معماری کامپیوتر پیشرفته

سیستم حافظه :

- اجزاء سیستم حافظه
- ساختار حافظه (ساختار درختی حافظه)
- حافظه های نهان Cach memory
- الگوریتم جایگزینی
- استراتژی های نوشتن در حافظه

حافظه اصلی :

سریع - دسترسی تصادفی - گران - درون CPU : از این نوع حافظه ها برای ذخیره سازی برنامه و داده هایی استفاده می شود که هم اکنون به پردازش توسط CPU نیاز دارند.

حافظه های جانبی :

کند - ارزان - دسترسی مستقیم و به دور از CPU قرار دارند.

مشکلات سیستم های حافظه :

ما به چه چیزی احتیاج داریم ؟

ما به حافظه ای نیاز داریم که بتوانیم مقادیر بسیار زیادی از برنامه ها را با سرعت مناسب به میکرو پروسسورها جا دهیم.

مشکل اصلی این نوع حافظه ها :

- میکرو پروسسور با سرعت بالایی کار می کند و به حافظه خیلی بزرگی نیاز دارد.
- حافظه ها از میکرو پروسسور ها کند ترند.

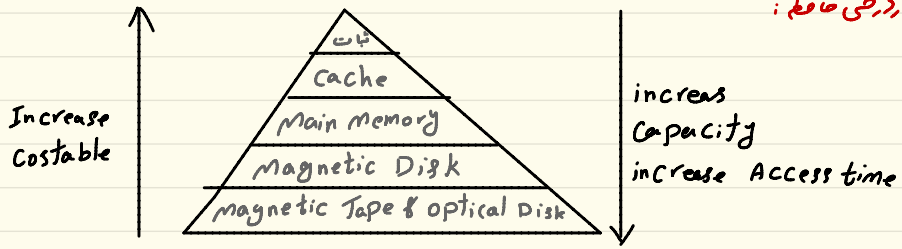
واقعیت ها :

- حافظه های بزرگتر کند تر نیز هستند.
- حافظه های سریع تر به ازاء هر بیت هزینه بیشتری دارند.

یک راه حل :

این امکان وجود دارد که ما بتوانیم حافظه های مرکبی درست کنیم که این حافظه ها از ترکیب حافظه های کوچک و سریع و حافظه های اصلی کند و با حجم زیاد باشند اما رفتاری مشابه حافظه های حجم بالا و سریع داشته باشند. براساس دو اصل حافظه وسط این دو اصل ما می توانیم ساختاری از حافظه ایجاد کنیم که شامل کلیه حافظه جانبی باشد (Disk store)

ساختار درختی حافظه :



ویژگی‌های عمومی:

ثبات های پروسسور Register Processor:

۱- شامل ۳۲ ثبات ۳۲ بیتی = ۱۲۸ KB

۲- زمان دسترسی: کمتر از نانو ثانیه

حافظه نهان درون CPU L1 cache (on-chip cache memory)

- ظرفیت آن ها ۱۱۸ الی ۳۲ کیلو بایت

- زمان دسترسی، حدوداً ۱۰ نانو ثانیه

حافظه های نهان بیرون CPU L2 cache (off-chip cache memory):

- ظرفیت: در حدود ۱۰۰ KB

- زمان دسترسی: ده ها نانو ثانیه

حافظه اصلی Main Memory:

ظرفیت: ده ها گیگا بایت

زمان دسترسی: در حدود ۱۰۰ نانو ثانیه

دیسک سخت: Hard Disk

ظرفیت: ترا بایت

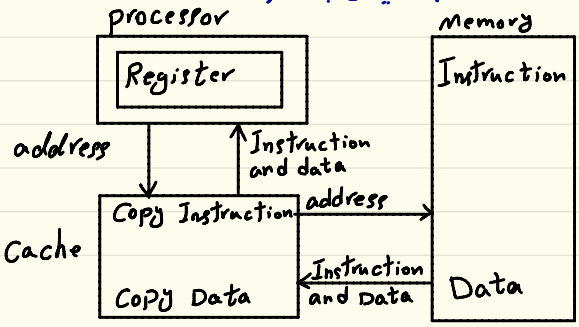
زمان دسترسی: ده ها میلی ثانیه

ساختار حافظه:

- داده هایی که در ثبات نگهداری می شوند تحت کنترل مستقیم کامپایلر و یا برنامه نویس قرار دارند.
- محتوی بقیه بخش های سلسله مراتب حافظه به صورت اتوماتیک مدیریت می شوند.
- کنترل مدیریت انتقال داده ها و دستورالعمل ها بین حافظه های نهان به عهده سخت افزار می باشد.
- کنترل انتقال اطلاعات از حافظه اصلی به حافظه های پشتیبان بر عهده سیستم عامل می باشد.

حافظه نهان:

حافظه نهان کوچک، خیلی سریع می باشد و در خود یک کپی از اطلاعاتی که اخیراً مورد استفاده قرار دارد از حافظه اصلی دارد. این که کدام مقدار از زمان در حافظه نهان نگهداری شود و یا کدام خارج شود بدون دخالت برنامه نویس انجام می گردد.



CPU فقط بر روی آیتم هایی که در Cache قرار دارند فرآیند می شود می توان بازدهی را بالا برد.

به طور کلی کارایی یک سیستم وابستگی بسیار زیادی به نحوه دسترسی به حافظه نهان توسط CPU دارد

- فعالیت های پردازنده زمانی بالاترین نرخ اجرا را خواهند داشت که تمامی آیتم هایی که پردازنده به آن نیاز دارد در حافظه نهان وجود داشته باشد.
کارایی یک سیستم وابستگی بسیار زیادی با استراتژی های بکار رفته توسط پردازنده برای دسترسی به حافظه نهان دارد.

hit : دسترسی به آیتم توسط پردازنده در حافظه نهان

miss : عدم دسترسی به یک آیتم در حافظه نهان توسط پردازنده

نسبت دسترسی به همه آیتم های مورد نیاز پردازنده در حافظه نهان را hit Rate و عکس آن را Mis Rate گویند.

- بهترین حالت حافظه نهان زمانی است که Mis Rate کمترین مقدار خود را داشته باشد.

- حافظه نهان بسیار کوچک است درحدود k و حافظه Ram در حد MB می باشد.

یک آیتم زمانی در حافظه نهان قرار داده می شود که به آن نیاز باشد.

این فرایند چطور انجام می شود؟

جواب محلیت یا locality می باشد.

در طی فرایند اجرای برنامه حافظه توسط پردازنده آدرس دهی میگردد، این آدرس دهی هم برای دستورالعمل ها و هم برای داده ها می باشد.

محلیت موقت temporal locality: اگر پردازنده به یک آیتم در حافظه ارجاع داشته باشد به زودی به همان آیتم ارجاع خواهد داشت. (محلیت در زمان)

محلیت خاص spacial locality: (محلیت در فضا) اگر پردازنده به آدرسی در حافظه ارجاع داشته باشد به آیتم های نزدیک آن نیز ارجاع خواهد داشت.

