر بود که جواب خوش است و اگر k_2+x_2 از m_2 بزرگتر بود به اندازه پیمانهm_2 کم می کنیم و به r_2+L_2 یک اضافه می کنیم یا همان باقی مانده را حساب می کنیم .

$K+x=r_1m_1+k_1+L_1m_1+x_1=(r_1+L_1)m_1+k_1+x_1$

پس وقتی به اندازه پیمانه کم می کنیم می شود mod

پس اثبات شد که برای جمع دو عدد می توان پیمانه ها یا باقی مانده ها را هم جمع کنیم .

تمرین : اثبات انجام شدن ضرب و تفریق – اثبات انجام نشدن تقسیم

پیمانه ها را جمع می کنیم ،اگر از پیمانه کوچکتر بود خودش اما اگر نبود به اندازه M کم میکنیم به آن طرف ۱ اضافه می کنیم.

تمرین : اثبات ضرب و تقسیم به عنوان تمرین برای شما (اثبات شدن ضرب و نشدن تقسیم)

۱- اثبات کنید ضرب و تفریق در مبنای RNS درست است و تقسیم درست نیست.

$$x(x_2,x_1,x_0)\rightarrow\begin{cases}x=m_2k_2+x_2\\x=m_1k_1+x_1\\x=m_0k_0+x_0\end{cases}$$

$$y(y_2,y_1,y_0)\rightarrow\begin{cases}y=m_2k'_2+y_2\\y=m_1k'_1+y_1\\y=m_0k'_0+y_0\end{cases}$$

$$x*y = (m_2k_2+x_2)*(m_2k'_2+y_2)=m_2^2k_2k'_2+m_2k_2y_2+m_2k'_2x_2+x_2y_2 = m_2(m_2k_2k'_2+k_2y_2+k'_2x_2)+x_2y_2 = m_2k''_2+x_2y_2$$

$$x*y = (m_1k_1+x_1)*(m_1k'_1+y_1)=m_1^2k_1k'_1+m_1k_1y_1+m_1k'_1x_1+x_1y_1 = m_1(m_1k_1k'_1+k_1y_1+k'_1x_1)+x_1y_1 = m_1k''_1+x_1y_1$$

$$x*y = (m_0k_0+x_0)*(m_0k'_0+y_0)=m_0^2k_0k'_0+m_0k_0y_0+m_0k'_0x_0+x_0y_0 = m_0(m_0k_0k'_0+k_0y_0+k'_0x_0)+x_0y_0 = m_0k''_0+x_0y_0$$

$$\Rightarrow (x_2y_2,x_1y_1,x_0y_0)$$

$$x(x_2,x_1,x_0)\rightarrow\begin{cases}x=m_2k_2+x_2\\x=m_1k_1+x_1\\x=m_0k_0+x_0\end{cases}$$

$$y(y_2,y_1,y_0)\rightarrow\begin{cases}y=m_2k'_2+y_2\\y=m_1k'_1+y_1\\y=m_0k'_0+y_0\end{cases}$$

$$x-y=m_2k_2+x_2-(m_2k'_2+y_2)=m_2(k_2-k'_2)+x_2-y_2$$

$$x-y=m_1k_1+x_1-(m_1k'_1+y_1)=m_1(k_1-k'_1)+x_1-y_1$$

$$x-y=m_0k_0+x_0-(m_0k'_0+y_0)=m_0(k_0-k'_0)+x_0-y_0$$

نکته:

در هریک از تفریق های بدست آمده که حاصل عدد منفی شده به اندازه یک پیمانه به ان اضافه میکنیم.

$$\Rightarrow x-y\ (x_2-y_2,x_1-y_1,x_0-y_0)$$

$$\frac{x}{y}=(m_2k_2+x_2)/(m_2k'_2+y_2)$$

این عبارت الزاما برابر با $\frac{m_2k_2}{m_2k_2'} + \frac{x_2}{y_2}$ یا عبارتی که بتوان آن را بصورت $m_2k_2'' + \frac{x_2}{y_2}$ نوشت نیست و از طرفی در $\frac{x_2}{y_2}$ امکان دارد y_2 صفر باشد که حاصل تعریف نشده است و باقیمانده هیچ تقسیمی در ریاضی تعریف نشده نیست بنابراین در دلیل تقسیم در RNS صحیح نیست.

مثال) محدوده (۶۰ تا ۱) یا (۵۹ تا۰)= ۶۰

(۳ و ۴ و ۵) پیمانه

۱۸ =(۰ و ۲ و ۳) forward conversion

۱۷ =(۲ و ۱ و ۲)

(۲ و ۳ و ۰) ۳۵

-۷ ⟶ -۷ + ۳۰ = ۲۳⟶ (۲ و ۳ و ۳)

-۸ ⟶ -۸ + ۳۰ = ۲۲⟶ (۱ و ۲ و ۲)

-۱۵ ⟶ -۶⟵ (۰ و ۱ و۰) = ۴۵

برای تحت پوشش قرار دادن عدد منفی : هر عدد را ۳۰+ می کنیم عدد را یک shift می دهیم ، جواب ۳۰- شود تا برگردد به حالت قبلش

(RNS)

عدد را از ۶۰- کم کردیم زیر ۲ تا ۳۰ تا اضافه شده است ⟶ ۶۰-

راه حل های دیگر :

۱: پیمانه ها زیاد شود اما نسبت به هم اول باشند، (به جای ۳ پیمانه ۴ پیمانه)

۲: اعداد بزرگتر شود.

اگر پیمانه ها زیاد تر شود موازی سازی بیشتر می شود و موازی می توان کار کرد اما : ۱) عدد بردن به پیمانه ها سخت تر می شود ۲) برگرداندن آنها هم سخت تر می شود.

اما اگر اعداد بزرگتر شوند موازی سازی کمتر می شود محدوده اعداد بزرگ می شود محاسبات سخت تر می شود اما forward و backward کم می شوند و باید بر اساس نیاز انتخاب کرد و کار را انجام داد.

۲-۱-۱ برگرداندن اعداد از پیمانه (RNS)به مبنای به مبنای ۱۰

(روش چینی): backward

زمانی که پیمانه را داشته باشیم و از ما عدد مربوط به آن را بخواهند از این روش استفاده مـی کنـیم. بـرای این کار به صورت زیر عمل می کنیم:

اون چه عددی که باقی مانده بر ۵ ← ۲ باقی مانده به ۴ ← ۲ و باقی مانده به ۳ ← ۱ می شود ؟

پیمانه (۳ و ۴ و ۵) ضرب ۴ و ۳

(۱ و ۱ و ۱) = ۱

(۰ و ۰ و ۱) = ؟

(۰ و ۱ و ۰) = ؟

۴ × ۳ = ۱۲

۱۲ × ۲ = ۲۴

۱۲ × ۳ = ۳۶

۵ × ۳ =۱۵

۱۵ × ۲ = ۳۰

۱۵ × ۳ = ۴۵

۳۶ = (۰ و ۰ و ۱)

۴۵ = (۰ و ۱ و ۰)

۴۰ = (۱ و ۰ و ۰)

(۱ و ۰ و ۰) = ؟

۵ × ۴ = ۲۰

۲۰ × ۲ = ۴۰

(مثال

(۰ و ۱و ۰) × ۳ + (۰ و ۰ و ۱) × ۴ = (۱ و ۳ و ۴) = ۱۹

+ ۱ × (۱ و ۰ و ۰) ⟶ ۴ × ۳۶ + ۳ × ۴۵ + ۱ × ۴۰ = ۳۱۹

۶۰ = ۱۹mod۳۱۹

(مثال

۶۰=۲۷mod = (۲ و ۳ و ۰ ⟶ ۲× (۰ و ۰ و ۱) + ۳(۰ و ۱ و ۰) = ۲ × ۳۶ + ۳ × ۴۵ =۲۰۷

۲۷

مثال) عدد (۱۰۲ و ۹۱) را به پیمانه برده و backward انجام دهید؟ (۷ و ۸ و ۹) پیمانه

۷۲= ۸ × ۹ = (۱ و ۰ و ۰)

۷۲ × ۲ = ۱۴۴

۷۲ × ۳ = ۲۱۶

۹۱ + ۱۰۲

(۷ و ۸ و ۹) پیمانه = (۷ × ۸ × ۹)

۹۱ (۱ و ۳ و ۰)

۷۲ × ۴ = ۲۸۲

(۴ و ۶ و ۳) ۱۰۲

(۴ و ۶ و ۳) ۱۹۳

(۰ و ۰ و ۱) = ۸ × ۷ = ۵۶

۵۶ × ۲ = ۱۱۲

۵۶ × ۳ =۱۶۸

۵۶ × ۴ = ۲۲۴

۵۶ × ۵ = ۲۸۰

(باقیمانده بر ۹ عدد ۱ شود)

(۰ و ۱و ۰) = ۹ × ۷ = ۶۳

۶۳ × ۲ = ۱۲۶

۶۳ × ۳ = ۱۸۹

۶۳ × ۴ =۲۵۲

۶۳ × ۵ = ۳۱۵

۶۳ × ۶ = ۲۷۸

۶۳ × ۷ = ۴۴۱

(باقیمانده بر ۸ عدد ۱ شود)

مزایا و معایب این روش :

محاسبات در این مبنا بسیار خوب است زیرا آسان می باشد و موازی سازی داریم . وقتی موازی سازی پیش می آید به آن علاقه مند می شویم چون سرعت زیاد می شود و کاری که قبلا در ۱ ثانیه انجام میشد حالا در نیم ثانیه انجام میشود. و سرعت افزایش می یابد.

این محاسبات یکبار برای همیشه انجام می شود و همیشگی ست. پس forward خوب است و محاسبات عالی انجام میشود اما برای backward خوب نیست.

تمرین : برای backward یک روش غیر چینی ارائه دهید. (تمرین خارج از امتحان، جواب در پیوست ۱)

کاربرد:

در محاسبات کامپیوتری مورد استفاده قرار می گیرد. از این روش برای ارسال داده ها هم میتوان استفاده کرد. داده ها را کوچک کرده(RNS) و ارسال می کنیم.

کاهش توان مصرفی و کاهش خطا را داریم. به دلیل کوچک شدن در شبکه های بیسیم استفاده می شود. اما در multi casting استفاده شده است.

تمرین: اعداد منفی را چگونه به مبنای RNS میبریم؟

برای این کار باید به جای استفاده از سیستم RNS از SNS(signed residue System) استفاده کرد. در این سیستم اعداد دارای علامت هستند که برای ایجاد علامت یک بیت یا صفر می شود یا یک. برای جمع و تفریق اعداد علامت دار باید از روش زیر استفاده کرد.

برای ایجاد سیستم SNS باید ابتدا بازه M بازه اعداد را به دو قسمت تقسیم کنیم:

$$\begin{cases} X < {}^{M}/_{2} & \text{اعداد مثبت} \\ X \geq {}^{M}/_{2} & \text{اعداد منفی} \end{cases}$$

که برای بازه منفی از فرمول زیر استفاده می کنیم

$$(X - M) mod\ M = X\ mod\ M$$

برای محاسبه بازه منفی باید ابتدا Complement عدد را به صورت زیر برای X mod M محاسبه کنیم:

$$X^c = (M - X)\ mod\ M$$

حال $X = [x_i]$ و چون $[x_i] = X\ mod\ m_i$ پس در نتیجه می توان گفت که Complement عدد X می تواند Complement عدد x_i هم بشود.پس:

$$x_i^c = (m_i - x_i) mod\ m_i$$

$$[x_i]^c = [x_i^c] = X^c$$

پس عدد $-x_i$ با عدد $m_i - x_i$ برابر است

مثال: تفریق دو عدد ۸ و ۹ از هم در مبنای ***5,3,2*** که برای این کار به صورت زیر عمل می کنیم:

8[3.2,0]

9[4,0,1]

بدست آوردن complement عدد ۸

$$(8)^c = [2,1,0], 5-3\ , 3-2\ , (2-0) mod\ 2$$

برای محاسبه ۹-۸ ابتدا Complement عدد ۹ را محاسبه می کنیم

$$(9)^c = [1,0,1]$$

سپس ۸ را با عدد Complement عدد ۹ جمع می کنیم:

$$\begin{array}{rcccl} 8 & = & 8 & = & [3,2,0] \\ \underline{-9} & = & (9)^c & = & \underline{+[1,0,1]} \\ -1 & & & & [4,2,1] \quad = 29 \text{ or } -1 \end{array}$$

سپس عدد ۲۹ که بدست آمد ابتدا بررسی می کنیم که در چه بازه ای قرار دارد. اگر در بازه دوم است باید آن را از بازه بالا کم کنیم تا جواب صحیح بدست آید.به صورت زیر:

$$29 - 30 = -1$$

جواب صحیح عدد ۱- است.

۲-۱ اعداد در مبنای RTIT :

در جلسات قبل یکی از تکنیک های جدیدی که مطرح شده است جهت افزایش کارایی سیستم در حینی که سیستم کامپیوتر ما بر مشکلات قانون moor قائل شود را گفتیم. (RNS)

ما در کامپیوتر مبنای دودویی را یاد گرفتیم. که از صفر و یک تشکیل شده بود. ولی کامپیوتر ها از ابتدا صفر و یک نبودند بلکه از هزاران لامپ خلا تشکیل شده بودند و در مبنای ۱۰ بودند!!

مبناهای بزرگتر پیچیده تر هم هستند. و هوش بالاتری می خواهند. مثلا مبنای ۲ ساده تر می باشد. در حقیقت ساختن AND ,OR در آن ساده تر از مبناهای بالاتر می باشد.

به تازگی به این نتیجه رسیدند که مبنا را از ۲ کمی بالاتر ببرند. یعنی بـه جـای اینکـه فقـط ۰ و۱ باشـد، مبنای ۳ داشته باشیم، یعنی ۰ و ۱ و ۲.(Ternary Digit) .

چون ساختن قطعات بر مبنای ۳ سخت بود، پس مبنای جدیدی به نام RTit به وجـود آمـد. (۰و۱و۲). کـه البته مبنا همان ۲ می باشد و آن عدد اضافه را Redundant میگویند.

سیستم هایی که Redundant دارند، قابلیت اطمینان بالاتری هم دارند. یعنی بیت اضافی که وجـود دارد، در صورتی که بیت های اصلی خراب شوند، جایگزین می شود.

مثال:

$$(1000)_{RTit} = 0*2^0+0*2^1+0*2^2+1*2^3 = 8$$

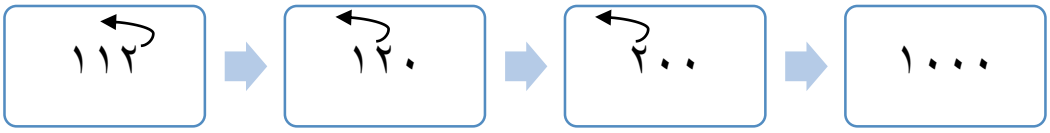
$$(200)_{RTit} = 0*2^0+0*2^1+2*2^2 = 8$$

دو مساله پیش می آید:

۱. برای عدد ۸ حالت های مختلفی به وجود می آید. و یک مزیت می باشد چون اگر خطایی رخ دهـد ما می توانیم نتیجه را به حالت های مختلف ارائه دهیم.

۲. به یک طریقی این اعداد را می توان به دست آورد. یعنی اتفاقی همه ی این ها ۸ نشدند!!

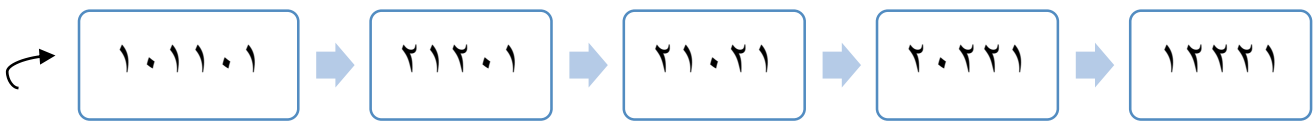
۱-۲-۱ روش بدست آوردن اعداد در RTit :



توضیح: عدد ۲ می تواند از رقم پایین تبدیل به ۱ شود. یک مرحله بالا رود.

اگر عدد ۱ از یک مرحله بخواهد پایین رود چه می شود؟ ۲ می شود. (یک مرحله بالا رود ۱ و یـک مرحلـه پایین تر رود برعکس می شود.)

مثالی دیگر:



نکته : برای تبدیل عددی از مبنای ۱۰ به RTit ، عدد باید از مبنـای ۱۰ اول بـه مبنـای ۲ تبـدیل شـود و سپس به RTit تبدیل شود. مبناهای جدید، به تبع، and , or های جدیدی هم دارند. همانطور کـه مـی دانید and در مبنای ۲ به شرح زیر است:

Binary digit		and	Or
0	0	0	**0**
0	1	0	**1**
1	0	0	**1**
1	1	1	**1**

در RTit ، مینیمم اعداد می شود and و ماکسیمم اعداد می شود or .

RTit		AND	OR
0	0	0	**0**
0	1	0	**1**
0	2	0	**2**
1	0	0	**1**
1	1	1	**1**
1	2	1	**2**
2	0	0	**2**
2	1	1	**2**
2	2	2	**2**

NOT در این مبنا به شکل زیر است:

NOT	M	N	P
0	2	2	**2**
1	0	1	**2**
2	0	0	**0**

در این جدول، P (مثبت)و N (نرمال)و M (منفی) می باشد.

NAND در این مبنا به شکل زیر است: در حقیقت ستون AND را NOT میکنیم. که برای هـر بیـت ۳ حالت NOT پیش می آید.

NAND

RTit		AND	M	N	P
0	0	0	2	2	**2**
0	1	0	2	2	**2**
0	2	0	2	2	**2**
1	0	0	2	2	**2**
1	1	1	0	1	**2**
1	2	1	0	1	**2**
2	0	0	2	2	**2**
2	1	1	0	1	**2**
2	2	2	0	0	**0**

تمرین : NOR, XOR, XNOR را در مبنای RTit بنویسید.

جواب های تمرین به شرح زیر می باشد:

NOR در این مبنا به شکل زیر است: در حقیقت ستون OR را NOT می کنیم. که برای هر بیت ۳ حالت NOT پیش می آید.

NOR

RTit		OR	M	N	P
0	0	0	2	2	**2**
0	1	1	0	1	**2**
0	2	2	0	0	**0**
1	0	1	0	1	**2**
1	1	1	0	1	**2**
1	2	2	0	0	**0**
2	0	2	0	0	**0**
2	1	2	0	0	**0**
2	2	2	0	0	**0**

فرمول XOR: $A'.B + A.B'$

A´	B´	AB´	A´B	A´B´	A XOR B
2 2 2	2 2 2	0 0 0	0 0 0	222 222 222	000 000 000
2 2 2	0 1 2	0 0 0	1 1 1	012 012 012	111 111 111
2 2 2	0 0 0	0 0 0	2 2 2	000 000 000	222 222 222
0 1 2	2 2 2	1 1 1	0 0 0	000 111 222	111 111 111
0 1 2	0 1 2	0 1 1	0 1 1	000 011 012	011 111 111
0 1 2	0 0 0	0 0 0	0 1 2	000 000 000	012 012 012
0 0 0	2 2 2	2 2 2	0 0 0	000 000 000	222 222 222
0 0 0	0 1 2	0 1 2	0 0 0	000 000 000	000 111 222
0 0 0	0 0 0	0 0 0	0 0 0	000 000 000	000 000 000

فرمول XNOR: $A'.B + A.B'$

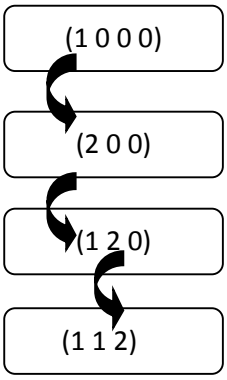
در حقیقت جدول XOR، باید NOT شود.

جدول آن به صورت زیر می باشد:

A XOR B	A XNOR B
000 000 000	222 222 222
111 111 111	012 012 012
222 222 222	000 000 000
111 111 111	000 111 222
011 111 111	111 111 112
012 012 012	111 111 111
222 222 222	000 000 000
000 111 222	011 111 111
000 000 000	222 222 222

۱-۲-۱ روش های به دست آوردن نمایش های مختلف اعداد در مبنای RTIT:

در مبنای RTIT برای یک عدد می توان نمایش های مختلفی بدست آورد. برای این کار از سمت چپ پیمانه یک ارزش پایین تر میرویم، برای مثال اگر عدد ۱ بخواهد یک ارزش پایین بیاید به عئئ ۲ تبدیل می شود، اگر عدد ۲ بخواهد یک ارزش پایین برود، چون عدد ۲ برابر با ۱+۱ است، یکی از یک هایش باقی می ماند و ۱ بعدی یک ارزش پایین می رود و به ۲ تبدیل می شود.



مثال:

۱-۳ اعداد در مبنای BR Tit (Balanced RTit)

در این قسمت مبنا به صورت زیر است: $\{-1,0,1\}_2$

اما برای ۱- ، از این نمایش استفاده می کنیم: $\{1, 0, \bar{1}\}$

جدول نمایش اعداد در مبنای BRTit :

مبنای ۱۰	مبنای BRTit	مبنای ۱۰	مبنای BRTit
7	1 1 1	-7	$\bar{1}\bar{1}\bar{1}$
6	0 1 1	-6	$\bar{1}\ \bar{1}\ 0$
5	1 0 1 1 1 $\bar{1}$	-5	$\bar{1}\ 0\ \bar{1}$ $\bar{1}\ \bar{1}\ 1$
4	1 0 0	-4	$\bar{1}\ 0\ 0$
3	0 1 1 1 0 $\bar{1}$ 1 $\bar{1}$ 1	-3	0 $\bar{1}\ \bar{1}$ $\bar{1}$ 0 1 $\bar{1}$ 1 $\bar{1}$
2	0 1 0 1 $\bar{1}$ 0	-2	0 $\bar{1}$ 0 $\bar{1}$ 1 0
1	0 0 1 0 1 $\bar{1}$ 1 $\bar{1}$ $\bar{1}$	-1	0 0 $\bar{1}$ 0 $\bar{1}$ 1 $\bar{1}$ 1 1
0	0 0 0		

تمرین: جدول ۴ بیتی BRTit را بنویسید.