مشکلات حافظه نهان:

- چه مقدار حافظه نهان نیاز می باشد؟
- اگر hit یا miss داشته باشیم چطور برخورد کنیم؟
- اگر یک miss داشته باشیم و فضای خالی برای جایگزینی آیتم وجود نداشته باشد اطلاعات چطور به آیتم جایگزین گردد؟
- چطور باید بین حافظه نهان و Ram سازگاری بوجود آوریم.

جداسازی حافظه نهان داده و دستورالعمل:

در این روش یک حافظه نهان مستقل برای داده و یک حافظه نهان برای دستورالعمل در نظر گرفته می شود.

مزایای مجتمع بودن حافظه های نهان داده و دستورالعمل:

توانایی بهتری برای ایجاد توازن بین واکشی دستورالعمل و داده ها برای اجرای پویای برنامه ها دارد.

- طراحی و پیاده سازی آن ساده تر است.

مزایای جداسازی حافظه های نهان:

رقابت پردازنده برای واکشی دستورات و داده ها کاهش می یابد

واکشی داده ها می تواند به صورت سوزنی و همزمان با دسترسی به حافظه توسط دستورالعمل اجرا انجام می گیرد

سازمان دهی حافظه نهان: (ساختار حافظه نهان)

مثال:

- حافظه نهان ما 64 KB می باشد.
- داده ها بین cache و حافظه اصلی توسط بلوک های 4 byte بایتی جابجا شوند.

در این مثال ما اصطلاحاً می گوئیم حافظه نهان با خطوط ۴ بایتی سازمان دهی شده است.

- حافظه اصلی هم 16 MB باشد.

هر بایت با ۲۴ بیت آدرس دهی می شود $\Leftarrow$ $2^{24} = 16$ MB

در نتیجه:

- حافظه نهان شامل 2^{14} (16 K) بلوک (خط) می باشد.
- حافظه اصلی شامل 2^{22} (4 MB) بلاک می باشد.

سوال:

۱- وقتی می خواهیم یک بلوک را از حافظه اصلی به cache بیاوریم. آن وقت بلوک را در کدام قسمت cache می بایست قرار داد؟

۲- وقتی در یکی از آدرس های حافظه به دنبال محتوی می گردیم

- در کدام خط (بلوک) cache می بایست جستجو کنیم.
- چطور می توانیم بفهمیم چه مقدار از اطلاعات ما miss و چند درصد hit داشته باشیم.

مدیریت حافظه کش به روش Direct mapping (نگاشت مستقیم)

- در این روش هر خانه از حافظه اصلی بر روی یک بلوک ثابت از حافظه نهان نگاشت می شود.
- یک خانه حافظه ترکیبی از سه فیلد دیگر است:

۱- کم ارزش ترین بیت ها (در مثال فوق ۲ بیت) شناسه یک بایت در بلوک حافظه کش می باشد 2 bit

۲- در مثال فوق ۲۲ بیت دیگر شناسه بلوک در حافظه اصلی می باشد.

در منطق کش این ۲۲ بیت خود به دو بخش دیگر تقسیم می شود:

بخش اول: ۱۴ بیت $\Leftarrow$ یک خط کش را مشخص می کند.

بخش دوم: ۸ بیت $\Leftarrow$ یک tag است که به همراه خط در کش ذخیره می شود.

* از tag برای مشخص کردن مقداری که می بایست در خط کش قرار گیرد استفاده می شود.

مزایای Direct mapping:

- ساده و ارزان
- فیلد tag بسیار کوچک کردن، از tag برای مقایسه hit و miss استفاده می شود.

معایب Direct mapping:

- به ازاء هر بلوک در رم فقط یک بلوک مخصوص در کش وجود دارد که این امر ممکن است باعث خالی بودن بسیار زیاد cache شود.

در این روش امکان بالا رفتن miss بسیار زیاد است و به طبع آن hit پایین می آید.

نگاشت انجمنی مجموعه: Set Associative mapping

در این روش به ازای هر چند بلاک در آیک یا چند بلاک در Cache وجود دارد.

- یک بلاک حافظه در هر خطی از مجموعه می تواند نگاشت شود.
- اگر یک بلاک بخواهد در یک خط خاص از حافظه نهان قرار گیرد از الگوریتم های جایگزینی استفاده می شود
- آدرس حافظه باسه فیلد در Cache تفسیر می شود هانند روش Direct mapping
- چندین tag برای تعیین hit, miss وجود دارد
- تعداد خط های مجموعه با طراح می باشد : ۲ خط در طرف set Associative
۴ خط : set Associative چهار طرفه

این روش مزایای Direct map را دارای باشد :

۱- فیلد tag کوتاه
۲- دسترسی سریع
۳- نسبتاً ساده می باشد

- SAM سی در حد فاصل کمبود های DM دارد که یکی از کمبود وجود چند بین خط در حافظه نهان برای یک بلاک در Ram است.
- مدار حافظه نهان برای پیاده سازی SAM نسبت به DM پیچیده تر است.
- در مدل SAM های ۲ تایی و ۴ تایی بهترین نتیجه را داده اند. افزایش تعداد خط ها در مجموعه باعث افزایش کارایی نمی گردد.

Associative mapping ،

در این روش هر بلاک از Ram می تواند برادی یک خط حافظه نهان نگاشت شود
- اگر یک بلاک بخواهد در یک خط قرار گیرد از الگوریتم های جایگزینی استفاده می شود.
- آدرس دهی حافظه توسط دو فیلد در منطق حافظه نهان تفسیر می گردد.

- مزایای روش :
- این روش بالاترین انعطاف را برای جایگزینی یک خط حافظه نهان به هنگام خواندن یک بلاک جدید در Cache دارد

معایب :

- پیچیده می باشد.
- فیلد tag بسیار بزرگ می باشد.

- الگوریتم های جایگزینی :

وقتی یک بلاک می خواد در Cache قرار گیرد ، ممکن است در آن خط بلاک دیگری وجود داشته باشد
- در روش DM با انتخاب دیگری نخواهیم داشت
- با دو روش دیگر این جایگزینی نیاز به الگوریتم های خاص خود را دارد
- در روش SAM هر یک مجموعه از خط ها را می توانیم برای جایگزینی کاندید کنیم و در روش AM ها می توانیم از تمامی خط ها برای جایگزینی استفاده کنیم

الگوریتم های جایگزینی :

جایگزینی تصادفی : یک خط به صورت تصادفی کاندید می گردد.

نمای روش های دیگر مبتنی بر گذشته بلاک های باشه :

LRU : Least Recently used : خطی جایگزین می گردد که برای مدت طولانی آدرسی دهی نشده باشه .

Fifo : خطی جایگزین می گردد که برای مدت زمان طولانی در Cache مانده باشه

LFU : Least Frequently used : خطی جایگزین می گردد که کمترین ارجاعات را داشته باشه .

- الگوریتم های جایگزینی توسط مدیریت Cache به صورت سخت افزاری پیاده سازی می شوند

- LRU کارایی بیشتری دارد و پیاده سازی آن ساده. تو فیچد بهتری نیز خواهد داشت

- Fifo پیاده سازی ساده ای دارد

- جایگزینی تصادفی پیاده سازی بسیار ساده ای دارد و در عین حال فیچد بسیار شگفت آوری نیز در بر خواهد داشت .

استراتژی های نوشتن:

write-through : تمامی فرآیندهای نوشتن به حافظه اصلی ارجاع داده می شوند، اگر آدرسی در حافظه تغییر کند این تغییر به همان آدرس بلافاصله در Cache نیز تغییر خواهد کرد.

write-through with Buffered write: در روش قبلی پردازنده برای نوشتن با کندی حافظه اصلی روبرو بود در این روش داده ها در یک بافر سریع که بین CPU و حافظه اصلی قرار دارد نوشته می شود و این بافر خود وظیفه انتقال اطلاعات به حافظه اصلی را بر عهده می گیرد .

این روش سرعت بالا و پیچیدگی سخت افزاری بیشتری دارد.

copy back : در این روش تا زمانی که نیاز به جایگزینی اطلاعات در Cache نباشد، باقی می مانند در این روش عملیات نوشتن فقط وقتی انجام می شود که اختلافی بین Cache و Ram وجود داشته باشد. این روش کارایی بالا و پیچیدگی سخت افزاری زیادی دارد. مسئله انسجام بین Cache و Ram بسیار دقیق. تو سخت ترین آن است که برای حل آن نیاز به سیستم های چند پردازنده می باشد.

خطلوله : Pipeline

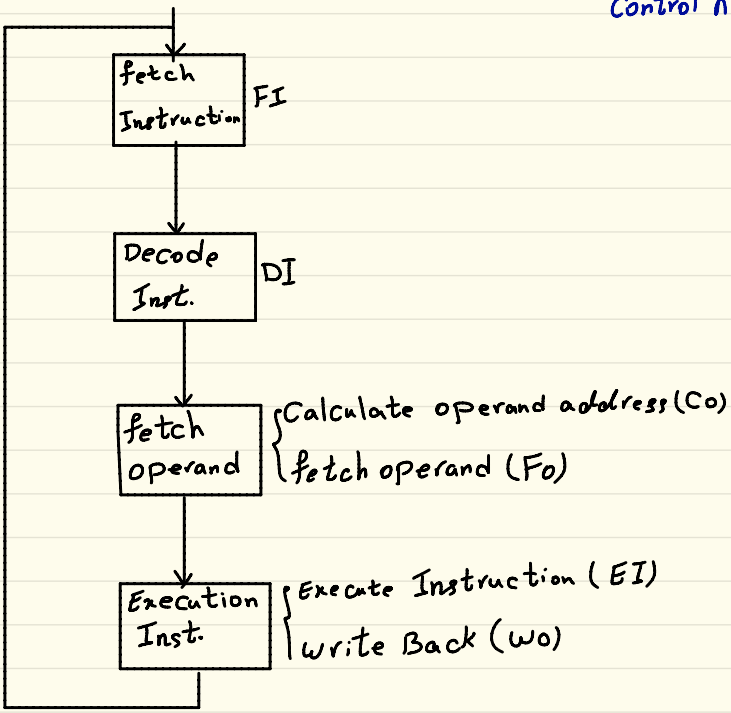
- جرة دستور العمل
- خطلوله دستور العمل ها
- خطات خطلوله (Hazards)

structure hazard.

Data hazard.

Control hazard.

جرة دستور العمل :



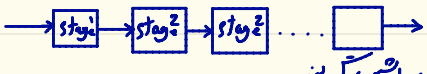
خط لوله دستورالعمل :

- دستورالعمل ها می توانند بسیار پیچیده باشند و شامل چندین عملیاتی که می بایست با موفقیت اجرا شوند. برای اجرای این چنین دستورالعمل هایی به صرف افزارهای زیادی نیاز می باشد که در این قسمت فقط یکی از این بخش ها را بررسی می کنیم :

- خط لوله یک تکنیک پیاده سازی برای اجرای چندین دستورالعمل می باشد که در این روش دستورالعمل ها با هم هم پوشانی دارند. این روش به دون نیاز به صرف افزار اضافی قابل پیاده سازی است اما با تجزیه بخش های مختلف صرف افزار برای دستورالعمل های مختلف در همان زمان

- ساختار خط لوله در CPU ها نقطه اسپیل در کارخانجات می باشد. این کار با نگه نگه کردن چرخه دستورالعمل در چندین گام انجام می گیرد هر کدام از گام ها کاری را نا بایند طول می کشد تا کامل شود و بعد از کامل شدن به مرحله بعدی رود.

- هر مرحله یا گام را یک Pipe گویند.



- گام های یک خط یکی پس از دیگری به هم متصل می باشند.

- برای حرکت از یک گام به گام بعد مدت زمانی سپری خواهد شد که این مقدار زمان را یک سیکل ماشین گویند

سیکل : یک پالس ساعت

تسریع در اجرا با استفاده از pipeline : Acceleration by pipeling

برای مثال یک خط لوله دو مرحله ای را مورد بررسی قرار می دهیم. در این خط لوله دو مرحله FI و EI وجود دارد.

clock cycle	1	2	3	4	5	6	7
instr i	FI	EI					
i+1		FI	EI				
i+2			FI	EI			
i+3				FI	EI		
i+4					FI	EI	
i+5						FI	EI

* در این مثال مدت زمان اجرای هر دستورالعمل را T_{ex} در نظر می گیریم :

T_{ex} = Time of Execution Time

* در این مثال ۵ دستورالعمل اجرا شده است.

* زمان اجرا برای ۵ دستورالعمل با استفاده از فرمول زیر قابل محاسبه می باشد :

بدلیل این که هر دو دستورالعمل با هم اجرا می شوند. $\rightarrow$ $(\frac{T_{ex}}{2}) \times 5 = \frac{5T_{ex}}{2} = 2.5\, T_{ex}$

حال مثال فوق را در یک Pipline با یک چرخه ۶ دستورالعمل و ۴ دستگاه در ۱۲ پالس مورد بررسی قرار می دهیم: $(\frac{T_{ex}}{4}) \times 12 = 3\, T_{ex}$

- خط لوله ابتدا خالی است و به مرور زمان پر می شود.

- در پالس ششم هر ۴ فعالیت وجود دارند و هم فعالیت با بخش خاصی از سیستم کار می کند.

- در Fetch داده به طور همزمان نمی تواند اجرا شود و راه حل رفع این مشکل جداسازی حافظه زمان داده از حافظه زمان دستورالعمل می باشد.

- هر چقدر مراحل بیشتر شود overhead افزایش می یابد. علاوه بر overhead همزمان سازی نیز افزایش می یابد.

- و حتی تعداد stage ها بالا رود level سازی سازی افزایش پیدا می کند اما با بالا رفتن تعداد stage ها نمی توان همه stage ها را پر نگه داشت.

مثل :

پردازنده های خانواده Pentium , 80486

۵ گام برای دستورالعمل های صحیح

۸ گام برای دستورالعمل های اعشاری

مخاطره های خط لوله: Pipeline Hazard

تعریف Hazard:

هازاردها موقعیتی را بیان می کنند که در آن یک دستورالعمل نتواند در زمانی که می بایست اجرا گردد، اجرا شود و یا به عبارتی دستورالعمل نتواند در پالس اختصاص داده شده به آن اجرا گردد.