جداول AND, OR, NOT برای اعداد در مبنای BRtit:

	AND	OR
$\bar{1}$ $\bar{1}$	$\bar{1}$	$\bar{1}$
0 $\bar{1}$	$\bar{1}$	0
1 $\bar{1}$	$\bar{1}$	1
$\bar{1}$ 0	$\bar{1}$	0
0 0	0	0
0 1	0	1
$\bar{1}$ 1	$\bar{1}$	1
1 0	0	1
1 1	1	1

جدول NOT:

	M	N	P
$\bar{1}$	1	1	1
0	$\bar{1}$	0	1
1	$\bar{1}$	$\bar{1}$	$\bar{1}$

فواید نمایش اعداد به صورت BRTit :

- نمایش اعداد منفی آسان تر است.
- افزونگی خواهیم داشت.
- محدوده گسترده تر با تعداد خانه کمتر نمایش داده می شود.

کاربرد ها:

- استفاده در سخت افزار
- در بعضی از ماشین های لباس شویی

تمرین: جدول XNOR , XOR و NAND در مبنای BRtit بنویسید؟

	AND	OR	NAND	NOR					XOR	XNOR
A B	**AB**	**A+B**	$\overline{AB}$	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}$B	$\overline{A}\overline{B}$	$A \oplus B$	$\overline{A \oplus B}$
			M NP	M NP	M NP	M NP	M NP	M NP		
$\bar{1}\bar{1}$	$\bar{1}$	$\bar{1}$	1 1 1	1 1 1	1 1 1	1 1 1	$\bar{1}\bar{1}\bar{1}$	111 111 111	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	111 111 111
$\bar{1}$ 0	$\bar{1}$	0	1 1 1	1 0 1	1 1 1	$\bar{1}$ 0 1	0 0 0	$\bar{1}01\bar{1}01\bar{1}01$	000 000 000	000 000 000
$\bar{1}$ 1	$\bar{1}$	1	1 1 1	$\bar{1}\bar{1}\bar{1}$	1 1 1	$\bar{1}\bar{1}\bar{1}$	1 1 1	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	111 111 111	111 111 111
0 $\bar{1}$	$\bar{1}$	0	1 1 1	$\bar{1}$ 0 1	$\bar{1}$ 0 1	1 1 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$ 000 111	000 000 000	000 000 000
0 0	0	0	$\bar{1}$ 0 1	$\bar{1}$ 0 1	$\bar{1}$ 0 1	$\bar{1}$ 0 1	$\bar{1}$ 0 0	$\bar{1}\bar{1}\bar{1}\bar{1}00$ $\bar{1}01$	$\bar{1}00$ 000 000	000 000 000
0 1	0	1	$\bar{1}$ 0 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}$ 0 1	$\bar{1}\bar{1}1$	$\bar{1}$ 0 1	$\bar{1}\bar{1}\bar{1}\bar{1}0$ $\bar{1}\bar{1}1$	$\bar{1}01\bar{1}01001$	001 001 001
1 $\bar{1}$	$\bar{1}$	1	1 1 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	1 1 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	111 111 111	111 111 111
1 0	0	1	$\bar{1}$ 0 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	$\bar{1}$ 0 1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$ 000 111	000 000 111
1 1	1	1	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	111 111 111

فصل دوم

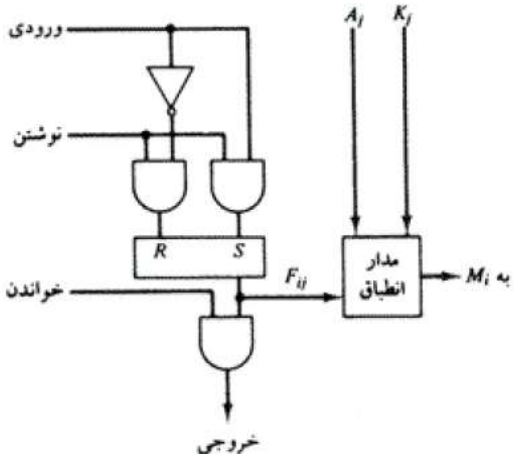
سازمان حافظه

۲-۱ حافظه تداعیگر:

سرعت بالایی دارد و سرعت جستجو در آن بالاست و به همین دلیل در حافظه های CASH از آنها استفاده می شود واشکال های آن مدارات پیچیده و بزرگ و هزینه بالای آن است

۲-۱-۲ برسی مدار سخت افزاری حافظه تداعیگر

مدار هر بیت آن بصورت زیر است:



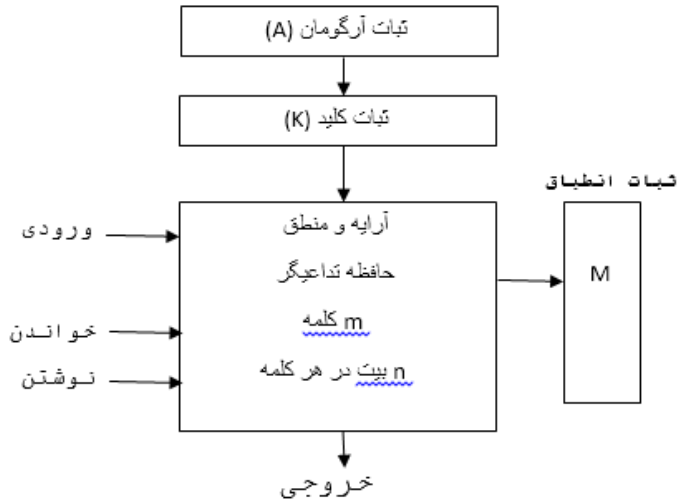
۲-۱-۲ مدار انطباق

خروجی فلیپ فلاپ بصورت زیر است:

S	R	Q
0	0	بدون تغییر
0	1	0
1	0	1
1	1	نامعتبر

برای ساختن حافظه دینامیک از خازن استفاده می کنیم و برای ساختن حافظه استاتیک از لچ و فلیپ فلاپ استفاده می شود و ساختن حافظه استاتیک از نظر منطقی آسانتر ولی هزینه طراحی حافظه دینامیک کمتر است.

ساختار سخت افزاری:



دیاگرام حافظه تداعیگر

ثبات A :n بیتی است و داخل آن داده مورد جستجو قرار می گیرد.

ثبات K : یک ثبات n بیتی است و بیت های متناظر با بیتهای این ثبات که برابر یک است در جستجو مهم هستند و بیت های متناظر این ثبات که صفر هستند جستجو نمی شوند در واقع ثبات ماسک می باشد.

مثال عددی: فرض کنید که ثبات آرگومان A و ثبات کلید K دارای آرایش بیتی زیر باشند تنها سه بیت سمت چپ A با کلمات مقایسه زیرا K در این مکان ها دارای ۱ است.

A ۱۰۱ ۱۱۱۱۰۰

K ۱۱۱ ۰۰۰۰۰۰

انطباق ندارد ۱۰۰ ۰۰۱۱۱۱ کلمه۱

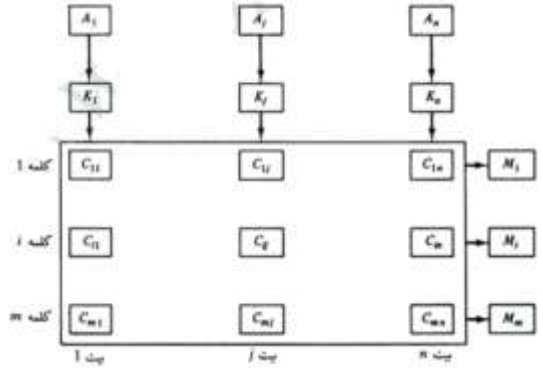
انطباق دارد ۱۰۱ ۰۰۰۰۰۱ کلمه۲

کلمه۲ با میدان آرگومان پوشش نیافته تطبیق دارد زیرا سه بیت سمت چپ آرگومان و کلمه برابرند.

نکته : در حافظه تداعیگر معمولا خطوط آدرس مطرح نمی شود.

حافظه را می توان مانند یک آپارتمان چند طبقه در نظر گرفت که هر طبقه دارای تعدادی مساوی اتاق است که هر کدام از اتاق ها بک بیت هستند اگر تعداد طبقات m و تعداد اتاق ها در هر طبقه n بیت

باشد یعنی حافظه دارای m کلمه n بیتی است و برای آن یک ثبات انطباق m بیتی در نظر می گیـریم که شکل آن بصورت زیر می باشد:



رابطه بین حافظه و ثبات های خارجی در حافظه تداعیگر در شکل بالا نشان داده شده است سلول های آرایه با حرف C و دو اندیس مشخص شده اند اولین اندیس شماره کلمه و دومین اندیس مکان بیـت را در کلمـه مشخص می کند و بیت A_j در ثبات آرگومان با همه بیت های ستون j از آرایه مقایسه می شود بشرطی که $K_j=1$ باشد و این عمل برای همه ستونهای $j=1,2,....,n$ انجام می شود. اگر بین همه بیـت هـای پوشـش نیافته آرگومان و بیت های کلمه تطابق وجود داشته باشد بیت متناظر در ثبات انطباق یعنی M_i برابر ۱ می شود و اگر یک یا چند بیت پوشش نیافته و کلمه مطابقت نداشته باشند M_i برابر صفر می شود.

ثبات انطباق بصورت زیر طراحی می شود:

$$\begin{cases} xj & \text{اگر } kj = 1 \\ 1 & \text{اگر } kj = 0 \end{cases}$$

رابطه M_i را بصورت زیر میتوان بدست آورد :

ابتدا بیت های کلید را نادیده میگیریم و آرگومان A را با بیت های ذخیره شده در سلول کلمه مقایسه می کنیم کلمه i برابر با آرگومان A است اگر به ازای $j=1,2..n$ داشته باشیم $A_j=F_{ij}$ و آن را می تـوان بصـورت زیر نشان داد:

$$X_j=A_jF_{ij}+A'_jF'_{ij}$$

X_j زمانیکه دو بیت A_jو F_{ij} برابر باشند یک می شود و درغیر اینصورت صفر خواهد شد برای اینکه کلمـه i و آرگومان A برابر باشد باید تمام X_j ها یک شود این وضعیت شرط یک شدن بیـت انطبـاق M_i اسـت یعنـی عبارت زیر:

$$M_i=X_1X_2X_3.....X_n$$

که برابر AND کردن جفت بیت های انطباق یافته است.

اکنون بیت K_j را در مدار اعمال می کنیم که بصورت زیر می باشد:

$$X_j+K'_j=$$

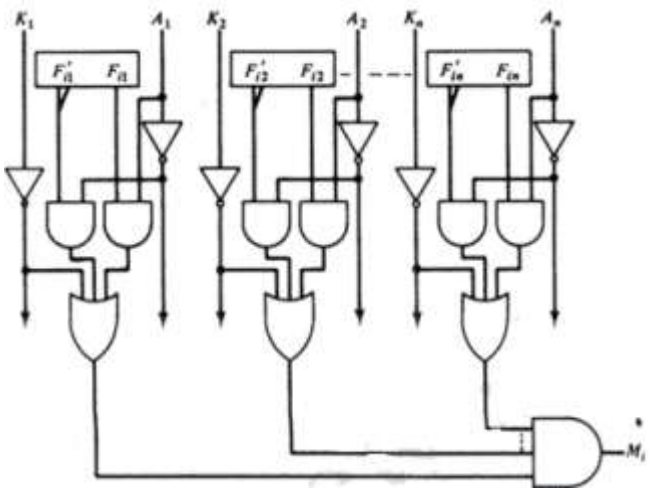
و فرمول بیت انطباق را می توان بصورت زیر نوشت:

$M_i=(x_1+K'_1)(X_2+K'_2).....(X_n + K'n)$

که با جاگذاری تابع بول آن بصورت زیر درمی آید:

$M_i=\prod_{j=1}^{n}(AjFij + A'jF'ij + K'j))$

و شکل مدار بصورت زیر است:



قیمت حافظه تداعیگر بیشتر از حافظه معمولی است و سرعت جستجوی آن بالاتر است

۲-۲ حافظه Cache

چون سرعت حافظه اصلی نسبت به سرعت CPU پایین تر است از یک حافظه مابین حافظه اصلی و CPU بنام کش قرار می گیرد که سرعت بالایی دارد و هزینه ساخت آن نیز بالا است بنابراین ظرفیت آن پایین و تنها بلوک هایی از حافظه اصلی درون کش قرار می گیرد و CPU در صورت عدم وجود در کش به حافظه اصلی مراجعه می کند و وجود کش به دو دلیل محلیت مکانی و محلیت زمانی است.

مبانی Caching

Caching یک تکنولوژی استفاده شده برای زیر سیستم های حافظه، در کامپیوتر است. مهمترین هدف یک Cache افزایش سرعت و عملکرد کامپیوتر بدون تحمیل هزینه های اضافی برای تهیه سیستم است. با استفاده از Cache عملیات کاربران با سرعت بیشتری انجام خواهد شد.

وقتی پردازنده نیاز دارد که داده‌ای را بخواند یا بنویسد، ابتدا چک می‌کند که در Cache موجود است یا نه، این کار به وسیله مقایسه آدرس مکان حافظه با همه تگ‌های موجود در Cache ((که ممکن است حاوی آدرس باشد)) صورت می‌پذیرد.

اگر پردازنده آدرس مکان مورد نظر حافظه را در Cache بیابد، می‌گوییم کـه یـک "برخـورد Cache" رخ داده، در غیر این صورت گوییم که یک "خطای Cache" روی داده است. در صـورت برخـورد پردازنـده بـه سرعت داده‌ها را از خط Cache می‌خواند یا می‌نویسد. نسبتی از دسترسی‌ها که منجر به برخورد می‌شود را "نرخ برخورد" گویند، که مقیاسی برای اندازه‌گیری کارایی یک الگوریتم یا برنامه می باشد. در صـورت بـروز خطا، Cache مدخلی دیگر را در نظر می‌گیرد.

ساختمان مدخل Cache:

مداخل سطری Cache معمولاً ساختاری این چنینی دارند:

Valid Bit	Data Blocks	Tag

بلوک‌های داده حاوی داده‌های واکشی شده از حافظه اصلی می‌باشند. بیت اعتبار مشخص می‌کند که مدخل مذکور، داده معتبر دارد یا نه.

دو مثال ساده:

فرض کنیم که شما هر روز به رستوران می روید. هر روز راس ساعت ۵ بعد از ظهر سفارش غذا مـی دهیـد. هر روز ۴ نوع غذا را به ترتیب خاص سفارش می دهید. راس ساعت ۵ همبرگر سفارش می دهیـد، گارسـون سفارش شما رو بررسی می کند، به آشپزخونه می رود، بعد از یک دقیقه همبرگر را برای شما می آورد، شما همبرگر را خورده و سفارش سوسیس می دهید. مجدداً سفارش توسط گارسون به آشپزخانه منتقل شـده، و بعد از یک دقیقه غذا آماده می شود. به همین ترتیب شما ۳ غذای دیگه را سفارش داده، و برای هـر غـذا ۱ دقیقه معطل می شوید. خوب شما هرروز همین غذاها را سفارش داده، و برای آماده شدن هر غـذا ۱ دقیقـه معطل می شوید. گارسون با خودش فکر می کند که برای اینکه هم خودش کمتر کار کند و هم شما کمتـر معطل شوید، ۱ میز آماده دیگر از غذاهای شما را تهیه کنـد، و بلافاصـله بعـد از سـفارش شـما غـذا را روی میزتان قرار دهد. در اینجا گارسون "Bus"، آشپزخانه "Ram"، و میز آماده "Cache" در نظر گرفته مـی شوند. بعد از چند روز شما همبرگر را می خورید، طبق عادت گارسون برای شما سوسیس می آورد، اما شما می گویید که امروز پیتزا می خواهم، اینجا گارسون مجددا مجبور می شود که ۱ دقیقه شما را در انتظار نگه دارد، تا پیتزا را برایتان بیاورد. در اینجا گارسون میز دومی را تهیه می کند که بر اسـاس انتخـاب هـای دوم شما چیده شده است. بدین ترتیب شما اگر غذایی را سفارش دهید که در میز اول نباشـد، امـا در میـز دوم باشد بلافاصله غذا را میل می کنید و معطل نمی شوید. میز دوم در اینجا Cache سـطح دو (Cache L2) است.

کتابداری را در نظر بگیرید که در یک کتابخانه مسئول تحویل کتاب بـه متقاضـیان اسـت. فـرض کنیـد در سیستم فوق ((درخواست و تحویل کتاب)) از مفهوم Cache استفاده نمی گردد. اولـین متقاضـی کتـابی را

درخواست می نماید ((فرض شده است که متقاضی خود نمی تواند مستقیما کتاب مورد نظر را از قفسه مربوطه بردارد))، کتابدار، کتاب مورد نظر را از قفسه مربوطه پیدا، و آن را به متقاضی تحویل می دهد. متقاضی پس از ساعاتی مراجعه، و کتاب را تحویل می دهد. کتابدار، کتاب تحویلی را مجددا در قفسه مربوطه قرار می دهد. پس از لحظاتی یک متقاضی دیگر مراجعه و همان کتاب قبلی را درخواست می نماید، کتابدار مجددا می بایست به بخش مربوطه در کتابخانه مراجعه، و پس از بازیابی کتاب، آن را در اختیار متقاضی دوم قرار دهد. همانگونه که ملاحظه می گردد، کتابدار مکلف است برای تحویل هر کتاب (حتی کتاب هائی که فرکانس استفاده از آنان توسط متقاضیان زیاد باشد)، به بخش مربوطه مراجعه، و پس از یافتن کتاب آن را در اختیار متقاضیان قرار دهد. فرض کنید بخشی را با ظرفیت حداکثر ده کتاب در مجاورت (نزدیکی) کتابدار آماده نمائیم. کتاب هایی که توسط متقاضیان برگردانده می شود، در بخش فوق ذخیره خواهند شد. مثال فوق را با در نظر گرفتن سیستم Cache ایجاد شده برای کتابدار، مجددا دنبال می نمائیم. در ابتدای فعالیت روزانه، بخش Cache خالی بوده و هنوز در آن کتابی قرار نگرفته است. اولین متقاصی مراجعه، و کتابی را درخواست می نماید. کتابدار می بایست به بخش مربوطه مراجعه، و کتاب را از قفسه مربوطه برداشته، و در اختیار متقاضی قرار دهد. متقاضی پس از تحویل کتاب، چند ساعت بعد مراجعه، و کتاب را تحویل کتابدار خواهد داد. کتابدار، کتاب تحویلی را در بخش پیش بینی شده، برای Cacheقرار می دهد. لحظاتی بعد متقاضی دیگر مراجعه، و درخواست همان کتاب را می نماید.کتابدار در ابتدا بخش مربوط به Cache را جستجو، و در صورت یافتن کتاب، آن را به متقاضی تحویل خواهد داد. در این حالت ضرورتی برای مراجعه کتابدار به بخش و قفسه های مربوطه نخواهد بود. در روش فوق زمان تحویل کتاب به متقاضی، بهبود چشمگیری پیدا خواهد کرد. در صورتی که کتاب درخواستی توسط متقاضی، در بخش Cache کتابخانه نباشد، چه اتفاقی خواهد افتاد؟ در ابتدا مدت زمانی صرف خواهد شد که کتابدار به این اطمینان برسد که کتاب درخواستی در بخش Cache موجود نمی باشد (جستجو)، یکی از چالش های اصلی در رابطه با طراحیCache ، به حداقل رساندن زمان جستجو در Cache است. سخت افزارهای جدید، زمان فوق را به صفر نزدیک کرده اند. پس از حصول اطمینان از عدم وجود کتاب در بخش Cache، کتابدار می بایست با مراجعه به بخش مربوطه آن را انتخاب و در ادامه در اختیار متقاضی قرار دهد.

با توجه به دو مثال فوق، چندین نکته مهم در رابطه با Cache استنباط می گردد:

تکنولوژیCache ، استفاده از حافظه های سریع ولی کوچک، بمنظور افزایش سرعت یک حافظه کند، ولی با حجم بالا است.

زمانیکه از Cache استفاده می گردد، در ابتدا می بایست محتویات آن به منظور یافتن اطلاعات مورد نظر بررسی گردد. فرآیند فوق را Cache Hit می گویند. در صورتی که اطلاعات مورد نظر در Cache موجود

نباشند (Cache miss)، کامپیوتر می بایست در انتظار تامین داده های خود از حافظه اصلی سیستم (حافظه ای کند ولی با حجم بالا) باشد.

اندازه Cache محدود بوده وسعی می گردد که ظرفیت فوق حتی المقدور زیاد باشد، ولی به هرحال اندازه آن نسبت به رسانه های ذخیره سازی دیگر بسیار کم است.

این امکان وجود خواهد داشت که از چندین لایه Cache استفاده گردد.

۱-۲-۲ محلیت زمانی و مکانی حافظه کش

همانطور که قبلا گفته شده دو دلیل برای استفاده از حافظه وجود دارد؛ محلیت زمانی و محلیت مکانی.

۱) محلیت زمانی (Temporal locality):

داده ها و دستوراتی که اخیرا مورد استفاده قرار گرفته اند، مجددا مورد رجوع قرار خواهند گرفت.

۲) محلیت مکانی (Spatial locality):

داده ها یا دستوراتی که در خانه های نزدیک به هم ((در حافظه)) قرار دارند، با فاصله زمانی اندکی از هم مورد رجوع قرار خواهند گرفت.

۲-۲-۲ کارایی cache

کارایی حافظه cache به این جمله بستگی دارد :

پیش بینی محلیت زمانی و مکانی که ما داشتیم، چقدر موفق بوده است ؟؟؟

حالا مولفه های زیر را جهت محاسبه کارایی تعریف میکنیم:

متوسط زمان خواندن از حافظه Cache = T_{Avg}

متوسط زمان خواندن از Cache = T_{Cache}

متوسط زمان خواندن از = T_{RAM}

احتمال اینکه داده در داخل کاشه باشد.= (Hit ratio) = P

احتمال اینکه داده در داخل کاشه نباشد.= (Miss ratio) = (1-P)

در ضمن :

$T_{RAM} > T_{Cache}$

زیرا حافظه cache سریع تر از RAM می باشد.

$$T_{Avg} = P\ T_{Cache} + (1-p)(T_{RAM} + T_{Chache})$$

اول به سراغ Cache می رود، اگر داده مورد نظر آن جا بود که می خواند و اگر که نبود، به حافظه RAM مراجعه و از آنجا می خواند و به Cache می دهد و دوباره از آن می خواند. حالا چرا دوباره داخل Cache می گذارد؟؟؟ به دلیل محلیت زمانی و محلیت مکانی.

$$P\ T_{Cache} + T_{RAM} + T_{Cache} - P\ T_{RAM} - P\ T_{Cache} = T_{Cache} + (1-P)\ T_{RAM}$$

اگر P مقدارش کم باشد، یعنی به صفر نزدیک تر باشد، $T_{Cache} + (1-P)\ T_{RAM}$ که در بالا بدست آوردیم، بالا میرود. و اگر P مقدارش بالا رود یعنی به یک نزدیک شود، مقدار تاثیر T_{RAM} کم می شود.

نکته) P به اندازه Cache و RAM و اندازه بلوک ها و تکنیک های نگاشت بستگی دارد.

حالا اینکه سیستم Cache داشته باشد یا نداشته باشد، به مقدار P بستگی دارد.

پس حافظه Cache سرعت بالایی دارد. خیلی از قسمت ها در سیستم بافر وجود دارد. اگر قرار باشد که حافظه در داخل خودش فقط اطلاعات را نگه داری کند، همان بافر کافیست! و اما اگر قرار باشد که جستجویی انجام شود ، Cache و حافظه تداعی گر نیاز است. در حقیقت حافظه تداعی گر از Cache استفاده می کند. اما بافر در دل خود از حافظه RAM استفاده می کند. حافظه Cache نیاز دارد که روی خودش هم جستجو انجام شود. بافر یک حافظه میان گیر می باشد. و با سخت افزار بافر متفاوت می باشد. جستجو را انجام نمی دهد. مثل Cache نیست که چیزهای مهم در درون آن ذخیره شوند. حتی توی CPU هم ممکن است وجود داشته باشد.

CPU که می خواهد با RAM ارتباط برقرار کند، همیشه یک کپی از قسمت هایی که مد نظرش هست را به دلیل محلیت زمانی و مکانی در داخل Cache قرار می دهد و اگر این کار به خوبی و درستی انجام شود، سرعت بالا می رود.سوالی که اینجا پیش می آید این است که این عملیات چگونه انجام می شود؟

جواب : با استفاده از نگاشت ها ... که در ادامه با ۳ نوع نگاشت آشنا خواهیم شد ....

۳-۲ نگاشت

انواع نگاشت یا mapping از حافظه اصلی به cashe :

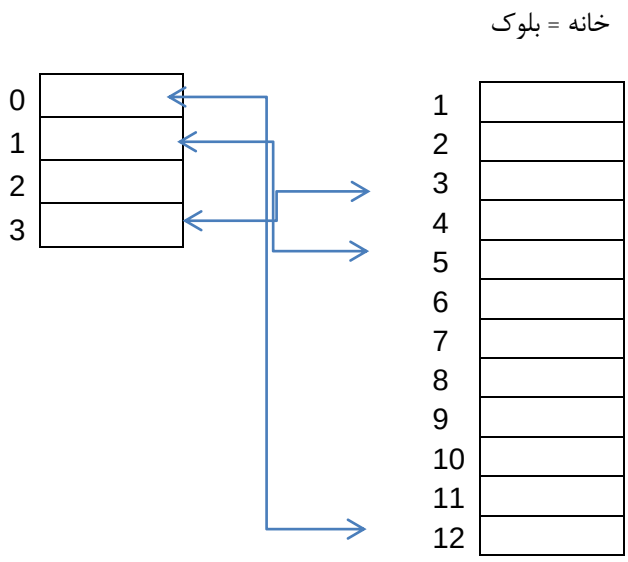
- DIRECT Mapping یا نگاشت مستقیم
- ASSOCIATIVE Mapping یا نگاشت تداعیگر
- SET ASSOSIATIVE Mapping یا نگاشت تداعیگر مجموعه ای

۱-۳-۲ نگاشت مستقیم

حافظه تداعیگر در مقایسه با حافظه های دستیابی تصادفی بدلیل مدار اضافی همراه هر سلول گرانتر است در نگاشت مستقیم، جایی که بلوک باید در آن قرار گیرد مشخص است، و در روش های اشتراکی بلوک در چند محل متفاوت می تواند قرار گیرددر نگاشت مستقیم، جایی که بلوک باید در آنقرار گیرد مشخص است در روش های اشتراکی بلوک در چند محل متفاوت می تواند قرار گیرد در درجه ی اول مکانی انتخاب می شود که بیت اعتبارآن غیر فعال است. در غیر این صورت از بلوکی که کم تر مورد استفاده قرارگرفته است، از حافظی نهان خارج می شود.

فرمول نگاشت مستقیم :

شماره بلوک کاشه = تعداد بلوک کاشه mod شماره بلوک حافظه : DIRECT Mapping



شماره ۵ حافظه را بخواهیم ببریم داخل کش ۱= ۴mod ۵ پس باید داخل خانه ۱ کش برود.

شماره ۱۲ حافظه را بخواهیم ببریم داخل کش ۰= ۴mod ۱۲ پس باید داخل خانه ۰ کش برود.

شماره ۳حافظه را بخواهیم ببریم داخل کش ۳= ۴mod ۳ پس باید داخل خانه ۳ کش برود.

شماره ۹ حافظه را بخواهیم ببریم داخل کش ۱= ۴mod ۹ پس باید داخل خانه ۱ کش برود، ولی خانه۱ پر است در اینجا باید خانه یکم را write back کنیم ولی اگر در داخل کش محتوای خانه یکم تغییر کرده و با حافظه اصلی نابرابر است باید Write backکنیم اما اگر مقدار داخل کش تغییر نکرده و با حافظه یکی است دیگر به write back نیازی نیست پاکش می کنیم.

حال اگر فرض کنیم برنامه ما قسمتی در خانه ۱ قسمتی در خانه ۵ و قسمتی در خانه ۹ می باشد خانه ۱ را باید در خانه یکم کش بگذارد بلوک ۵ را هم در خانه ۱ باید بگذارد پس باید محتوای خانه ۱ را برگرداند بلوک نهم را در خانه یکم باید بگذارد باز باید محتوای خانه یک را برگرداند یعنی راندمان پایین می آید چون از کش درست استفاده نکردیم چون با direct mapping می گوییم حتما باید در یک خانه خاصی باشد یعنی با direct mapping امکانش هست که ازکش درست بهره برداری نشود.

مشکل direct mapping: cashe هنوز فضای خالی دارد ولی باید از یک سری خانه ها بیشتر استفاه شود و همین با عست می شود p (بهره وری)پایین باید یعنی برخی از خانه ها خالی هستند در حالی که برخی چندین بار خالی و پر می شود .

مزیت direct mapping : جستجو در خانه حافظه کش راحت تر است زیرا لازم نیست تمام کش جستجو شود مثلا اگر به بلوک ۱۴ نیاز داریم فقط بلوک ۶ از کش جستجو می شود (۱۴mod۸=۶) و از لحاظ پیاده سازی راحت تر است.

۲-۳-۲ نگاشت تداعیگر

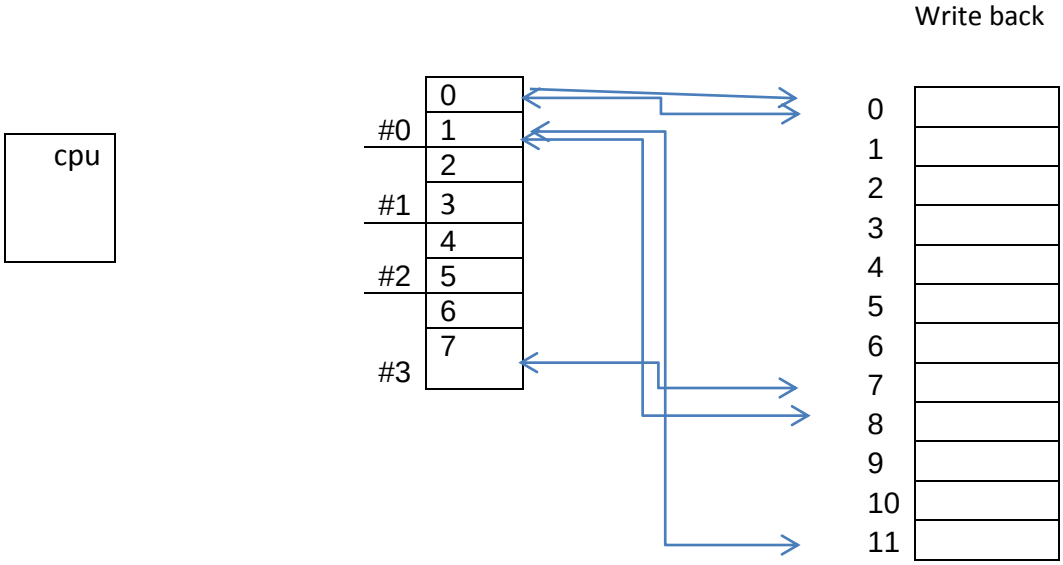
گفتیم باید کل حافظه جستجو شود توی این روش هم شاخص را باید در کش ذخیره کنیم و هم نشانه رابابد ذخیره کنیم یعنی در هرکجای کش که خواست بنویسیدو برای پیدا کردن داده باید کل حافظه جستجو شود در این روش هم شاخص را باید در کش ذخیره کنیم و هم نشانه را باید ذخیره کنیم یعنی تمام ست های آدرس در داخل حافظه کش ذخیره می گردد(هر جایی از حافظه می تواند دخیره شود) در یک قسمت داده و در قسمت دیگر آدرس و وقتی کش کامل پر شود باید write back کنیم و برای write back می توانیم از الگوریتم های مختلف مثل fifo و غیره استفاده کنیم.

مزیت : استفاده بهینه از حافظه کش

مشکل : چون باید کل خانه های کش جستجو شود سرعت جستجو پایین می آید .

۳-۳-۲ نگاشت تداعیگر مجموعه ای

در این روش کش مجموعه مجموعه می شود برای مجموعه بندی باید تعداد بلوک های همه مجموعه ها یکسان باشد.



حافظه کش همیشه باید ۲ به توان x باشد

شماره مجموعه کش = تعداد مجموعه ها mod شماره بلوک حافظه

۷ ⟶ #3

۷ ⟶ #3

۱۲ ⟶ #0

این روش میانه است باید ببینیم کدام بهتر می شود، اگر اندازه مجموعه بزرگ باشد یعنی کل کش یک مجموعه باشد به associative نزدیک می شود و دارای معایب و مزایای associative است. اگر اندازه مجموعه کوچک باشد به روش direct نزدیک می شود و به معایب و مزایای direct است. سرعت جستجوی این روش میانه این دو روش است .

مثال) حافظه ما یک حافظه 32m دارد و داده ها ۱۶ بیتی است یعنی ۳۲*۱۶ است

این کامپیوتر 1 kb کش دارد از روش نگاشت مستقیم استفاده می کنیم.

۱) مکانیسم عملکرد سیستم را توضیح دهید.

۲) هر بیت نشانه به چند شاخص و هر طبقه کش چقدر بیت نیاز دارد.

در هر طبقه ۱۶ بیت $=2^{25}32*2^{20}$

۲۵ خط آدرس

هر طبقه ۳۱ بیت دارد ۱۰ شاخص

۱۵ بیت برای نشانه ۱۵ نشانه

۱۶ بیت برا ذخیره داده

در تداعی گر هم شاخص و هم نشانه را باید ذخیره کنیم

هم آدرس و هم داده کامل ذخیره می شود.

MIPS دومی بیشتر است.

۲-۴ خط لوله (Pipeline)

۲-۴-۱ روش ترتیبی(بدون خط لوله)

بطور مثال اجرای هر دستور اسمبلی در یک برنامه دارای ۴ مرحله می باشد (تعداد مراحل بستگی به طراحی سخت افزار دارد)

مرحله اول Fetch یا واکشی یک دستور از حافظه و قرار دادن درون ثبات است. دومین مرحله از اجرای هر دستور Decode یا خواندن و باز کردن دستور است. سومین مرحله Execute یا همان اجرای کد است. و آخرین مرحله WriteBack یا بازنویسی نتایج اجرای دستور در حافظه اصلی است.

طبق دید معمولی در روال عادی پردازنده تمام این مراحل برای دستور اول انجام می شد و بعد تمام مراحل برای دستور دوم و همینطور دستور سوم و ...

F_1	D_1	Ex_1	WB_1	F_2	D_2	Ex_2	WB_2 .....

اگر زمان اجرای هر دستور را T_n و زمان اجرای هر مرحله از دستور را T_κ در نظر بگیریم در نتیجه زمان اجرای هر دستور برابر است با kT_κ

(در اینجا تعداد مراحل ۴ در نظر گرفته شده است یعنی k=4) و زمان اجرای n دستور برابر است با nkT_K

۲-۴-۲ روش با خط لوله

پایپلاین کردن یا pipelining تکنیکی است که در آن اجرای چند دستور باهم همپوشانی پیدا می‌کنند . امروزه پایپلاین کردن یک امر حیاتی برای افزایش سرعت پردازنده‌ها به‌حساب می‌آید. در این بخش برای توضیح اصطلاحات و مفاهیم پایپلاین کردن از مقایسه با دنیای بیرون استفاده خواهیم کرد.

کسانی که در اتاق رختشوئی کار میکنند و تعداد زیادی لباس را شستشو می کنند، به طور ناخودآگاه از تکنیک پایپلاین کردن استفاده می کنند .یک روش غیر پایپلاین برای شستن این تعداد لباس به صورت زیر است:

۱ .تعدادی لباس کثیف را داخل شوینده قرار (washer) دهید .

۲ .بعد از تمام شدن عملیات شستن، لباس های خیس را داخل خشک کننده (dryer) قرار دهید.

۳ .وقتی که کار خشک کننده تمام شد، لباسهای خشک را بر روی یک میز قرار داده و تا کنید.

۴ .وقتی که کار تا کردن تمام شد، لباس های تا شده را داخل کمد قرار دهید و یا از همکارتان بخواهید که آنها را بیرون ببرد.

وقتی که لباس های تا شده را داخل کمد قرار دادید، دوباره یک کار جدید را با شستن دسته دیگری از لباس های کثیف، شروع کنید.

در پردازنده‌ها نیز اجرای دستورات می‌تواند به‌صورت پایپلاین شده انجام شود. به این صورت که به علت بهره‌وری پردازنده و بیکار نماندن آن در زمانه‌ای بین اجرای هر مرحله از اجرای دستور ، هر مرحله بعد از اتمام کار به سراغ دستور بعد رفته و مرحله مربوط به خود را در دستور بعد اجرا کند.

یعنی مرحله Fetch پس از اتمام کار خود به سراغ دستور بعد رفته و دستور بعد را Fetch کند. همین‌طور مرحله Decode و مراحل Execute و WriteBack .

F_1	D_1	Ex_1	WB_1			
	F_2	D_2	Ex_2	WB_2		
		F_3	D_3	Ex_3	WB_3	
			F_4	D_4	Ex_4	WB_4

در اینجا همانند روش ترتیبی دستورات پشت سر هم اجرا می‌شوند ولی از مکانیزم دیگری برای این کار استفاده می‌شود.

در این حالت زمان اجرای n دستور برابر است با kTк + (n-1) Tк چون از دستور دوم به بعد به Tк زمان نیاز است.

مقایسه کارایی در دو حالت فوق:

ما می‌توانیم میزان افزایش سرعت پردازنده پایپلاین شده به پردازنده غیر پایپلاین را که در بالا صحبت کردیم در قالب فرمول بیان کنیم .اگر همه مراحل کاملاً بالانس باشند (تأخیرشان یکسان باشد)، در این صورت زمان بین دستورات در پردازنده پایپلاین شده (با فرض شرایط ایدآل) برابر خواهد بود با:

$$\frac{\text{کارایی با خط لوله}}{\text{کارایی بدون خط لوله}} = \frac{\text{زمان اجرای بدون خط لوله}}{\text{زمان اجرای با خط لوله}} = \frac{nkT\text{к}}{[k+(n-1)]T\text{к}} = \frac{nk}{k+n-1}$$

$$= \lim_{n\to\infty} \frac{nk}{k+n-1} = k$$

در شرایط ایده آل میزان افزایش سرعت پایپلاین کردن مساوی تعداد مراحل پایپلاین خواهد بود.

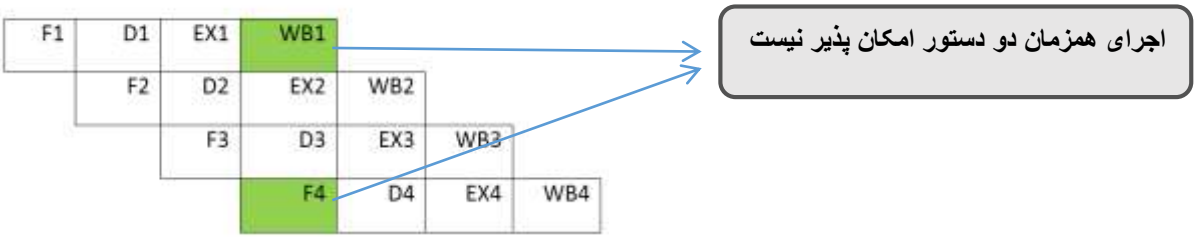
طبق مقایسه بالا کارایی سیستم در حالت دوم k برابر بیشتر از حالت اول است. یعنی دستورات k برابر سریع‌تر انجام می‌شوند.

در اینجا چون k را ۴ فرض کردیم پس سرعت انجام دستورات ۴ برابر سریع‌تر انجام می‌شوند.

۳-۴-۲ Hazard های خط لوله

هر چقدر K بیشتر شود هازارد هم بیشتر می شود و هر کدام مشکلات و مسائل خاص خودش را دارد و باعث می شود سرعت ما K برابر نشود و کمتر شود. و وقتی هازارد بیشتر شود مشکلات هم بیشتر می شود. پس عملاً اولین مسئله این است که هر چقدر K را بیشتر کنیم افزایش کارایی مان بیشتر می شود ولی به مثابه آن هازاردها هم بیشتر می شود.

اولین hazard در خط لوله:



برای مثال زمانی که ما fetch می کنیم یعنی دستوراتی را می خواهیم از حافظه واکشی کنیم، اگر همان زمان دستورwb (یعنی نتیجه ی اجرای دستورات را روی حافظه بنویسیم) بخواهد انجام شود یعنی همان زمان یک دستور می خواهد بخواند و یک دستوردیگرمی خواهد بنویسد که این امکان برای حافظه های معمولی امکان پذیر نیست.

به طور مثال در شکل زیر هر دو دستور wb1 وf4 می خواهند هم زمان اجرا شوند اما چون f4 باید بخواند و wb1 می خواهد بنویسد امکان خواندن و نوشتن هم زمان روی حافظه معمولی نیست که به این مشکلHAZARD می گویند.

این هازارد ها باعث می شود کارایی خط لوله به اون حد K برابری خودش نرسد و هر چه تعداد مراحل را هم بیشتر کنیم این هازارد ها هم بیشتر می شوند

راه حل : استفاده از حافظه های DUALPORT که هم زمان هم می توان خواند و هم می توان نوشت در این صورت مشکل هازارد اول حل می شود اما این نوع حافظه گران تر است.

دومین hazard خط لوله :

اگر فرض کنیم دستور اول، دستور jump باشد(یعنی پرش کردن به نقطه ای) زمانی که مرحله ی اول fetch است در این مرحله یعنی fetch کردن مشخص نمی شود که دستور اول jump است چون فقط وظیفه واکشی دارد مثلا اگر دستور اول پرش به دستور پنجم باشد ،در زمان واکشی دستور اول این مشخص نمی شود پس بنابراین وقتی می خواهد دستور بعدی fetch شود دستور دوم یعنی add واکشی می کند و دستور ۵ را fetch نمی کند .

Ex:
دستورات Jump (پرش به دستور پنجم)
Add
Sub
Mul
دستور پنجم ۵:

در مرحله بعد یعنی مرحله ی اجرا هنوز هم هیچ دیدگاهی نسبت به نوع دستور وجود ندارد فقط زمانی که اجرا تمام شود نوع دستور مشخص می شود پس زمانی که اجرای دستور اول تمام می شود تازه متوجه می شود که دستور اول jump بوده و دو دستور بعدی ، بی خودی fetch شده اند ونیاز به واکشی کردن آنها نبوده است و این خودش یک hazard است.

راه حل :

یک مدار به نام prefetch وجود دارد که کارش بررسی دستورات است مثلا در مثال بالا بررسی می کند ببیند دستور jamp است یا نه اما ممکن است باعث افزایش هزینه یا افزایش زمان اجرا شوند.

تمرین: چهار مورد از هازادهای (مخاطره های) خط لوله را بنویسید ؟

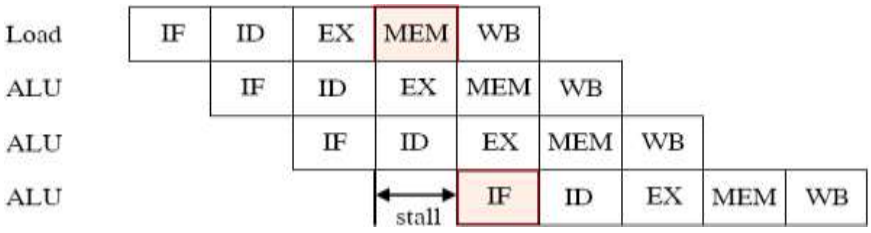
در معماری‌های خط لوله بزرگترین مانع وجود موانع یا در اصطلاح stall های خط لوله است. مانع هنگامی رخ می دهد که مجموعه جدید ورودی را نتوان در خط لوله بواسطه شرایطی به نام مخاطره یا hazard به اجرا در آورد. هنسی و پاترسون مخاطرات را به سه دسته تقسیم می کنند:

۱. مخاطرات ساختاری، بدین واسطه رخ می دهند که منابع ماشین نتوانند کلیه ترکیبهای همپوشانی دستورالعمل را پشتیبانی کنند.