به چنین دستورالعمل هایی اصطلاحا stalled گویند.
متوقف شده.

وقتی یک دستورالعمل stalled شد. باشد تمامی دستورالعمل های بعد از آن نیز stalled خواهند شد. و در زمان رویداد این یک stalled هیچ دستورالعملی از خط واکشی نخواهد شد.

انواع Hazard ها:

1- Structual Hazard (ساختاری)
2- Data Hazard (داده)
3- Control Hazard (کنترلی)

هازاردهای ساختاری Data Hazard:

هازاردهای ساختاری زمانی رخ می دهند که یک منبع فعالی مانند حافظه در یک زمان توسط دو دستور مختلف مورد استفاده قرار گیرد.

چرخه دستورالعمل ها:

FI: fetch Instruction
DI: Decode Instruction
CO: Calculate operand Address
FO: fetch operand
EI: Execute Instruction
WO: write Back operand

مثال :

دستور ADD R4, X در مرحله FO که خط لوله عملوند X را از حافظه واکشی می کند، آنگاه این عملوند از حافظه واکشی نشده است حافظه اجازه دسترسی به دستورالعمل دیگری را نمی دهد.

clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
ADD R4, X	FI	DI	CO	FO	EI	WO						
inst i+1		FI	DI	CO	FO	EI	WO					
inst i+2			FI	DI	CO	FO	EI	WO				
inst i+3				~~FI~~	DI	CO	FO	EI	WO			
inst i+4												

FO: واکشی عملوند از حافظه

FI: واکشی دستورالعمل

* علت بوجود آمدن یک پالس stalled در دستور سوم دسترسی هم زمان به حافظه توسط دستور اول و دستور سوم می باشد.

* همانطور که مشاهده می شود جریمه حاصل از اجرای این دو دستور یک پالس ساعت می باشد.

راه حل های رفع این نوع هازارد ها:

۱_ افزایش منابع

۲_ pipeline کردن functional unit هایی مانند Alu

۳_ یک تکنیک جلوگیری از این هازاردها جداسازی کش داده از دستورالعمل می باشد.

Data Hazard :

این نوع هازارد ها زمانی اتفاق می افتند که یک دستور بخواهد از نتیجه‌ی دستور دیگر استفاده کند در حالی که نتیجه دستور دیگر هنوز مشخص نمی باشد.

مثال :

دو دستورالعمل به نام های I1, I2 داریم، در یک حافظه دستور I2 می تواند قبل از پایان یافتن I1 اجرا شود. اگر در یک گام خاص از حافظه I2 نتیجه I1 نیاز داشته باشد اما این نتیجه هنوز آماده نشده باشد در آن زمان یک Data hazard اتفاق افتاده است.

I1: mul R2,R3 → R2 ← R2 * R3

I2: Add R1,R2 → R1 ← R1 + R2

clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
mul R2,R3	FI	DI	Co	Fo	EI	WO						
ADD R1,R2		FI	DI	Co			Fo	EI	WO			
inst j+2			FJ	DI			Co	Fo	EI	WO		

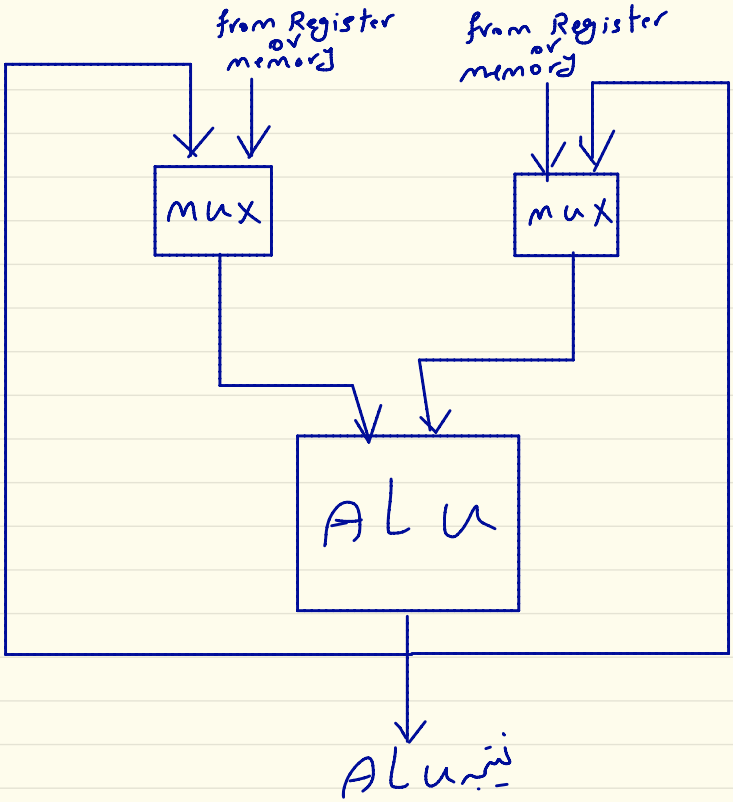
دستور ADD زمانی می تواند اجرا شود که نتیجه دستور mul مشخص شده باشد.
در نتیجه اجرای این دو دستور ها اندکی دوباره سازی جریمه خواهیم داشت.

راه کار :

استفاده از روش bypassing یا forwarding

در این روش از خروجی های ALU یک لینک به ورودی آن وارد می شود. بدین ترتیب می توان از نتیجه دستورالعمل قبل از ذخیره نتیجه در حافظه استفاده کرد.

دیاگرام مسئله بعد



دیاگرام روش forwarding

در این روش نتیجه ALU همیشه عنوان ورودی به خود ALU برگردد.

اگر کمی اضافه تشخیص دهد برای اجرای عملیات جاری نیاز به نتیجه دستورالعمل قبلی است، آن گاه به جای خواندن نتیجه از حافظه، نتیجه از ALU خوانده می‌شود. در نتیجه اجرای این فرآیند یک پالس از Stallها کاهش پیدا می‌کند.

clock cycle	1	2	3	4	5	6	7	8	9	10	
Mul R2, R3	FI	DI	CO	FO	EI	WO					
ADD R1, R2		FI	DI	CO		FO	EI	WO			

در نتیجه این روش حرکت از دو پالس ساعت به یک پالس ساعت کاهش پیدا می‌کند.

Control Hazard:

این هازاردها توسط دستورات پرش بوجود می آیند که این دستورات پرش می توانند شرطی و یا غیرشرطی باشند.

دستورات پرش غیرشرطی:

مثال:

```
-----
BR target  ⟶ Goto Target
--------
target ---------
Target i+1 -----
```

Clock cycle	1	2	3	4	5	6	7	8	9	10	11
BR target	FI	DI	CO	FO	EI	WO					
target		~~FI~~			FI	DI	CO	FO	EI	WO	
target +1						FI	DI	CO	FO	EI	WO

* تا زمانی که دستور BR target اجرا نشود آدرس target مشخص نمی شود.

* علت اجرا نشدن FI خط 4:

FI خط خورده مربوط به دستوراتی است که بعد از پرش اجرا می شود در صورتیکه قبل از این DI خط اول تمام شود ما نمی دانیم اصلا پرش انجام خواهد گرفت یا خیر در نتیجه دستور FI به طور کل حذف می شود.

دستورات پرش شرطی:

مثال:

ADD R1,R2 : $R1 \leftarrow R_1 + R_2$

BEZ Target : branch if zero

instruction i+1 :

Target

اگر شرط انجام نشود:

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
ADD R1,R2	FI	DI	CO	FO	EI	WO						
BEZ Target		FI	DI	CO	FO	EI	WO					
target			~~FI~~			FI	DI	CO	FO	EI	WO	

* تا زمانی که دستور DI پالس سوم مرحله دوم اجرا نشود آدرس target مشخص نخواهد شد، در نتیجه دستور بعد از target واکشی نخواهد شد.

* در انتهای پالس پنجم مرحله دوم تمام شده و هم آدرس target مشخص می‌شود.

(تا زمانی که FO انجام نشود مشخص نیست پرش انجام شود یا خیر.)

اگر شرط انجام نشود:

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
ADD R1,R2	FI	DI	CO	FO	EI	WO						
BEZ Target		FI	DI	CO	FO	EI	WO					
inst i+1			FI			DI	CO	FO	EI	WO		

* در انتهای پالس پنجم مشخص می‌شود که شرط انجام شده است یا خیر.

* در پرش های شرطی، چه شرط انجام شود و چه شرط انجام نشود ما جریمه خواهیم داشت چون همیشه می بایست منتظر نتیجه پرش باشیم.

* دستورات پرش یکی از مسائل بسیار کوچک در بهینه سازی خط لوله CPU هستند، ما نمی توانیم این هزارد ها را به طور کامل از بین ببریم اما راه کار هایی برای کاهش آنها وجود دارد.

راه حل های کم کردن هزارد ها:

دستورالعمل های پرش به طور چشمگیری بر روی کارایی خط لوله تاثیر می گذارد.

عملیات های کنترل (پرش شرطی و غیر شرطی) اجزو دستورالعمل هایی هستند که بیشتر از ده تا ی قسمت های برنامه مورد استفاده قرار می گیرند.

برخی آمار ها در این زمینه:

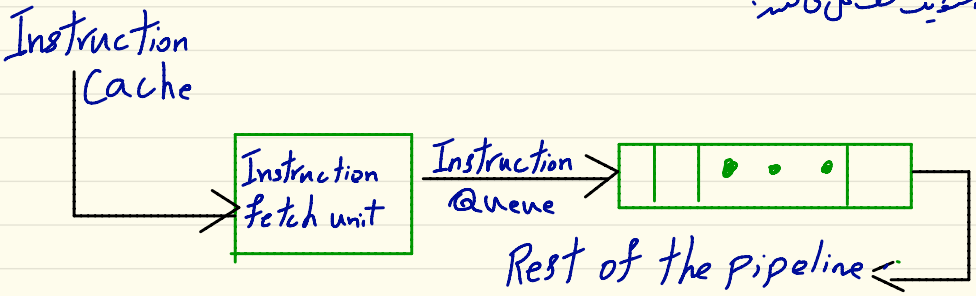
- از ۲۰ الی ۲۵ دستورالعمل هایی که در یک برنامه اجرا می شوند جزو دستورات پرش می باشند. (شرطی و غیر شرطی)
- در ۶۵ از پرش ها، پرش انجام می گیرد.
- دستورالعمل های پرش شرطی بسیار بیشتر از دستورالعمل های پرش غیر شرطی می باشند.
- در بیشتر از ۵۰ از پرش های شرطی پرش انجام می گیرد.

در نتیجه کاهش جریمه های ناشی از دستورالعمل های پرش بسیار با اهمیت می باشد.

واحد واکشی دستورالعمل ها و صف دستورالعمل: Instruction fetch units and instruction Queues

در بیشتر پردازنده ها در سطح بالایی تحت عنوان واحد واکشی دستورالعمل وجود دارد که وظیفه آن ذخیره دستورالعمل های بعد از قبل از این که CPU به دستورالعمل نیاز داشته باشد.

این واحدها به عنوان یک صف عمل می‌کنند.



واحد واکشی این توانایی را دارد که بتواند دستورات پرش را از همین تشخیص دهد و در صورت پرش بودن آن آدرس پرش را به دست آورد و بنابراین با این روش جریمه‌های ناشی از پرش‌ها به طور معمول وجود کاهش پیدا می‌کند:

واحد واکشی آدرس پرش را محاسبه و دستورالعمل بعد از پرش را واکشی می‌کند.

بدین ترتیب دیگر گام‌های خط لوله به کار خود ادامه خواهد داد، بدون توقف (بدون stalled) جریمه‌های ناشی از پرش‌ها شاید بتوان گفت به طور کامل نادیده گرفت.

روش کاهش جریمه ناشی از پرش:

روش Delay Branching (تأخیر در پرش):

با به تأخیر انداختن پرش به cpu این اجازه داده می‌شود که cpu وقت خود را صرف کارهای مفیدتری خواهد کرد.

با تکنیک تأخیر در پرش cpu بلافاصله بعد از پرش رفته و آن را واکشی می‌نماید. دستوراتی که بعد از دستور پرش قرار می‌گیرند اصطلاحاً branch Delay slot گویند. (شکاف تأخیر پرش)

برای خوب بیشتر مثالی بیان می‌شود، در این مثال یک برنامه‌نویس برنامه‌ای به شکل زیر می‌نویسد:

```
Mul  R3,R4 : R3 ← R3 + R4
SUB  #1,R2 : R2 ← 1
ADD  R1,R2 : R1 ← R1+R2
BEZ  TAR   : branch if zero
MOV  #10,R1 : R1 ← 10
- - - - - - - -
TAR - - - - - - - -
```

(Mul R3,R4 →) این دستورالعمل تأثیری در دیگر دستورالعمل ندارد

(MOV #10,R1 →) این دستورالعمل فقط زمانی اجرا می شود که شرط انجام نگیرد

کامپایلر (اسمبلر) دستورالعمل هایی را می تواند جای اصلی آن ها تغییر دهد که اجرا شدن آن ها بعد از پرش تأثیری در منطق برنامه نداشته باشد.

جابجایی دستورالعمل ها توسط کامپایلر همانند شکل زیر خواهد بود.

```
SUB  #1,R2
ADD  R1,R2
BEZ  TAR
mul  R3,R4  →
MOV  #10,R1 →
- - - - - - - - - -
TAR - - - - - - - - -
```

(mul R3,R4 →) جابجا شدن این دستور تأثیری در منطق برنامه نخواهد داشت.

(MOV #10,R1 →) این دستور فقط وقتی اجرا می شود که شرط انجام شود.