۲. مخاطرات داده ای، هنگامی که نتیجه حاصل از یک دستورالعمل مورد نیاز دستورالعمل بعدی باشد.

۳. مخاطرات کنترلی، هنگام پردازش دستورالعملهای انشعاب ایجاد می شود.

مخاطره ساختاری می تواند در صورت عدم وجود منبع رخ دهد. برای نمونه اگر ماشینی صرفاً یک درگاه برای حافظه داشته باشد و نتوان داده و دستورالعمل را به طور همزمان از حافظه خواند این مشکل می تواند رخ دهد. برای نمونه به شکل ذیل توجه نمایید.



شکل: شکل هازارد ساختاری در درگاه حافظه ماشین های DLX

همانگونه که در بالا مشخص است نمی توان چهارمین fetch از حافظه را انجام داد زیرا دستورالعمل اولی در این زمان نیاز به دسترسی به حافظه دارد. برای ممانعت از مواجه شدن با همچنین وضعیتی در هنگام تولید کد سعی به استفاده بیشتر از ثباتها می شود. برای نمونه به دستورالعملهای ذیل توجه نمایید:

ADD R1,R2,R3

SUB R4,R1,R5

این دو دستورالعمل بدون هیچگونه تاخیری قابل اجرا هستند زیرا نتیجه دستورالعمل اول در دومی بلافاصله استفاده می شود. اما، در مورد دو دستورالعمل زیر نمی توان از ایجاد مانع یـا در اصـطلاح stall جلـوگیری نمود.

LW R1,0(R2)

ADD R3,R1,R4

علت بوجود آمدن مانع در شکل ذیل مشخص شده است:

LW R1,0(R2)	IF	ID	EX	MEM	WB		
ADD R3,R1,R4		IF	ID	stall	EX	MEM	WB

شکل: هازاد داده ای

می توان با اسـتفاده از امکـان زمانبنـدی دسـتورالعملها یـا در اصـطلاح Instruction Scheduling یـا Instruction Reordering این مشکل را حل نمود. برای این منظور با در نظـر گـرفتن وابسـتگی هـای داده ای و کنترلی امکان جابجایی دستورالعملهایی که ثبـات R1 را اسـتفاده نمـی کننـد و مشـکل stall را ایجاد نمی کنند، را به میان دو دستورالعمل فوق بررسی باید نمود. در صورت یافتن دستورالعمل مناسب آنرا به بین دو دستورالعمل می توان انتقال داد.

مشکل فوق در اجرای عبارتی مثل A+B+C نیز امکان پذیر است. برای نمونه به شکل ذیل توجه نمایید. در اینجا چنانچه اجرا از چپ به راست انجام شود دستورالعمل جمع دومـی مـی بایسـت بـرای یـک سـیکل در انتضار اجرا باقی بماند.

ADDD R3,R1,R2	IF	ID	EX1	EX2	MEM	WB		
ADDD R3,R3,R4		IF	ID	stall	EX1	EX2	MEM	WB

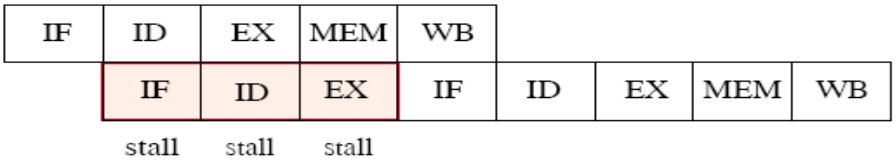
شکل : هازارد داده ای ایجاد کننده مانع

بازهم می توان گفت که زمانبند کامپایلر می تواند مانع این مشکل شود. برای نمونه فرض کنید که عبارتی برای جمع چهار متغیر به صورت A+B+C+D قرار است اجرا شود. در اینجا بعد از هر عمل جمع بـه تعـداد دو سیکل انتظار برای دریافت نتیجه دستورالعمل قبلی نیاز است. برای رفع مشکل کامپایلر می توانـد عمـل جمع را به صورت ذیل گروه بندی کند:

(A+B) + (C+D)

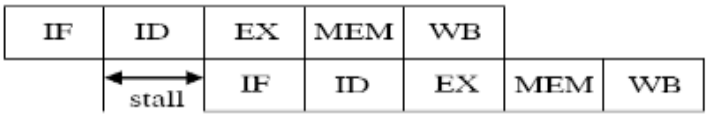
در این صورت جمع دومی می توان در یک سیکل بعد از جمع اولی به اجرا در آید. در مجموع یک سیکل مجموع سریعتر بدست می آید.

مخاطرات کنترلی همانطور که در بالا توضیح داده شد بواسطه دستورالعملهای پرش ایجادمی شوند. در ماشینهای DLX مخاطرات کنترلی می تواند مانعی به طول سه سیکل زمانی ایجاد نماید. پردازنده شروع به آوردن دستورالعمل بعدی از حافظه می کند بدون اطلاع از اینکه پرش انجام می شود. تشخیص عمل پرش بعد از مرحله MEM از دستورالعمل پرش مشخص می شود. در این صورت دستورالعمل بعدی می بایست از درون حافظه بیرون کشیده شود قبل از اینکه هر گونه تغییری در ثباتها و یا حافظه ایجاد شود. برای نمونه به شکل ذیل توجه نمایید.



شکل هازارد ایجاد شده بواسطه دستورالعمل پرش شرطی

از آنجایی که سه سیکل پنالتی زیادی می باشد، طراحان ماشین سعی زیادی به کاهش این هزینه نموده اند. یک راه، استفاده از سخت افزاری است که در ضمن دیکد دستورالعمل بتواند تشخیص پرش را بدهد. این عمل در صورت آزمون مقایسه با صفر امکان پذیر است. در این صورت stall به یک عدد به صورت زیر تقلیل داده می شود:



شکل تقلیل هازارد ایجاد شده بواسطه دستورالعمل پرش شرطی

۲-۴-۴ انواع پیاده سازی های خط لوله:

۲-۴-۴-۱ روش سنکرون

برای پیاده سازی این نوع از خط لوله، از یک قطعه ای به نام latch استفاده می کنیم. زمانی که پایه لود مدار فعال باشد، latch ورودی ها را در خروجی قرار می دهد و آن را نگه می دارد. در این سیستم پایه لود (load) و چیزی به اسم clock عمل همزمان سازی را انجام می دهند. پس همان طور که گفتیم زمانی که پایه لود فعال می شود، latch ها هرچه در ورودی خود دارند در خروجی نمایش می دهند.

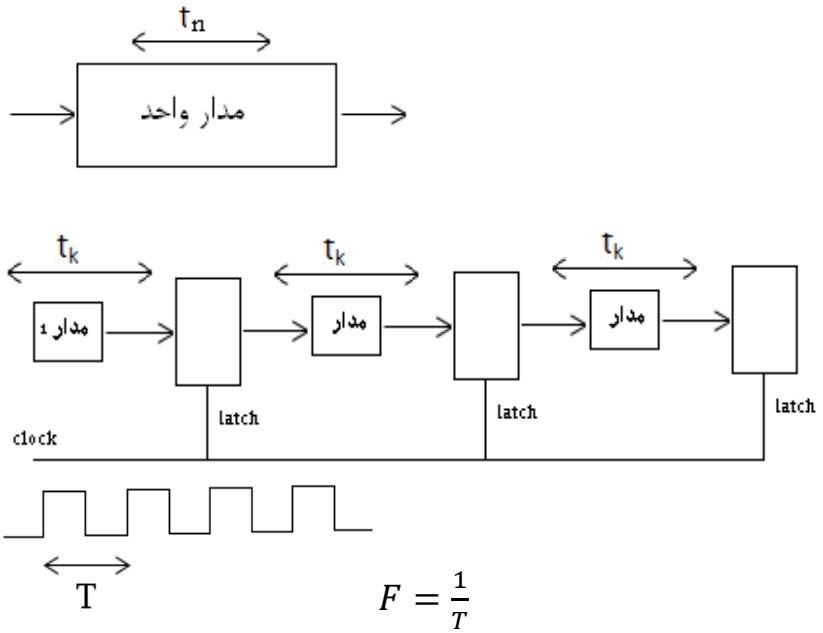
مثال: در مثال شستن لباس ها، قبلاً یک دانه تشت (ظرفهایی که در آنها لباس می شویند) جواب می داده ولی حالا باید یک ظرف میانی باشد. تا با یک طریقی اینها با هم کار کنند. شاید آدم ها چشم و گوش داشته باشند و بینند که بعدی خالی است و اگر هم داده ها روی هم بیفتد هیچ اشکالی پیش نمی آید چون لباس ها را روی هم بگذاریم خراب نمی شود ولی در مورد داده ها وقتی داده اول وجود دارد و داده ی دوم را می گذاریم روی آن در همان خانه ی حافظه چه اتفاقی می افتد؟ داده اولی پاک می شود و از بین می رود.

اول از همه اینها چشم و گوش ندارند. باید یک فکر اساسی کنیم. در طریقه sync ما یک طبال داریم و کارش این است که هر وقت من طبل زدم شما باید به نفر بعدی سریع تحویل بدهید این ۴ تایی که اینجا نشسته اند هم کور و هم لال هستند هیچی نمی بینند و فقط صدای طبل را می شنوند. هر وقت صدای طبل را شنیدند باید کارشان تمام شود و بدهند به نفر بعدی و وقتی این را تحویل نفر بعدی می دهد از نفرقبلی، نفر ۲ از نفر قبلی تحویل می گیرد و آن نتایج اش را به نفر بعدی یا ۳ می دهد.

سوال: این طبال به کسی باید نگاه کند؟

به کسی که از همه کندتر است باید نگاه کند، زمانی که طبال طبل می زند باید بر اساس مصرف کننده پر تاخیرترین قسمت باشد. هرچقدر آن قسمت بیشتر تاخیر داشته باشد طبال می زند. پس اگر تعداد مراحل هم زیاد باشد بهتر است. چرا؟ چون طبال سریعتر طبل می زند.

مدار سنکرون: مدار واحد را تکه تکه کردیم.



$$F = \frac{1}{T}$$

اگر تاخیر قسمت های مختلف یکسان باشد. نتیجه می گیریم کار بهتر انجام می شود. چرا؟ چون با هم همزمان کار انجام می دهند. و به جای طبال اینجا clock را داریم.

در مدل سنکرون بینشان Latch می گذاریم. قبلاً یک مدار واحد داشتیم که زمان تاخیرش t_n بود و حالا مدار را تکه تکه کردیم و زمان هر مرحله اش t_k است. فرض می کنیم که اندازه تاخیرهایشان با هم برابر است ولی اگر تاخیر یک قسمت بیشتر باشد بد است. ما باید به آن نگاه کنیم. بهتر است است تاخیر قسمت های مختلف یکسان باشد.

حال ما طبال را در دنیای واقعی داریم ولی اینجا ما یک clock pals (پالس) می گذاریم. Clock یک سری ۰ و ۱ متوالی است که توسط یک دستگاه کریستال ایجاد می شود. فاصله ی بین دو لبه ی بالا رونده را پریود می گویند و با T نشان می دهند و فرکانس هم $\frac{1}{T}$ است. که T واحدش زمان و فرکانس واحدش HZ است. و از آنجایی که Latch کار قفل را انجام می دهد. و از آنجا که باید فرکانسش فعال شود هر چه در ورودی است در خروجی می گذارد. و در خروجی اش نگه می دارد تا زمانی که پایه فرمانش فعال شود. و در خروجی خویش قفل می کند.

پس این پایه فرمان Latch ها یا پایه قفل Latch ها مثل طبل می ماند. بهتر است که وقتی حساس به لبه بالا رونده باشد و طول پریودمان برابر T_k باشد و فرکانس $\frac{1}{T_k}$ باشد.

هرچقدر فرکانسمان بالا رود T_kکم می شود. پس ما علاقه مند هستیم که فرکانسمان را بالا ببریم چون هر چه فرکانس را بالا ببریم T_k کم شده و هر چه T_k کم شود یعنی اینکه تعداد مراحلمان زیاد شده و درباره آن مشکلات خاص و هازارد ها پیش می آید.

قیمت خط لوله و تاخیرش چه می شود؟

فرض کنیم قیمت کل واحدمان P_T، قیمت یک Latch را P_L حساب کنیم، قیمت کل می شود:

قیمت کل $= P_T + KP_L$

چرا K: چون Latch ها یکی نیست چند تاست.

فرض کردیم که قسمت قسمت کردن مدار هزینه ندارد.

تاخیر هر کدام از اینها T_k است. وقتی که همه را پشت سرهم بگذاریم می شود t_n و هر یک از این Latch ها یک تاخیر دارد t_L پس تاخیر کلمان می شود:

تاخیر کل $= \frac{t_n}{k} + t_L$ ↑ تاخیر یک مرحله و t_L تاخیر یک Latch است.

تعداد K بهینه را می خواهیم حساب کنیم. چون K ↑ قیمت ، تاخیر را طوری می خواهیم حساب کنیم که K بهینه باشد. قبلا گفتیم که K بیشتر باشد بهتر است بعد گفتیم K باشد هازارد داریم.

$$\frac{\text{قیمت}}{\text{تاخیر}} = \frac{p_T + kp_L}{\frac{t_n}{k} + t_L}$$

وقتی K افزایش یابد تعداد Latch بیشتر می شود و وقتی تعداد Latch ها بیشتر شود هزینه و تاخیر زیاد می شود. بنابراین با افزایش K قیمت کل و تاخیر کل سیستم بیشتر می شود. حال دنبال این هستیم که یک K بدست آوریم که هم تاخیرش کم شود و هم قیمت زیاد افزایش نداشته باشد.

چگونه k بهینه را بدست آوریم؟ از نسبت قیمت به تاخیر نسبت به k مشتق می گیریم و صورت مشتق را مساوی صفر قرار می دهیم.

$$\frac{p_T + kp_L}{\frac{t_n}{k} + t_L} = \frac{p_L\left(\frac{t_n}{k} + t_L\right) - \left(-\frac{t_n}{k^2}\right)(p_T + kp_L)}{\left(\frac{t_n}{k} + t_L\right)^2} = 0$$

$$p_L\left(\frac{t_n}{k} + t_L\right) + \left(\frac{t_n}{k^2}\right)(p_T + kp_L) = 0$$

$$K^* = \sqrt{\frac{p_n t_n}{p_L t_L}}$$

وقتی سنکرون را حساب کنیم همان فرمول را حساب کنیم به این نتیجه می رسیم که اگر قیمت مدار را افزایش داده و قیمت Latch را کاهش دهیم بهتر است تعداد مراحل زیاد شود و از آن طرف اگر قیمت مدار افزایش و تاخیر Latch کاهش یابد بهتر است تعداد مراحل زیاد شود. یعنی وقتی تاخیر و قیمتمان پایین است بهتر است تعداد مراحل کم شود و همین طور بالعکس.

با افزایش قیمت Latch و افزایش تاخیر Latch بهتر است تعداد مراحل کم شود. چرا؟ چون Latch ها تاخیر دارند و از طرفی با کاهش قیمت Latch و کاهش تاخیر Latch بهتر است تعداد مراحل n بیشتر شود.

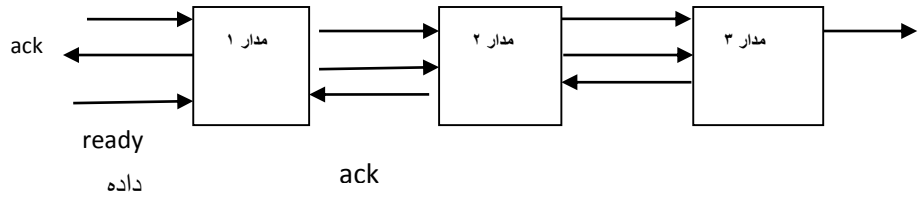
نکته: ابزارهای طراحی سنکرون در بازار فراوان است و این روش سنکرون خیلی خوب است و اینجا یک اشکالی که بوجود می آید این است که ما باید یک مداراتی اضافه کنیم به اسم Latch که این هم هزینه دارد و این باعث می شود که ما نتوانیم زیاد kمان را افزایش دهیم.

راه حل دیگر:

۲-۴-۴-۲ روش آسنکرون:

در این سیستم دیگر نیازی به clock نداریم و از چیزی به اسم hand shaking استفاده می کنیم. در این روش دو قطعه برای یکدیگر سیگنال هایی ارسال می کنند تا از میزان آمادگی یکدیگر با خبر شوند.

مثال: در مثال شستن لباس ها هر کسی با بغل دستی اش در ارتباط است. یعنی دست تکان می دهد به این صورت است که وقتی مدار اصلی را تکه تکه کردیم مدارها خودشان را پشت سر هم می گذاریم. مدار ۱ که آماده شد تحویل مدار ۲ بدهد. و مدار ۲ که آماده شد تحویل مدار ۳ بدهد. در سنکرون مدار نتیجه اش را می گذارد داخل Latch و Latch تحویل مدار ۲ می دهد. ولی در اینجا مدار هر وقت کارش را انجام داد به بعدی می گوید من آماده ام و داده ی جدید را بگیر و بعدی هم خواست می گوید من گرفتم و اگر آماده نبود جوابش را نمی دهد. و صبر می کند کار قبلی اش تمام شود و سپس به بعدی تحویل می دهد. به این کار می گویند دست تکان دادن. در این روش هر کسی با خودش و با بعدی و قبلی خودش ارتباط دارد از طریق سیگنال Ack ،ready

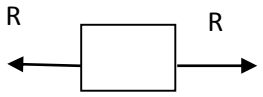


مزیت: قیمت Latch، چون هزینه ای برای Latch ها داشتیم هزینه Latch حذف شده و تأخیر Latch هم نداریم.

بدی: مدارش یک مقدار پیچیده تر می شود چرا؟ باید با طرف راست و چپش هماهنگی کند و این پیچیدگی مداری خواهد داشت. و مداری برای hand shunting(دست تکان دادن) به هر کدام از آنها باید اضافه کنیم. ولی این بهتر از قبلی است.

اشکال بزرگ: چون تحلیل این ها به صورت دستی امکان پذیر نیست و نرم افزار خوبی که این کار را انجام دهد عملاً وجود ندارد پس این روش، روش خوبی نیست.

در سنکرون ابزار زیاد است، چرا؟ چون پیچیدگی کمتر است. چون طبال کار راخیلی آسان می کند clock را که می گذاریم می گوید سر clock کار را انجام بده و تحویل بده ولی در آسنکرون مشخص نیست کجاست؟ در سنکرون همه ی عملیات ها در لبه ی بالا رونده clock انجام می شود و هر گاه لبه ی بالا رونده رخ داد یک سری کار را انجام می دهد و در مراحل بعدی تا لبه ی بالا رونده بعدی خداحافظی می کند. آن لحظه ای که لبه ی بالا رونده اتفاق می افتد آنجا یک کاری اتفاق می افتد، و در آنجا سیستم را می توان راحت طراحی کرد. و سیستم گسسته است.

وقتی آسنکرون است 

ما نمی دانیم که آیا این هم آماده است کـه بگیـرد و هـم آماده است که بفرستد؟

در گذشته انسان ها سیستم را تحلیل می کردند. ولی الان ابزارها تحلیل می کنند. ابزارهایی که تحلیل مـی کند و وقتی می خواهیم روی سیستم خودمان طراحی کنیم گیر می کند و به مشکل می خورد. و به راحتی تحلیل نمی کند. پس باید نرم افزار یا ابزار تحلیل سیستم آسنکرون تهیه کنیم. ابزارهـا بیشـتر بـرای sync است و برای Async ابزار کم است.

روال ساخت IC:

وقتی IC می خواهیم بسازیم از خاک رس یا خاک قرمزی که در آن سیلیکتون است استفاده می شود. و این را به کارخانه می برند و تبدیل به IC می شود.

دو زبان HDL داریم: ۱- VHDL ۲- very Log

چرا در سیستم آسنکرون کسی برایش نرم افزار طراحی نکرد؟

چون گسسته نیست و پیوسته است برای سیستمی که پیوسته است ما همیشه می آییم گسسته سازی مـی کنیم. وقتی یک سیستم پیوسته را گسسته سازی روی آن انجام می دهیم یـک سـری جزئیـاتی روی آن از بین می رود. وجزئیات که از بین می رود باعث می شود که مدار تغییر کند و ما اصلاً جواب نداریم. با دقـت بالا گسسته سازی کنیم مشکل حل می شود. پس عملاً ابزارهایی که این را تبدیل کنیم به IC برای سیستم های sync سخت است. پس عملاً همه ی چیزهایی که داخل com هستند بیشترsync است و یعنی همه ی آنها طبال را دارند.

فصل سوم

چندپردازندگی

۳-۱ طبقه بندی سیسیتم ها:

یک طبقه بندی بنام flyn داریم که چندپردازنده ها را به چهار سیستم یا نوع تقسیم می کند:

1- SISD: Single Instruction Single Data

2- SIMD: Single Instruction Multiple Data

3- MISD: Multiple Instruction Single Data

4- MIMD: Multiple Instruction Multiple Data

۱- پردازنده های معمولی جزء این دسته هستند. یک دستور، یک داده

۲- یک دستو به صورت همزمان بر روی چند پردازنده اجرا می شود.کاربردش در پردازش ماتریس می باشد. به طور مثال جمع ماتریس یکباره انجام می شود.

۳- چندین دستور به طور همزمان بر روی یک داده اجرا می شوند.پیاده سازی این روش انجام نشده است.

۴- واقعا چند پردازنده ای است. یعنی چند تا دستور توسط چند پردازنده انجام می شود.

مثال: در SISD (یک دوربین، یک داده) یک دوربین در جاده می گذاریم و هر ماشینی که رد می شود دوربین باید پلاکش را بخواند و پردازش سیگنال یا پردازش دستور را انجام دهد.