حال اگر کامپایلر نتواند دستوری برای انتقال به بعد از پرش پیدا نکند چه کاری انجام می دهد؟

در این نوع موارد کامپایلر از دستور Nop استفاده می کند این دستور به CPU می گوید که هیچ کاری انجام ندهد.

در این موارد جریمه حاصل به همان اندازه جریمه استفاده نکردن از تکنیک بدون پرش می باشد. برنامه ای که ایجاد شده توسط کامپایلر برای چنین مواردی به صورت زیر خواهد بود.

```
Mul  R2,R4
SUB  #1,R2
ADD  R1,R2
BEZ  TAR
 Nop
MOV  #10,R1
- - - - - - -
TAR - - - - - - - - -
```

Branch prediction : پیش بینی پرش

در مثال قبل ما این طور در نظر گرفتیم که پرش انجام نشود و ما دستورعمل بعدی پرش را واکشی نمودیم. در این مورد شرط انجام شده و دستورالعمل بعدی آن دور ریخته شد. حال ما حالت های زیر را داریم: اگر پرش انجام نگیرد (پیش بینی درست) ۱ / اگر پرش انجام گیرد (پیش بینی برآورده نشود) ۲ — جریمه حاصل از پرش

حال اجازه دهید پیش بینی همیشه انجام پرش در نظر بگیریم. ما برای این راه حل احتیاج داریم که دایره واکشی دستورالعمل آدرس هدف (پرش) را محاسبه نماید.

Branch is taken:

Clock cycle	۱	۲	۳	٤	٥	٦	۷	۸	۹	۱۰
ADD R1,R2	FI	DI	CO	FO	EI	WO				
BEZ TAR		FI	DI	CO	FO	EI	WO			
Mul R3,R4			FI	DI	CO	FO	EI	WO		
the target				FI	[X]	DI	DO	FO	EI	WO

جریمه حاصل : یک پالس ساعت (پیش بینی برآورده شده است)

Branch is not taken:

Clock cycle	۱	۲	۳	٤	٥	٦	۷	۸	۹	۱۰
ADD R1,R2	FI	DI	CO	FO	EI	WO				
BEZ TAR		FI	DI	CO	FO	EI	WO			
Mul R3,R4			FI	DI	CO	FO	EI	WO		
move 100,R1				[X]	[X]	DI	DO	FO	EI	WO

جریمه : دو پالس ساعت (پیش بینی برآورده نشده است)

- پیش بینی درست انجام پرش بسیار با اهمیت است و می تواند تأثیر بسیار زیادی در کارایی داشته باشد.
- بنابراین نتیجه شرط دستورالعملی که واکشی می شود به عنوان ادامه دستورالعمل ها بوده و می تواند در صف دستورالعمل قرار گیرد. اگر بعدا به محاسبه شرط پرش، پرش براساس پیش بینی ما درست انجام شود دستورالعملی که در صف دستورالعمل به عنوان دستورالعمل بعدی است واکشی می شود از طرف دیگر اگر پیش بینی درست نبوده و شرط انجام نشود دستورالعمل واکشی شده دور ریخته شده و دستورالعمل صحیح به جای آن قرار می گیرد.
- برای بهره بردن هر چه بیشتر از تکنیک پیش بینی پرش ما می توانیم دستورالعمل هایی داشته باشیم که علاوه بر واکشی اجرا نیز شوند. که به این نوع اجرا Speculative execution گوییم.
- Speculative execution اجرای حدسی یا اجرای بخش هایی از کد که ممکن است بعدا لازم شوند بدون یک عمل شرطی نیاز نباشد، اجرای حدسی دستورالعمل عملا موجب افزایش متوسط عملکرد می شود.

استراتژی های پیش بینی شرط به دو دسته تقسیم می شوند:

۱- پیش بینی استاتیک ۲- پیش بینی دینامیک

پیش بینی استاتیک پرش:

در این نوع تکنولوژی های پیش بینی هیچ گونه توجهی به تاریخچه پرش ها نمی شود

روش های استاتیک:

- پیش بینی شود که شرط هرگز انجام نشود مانند پردازنده های سری Motorola 68020: غرض از این است که پرش هرگز انجام نشود.
- پیش شود پرش همیشه انجام گیرد: غرض از این است که پرش همیشه انجام گیرد.
- پیش بینی به جهت شرط بستگی داشته باشد: (power pc 601)
  - پرش برای پرش های رو به عقب همیشه انجام گیرد.
  - پرش برای پرش های رو به جلو همیشه انجام نگیرد.

پیش بینی پرش داینامیک:

این نوع تکنولوژی های پیش بینی دقت پیش بینی را ذخیره کرده شته پرش های شرطی بصورت می کنند.

شمای پیش بینی یک بیتی: در این روش برای نگهداری نتیجه آخرین پرش یک بیت در نظر گرفته می شود و بستگی براساس نتیجه آخرین پرش پیش بینی خود را انجام می دهد.

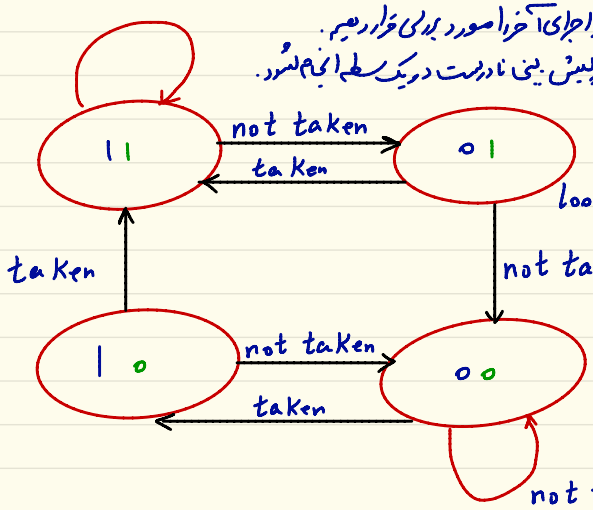
کمبود: در این تکنیک اگر شرط همیشه انجام گیرد و یا همیشه انجام نگیرد به جای یکبار پیش بینی اشتباه ما دو بار اشتباه پیش بینی خواهیم کرد.

```
--------
loop-------.
--------
BNZ loop
-------
```

شمای پیش بینی دوبیتی:

با پیش بینی ساخته شده براساس دوبیت می توانیم دو اجرای آخر اصورد بررسی قرار دهیم.

در این روش شرط فقط وقتی تغییری کند که دو پیش بینی نادرست در یک سطر انجام شود.



مثال:

```
-------
loop -------
-------
BNZ loop
-------
```

After the first execution of the loop the bits attached to BNZ will be 01 now, there will be always one false prediction for the loop, at its exit

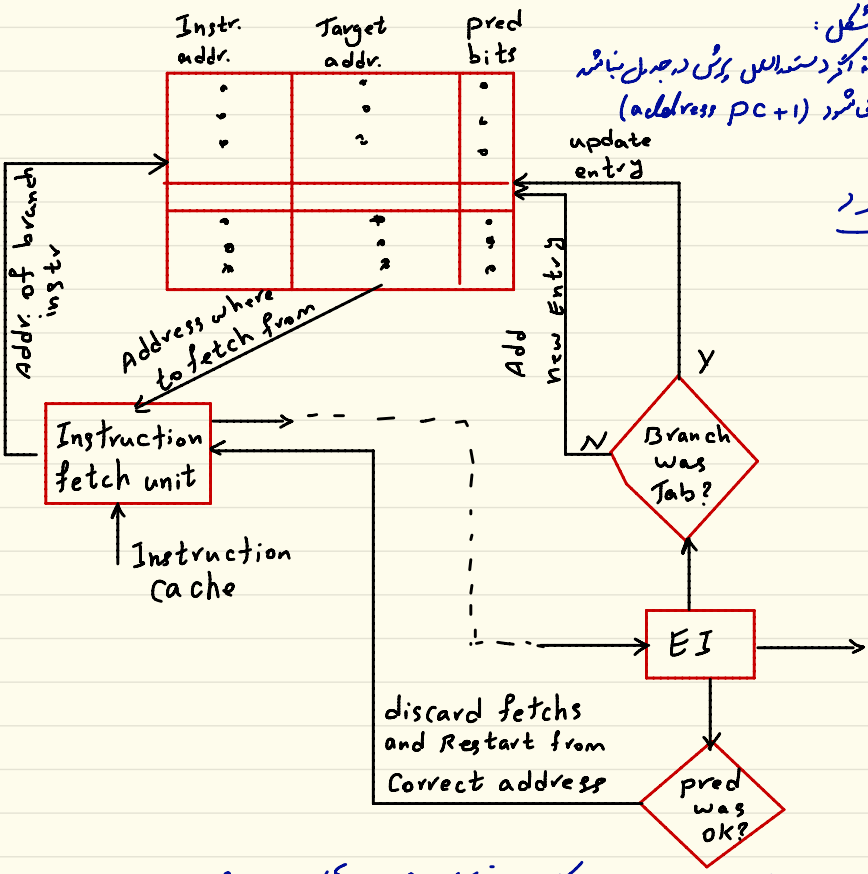
روش جدول گذشته پرش Branch history Table

جمع آوری اطلاعات گذشته فقط برای پیش بینی پرش های شرطی به کار نمی رود بلکه می تواند باعث جلوگیری از محاسبه مجدد آدرس هدف پرش گردد. با این دو بیت استفاده شده برای پیش بینی آدرس هدف پرش می توانند برای استفاده های بعدی در جدول گذشته پرش ذخیره شود (branch history Table)

برخی توضیحات در مورد شکل:

- آدرسی که برای واکشی از: اگر دستورالعمل پرش در جدول نباشد دستورالعمل بعدی واکشی می شود (address pc+1)

ادامه بررسی شود



خلاصه:

- دستورات پرش تاثیر خیلی بدی بر روی کارایی خط لوله می توانند بگذارند در نتیجه کاهش جریمه های حاصل از پرش بسیار با اهمیت است.
- واحد واکشی دستورالعمل می تواند پرش شرطی را تشخیص داده و آدرس پرش را نیز محاسبه نماید. با ایجاد کش دستورالعمل و نگهداری دستورالعمل های دیکد شده نرخ واکشی اطلاعات بسیار زیاد خواهد شد حتی امکان دارد جریمه های ناشی از پرش های غیر شرطی کاهش یابد. برای پرش های شرطی این امکان وجود ندارد زیرا کدی باید منتظر نتیجه پرش باشد (که آیا پرش انجام بگیرد یا خیر)
- تکنیک تاخیر پرش (branching Delay) یک تکنیک مبتنی بر کامپایلر است و هدف آن کاهش جریمه های ناشی از پرش با جابجایی دستورالعمل ها و انتقال آن ها به قبل از پرش می باشد (قرار دادن دستورالعمل ها در branch Delay slot)

- کاهش کارایی جریمه های ناشی از پرش های شرطی به استراتژی های پیش بینی هوشمند نیاز دارد. تکنیک پیش بینی پرش استاتیک هیچ گونه توجهی به گذشته اجرا ها ندارد. تکنیک های دینامیک پیش بینی مبتنی بر ذخیره سازی گذشته پرش های شرطی می باشد.
- جدا مل گذشته پرش هم اطلاعات مربوط به رخداد پرش ها و هم آدرس هدف را در خود ذخیره می نماید.

پردازنده های Super scaler:

- معماری Super scalar چیست؟
- Super Pipling
- ویژگی های معماری های Superscalar
- وابستگی داده ها
- سیالیت های اجرای موازی دستورالعمل ها
- تغییر نام ثبات ها

معماری Superscalar چیست؟

- در این معماری چندین دستورالعمل می توانند باهم شروع شده و به طور مستقل از هم اجرا شوند.
- خط اجازه می دهد که چندین دستورالعمل بتوانند در یک زمان (پالس ساعت) وارد اجرا شوند اما این اجرای دستورالعمل ها می بایست در مرحله های Stage ها از الزاما خط لوله باشد.
- معماری های Superscalar دارای تمامی ویژگی های خط لوله می باشد و علاوه بر این ویژگی ها می توانند اجرای چندین دستورالعمل را در یک مرحله از خط لوله شبیه سازی نمایند. این معماری این توانایی را دارد که بتواند چندین دستورالعمل را در یک پالس ساعت اجرا نماید.

در زیر دو نمونه از این معماری که براساس کارایی مرتب شده اند بیان شده اند:

1- Super piplining
2- Super scalar

Super piplining:

این روش بر تقسیم مرحله های خط لوله به چندین قسمت بنا شده است که با این ترتیب تعداد دستورالعمل هایی که در یک کلاک می توانند خط لوله اجرا شود افزایش می یابد.

- با تقسیم هر مرحله از خط لوله به دو قسمت دوره تناوب پالس ساعت آ به نصف یعنی $\frac{T}{2}$ کاهش می یابد و از این رو باعث افزایش ظرفیت خط لوله می گردد.
- برای این که این معماری بتواند بهترین بازده را داشته باشد می بایست ما بتوانیم تعداد دفعه گام های خط لوله را به اندازه کافی افزایش بیش از حد تعداد مراحل باعث کاهش کارایی می گردد.
- یکی از راه حل ها برای بهبود سرعت معماری Super scalar می باشد.