در SIMD از ۴ دوربین استفاده می کنیم و ۴ عکس ازمحیط می گیریم، در همه تصاویر به دنبال پلاک خودرو هستیم. پس یک دستور است وقرار است یک دستورالعمل روی آن اجرا شود. دستورالعمل در همه آنها یکسان است و تصویر را به داده یا پلاک تبدیل می کنند. MISD دو دستورالعمل داریم، یک دستور تشخیص پلاک و دو تشخیص تعداد افراد داخل خودرو.

پس دراین صورت یک عکس از خودرو گرفته و به یک پردازنده دستور می دهیم پلاک ماشین را تشخیص بدهد و به پردازنده دعدی دستور می دهیم تعداد افراد داخل ماشین را بدست آورد. پس دو دستور بر روی یک داده انجام می شود. ولی عملا این روش پیاده سازی نشده است. MIMD چند دستور توسط چند پردازنده انجام می شود.

برای بوجود آوردن چندپردازندگی برای Async ها:

در اینجا به جای اینکه واحد Fمان یکی F کند ۲ تا F می کند اما حواسمان باشد، ۲ عدد پردازنده داخل سیستم وجود دارد. در قسمت D هر دو را همزمان D می کند پس ما ۲ عدد مدار داریم.

آیا می شود ۲ دستور را همزمان در حافظه انجام داد؟آن مدار حافظه مان باید یک مقدار سخت تر باشد. باید یک مدار متنوع تری باشد تا بتوانیم ۲ بار از روی آن همزمان بخوانیم یا حافظه مان را Inter leveling می کند. یعنی یک دستوری در حافظه ی اولی می گذاریم و یک دستور را در حافظه ۲ می گذاریم، f می کنند. به این کار Inter leveling کردن می گویند.

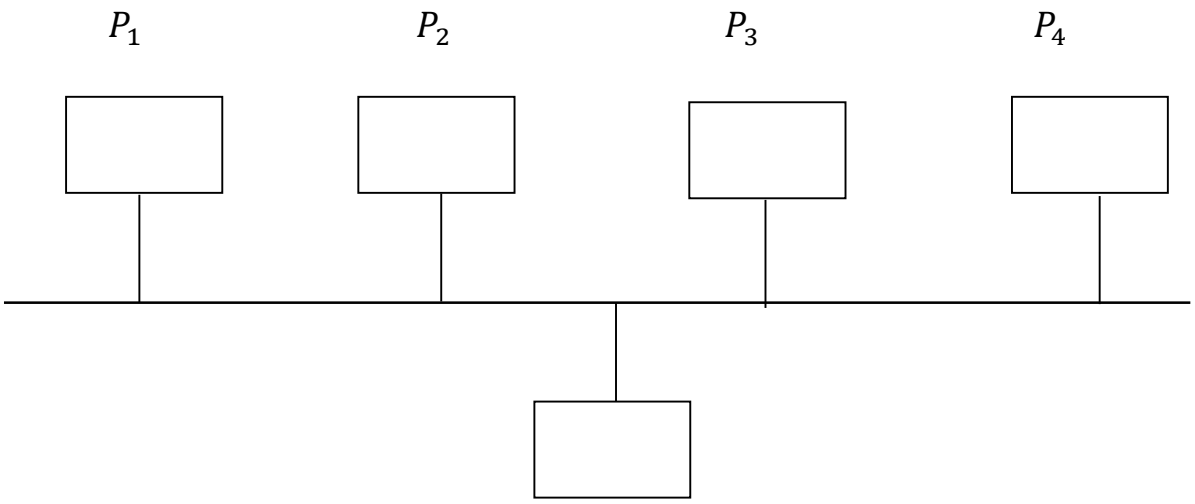
F1	D1	Ex1	Wb1			
F2	D2	Ex2	Wb2			
	F3	D3	Ex3	Wb3		
	F4	D4	Ex4	Wb4		
		F5	D5	Ex5	Wb5	
		F6	D6	Ex6	Wb6	
			F7	D7	Ex7	Wb7
			F8	D8	Ex8	Wb8

درهم ریختن و پشت سر هم قرار ندادن بعضی از این کلال ها به عمل ها کاری نداریم و عملیات را سطح بالاتر نگاه می کنیم و مطمئناً هازاردهایش بیشتر است مشکلاتش بیشتر است و دردسرهای بیشتری دارد و سخت افزار بیشتری دارد و هزینه اش نیز بیشتر است. ما با این ها هیچ کاری نداریم می خواهیم ببینیم اگر این کار را کنیم سرعت چقدر بهتر است همه ی آنها را می دانیم.

تمرین: سیستم های grid را با سیستم های چند پردازنده مقایسه کنید؟

در grid سیستم های مختلف با هم ارتباط دارند در صورتی که در چند پردازنده ای cpu ها به طور مستقیم با هم در ارتباط هستند. در grid ارتباط ازطریق اینترنت می باشد در صورتی که در چندپردازنده ها ارتباط از طریق bus می باشد. در grid پردازنده ها با هم یک کار را انجام میدهند در صورتی که در چند پردازنده هر پردازند کارخودش را انجام می دهد.

چگونگی اتصال پردازنده ها با یکدیگر:



۳-۲ ساختار های اتصالات متقابل

۳-۲-۱ سیستم چند پردازنده معمولی(ساده):

در این روش همه پردازنده ها ازطریق یک BUS بهم متصل شده اند.از لحاظ هزینه، هزینه چندانی ندارد چراکه قبلاگذرگاه بین حافظه و پردازنده وجود داشته است و در این روش تنها چند عدد پردازنده به آن اضافه شده است.

مشکل این روش این است که :

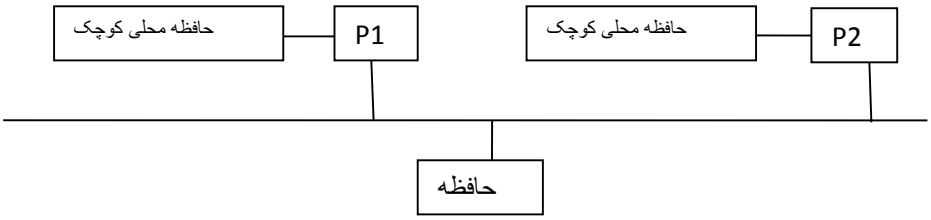
۱-در هر لحظه تنها یک پردازنده می تواند از حافظه استفاده کند.بدلیل ارتباط از طریق یک باس.

۲-اگر تعداد پردازنده ها زیاد شود هزینه بالا رفته ولی کارایی بالا نمی رود.

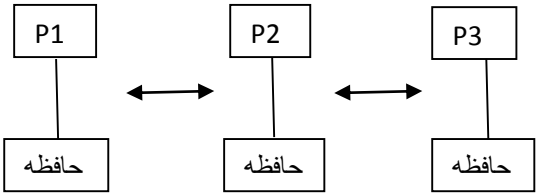
راه حل :یک پردازنده محلی به هر پردازنده اختصاص دهیم . در این صورت هر پردازنده به جای اینکه برای واکشی تک تک دستورات دفعات زیادی به حافظه مراجعه کند می تواند با یک مرحله مراجعه با حافظه اطلاعات زیاددی را واکشی کرده و در حافظه خودش قرار دهد.در این صورت باس آزاد بوده و سرعت کار بالا می رود.ولی باز مشکلاتی نیز وجود دارد، هدف ما از چند پردازنده ای موازی سازی و ارتباط پردازنده ها باهم می باشد. اما اگر در این جا یک پردازنده به نتیجه کار پردازنده دیگر احتیاج داشت باید صبر کند تا پردازنده دیگر کارش به پایان برسد و سپس نتیجه کار را بر روی حافظه بنویسد و بعد این پردازنده بتواند از آن اطلاعات استفاده کند، و یا می توان گفت اگر دو پردازنده بخواهند بر روی یک داده تغییراتی اعمال کنند

و هر دو داده را واکشی کرده و آن را تغییر دهند، سوالی که در این جا مطرح است این است که کدامیک باید بر روی حافظه نوشته شود؟

یک دفعه به اندازه ی حافظه اش به اندازه ای که می خواهد پردازش کند می رود یکی می گذارد توی حافظه ی محلی اش و همین طور p3, p2 هم همینطور واین وسط پردازش را روی اطلاعات محلی شان انجام می دهد و دراین صورت کارایی مان افزایش می یابد.

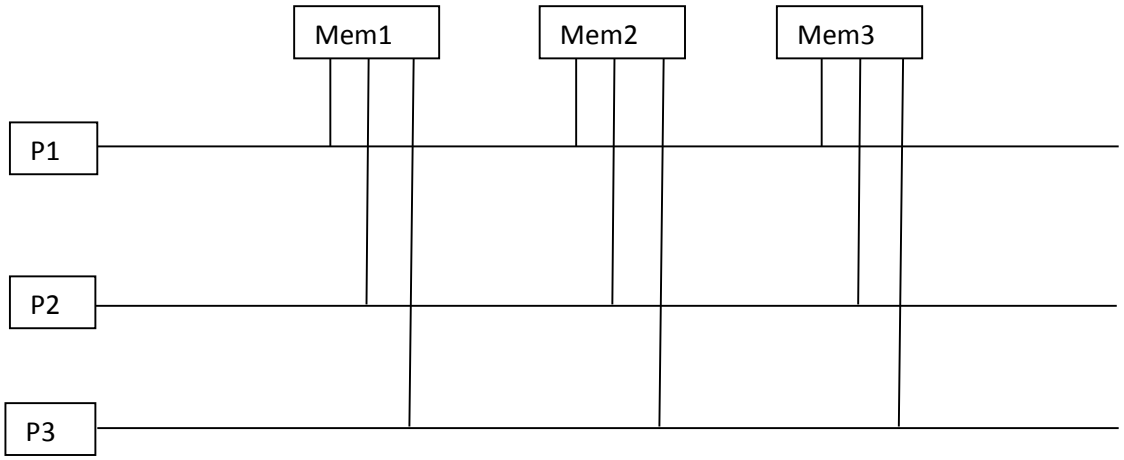


مشکل : مثلا خانه ی حافظه مقدار ۱۰ را دارد و هر دو تا این ۱۰ را برمی دارند و در حافظه محلی خودشان می ریزد و هرکدام از اینها قرار است که یکی *۲ یکی *۳ و یکی *۴ کند و حال یک خانه حافظه را می خواهند تغییر دهند هر کدامشان و هر کدام مقدار داده ای را می آورد و درخانه حافظه خودش و یکی *۲ می شود و ۲۰ می شود و یکی ۳۰ و دیگری ۴۰و هر سه که می خواهند مقدار را برگردانند و درحافظه اصلی نوشته شود و این مشکل است .اگر اینها خواستند برگردانند و هر کدام تغییری روی داده داشته باشند کدامیک باید بازنویسی شوند؟ ویک پیشنهاد می دهیم: و می گوییم این پردازنده منتظر بماند و هر وقت کارش تمام شد این برود، بخواندش وتغییر دهد وهر موقع این کارش تمام شد دیگری برود ریال بخواند و تغییر دهد.و همین طور پردازنده ها پشت سر هم این کار را انجام دهند و الان چه اشکالی پیش می آید؟یعنی p_1 وقتی تغییر می دهد p_2 باید منتظر این باشد که جواب را برگرداند واین روش بدتر می شود و اینکه پردازنده ها هر کدام حافظه خودشان را داشته باشند و بعدا باهم ارتباط داشته باشند و این دیگرچند پردازشی نمی شود و می شود شبکه با شبکه gride آن چند پردازنده ای که ارتباط شبکه ای دارند، البته ما به دنبال این هستیم که پردازنده ها خودشان باهم در ارتباط باشند.



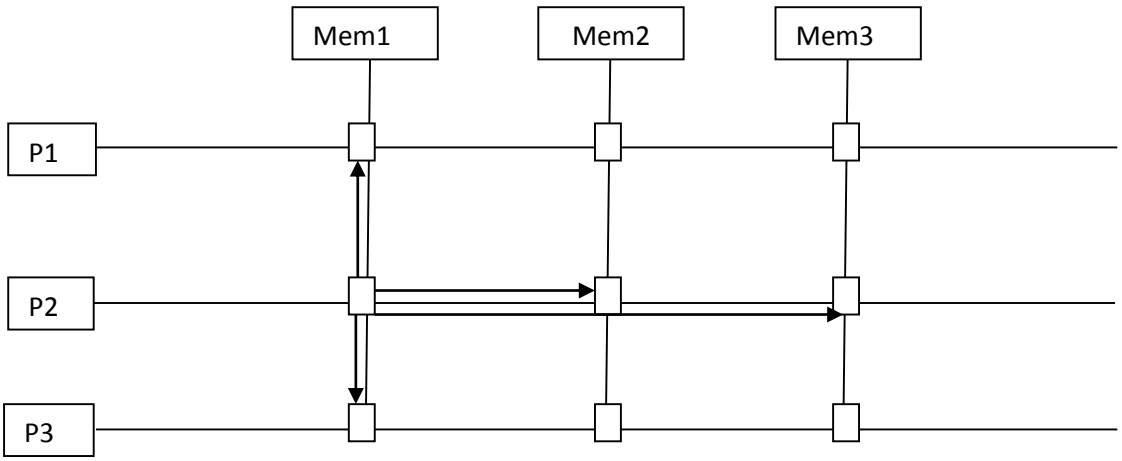
۳-۲-۲ حافظه چند پورتی:

حافظه چند پورته : این حافظه ها با حافظه های معمولی یک تفاوت دارند ومی توانند همزمان ۲ دستور را باهم پردازش کنند اگر یک دستور به آن بدهیم و ما به حافظه دستور خواندن و نوشتن می دهیم، نباید بادستور Add,Sub اشتباه کنیم در این شکل از یک حافظه چند پورت خارج می شوند که می تواند همزمان چندین cpu به آن وصل باشدوهمزمان از چندcpu دستور بگیرد. اگر یک cpu به آن دستور دهد سریعتر عمل می کند واگرچندتا cpu به آن دستور دهد یک مقدارکندتر عمل می کند. البته بعضی از مدل ها همزمان دارند دستور می دهند وسریعترهم عمل می کند ولی گران تمام می شود. این حافظه ها ازحافظه های معمولی گرانتر است. در اینجاچه فرقی کرده ؟ در اینجا پردازنده ها یمان به چند تا حافظه وصل است بایدطوری برنامه هایمان راتغییردهیم ودسته بندی کنیم که برنامه هایی که مربوط بهp1 هستند داخل mem1 باشد و برنامه هایی که مربوط بهp2 است داخل mem2 باشد.وبرنامه هایی که مربوط بهp3 هستند داخل mem3 باشد. پس سوال مطرح می شودکه چرا p3 به mem1 وصل است؟شایدبه اطلاعات نیاز پیداکرد ولی کمتر به اطلاعات نیاز پیداکرده ولی p3 بیشتر باmem3 کترمی کند. ولی ارتباط هم با mem1 دارد به خاطر بعضی مواقع.البته mem1 توانایی جواب دادن به هردو یعنی p1,p3 رادارد ولی مایک کار می کنیم که هرکدام ازاینها داده های مربوط به خودشان را در حافظه جداگانه داشته باشند و ازاین طرف هروقت به حافظه دیگری نیازپیدا کرد مشکل پیش نیاید.



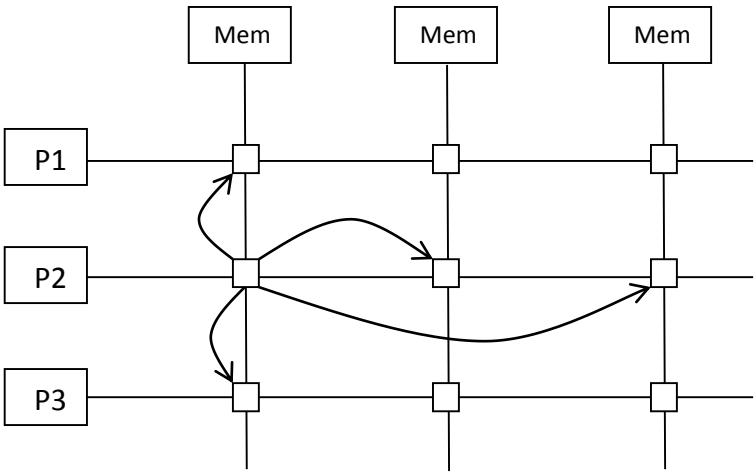
یک مشکل اساسی دارد : هزینه – ما باید برای حافظه چند پورته پول بدهیم و از یک طرف هم مشکل scalabity (توسعه پذیری) را داریم وبه همین راحتی نمی شود زیاد کرد، وقتی زیاد می شود باز هم به مشکل می خوریم البته تعداد پردازنده های ما با حافظه ها لزوما برابر نیست و کمتر هم می تواند شود ولی وقتی تعداد پردازنده ها زیاد شود و تعداد حافظه ها کم باشد مثل قبل می شود و اگر نشود حافظه با پورت زیاد گران می شود.

۳-۲-۳حافظه با سوئیچ تقاطعی:



سوئیچ ها نسبتا ساده هستند.

عملکرد سوئیچ: به این صورت است که می بیند به این حافظه چیزی وصل است یا نه وقتی p1 می خواهد با حافظه ۱ ارتباط برقرار کند می آید به سوئیچ دستور می دهد و این سوئیچ به حافظه ۱ متصل می شود ودر حینی که p1با mem1 متصل است این a اگر بخواهد به mem1 متصل شود این سوئیچ اجازه نمی دهد و این سوئیچ به a2 می گوید اگر می خواهید بنویسی بر روی mem2 بنویسید. پس نوشتن به حافظه و خواندن از حافظه از طریق این سوئیچ امکان پذیر می شود.



مزایا :

۱. حافظه تک پورت است.

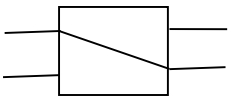
۲. گسترش پذیری راحت است هر چقدر می خواهیم پردازنده استفاده کنیم و هر چقدر می خواهم حافظه استفاده کنیم.

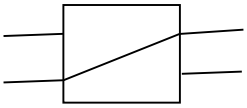
اشکال : ما برای ۳ دستور ۹ تا سوئیچ می کنیم و برای n دستور n^2 سوئیچ می کنیم. تعداد سوئیچ ها رابطه مستقیمی با تعداد پردازنده هایمان ندارد برای ۴ تایی یا ۵ تایی خوب است اما برای مثلا ۲۵۶ پردازنده کـه $۲۵۶^{۲۵۶}$ سوئیچ نیاز داریم خوب نیست .به صرفه و معمول نیست.خطایابی سخت و تاخیرمان بالا می رود.

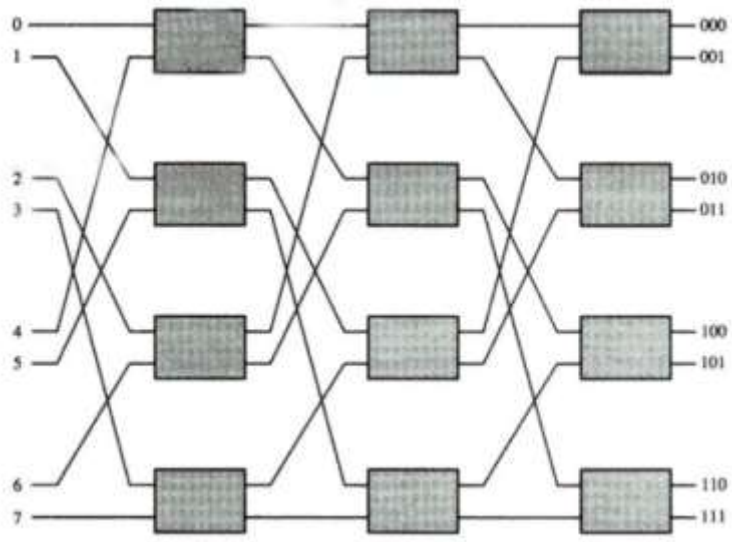
۳-۲-۴ استفاده از سوئیچ های بانیان(شبکه سوئیچ های چند طبقه ۲*۲)

مدل banyan: در ارتباط پردازنده ها به هم کاربرد دارد و در طراحی سوئیچ و روتر کاربرد دارد. ساختار Ω امگا دارد. مدل های دیگری هم دارد.در مثال قبلی به سوئیچ باید نگاه میکرد که دیگران اشغال هستند یا نه ولی در اینجا سوئیچ اگر ورودی اش صفر باشد ورودی به خروجی مستقیم وصل می شود و اگـر ورودی اش یک باشد باید به پایین وصل شود .اگر دو تا صفر و دوتایک باشد،internal blocking می شود و همچنین خالی نمی شود پس ما براساس آدرس وصل می شویم ، سمت چپ به پردازنده ها وسمت راست بـه حافظـه ها.از طریق این سوئیچ ها به هم متصل می شویم.

پردازنده فقط شماره حافظه می فرستد. سوئیچ طوری عمل می کند که مستقیم وصل شود.

برای ۱ 

برای ۰ 



یک مشکل اساسی دارد:

هزینه:مابایدبرای حافظه چندپورته پول بدهیم وازیک طرف هم مشکل توسعه پذیری Scalability راداریم وبه همین راحتی نمی شودزیادکردوزیاد که می شودبازهم به مشکل برمی خوریم.البته تعدادپردازنده هاباحافظه هالزومابرابرنیست ومی تواندکمترشود.ولی وقتی تعدادپردازنده هازیادشودوتعدادحافظه هاکم باشدمثل قبلی می شودواگرنشودحافظه باپورت زیادگران می شود.

ازسوئیچ استفاده می کنیم:

سوئیچ ها ی مدارنسبتا ساده هستند.

عملکردSwitch :

Switch به نحوی عمل می کندکه می بینیدبه این حافظه چیزی وصل است یاخیر؟

وقتی P1 می خواهدباحافظه ۱ ارتباط برقرارکند به Switch دستورمی دهد واین Switch به حافظه ۱ متصل می شوددراین حین که P1 به حافظه ۱ متصل است اگرP2 بخواهد به حافظه ۱ متصل شوداین سوئیچ اجازه نمی دهد و به P2 می گویداگرمی خواهی بنویسی برروی حافظه ۲ بنویس، پس نوشتن به حافظه وخواندن ازحافظه ازطریق این سوئیچ امکان پذیرمی شود

مزایا:

۱- دراینجا ازیک جهت مدار،مدارپیچیده ای نیست پس سوئیچ هایمان هزینه بالایی ندارند.

۲- درروش قبل برای ۸ پردازنده ۶۴ سوئیچ می خواستیم ولی دراینجا۱۲ سوئیچ می خواهیم.

برای n پردازنده $\frac{n}{2}$ سطرداریم یعنی برای ۸ پردازنده ۴سطرداریم.

تعدادستون ها هم برابرتعدادبیت هامی باشد پس برابرlog n می باشدکه ستون اول به بیت اول، ستون دوم به بیت دوم، ستون سوم به بیت سوم .

تعدادسوئیچ ها= $\frac{n}{2}$

تعدادبیت = تعدادستون= log n

برتری ها:

تعداد کمتری سوئیچ نیاز دارد. بنابر این هم مدار سوئیچ ساده است و هم قیمت کمتری دارد.

مشکلات:

۱- Internal blocking

فرض کنید ۱۰۰ می خواهد از پردازنده P_0وارد سوئیچ شود. ۱ وارد سوئیچ اول شده و از پورت دوم سوئیچ خارج می شود. سپس وارد پورت اول سوئیچ دوم می شود و اینجا به مشکل بر می خورد چون این سوئیچ قبلا توسط ۱۰۱ اشغال شده است. با وجود اینکه حافظه ۱۰۰ اشغال نیست ولی امکان اتصال وجود ندارد. در اینجا Internal blocking رخ داده است.

راه حل: اگر ورودی ها را مرتب کنیم Internal blocking رخ نمی دهد. اما در اینجا درخواست ها مرتب نبودند. در شبکه Banyan برای مرتب کردن درخواست ها از یک شبکه Batcher قبل از شبکه Banyan استفاده می کنیم. (منظور از شبکه اتصال آنهاست نه شبکه کامپیوتری) یعنی اینکه ابتدا شماره درخواست هایمان از حافظه را به Batcher می دهیم تا آنها را به ترتیب ۱ و۴و۵ مرتب کند و مشکل Internal blocking حل می شود.

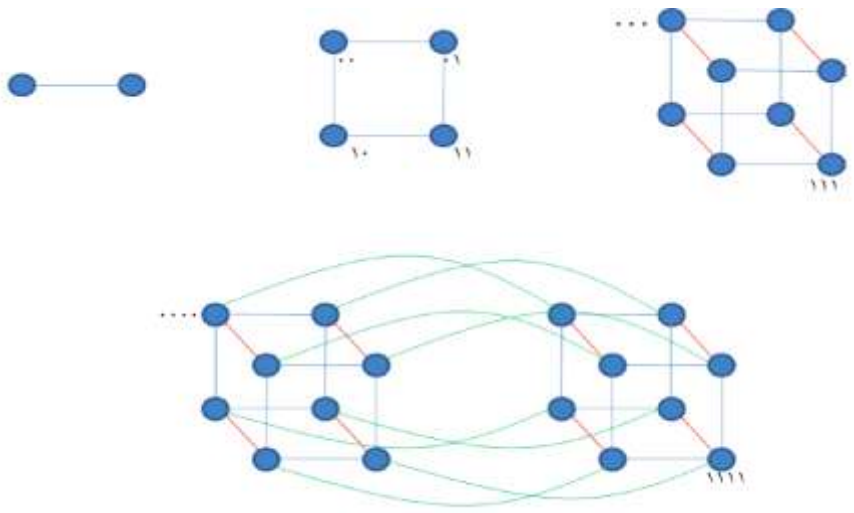
۲- output contention

پردازنده های نمی توانند تشخیص دهند کدام حافظه اشغال و کدام حافظه آزاد است. درخواست می دهند ولی امکان دارد حافظه اشغال باشد و در خروجی برخورد بوجود آید. یعنی دو پردازنده نمی توانند همزمان به یک حافظه وصل شوند. که به این مشکل output contention گویند.

۳-۲-۵ روش Hyper cube:

در این روش پردازنده هایی که شماره باینری آنها فقط و فقط در یک بیت تفاوت دارند به هم وصل می شوند.

اگر n پردازنده داشته باشیم ، هر پردازنده به $\log_2^n$ پردازنده دیگر متصل می شود.



به بحث پردازش موازی (Parallel) مرتبط است.

البته قبل از آن یک مفهوم جدید را مطرح می کنیم و دنبال این هستیم که موازی حل شود.(مسئله کوله پشتی، مسئله کوتاهترین مسیر و.....)

شبکه ارتباطات را با گراف مقایسه می کنیم. پردازنده ها با همدیگر با گراف مدل می شوند. مباحث در هر دو بکار می رود. یعنی در چند پردازنده ای و شبکه کاربرد دارد. مثلا Banyan در ارتباط بین پردازنده ها و حافظه ها بود. عینا در سوئیچ ها و روترها هم استفاده می شود.

این بحث هم در ارتباط پردازنده ها و هم در ارتباط شبکه ها مطرح می شود.

NOC: Network On Chip

SOC: System On Chip

SOC: تمام چیزهایی که داخل یک سیستم بود، داخل یک Chip به کار می رود.

Chip همان تراشه یا IC است. مثلا در Computer ما پردازنده داخل یک Chip است. ولی یک Chip هم برای کارت گرافیک می گذاریم و یک Chip هم برای LAN و.... و می گوییم کل یک سیستم داخل یک Chip پیاده سازی می شود. همه چیز ، حافظه، ارتباط با دستگاه ها، همه چیز داخل یک Chip قرار می گیرد که به آن SOC می گوییم و یک بحث قدیمی است.

NOC: داخل یک IC ما چند پردازنده داشته باشیم، چند حافظه داخل یک Chip داشته باشیم و این ها با همدیگر ارتباط داشته باشند. یعنی شبکه ای بینشان موجود باشد که خود بحث مفصلی دارد. اینکه شبکه بین آنها چطور باشد. Switchها چطور باشد، چون در شبکه واقعی Switch داریم.

تمرین: ۱- سوئیچ بچربانیان Batcher-Banyan را با رسم مدار توضیح دهید.

سوئیچ بانیان(Banyan switch)

یک سوئیچ چند مرحله ای است که در آن میکروسوئیچهای هر مرحله، اطلاعات را با استفاده از دنباله باینری آدرس، به سمت خروجی سوئیچ می کنند. اگر ورودی سوئیچ صفر باشد، ورودی مستقیما به خروجی وصل می شود.اگر ورودی سوئیچ یک باشد، ورودی به پایین وصل می شود.

اگرورودی ها ۲ تا صفر یا ۲ تا یک باشد، مشکل بلوکه شدن داخلی اتفاق می افتد یکی از راه هایی که برای حل این مشکل ارائه شده، استفاده از مرتب کننده قبل از سوییچ می باشد .

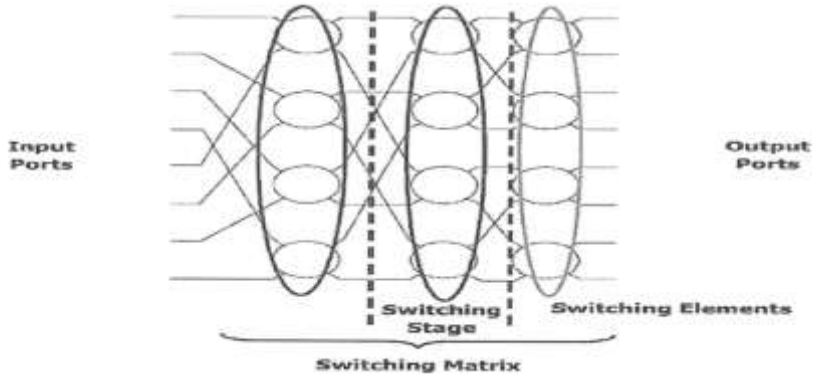
از مزایای سویئچ بانیان میتوان سادگی مدار و همچنین کم هزینه بودن به دلیل تعداد کمتر سوئیچ را نام برد. برای N ورودی و N خروجی به Log N در مبنای ۲ مرحله نیاز داریم در هر مرحله به N/2 میکروسوئیچ نیاز است ، در شکل ساختار این سوئیچ دیده می شود .

به عبارت دیگر مثلا برای ۸ پردازنده به ۱۲ سوئیچ نیازمندیم :

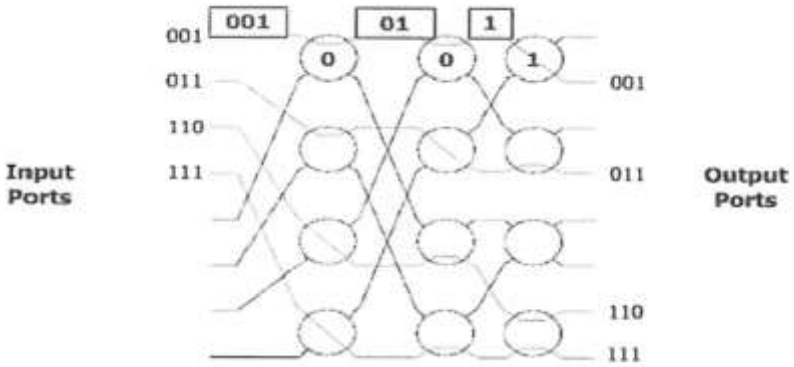
تعداد سوئیچ ها $= \frac{n}{2} . \log n$

تعداد سطرها $= \frac{n}{2}$

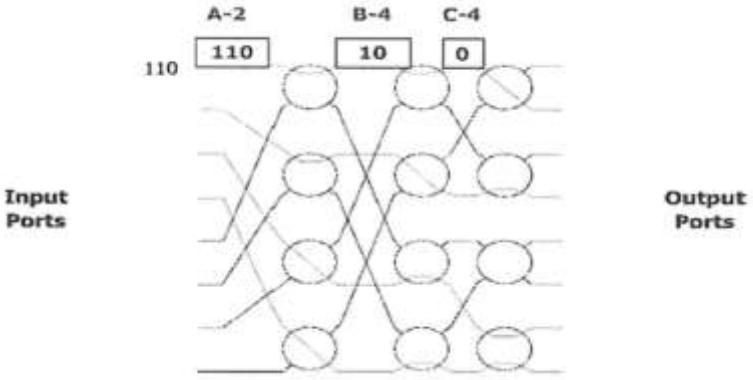
تعداد ستون ها همان تعداد بیت هایمان می باشد یعنی $\log n$



مرحله اول، بسته اطلاعاتی را بر اساس با ارزشترین بیت آدرس، مسیریابی و هدایت می کند، مرحله دوم بر اساس بیت بعدی، الی آخر.



فرض کنید، بسته ای وارد باب ۱ شده است و آدرس بسته ۱۱۰ یا باب ۶ است، اولین میکروسوئیچ (یا A-2) بر اساس اولین بیت که یک است آن را مسیریابی می کند. دومین میکروسوئیچ (B-4) آن را بر اساس دومین بیت که باز هم یک است. مسیریابی می کند و سومین میکروسوئیچ (یا C-4) بر اساس سومین بیت (صفر) آن را مسیریابی می نماید .



و لذا، بسته به باب خروجی (۶) مطابق آدرسش می رسد .

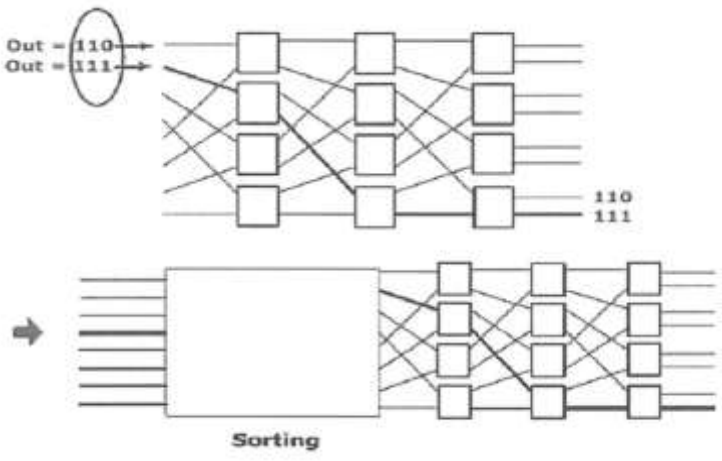
سوئیچ بچر – بانیان(Batcher – Banyan switch)

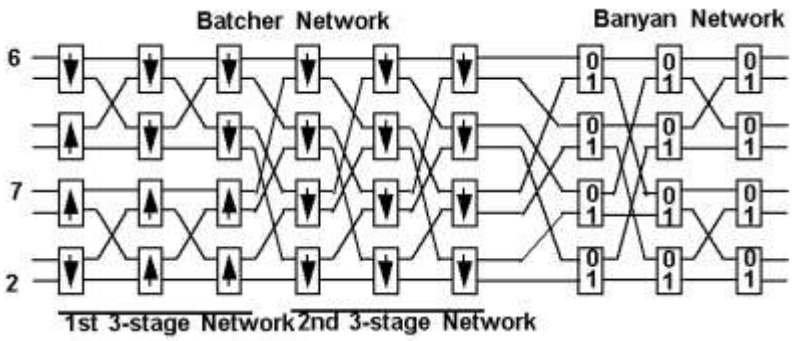
مشکلی که در سوئیچ Banyan وجود دارد که حتی اگر دو بسته آدرسهای خروجی یکسان نداشته باشند (به باب خروجی یکسانی فرستاده نشود) ممکن است به علت یک برخورد داخلی نتوانند مسیر یابی شوند .

این مشکل با ردیف کردن (sort کردن) بسته ها با توجه به باب مقصد ، توسط Batcher حل شده است .

Batcher سوئیچی طراحی نمود که قبل از سوئیچ Batcher می آید و بسته ها را متناظر به آدرس نهایی آنها ردیف می نماید که به مجموعه این دو سوئیچ Batcher – Banyan سوئیچ گفته می شود .

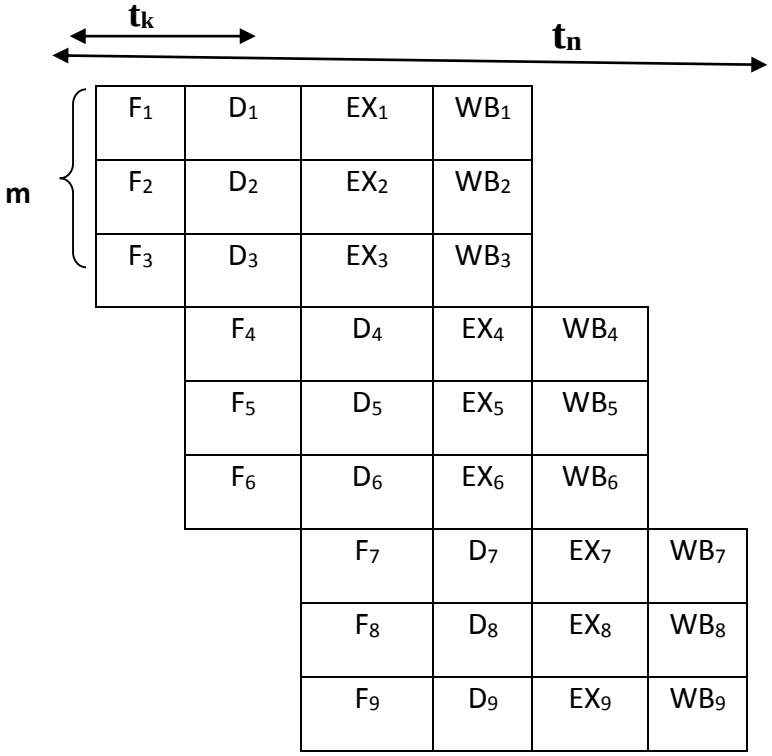
یکی از مشکلاتی که سوئیچ بانیان داشت این بود که احتمال برخورد داخلی حتی برای دو بسته ای که به پورت خروجی یکسانی ارسال شده اند وجود دارد. برای حل این مشکل می توان بسته های ورودی را بر اساس پورت مقصد (خروجی) مرتب کرد. برای این کار از سوئیچ مرتب کننده Batcher قبل از سوئیچ Banyan استفاده می شود. در این سوئیچ از یک قطعه سخت افزاری به نام تله Trap استفاده می کنند. این قطعه از عبور همزمان بسته های تکراری به سوئیچ بانیان جلوگیری می کند. در هر زمان برای هر مقصد فقط یک بسته می تواند ارسال شود، اگر تعداد این بسته ها بیش از یکی باشد باید منتظر بماند.





Super scalar ۲-۴

در super scaler همزمان چند دستور با هم اجرا می شوند، پس مراحل اجرا به صورت زیر است. مـثلا در این شکل تعداد دستوراتی که همزمان اجرا می شوند برابر m یعنی ۳ تا است. یعنی ۳ دستور با هم fetch می شوند وسپس با هم decode می شوند وسپس اجرا و back write می شوند.



$t_n=kt_k$

m=3

زمان اجرای n دستور $=t_n+t_k+\ldots\ldots$

m دستور اول

تعداد t_k ها $\frac{n-m}{m}$

شرح فرمول بالا: زمان اجرای m دستور اول t_n و $\frac{n-m}{m}$ دستور بعدی هر کدام یک t_k را به زمان اجرا اضافه می کنند. یعنی به تعداد $\frac{n-m}{m}$ در این فرمول t_k داریم.

$$\text{زمان اجرای n دستور اول با super scaler} = kt_k + \frac{n-m}{m}t_k = t_k\left(k + \frac{n-m}{m}\right) = t_k(k + \frac{n}{m} - 1)$$

$$\text{زمان اجرای n دستور با pipe line} = t_k(n + k - 1)$$

$$\text{زمان اجرای n دستور بدون خط لوله} = nkt_k$$

محاسبه کارآیی super scaler نسبت به super pipeline:

$$\frac{\text{کارایی } Super\ scaler}{\text{کارایی } pipeline} = \frac{\text{زمان اجرای با } super\ scaler}{\text{زمان اجرای باخط لوله}}$$

$$= \lim_{n\to\infty} \frac{t_k(n + k - 1)}{t_k(k + \frac{n}{m} - 1)} = m$$

✓نتیجه: کارآیی super scalerنسبت به خط لوله mبرابر بیشتر است.

m: تعداد مراحلی که موازی اجرا می شوند.

۳-۵ Super pipeline

در خط لوله مشابه شکل زیر یک دستور در ۴ مرحله اجرا می شود.

F_1	D_1	EX_1	WB_1		
	F_2	D_2	EX_2	WB_2	
		F_3	D_3	EX_3	WB_3

pipeline مزایای زیادی دارد پس برای بهبود کارآیی در خط لوله می خواهیم تعداد قطعات را زیاد کنـیم اما به ۲ دلیل نمی توان چون اگر تعداد قطعات بیشتر شود:

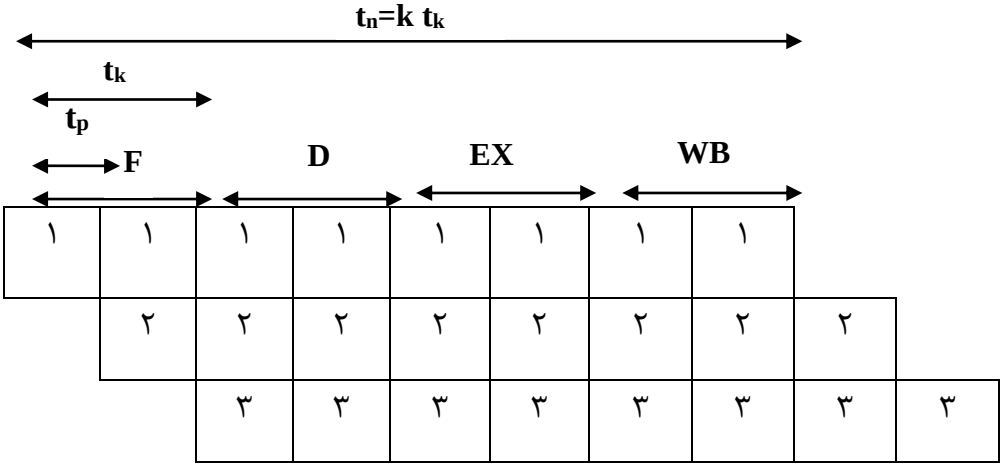
۱- مخاطرات بیشتر می شود.

۲- هزینه ها بالاتر می رود به دلیل اینکه در خط لوله از روش سنکرون استفاده می کنیم ونیاز به latchداریم.

به دلیل این مشکلات می گوییم به جای اینکه fetch را در یک مرحله انجام دهی fetchرا به صورت چند مرحله‌ای انجام بده یعنی در چند مرحله کوچکتر انجام شود وبقیه قطعات نیز به همین ترتیب به مراحل وقطعات کوچکتری تقسیم می شوند. مثلاً fetch به دو مرحله prefetch و WRT=write to register تقسیم می شود.

Prefetch	WRT	D_1	EX_1	WB_1			
		F_2	D_2	EX_2	WB_2		
			F_3	D_3	EX_3	WB_3	

مثلاً می توان هر قطعه را به دو قطعه کوچکتر تقسیم نمود مانند شکل زیر:



P=2

$t_k=pt_p$

زمان اجرای n دستور با super pipeline $=t_n + t_p + \cdots + t_p = kpt_p + (n-1)t_p = (kp + n - 1)t_p$

یک t_n برای دستور اول و تعداد (n-1) تا t_p برای n-1 دستور بعدی. چون تعداد دستورات را n در نظر گرفتیم با اجرای دستور اول n-1 دستور باقی می ماند.

$$\frac{\text{کارایی } Super\ pilpeline}{\text{کارایی } pipeline} = \frac{\text{زمان اجرای باخط لوله}}{\text{زمان اجرای با } super\ pipeline}$$

$$= \lim_{n\to\infty} \frac{pt_p(n+k-1)}{t_p(kp+n-1)} = p$$

در واقع در superpipeline تعداد مراحل pipeline را با روش دیگر وهزینه کمتر، افزایش می دهیم. مثلا هر مرحله pipelineرا ۲ مرحله ای می کنیم.

✓نتیجه: Super pipeline نسبت به خط لوله معمولی p برابر سریعتر اجرا می شود.

Super pipeline-scalar ۴-۵

معماری superpipline ارتقا روش pipline است که در آن منابع دو برابر می باشد. تا در نتیجه هم پوشانی بیشتری بین مراحل ایجاد گردد و در نهایت تعداد بیشتری دستور در یک کلاک انجام گیرد. در این تکنیک هر مرحله به دو زیر مرحله تقسیم میشود. در این روش Fetch دستور اول می تواند همزمان با Fetch دستور دوم اجرا شود. کارایی و بهره وری این روش بالاتر است. زیرا دو عمل مشابه می توانند همزمان اجرا شوند. و نیر زمان انتظار و زمان پاسخ دهی کاهش می یابد.

اشکال:

این روش یک اشکال عمده دارد، آن هم نیاز به سخت افزار اضافه است که باعث می شود هزینه بالاتر رود. و نیز افزایش سخت افزار حدی امکان پذیر است. زیرا وابستگی را بالا برده و نیز باعث بالا رفتن سربار Hazard میشود.

فرمول میزان بالا رفتن سرعت در معماری superpipline:

$$S = ((۱ - n) + k۲) / (((۱ - n) * ۹k) * z)$$

k : تعداد مراحلpipline

n : تعداد دستور

معماری superscaler علاوه بر ویژگی های معماری superpipline ، توانایی اجرای همزمان چندین دستور العمل را در یک مرحله از خط لوله را داراست.

پردازنده چند شاخه که بصورت پویا زمانبندی و برنامه ریزی میشوند را پردازنده superscaler می گویند.

در ساده ترین نوع این پردازنده ها، دستورات بصورت ترتیبی اجرا می شوند. و خود پردازنده تصمیم می گیرد که دو سیکل ساعت مورد نظر، یک یا چند دستور العمل را روی خط لوله بفرسترد.

دستیابی به کارایی خوب superscaler به تلاش هوشمندانه کامپایلر در راستای مهمانپذیری و برنامه ریزی دستورات و از میان برداشتن وابستگی بین آنها نیاز دارد، تا به این صورت نرخ ارسال همزمان دستورات بهبود یابد.

فرمول میزان بالا رفتن سرعت در معماری superscaler (در صورتی که مرحله خط لوله به دو مرحله تقسیم شود):

$$S = (۲ / (۱ - n) + k) / ((۱ - n) + k)$$

تفاوت کارایی روش superpipline-superscaler:

روش Super Scaler کارایی بالاتری نسبت به superpipline دارد، ولی دو برابر شدن سخت افزار باعث دو برابر شدن کارایی نمی شود که به دلیل Hazard است.

فصل چهارم

ارزیابی کارایی پردازنده ها

مقدمه:

این فصل توضیح خواهد داد که چگونه کارآیی سرعت یک کامپیوتر را اندازه بگیریم و آن را گزارش کنیم .در این فصل سعی خواهیم نمود که به سؤالات زیر جواب دهیم:

- چرا کارآیی مهم است؟
- چگونه ما می توانیم کارآیی را به دقت تعریف کنیم؟
- چگونه طراحی سخت افزار کارآیی نرمافزار را تحت تأثیر قرار میدهد؟
- چگونه در دنیای واقعی کارآیی اندازه گرفته شود؟
- چرا بعضی از سخت افزارها بهتر از بقیه عمل می کنند
- کدام یک از فاکتورهای کارآیی به سخت افزار وابستهاند؟
- ما برای اجرا شدن سریعتر یک برنامه به یک ماشین جدید نیاز داریم یا نیازمند یک سیستم - عامل جدید هستیم؟
- مجموعه دستورات یک ماشین چگونه کارآیی را تحت تأثیر قرار میدهند؟

در این فصل عامل های تعیین کننده در کارآیی کامپیوترها مورد بحث قرار خواهد گرفت .بررسی کردن کارآیی به این دلیل مهم است که کارآیی سختافزار اغلب کلیدی ترین اثر را در کل سیستم متشکل از سخت افزار و نرم افزار برعهده دارد .ما معمولاً در این فصل کارآیی و سرعت را به جای هم استفاده می کنیم .ولی در بعضی از موارد این دو عبارت کاملاً معادل هم نیستند .به عنوان مثال اگر هدف ما استفاده از ماشینی باشد که تعداد کار بیشتری را در واحد زمان انجام دهد، ماشینی که سریعتر است بعضی مواقع نمی تواند هدف ما را برآورده کند و ما مجبور هستیم در این مواقع از ماشینی استفاده کنیم که قابلیت های بیشتری داشته باشد) کاراتر باشد .(ما در این کتاب اغلب از اصطلاح کارآیی استفاده خواهیم کرد.

اظهار نظر کردن در مورد کارآیی یک سیستم واقعاً کار مشکلی است .سیستم های بزرگ نرم افزاری با جزئیات زیاد به همراه بازهٔ وسیعی از تکنیک های افزایش سرعت که توسط طراحان سخت افزار به کار گرفته می شوند، از جمله مواردی هستند که صحبت کردن دربارهٔ کارآیی را مشکل می نمایند. تقریبا کار غیر ممکنی است که برگه راهنمایی از مجموعه دستورات یک کامپیوتر را بردارید و با یک برنامه مشخص، تعیین نمایید که آن کامپیوتر برنامه شما را چقدر

سریعتر اجرا خواهد کرد .در واقع بـرای برنامـه هـای کـاربردی مختلـف، ممکـن اسـت مترهـای مختلفـی مناسب باشد و قسمت های مختلف یـک کـامپیوتر ممکـن اسـت نقـش متفـاوتی در تعیـین سـرعت کـل سیستم داشته باشند.

البته در انتخاب بین کامپیوترهای مختلف، کارآیی تقریباً همیشه یک مشخصه مهم به حساب میآیـد.انـدازه گیری و مقایسه ماشین های مختلف برای خریداران و در نتیجه برای طراحـان یـک امـر حیـاتی محسـوب میشود که این مطلب را فروشندگان کامپیوترها به خوبی درک می کنند. اغلب، فروشندگان دوست دارند که شما کامپیوترهای آنها را خیلی ساده نگاه کنید و این نگاه کردن ممکن است بدون توجه به نیاز خریدار، که برنامه های کاربردی خودش را بر روی آن اجرا خواهد کرد انجام بگیرد .در بعضی مواقع ادعاهایی در مـورد کامپیوترها می شود که دید مفیدی برای بعضی از برنامه های کاربردی در اختیار قرار نمی دهند .بنـابراین فهمیدن چگونگی اندازه گیری کارآیی یک کامپیوتر و محدودیت های اندازه گیری در انتخاب یک ماشـین، بسیار مهم می باشند. علاقمندی ما نسبت به کارآیی کامپیوتر ماورای این است که فقط کارآیی کامپیوتر را از بیرون آن اندازه بگیریم .ما نیاز به این داریم که بفهمیم چه چیزهایی کارآیی یک کامپیوتر را تحت تـأثیر قرار می دهند.

فهمیدن اینکه چرا بعضی قسمت های نرم افزار همان طور که ما انتظـار داریـم کـار نمـی کننـد، چـرا یـک مجموعه از دستورات می تواند به گونه ای پیاده سازی شود که بهتر عمـل کنـد، و اینکـه چگونـه بعضـی از قابلیت های سخت افزار کارآیی را تحت تأثیر قرار می دهند، همگی از مسائلی است که ما علاقمند بـه پیـدا کردن جواب مناسب برای آنها هستیم.

تعریف کارآیی

وقتی که گفته می شود کارآیی این کامپیوتر بالاتر از آن یکی است، منظور چیست؟ هر چند این سؤال ساده به نظرمی رسد ولی یک مقایسه با هواپیماهای مسافربری نشان می دهد که این سؤال چقدر قابل تأمل است.

۱ تعدادی از هواپیماهای مسافربری را به همراه سرعت متوسط آنها، برد مسافتی و ظرفیتشان - شکل ۲ نشان می دهد .اگر بخواهیم ببینیم که کدام یک از هواپیماهای موجود در جدول کارآیی بالاتری دارد، اول باید کارآیی را تعریف کنیم .به طور مثال، با در نظر گرفتن اندازهگیریهای مختلف، می بینیم که و هواپیمایی که بیشترین ظرفیت را دارد 747 ، concordeهواپیمایی که بالاترین سرعت متوسط را دارد می باشد.

۱ :مقایسه کارآیی هواپیماهای مختلف - شکل ۲

بنابراین اگر ما بخواهیم کارآیی این دو هواپیما را مقایسه کنیم .به دو صورت می توانیم این کار را انجام دهیم .می توانیم به این صورت تعریف کنیم که هواپیمای کاراتر هواپیمایی است که یک مسافر را از نقطه ای به نقطهٔ دیگر در کمترین زمان جابجا میکند .پس اگر شما علاقمند باشید که زودتر به مقصد کاراتر می باشد .روش دوم مقایسه، برای این مثال تعداد مسافر جابجا concorde برسید واضح است که

شده در واحد زمان است .بر اساس این معیار همان طور که در ستون آخر جدول نیز نشان داده شده است 747 کاراتر می باشد.

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

به طور مشابه ما می توانیم کارآیی کامپیوترها را به طرق مختلف تعریف نمائیم .اگر شما یک برنامه را بر روی دو کامپیوتر مختلف اجرا کنید، خواهید گفت که کامپیوتری سریعتر است که آن برنامه را سریعتر اجرا می کند .ولی اگر برنامه خود را بر روی یک ترمینال در مرکز کامپیوتری اجرا کنید که دارای دو کامپیوتر است که به طور مشترک توسط چندین نفر استفاده می شوند در این صورت خواهید گفت که کامپیوتری سریعتر است که تعداد زیادتری برنامه را در روز اجرا کند. همان طور که یک کاربر که مدت زمان بین (execution time)یا همان زمان اجرا (response time) دوست دارد که زمان پاسخ شروع و خاتمه یک کار است، بر روی یک کامپیوتر برای برنامه او کمتر شود، در یک مرکز کامپیوتر، افزایش پیدا کند . (throughput)مدیر مرکز اغلب دوست دارد که تعداد کار انجام گرفته در واحد زمان___

مثال: برنامه دلخواه ما بر روی کامپیوتر A که فرکانس کلاک آن ۴۰۰ مگاهرتز است در ۱۰ ثانیه اجرا می شود. ما میخواهیم به یک طراح کامپیوتر کمک کنیم که یک ماشین به نام B بسازد که برنامه ما را در ۶ ثانیه انجام دهد .طراح کامپیوتر به ما گفته است که افزایش فرکانس کلاک امکان پذیر است ولی این افزایش فرکانس طراحی بخش های دیگر cpu را تحت تأثیر قرار خواهد داد و باعث خواهد شد که ماشین B برای اجرای این برنامه تعداد کلاک هایی در حدود ۱/۲ برابر ماشین A نیاز داشته باشد. ما چه فرکانس کلاکی را برای رسیدن به این هدف از طراح بخواهیم؟

$F_A = 4\ GHz$

زمان اجرا A = ۱۰ s

زمان اجرا B = ۶ s

تعداد چرخه ساعت B = تعداد چرخه ساعت × 1.2

تعداد چرخه ساعت = زمان اجرا × آهنگ پالس ساعت

تعداد چرخه ساعت A = ۱۰ × ۴ × $۱۰^۹$

تعداد چرخه ساعت B = $1.2 \times 4 \times 10^9 = 48$

$$F_B = \frac{\text{تعداد چرخه ساعت}}{\text{زمان اجرا}} = \frac{48 \times 10^9}{6} = 8 \times 10^9 \text{ GHz}$$

۴-۱ (Clock per Instructor) CPI

تعداد پالس (کلاک) متوسط لازم برای اجرای هر دستور. ((این مقدار، در هر برنامه با برنامه دیگر متفاوت است.))

تعداد کلاک متوسط لازم برای اجرای هر دستور (CPI) × تعداد دستورات برنامه = تعداد کلاک های لازم برای اجرای یک برنامه

عبارت تعداد کلاک های متوسط مورد نیاز برای هر دستور CPI نامیده می شود. چون دستورات مختلف بسته به اینکه چه کاری انجام می دهند، ممکن است زمان اجرای متفاوتی داشته باشند CPI متوسط زمان اجرای ((با واحد کلاک)) همهٔ دستوراتی است که در داخل یک برنامه قرار دارند. روشی برای مقایسه دو پیاده سازی مختلف از یک مجموعه دستورات را نیز در اختیار قرار می دهد، چون در این دو پیاده سازی تعداد دستورات مورد نیاز برنامه مطمئناً یکی خواهد بود.

مثال: فرض کنید ما دو پیاده سازی از یک مجموعه دستورات واحد در اختیار داشته باشیم. ماشین A برای یک برنامه دارای پریود کلاک ۱ نانوثانیه بوده، و CPI آن ۲ می باشد و ماشین B دارای پریود کلاک ۲ نانوثانیه بوده، و CPI آن ۱.۲ می باشد. برای این برنامه کدام ماشین سریعتر است و چقدر؟

می دانیم تعداد دستوراتی که هر کدام از ماشین ها برای این برنامه اجرا می کنند یکی است. ما این تعداد دستورات را I می نامیم. در ابتدا تعداد کلاک های پردازنده را برای هر کدام از ماشین ها حساب می کنیم:

تعداد کلاک های لازم برای اجرای برنامه بر روی ماشین A = I × ۲

تعداد کلاک های لازم برای اجرای برنامه بر روی ماشین B = I × ۱.۲

حال زمان اجرا برنامه را برای هر کدام از ماشین ها حساب می کنیم:

زمان اجرای برنامه بر روی ماشین A = متوسط تعداد کلاکهای لازم برای اجرای برنامه (CPI) × پریود کلاک

$$= I \times 2 \times 1 \text{ ns} = 2 \times I \text{ ns}$$

همین طور برای ماشین B داریم:

زمان اجرای برنامه بر روی ماشین B = متوسط تعداد کلا های لازم برای اجرای برنامه (CPI) × پریود کلاک

$$= I \times 1.2 \times 2 \text{ ns} = 2.4 \times I \text{ ns}$$

واضح است که ماشین A سریعتر است چون زمان اجرای پایین تری دارد. برای اینکه بدانیم A چقدر سریعتر از B است، به صورت زیر عمل می کنیم:

$$\frac{\text{کارایی ماشین A}}{\text{کارایی ماشین B}} = \frac{\text{زمان اجرای برنامه بر روی ماشین B}}{\text{زمان اجرای برنامه بر روی ماشین A}} = \frac{2.4 \times \text{I ns}}{2 \times \text{I ns}}$$

بنابراین ماشین A برای این برنامه، ۲.۱ برابر سریعتر ار ماشین B می باشد.

CPI متوسط:

$$\text{CPI متوسط} = \frac{\sum_{i=1}^{n} \text{CPI}_i}{n}$$

مثال: یک طراح کامپایلر می خواهد بین دو قطعه کد برای یک ماشین مشخص یکی را انتخاب نماید، بر اساس پیاده سازی سخت افزار، سه کلاس مختلف از دستورات وجود دارد: کلاس A ، کلاس B ، و کلاس C ، که به ترتیب برای اجرا شدن نیاز به ۱ ، ۲ ، و ۳ کلاک دارند، قطعه کد اول دارای ۵ دستور است: ۲ مورد از کلاس A ، ۱ مورد از کلاس B ، و مورد از کلاس C . و قطعه کد دوم دارای ۶ دستور است: ۴ مورد از کلاس A ، ۱ مورد از کلاس B ، ۱ مورد از کلاس C :

الف) کدام قطعه کد سریعتر اجرا میشود و چقدر؟

ب) CPI را برای هر قطعه کد حساب کنید؟

الف) برای بدست آوردن تعداد کلاک از فرمول زیر استفاده می کنیم:

$$\text{تعداد کلاک های لازم} = \sum_{i=1}^{n} (\text{CPI}_i \times C_i)$$

که در آن CPI_i تعداد کلاک های لازم برای اجرای دستورات از نوع کلاس i می باشد، و C_i تعداد دستورات کلاس i می باشد.

تعداد کلاک های لازم برای قطعه کد ۱ = ۱ × ۲ = ۲ × ۳ = ۱۰ کلاک

تعداد کلاک های لازم برای قطعه کد ۲ = ۴ × ۱ = ۲ × ۱ = ۳ × ۱ = ۹ کلاک

چون هر دو قطعه کد بر روی یک ماشین اجرا می شوند و در نتیجه پریود کلاک برای هر دو یکی است، بنابراین قطعه کدی سریعتر است که تعداد کلاک کمتری لازم داشته باشد. بنابراین در این مثال با اینکه قطعه کد ۲ تعداد دستورات بیشتری دارد ولی سریعتر اجرا می شود.

ب) برای بدست آوردن CPI از فرمول زیر استفاده می کنیم:

$$\text{CPI} = \frac{\text{تعداد کلاک ها}}{\text{تعداد دستورات}}$$

قطعه کد اول CPI = $\frac{10}{5} = 2$

قطعه کد دوم CPI = $\frac{9}{6} = 1.5$

مثال:

زمان اجرای A = ۱۵

تعداد دستورالعمل B = تعداد دستورالعمل A $\times$ 0.6

$$\text{CPI}_B = 1.1\ \text{CPI}_A$$

زمان اجرای A = تعداد دستورالعمل A × CPI_A × اندازه پالس ساعت

زمان اجرای B = تعداد دستورالعمل B × CPI_B × اندازه پالس ساعت

تعداد دستورالعمل $0.6 \times A$ × $1.1\ \text{CPI}_A$ × اندازه پالس ساعت $= 0.6 \times 1.1 \times 15 = 9.9$

۴-۲ معیار MFLOP (Million Floating Point Operation Per Second):

این معیار تعداد عمل ممیز شناور (برحسب میلیون عمل) در یک ثانیه را نشان داده، و برای کارهای علمی و تحقیقاتی ((که محاسبه دارند)) معیار مهمی است.

این علامت یک مقیاس قدرت محاسباتی است، که معمولاً برای به کامپیوترهای بسیار بزرگ بکار می رود.

۴-۲-۱ MIPS (million instructions per second):

این واحد تعداد دستور ((برحسب میلیون دستور)) در یک ثانیه را نمایش داده، کـه واحـدی بـرای سـنجش کارایی یک پردازنده است. این معیار جایگزین معیار زمان است.

یک نوع معماری پردازنده از نوع معماری ریسک بوده، که توسط شرکت میپس تکنولوژی طراحی شده است. نسخه‌های اولیه آن ۳۲ بیتی بوده، و بعدها نسخه ۶۴ بیتی آن هم عرضه شد. چندین نسخه از این معمـاری وجود دارد که عبارتنـد از: MIPS 1، MIPS 2، MIPS 3، MIPS 4، MIPS 5، MIPS 32 و MIPS 64 . نسـخه‌هـای فعلـی MIPS 32 و MIPS 64 هسـتند. در MIPS 32 وMIPS 64 عـلاوه بـر مجموعـه

دستورالعمل‌ها، یک مجموعه ثبات کنترلی هم وجود دارد. پردازنده‌های MIPS عمدتاً در سامانه‌های توکار از جمله دستگاه‌های ویندوز سی‌ای، مسیریاب‌ها، کنسول‌های بازی‌های ویدیویی همانند پلی استیشن، پلی‌استیشن ۲ و پلی‌استیشن پرتابل استفاده شده‌اند.

$$\text{MIPS} = \frac{\text{تعداد دستور}}{\text{زمان اجرا} \times 10^6}$$

MIPS در یک کامپیوتر خاص برای برنامه های متفاوت، مقادیر متفاوتی خواهد داشت.

$$\text{MIPS} = \frac{\text{تعداد دستور}}{\frac{\text{تعداد دستور} \times \text{CPI}}{\text{نرخ کلاک}} \times 10^6} = \frac{\text{نرخ کلاک}}{\text{CPI} \times 10^6}$$

۴-۳ معیار SPEC (System Performance Evaluation Cooperative) :

این معیار مجموعه ای شامل ۱۲ "آزمون صحیح" و ۱۷ "آزمون ممیز شناور"، و متمرکز بر کارایی پردازنده بوده، که طی ۵ نسل تکامل یافته است.

برای ساده سازی استفاده از نتیجه این آزمون ها، نتایج این ۱۲ آزمون صحیح، باهم تلفیق شده، و در قالب یک عدد واحد گزارش می شود.

عملا به عنوان معیار استفاده شده، که یکسری نمودار به ما می دهد که با هم مقایسه می شوند. این معیار، بهترین معیار برای ارزیابی کارایی یک سیستم واقعی است.

۴-۴ قانون امدال(Amdahl's law):

قانون امدال در معماری کامپیوتر قانونی است که می توان به کمک آن ارزش کارهایی که قرار است انجام شوند را از قبل پیش بینی نمود. این کارها عمدتاً مربوط به بهبودهایی است که در سختافزار پردازنده ها داده می شود. در این قانون، مقدار بهبودهایی که انجام می شود به میزان مؤثر بودن آنها محدود می شود.

این قانون نشان می‌دهد که مقدار تسریع در یک سامانه پردازش موازی دارای یک "حد حداکثر" است. قانون امدال برای سامانه‌های موازی، با بار محاسباتی ثابت به کار می‌رود. امدال نشان داد که اگر بار محاسیاتی ثابت باشد، و در زمانی که واحدهای موازی به بی‌نهایت میل کند، مقدار تسریع دارای یک حد حداکثر است و از آن مقدار بیشتر نمی‌شود.

برای توضیح این قانون، سیستم واحدی دانشگاه را در نظر می گیریم. در این سیستم بعضی دروس یک

واحدی، بعضی دو واحدی، بعضی سه واحدی و بعضی نیز چهار واحدی اند. همانطور که می دانیم در محاسبه معدل ترم، دروسی بیشتر تأثیر دارند که تعداد واحد آنها بالاتر است. بنابراین در این سیستم

ارزش دروس چهار واحدی بیشتر از سه واحدی، ارزش دروس سه واحدی بیشتر از دو واحدی،

ارزش دروس دو واحدی نیز بیشتر از یک واحدی است. بنابراین ما بهتر است بیشترین وقت و

سرمایه گذاری را بر روی دروسی انجام دهیم که بیشترین تأثیر را در معدل دارند.

در معماری کامپیوتر قانون امدال می گوید: قبل از اینکه بهبود و یا کاری را انجام دهی باید به میزان مؤثر بودنش نیز توجه کنی، و آن کار را وقتی انجام دهی که تأثیرش در مقابل کاری که انجام می دهی قابل توجه باشد. درقانون امدال، اگر جایی را بهبود دادید، فقط آنجا بهبود می یابد.

در قالب فرمول، قانون امدال به صورت زیر بیان می شود:

$$\text{زمان اجرای برنامه بعد از بهبود} = \frac{\text{زمان اجرای بهبود یافته}}{\text{میزان بهبود}} + \text{زمان اجرای بهبود نیافته}$$

مثال: فرض کنید که تصمیم گرفته باشیم سخت افزار پردازنده را تغییر دهیم تا سرعت اجرای دستورات Floating Point دو برابر شود. دو برابر کردن سرعت یک ایده خوب است ولی باید ببینیم در قبال کاری که انجام می دهیم چقدر بهبود در کل زمان اجرای یک برنامه به وجود می آید. به عبارتی باید ببینیم دستورات Floating Point چه میزان در برنامه ما نقش دارند. فرض کنید برای یک برنامه فرضی فقط ۱۰ درصد زمان اجرای برنامه (T) شامل دستورات Floating Point باشد. برای این برنامه تأثیر بهبود سخت افزار را بررسی کنید.

با استفاده از قانون امدال می توان نوشت:

$$\text{زمان اجرای برنامه بعد از بهبود} = \frac{0.10\ \mathrm{T}}{2} + 0.90\ \mathrm{T} = 0.95\ \mathrm{T}$$

یعنی زمان اجرای کل برنامه فقط ۵ درصد بهبود خواهد داشت .این میزان بهبود خیلی بالا نیست بنابراین بر طبق قانون امدال دو برابر کردن سرعت اجرای دستورات Floating Point به امید بهبود زمان اجرای برنامه در این مثال ارزش فنی ندارد.

نتیجه قانون امدال: قسمت هایی از برنامه را بهبود دهیم که بیشتر استفاده می شوند چون بیشترین تـأثیر را

در کل زمان اجرای برنامه دارند. قانون امدال بیان می کند که موارد پر استفاده را بهبود دهیم و یا به

عبارتی می گوید: Make the common case fast .

$$\frac{80}{5} + 20 = 36$$

مثال:

	A	B	C
CPI	۱	۲	۳

	A	B	C	
کامپایلر ۱	۲	۱	۲	۵×۱۰^{۹}
کامپایلر ۲	۴	۱	۱	۶×۱۰^{۹}

F = 4 GHz

چرخه ساعت کد ۱ = $\left(2 \times ۱ + 1 \times 2 + 2 \times ۳\right) = 10 \times 10^9$

تعداد چرخه ساعت کد ۲ = $\left(4 \times ۱ + 1 \times 2 + ۱ \times ۳\right) = 9 \times 10^9$

کد ۱ سریع تر است.

زمان اجرای ۱ = $\frac{10 \times ۱۰^{9}}{5 \times ۱۰^{9}} = 2$

زمان اجرای ۲ = $\frac{9 \times ۱۰^{9}}{6 \times ۱۰^{9}} = 1.5$

$$\text{MIPS } 1 = \frac{5 \times 10^9}{2 \times 10^6} = 2500$$

$$\text{MIPS } 2 = \frac{6 \times 10^9}{1.5 \times 10^6} = 4000$$

MIPS دومی بیشتر است.

تمرین : تفاوت معماری دو معماری RISC و CISC :

۱- بدون آرگومان (پشته ای)

۲- تک آرگومان (انباره ای)

۳- دو آرگومان (CISC)

۴- سه آرگومان (RISC)

تفاوت های CISC و RISC :

با ظهور کامپیوترهای براساس پردازندهٔ PowerPC به عنوان رقیبی جدی برای‌کامپیوترهای مبتنی بر پردازنده‌های اینتل این بحث داغ شده است که پردازنده‌های RISCبهترند یا پردازنده‌های CISC.

RISC علامت اختصاری کلماتی است که در فارسی به صورت «پردازنده‌های کم‌دستورالعمل» ترجمه کرده‌ایم و بدین معناست که تعداد دستورالعملهایی که چنین‌پردازنده‌هایی می‌توانند بفهمند کمتر از تعداد دستورالعملهای پردازنده‌های معمول یاپردازنده‌های مشهور به CISC است. در مقابل، CISC علامت اختصاری کلماتی است که درفارسی به صورت «پردازنده‌های پر دستورالعمل» ترجمه کرده‌ایم. سازندگان‌پردازنده‌های RISC ادعا می‌کنند و به طور کلی از CISCها قویترند. اما اینتل و سایرطرفداران CISC سرسختانه عکس این مطلب می‌اندیشند.

آنچه در این مباحثات نادیده گرفته شده است اختلافات اصلی بین تراشه‌های RISCو CISC است. آیا تعداد دستورالعمل اختلاف اصلی بین این دو گروه از پردازنده‌هاست؟اگر چنین باشد، آیا بدین معنی نیست که پردازندهٔ RISC نرم‌افزار را آهسته‌تر اجرا می‌کند،چون باید تعداد بیشتری دستورالعمل را برای انجام کارهای پیچیده اجرا کند؟ مزیت تعداددستورالعمل کم، اگر مزیتی باشد، چیست؟ اصلاً مجموعه دستورالعمل چیست؟

تحقیق زیر سعی می‌کند به این پرسشها پاسخ دهد و مرزهای بین پردازنده‌های RISCو CISC را مشخص سازد. اگر تا به حال پاسخ به این سوالات را نمی‌دانسته‌اید حقایقی که‌خواهد آمد ممکن است شما را متعجب کند.

طرحان کامپیوترهای کم دستور (RISC) مدعی‌اند که به دو تا سه‌برابر کارآیی سیستم کامپیوترهای پیچیده دستور (CISC) با همان میزان پالس ساعت‌دست یافته‌اند. این افزایش کارآیی از طریق لوله‌گذاری

(Pipelining) که یک شکل شناخته‌شده موازی بودن و بهینه‌سازی مترجم‌هاست و معماری خاص RISC حاصل شده است.دستورهای RISC طوری طراحی شده که طول پایدار و با ثبات و زمان‌های اجرای باثباتی‌دارند. معماری RISC، ثابت بودن طول دستورالعمل و ثابت بودن زمان اجرا باعث‌می‌شوند تا بهینه سازی کامپایلر، که عامل اساسی در تعیین سرعت یک سیتسم‌کامپیوتری است، ساده‌تر شود.

طرفداران CISC، نه! طراحان CISC با دقت در حال ارزیابی خویشند تا دستوراتی راکه بیشتر از همه اجرا می‌شوند را مشخص کنند و با بهینه باسزی میکروکد یا پیاده‌سازی‌سخت‌افزاری (hardwiring) دستورالعمل ها، سرعت اجرای آنها را بالا ببرند. مدافعین CISCبراین باورند که می توانند به میانگین سیکل ساعت هر دستور، مانند مشابه آن در طراحی،RISC، دست یابند. این به معنای دستیابی به کارآیی، بدون قربانی کردن سازگاری باتعداد وسیع نرم‌افزارهای کامپیوترهای شخصی، است. (قابل ذکر است که هنوز تعدادنرم‌افزارهای همه منظوره برای ماشین‌های RISC بسیار کمتر از انواع CISC است).

افزون بر این بزرگ بودن مجموعه رجیسترهای ماشین‌های RISC باعث بروزاختلال در یک محیط تک کاربردی چند کاره می‌شود، چرا که محتوای یک مجموعه‌رجیستر، هنگامی که کاربر بین وظایف سوئیچ می‌کنند، باید ذخیره شود. در CISC پیچیدگی سخت افزار و در RISC پیچیدگی نرم افزاری وجود دارد.

برای عمل ضرب در سیستم های با معماری ریسک باید حتما از یک واحد سخت افزاری برای ضرب استفاده کرد. در سیستم های RISC برای انجام عملیات ضرب سیستم کامپایلر به این گونه رفتار می کند که یک حلقه FOR ایجاد کرده و ۲ را دو بار در خود ضرب می کند که این امر باعث سادگی سخت افزاری و سرعت بالا می شود. در تلفن های هوشمند از تکنولوژی RISC و در سیستم های دسکتاپ از تکنولوژی CISC استفاده شده است. تکنولوژی RISC باعث کم مصرف شدن باطری و پردازش سریع است و تکنولوژی CISC دارای قدرت پردازش بالا و مصرف انرژی بیشتری می شود. در کامپیوترهای قدیمی مک از پردازنده PPC استفاده می کردند که RISC بود. پردازنده‌های SPARC هم که در سرورهای Oracle (یا Sun قدیم) ارائه می شدند RISC هستند.

پیوست ۱

روش CRT – I (روش چینی تسریع شده):

$$X \equiv^{(M_1,M_2,\dots,M_n)} (r_1, r_2, \dots, r_n)$$

X از رابطه زیر بدست می آید:

$$X = r_1 + m_1 < k_1(r_2 - r_1) + k_2 m_2(r_3 - r_2) + \cdots + k_{n-1} m_2 m_3 \dots m_{n-1}(r_n - r_{n-1}) >_{m_2 m_3 \dots m_n}$$

که k_i به صورت زیر با محاسبه معکوس ضربی بدست می آید:

$$< k_1 m_1 >_{m_2 m_3 \dots m_n} \equiv 1$$

$$< k_2 m_1 m_2 >_{m_3 m_4 \dots m_n} \equiv 1$$

$$< k_{n-1} m_1 m_2 \dots m_{n-1} >_{m_n} \equiv 1$$

مثال:

$$X \equiv^{(2,3,5,7)} (1,0,4,0) \Longrightarrow |k_1 m_1|_{m_2 m_3 m_4} \equiv 1 \Longrightarrow |k_1 \times 2|_{105} \equiv 1 \Longrightarrow k_1 = 53$$

$$|k_2 m_1 m_2|_{m_3 m_4} \equiv 1 \Longrightarrow |k_2 \times 2 \times 3|_{35} \equiv 1 | 6k_2|_{35} \equiv 1 \Longrightarrow k_2 = 6$$

$$|k_3 m_1 m_2 m_3|_{m_4} \equiv 1 \Longrightarrow |k_3 \times 2 \times 3 \times 5|_7 \equiv 1 \Longrightarrow |30k_3|_7 \equiv 1 \Longrightarrow k_3 = 4$$

$$X = r_1 + m_1 < k_1(r_2 - r_1) + k_2 m_2(r_3 - r_2) + k_3 m_3 m_2(r_4 - r_3) >_{m_2}$$

$$X = 1 + 2 < 53 \times (0 - 1) + 6 \times 3 \times (4 - 0) + 4 \times 5 \times 3 \times (0 - 4) >_{105}$$

$$X = 1 + 2 \times (1 - 221)_{105} = 1 + 2 \times 94 = 189 \Longrightarrow X = 189$$

روش Mixed – Radix conversion:

$$* \; X \equiv^{(M_1,M_2,\dots,M_n)} (r_1, r_2, \dots, r_n)$$

فرض کنیم $(M_1, M_2, \dots, M_n)$ پیمانه‌های ما هستند که نسبت به هم اول هستند و $(r_1, r_2, \dots, r_n)$ مانده های ما در پیمان مذکور باشند بطوری که: $r_i \equiv^{mi} |X|$ هدف یافتن عدد X است.

فرض کنیم x نمایشی بصورت

$$** X = z_1 + z_2m_1 + z_3m_2m_1 + \cdots + z_nm_{n-1}m_{n-2} \dots m_1$$

داشته باشد که در آن $0 \le zi \le mi$ می باشد.

هدف یافتن ضرایب مجهول zi بصورتی است که معادله * برقرار باشد.

اگر از دو طرف رابطه ** به پیمانه m_1 همنهشتی بگیریم: $q \equiv^{M_1} X \equiv^{M_1} Z_1$ پس: $z_1=r_1$

سپس با محاسبه z_1 رابطه ** را بصورت

$** X = z_1 + z_2m_1 + \cdots + z_nm_{n-1}m_{n-2} \dots m_1$ می نویسیم حال از رابطه قبلی بر پیمانه m_2 همنشتی می گیریم که بدست می آید

$X - Z_1 \equiv^{M_2} z_2m_1 + 0 + 0 + \cdots$ چون اینها ضرایب m_2 دارند.

پس: $X - Z_1 \equiv^{M_2} z_2m_1$

با ضرب رابطه فوق در معکوس ضربی m_1 در پیمانه m_2:

$$|m_1^{-1}|_{m2}(x - z_1) \equiv^{m2} z_2$$

چون می دانیم $X \equiv^{M2} r_2$ پس رابطه با r بصورت: $z_{2\equiv^{m2}}|m_1^{-1}|_{m2}(r_2 - r_1)$ به همین ترتیب z_3 :

$$Z_3 \equiv^{m3} |(m_2m_1)^{-1}|_{m3} \times (r_3 - (z_2m_1 + z_1)_{m3}$$

و به همین ترتیب تا Z_n :

$$z_n \equiv^{mn} |(m_1m_2 \dots m_{n-1})^{-1}|_{mn} \times (r_n - (z_{n-1}m_{n-2}m_{n-3} \dots m_1 + z_{n-2}m_{n-3}m_{n-4} \dots m_1 + \cdots + z_2m_1 + z_1))|mn$$

با استفاده از رابطه ی $|(mi\ mj)^{-1}|_{mk} = ||mi^{-1}|_{mk}|mj|_{mk}|_{mk}$

$z_1 = r_1$ و $z_2 = ||mi^{-1}|_{m2}(r_2 - z_1|_{m_2}$

$* z_3 = ||m2^{-1}|_{m3}|m1^{-1}|_{m2}|_{m3}(r_3 - (z_2m_1 + z_1))_{m3}$

$* z_n = ||m_{n-1}^{-1}|_{mn}(|m_{n-2}^{-1}|_{mn}(|m_{n-3}^{-1}|_{mn}(\dots(|m_2^{-1}|_{mn}(|m_1^{-1}|_{mn}(r_n - z_1) - z_2 \dots) - z_{n-1}|_{mn}$

مثال برای روش Mixed –Radix :

$X \equiv^{(2,3,5,7)} (1,0,4,0)$

R= (1,0,4,0)

M= (2,3,5,7)

$X= (x_1, x_2, x_3, x_4) = ?$

$z_1 = r_1 = 1 \implies z_1 = 1$

$z_2 = ||mi^{-1}|_{m2}(r_2 - z_1|_{m_2} \implies z_2 = ||2^{-1}|_3(0-1|_{3=}|2x-1|_3 = 1 \implies z_2 = 1$

$* z_3 = ||m2^{-1}|_{m3}|m1^{-1}|_{m3}|_{m3}(r_3 - z_1) - z_2)_{m3} = ||3^{-1}|_5|2^{-1}|_5(4-1) - 1)|_5|(2) \times ((3) \times 3 - 1)|_5 = 1 \implies z_3 = 1$

$z_4 = ||m_3^{-1}|_{m4}(|m_2^{-1}|_{m4}(|m_1^{-1}|_{m4}(r_4 - z_1) - z_2)z_3)|_{m4} = ||5^{-1}|_7(|3^{-1}|_7(|2^{-1}|_7(0-1) - 1) - 1)|_7 = |3 \times (5 \times (4 \times (-1) - 1) - 1)|_7 = |-78|_7 = 6 \quad z_4 = 6$

$X = Z_4 m3 + m2 + m1 + Z_3 m2 + m1 + Z_2 m1 + Z_1 = 6 \times 5 \times 3 \times 2 + 1 \times 3 \times 2 + 1 \times 2 + 1 = 189 \quad X = 189$

روش معکوس با استفاده از مجموعه پیمانه های $\{2n+1, 2n, 2n-1\}$:

ایـن روش پیمانـه هـا بصـورت $\{2n+1, 2n, 2n-1\}$ هسـتند حالـت خاصـی از آن پیمانـه هـا $\left\{2^m + 1 و 2^m و 2^m - 1\right\}$ است در سخت افزار و پیاده سازی مدار است کاربرد زیادی دارد.

$$X \equiv^{(m1,m2,m3)} (r_1, r_2, r_3)$$

در این حالت جواب بصورت $X = \left|\sum_{i-1}^{3} wiri\right|_{m1m2m3}$ خواهد بود که اگر فرض کنیم

$$\begin{cases} M1= 2N+1 \\ M2= 2N \\ M3= 2N-1 \end{cases}$$

آنگاه wi بصورت زیر محاسبه می شوند.

$w_1 = \frac{(m_1+1)(m_2 m_3)}{2}$

$w_2 = m_1(m_2 - 1)m_3$

$w_3 = \frac{m_1 m_2 \times (m_3+1)}{2}$

در نهایت خواهیم داشت:

$$X = \left|\sum_{i-1}^{3} wiri\right|_{m1m2m3}$$

$$X \equiv^{(9,8,7)} (1,2,3)$$

مثال:

$$\text{چون } \begin{cases} 7 = 2 \times 4 - 1 \\ 8 = 2 \times 4 \\ 9 = 2 \times 4 + 1 \end{cases} \xrightarrow{\{2n+1,2n,2n-1\}} \begin{cases} m_3 = 2n - 1 = 7 \\ m_2 = 2n = 8 \\ m_1 = 2n + 1 = 9 \end{cases}$$

$$\text{طبق فرمولهای قبل } \begin{cases} w_1 = \dfrac{(7+1)(8 \times 9)}{2} = \dfrac{560}{2} = 280 \\ w_2 = 7 \times (8-1) \times 9 = 441 \\ w_3 = \dfrac{7 \times 8 + (10)}{2} = 288 \end{cases}$$

$$X = |w_1 k_1 + w_2 k_2 + w_3 k_3|_{9 \times 8 \times 7}$$

$$X = |280 \times 1 + 441 \times 2 + 288 \times 3|_{504}$$

$$X = |2026|_{504} = 15 \implies X = 15$$

تمرین: چگونه با RNS تقسیم انجام دهیم؟ ابتدا مفهومی معکوس ضربی را تعریف می کنیم.

تعریف: معکوس ضربی X^{-1} را معکوس ضربی X در پیمانه M می نامیم اگر $XX^{-1} \equiv^{M} 1$ مثلا معکوس ضربی ۷ در پیمانه ۱۱ می شود ۸ چون: $7 \times 8 \equiv^{11} 1$ برای تقسیم X در پیمانه m بر y بصورت زیر عمل می کنیم:

فرض کنیم $\frac{x}{y} \equiv^{m} q$

$$\implies x \equiv^{m} y \times q$$

اگر دو طرف رابطه بالا را در معکوس ضربی y ضرب کنیم.

$$\implies x \times y^{-1} \equiv^{m} y \times y^{-1} \times q \xrightarrow{y \times y^{-1} \equiv^{m} 1} x \times y^{-1} \equiv^{m} q$$

البته باید پیمانه طوری باشد که با Y مقسوم علیه مشترک نداشته باشند چون در اینصورت هنگامی که Y را به پیمانه انتخاب شده ببریم صفر می شود و تقسیم بر صفر تعریف نشده است.

پس از این رابطه می توانیم بجای $\frac{x}{y} \equiv^{m} q$ استفاده کنیم حال فرض کنیم پیمانه سه تایی (m_2, m_1, m_0) را داریم و می خواهیم نشان دهیم $\frac{x}{y}$ RNS است پس بجای $\frac{x}{y}$ می توانیم $x \times y^{-1}$ را به پیمانه (m_2, m_1, m_0) قرار دهیم:

$$\frac{x}{y^{-1}} \rightarrow \quad x\begin{cases} x = m_2k_2 + x_2 \\ x = \ m_1k_1 + x_1 \\ x = \ m_0k_0 + x_0 \end{cases}$$

$$y^{-1}\begin{cases} y^{-1} = m_2k_2 + y_2^{-1} \\ y^{-1} = m_1k_1 + y_1^{-1} \\ y^{-1} = m_0k_0 + y_0^{-1} \end{cases}$$

$$x \times y^{-1} = \ m_2^2k_2k_2 + x_2 \times y_2^{-1} + m_2k_2y_2^{-1} + m_2k_2x_2 = m_2(m_2k_2k_2 + k_2y_2^{-1} + k_2x_2) + x = m_2k_2 + x_2 \times y_2^{-1}$$

$$x \times y^{-1} = m_1^2k_1k_1 + m_1k_1y_1^{-1} + m_1k_1x_1 + x_1y_1^{-1} = m_1(m_1k_1k_1 + k_1y_1^{-1} + k_1x_1) + x = m_1k_1 + x_1 \times y_1^{-1}$$

$$x \times y^{-1} = m_0^2k_0k_0 + m_0k_0y_0^{-1} + m_0k_0x_0 + x_0y_0^{-1} = m_0(m_0k_0k_0 + k_0y_0^{-1} + k_0x_0) + x = m_0k_0 + x_0 \times y_0^{-1}$$

$$X \times y^{-1} \equiv^{(m_2,m_1,m_0)} (x_2 \times y_2^{-1}, x_1 \times y_1^{-1}, x_0 \times y_0^{-1}$$

مثال:تقسیم ۲۰ بر ۵ به پیمانه (۲و۳و۷) با روش RNS؟

ابتدا معکوس ضربی ۵ را به پیمانه ۷ و ۳ و ۲ بدست می آوریم:

$$5 \times 3 \equiv^7 1$$

پس معکوس ضربی ۵ به پیمانه ۷ برابر ۳ می شود

$$5 \times 2 \equiv^3 1$$

پس معکوس ضربی ۵ به پیمانه ۳ برابر ۲ می شود

$$5 \times 1 \equiv^2 1$$

پس معکوس ضربی ۵ به پیمانه ۲ برابر ۱ میشود

پس معکوس ضربی ۵ به پیمانه (۲و۳و۷) برابر (۱و۲و۳) می شود حال حاصلضرب ۲۰ در معکوس ضربی ۵ را بدست می آوریم

۲۰ به پیمانه (۲و۳و۷) برابر (۰و۲و۶) می شود پس :

حاصلضرب به پیمانه (۲و۳و۷) می شود:(۰و۱و۴)