اجرای دستورالعمل در خط لوله :

CC	1	2	3	4	5	6	7	8	9	10
i	FI	DI	CO	FO	EI	WO				
i+1		FI	DI	CO	FO	EI	WO			
i+2			FI	DI	CO	FO	EI	WO		
i+3				FI	DI	CO	FO	EI	WO	
i+4					FI	DI	CO	FO	EI	WO

اجرای دستورالعمل ها در حالت superpiplining

CC	1	2	3	4	5	6	7	8	9
i	FI FI	DI DI	CO CO	FO FO	EI EI	WO WO			
i+1		FI FI	DI DI	CO CO	fo fo	EI EI	WO WO		
i+2			FI FI	DI DI	CO CO	fo fo	EI EI	WO WO	
i+3				FI FI	DI DI	CO CO	fo fo	EI EI	WO WO
i+4									
i+5									

اجرای دستورالعمل ها در حالت superScalar

CC	1	2	3	4	5	6	7	8
i	FI	DI	CO	fo	EI	WO		
i+1	FI	DI	CO	fo	EI	WO		
i+2		FI	DI	CO	FO	EI	WO	
i+3		FI	DI	CO	fo	EI	WO	
i+4			FI	DI	CO	fo	EI	WO
i+5			FI	DI	CO	FO	EI	WO
i+6								

معماری Superscalar:

- این معماری اجازه می دهد که چندین دستورالعمل در یک پالس ساعت اجرا شده و کامل شوند.
- یک معماری superscalar شامل تعدادی خطوط لوله می باشد که به صورت موازی با هم کار می کنند.
- بسته به تعداد نرم افزارهای موازی در دستورالعمل، تعداد خطوط لوله می تواند به صورت موازی اجرا شوند.
- به مثال زیر یک برنامه شامل دو برنامه می باشد که می توانند همزمان با هم اجرا شده و کامل شوند هر برنامه خط لوله می تواند چندین دستورالعمل را به صورت موازی در stage های مختلف اجرا نماید.

دیاگرام معماری superscalar

محدودیت های اجرای سولزی :

- شرایطی که منع اجرای دستورالعمل ها به صورت سولزی در معماری Superscalar می باشد بسیار شبیه به شرایط ممانعت از اجرای دستورالعمل ها در خط لوله است (هازاردهای خط لوله)

- بعواقب این شرایط در معماری های sc بسیار بیشتر از خط لوله ساده می باشد زیرا که پتانسیل سولزی سازی در sc ها بیشتر می باشد لذا فرصت های بیشتری از دست داده می شود.

سه گروه از محدودیت در sc قرار گرفته اند :

- تداخل منابع : زمانی رخ می دهد که دو دستورالعمل برای گرفتن یک منبع مانند حافظه در یک زمان با هم به رقابت بپردازند. که حتی شبیه هازاردهای ساختاری در خط لوله می باشد. که با سولزی سازی واحدها sc کمی می کند این تداخل را کاهش دهد

- وابستگی کنترل : همانند هازاردهای کنترلی در خط لوله ساده می باشد

اگر طول دستورالعمل ها متغیر باشد، نمی توانند به طور سولزی واکشی شده و صادر شوند، برای این که یک دستورالعمل بتواند Decode شود ابتدا می بایست واکشی گردد به همین دلیل تکنیک پیاده سازی sc ها RISC می باشد که طول رفرنس دستورالعمل ها در آن ثابت است.

تداخل داده ها: Data conflict

تداخل داده ها، ریسک وابستگی چند بین دستورالعمل ها، Data ی همگی در برنامه بوجود می آیند. معماری های sc آزادی بیشتری برای تغییر ترتیب دستورالعمل برای صدور و کامل شدن دارند.

در این معماری به همین اندازه که توازنیت سولزی سازی بالا می آید دستورالعمل صادر می شود.

- معماری های sc از پتانسیل اجرای سولزی دستورالعمل در سطح برنامه بهره برداری می نمایند

- یکی از ویژگی های مهم sc های پیشرفته، معماری زمان بندی پویای دستورالعمل است :

- دستورات بصورت دینامیک برای اجرا صادر شده و بدون رعایت ترتیب اجرا شده و خارج می گردند.

- صدور خارج از نوبت : دستورالعمل ها براساس ترتیب خود صادر نشده و تنها براساس وابستگی و در نوبت بودن منابع خارج می گردند.

نتایج می بایست بازه ای که دستورالعمل ها براساس ترتیب اجرای شوند یکسان باشد :

- وابستگی داده ها به دقت در نظر گرفته شود

- وابستگی داده ها فقط بخشی از پتانسیل سولزی سازی دستورالعمل ها را شامل می شود.

- برای پیاده کردن ترتیب صدور دستورالعمل ها در آینه سولزی سازی، پردازنده تعدادی کافی از دستورالعمل ها را انتخاب می کند که برای اجرای این تعداد دستورالعمل نیاز به یک پنجره بزرگ می باشد.

پنجره اجرا :

پنجره اجرا :

مجموع دستورالعمل هایی که برای اجرا در یک زمان خاص در نظر گرفته می شوند، هر دستورالعمل در پنجره می تواند برای اجرای سولزی در نظر گرفته شود، موضوعات وابستگی داده ها و منابع موضوعاتی است که می بایست در یک پنجره در نظر گرفته شود.

- تعداد دستورالعمل های عرضه داده شده در یک پنجره می بایست زیاد باشد : - اولا این رفع ظرفیت واکشی اطلاعات - ثانیه شرط ها ← : مشکلات

```
for(i=0;i<=last;i++){
 if a[i]>a[i+1]{
  temp=a[i]
 a[i]=a[i+1]
 a[i+1]=temp
  chang++
   }
}
```

r7: آدرس خانه جاری آرایه a[i]

r3: آدرس بای دستوری بعدی a[i+1]

r8: تغییر ; r4: last , r6: i

```
L2  mov  r3,r7
    lw   r8,(r3)    R8←a[i]      ┐
    add  r3,r3,4                 │
    lw   r9,(r3)    r9←a[i+1]    │ basic block1
    ble  r8,r9,L3                │
    move r3,r7                   ┘
    sw   r9,(r3)    a[i]←R9      ┐
    add  r3,r3,4                 │ basic
    sw   r8,(r3)    a[i+1]←R8    │ block2
    add  r5,r5,1    chang++      ┘
L2  add  r6,r6,1    i++          ┐
    add  r7,r7,4                 │ basic
    blt  r6,r4,L2                ┘ block3
```

- پنجره ها را با استفاده از لبه های block گسترش داده می شود. (اجرای حدسی)
- با یکی از حدسی ها دستورالعملی که وارد پنجره می شود پیش بینی می گردد. دستورالعمل های basic block های پیش بینی شده به صورت آزمایشی اجرا می شوند.

وابستگی داده ها:

همه دستورات قرار داده شده در پنجره اجرای توانند با هم شروع شوند به شرطی که بین آن ها وابستگی داده وجود داشته باشد.

سه نوع وابستگی داده در یک پنجره می تواند وجود داشته باشد:

1- True Data Dependency

2- output Dependency

3- Antidependency

True Data Dependency:

زمانی اتفاق می افتد که خروجی یک دستور به ورودی دستورات قبل از خودش نیاز داشته باشد.

این وابستگی داده ها به طور ذاتی به برنامه کاربر بستگی دارد و با تکنیک های نامناسب اجرای

و یا کامپایلر نمی توان مشکل آن را برطرف نمود. این وابستگی تشخیص داده شده و قابل برطرف شدن می باشد، ساده ترین راه حل استفاده از یک stall در لوله

```
mul  R1,R3,R1   R1←R3*R1
ADD  R2,R4,R5   R2←R4+R5
```

```
L2   mov   r3, r7
     lw    R8, (r3)
     madd  r3, R3, 4
     lw    r9, (r3)
     ble   r8, r9, L3
```

add زمانی که نیمه mul مشغول شود و

Output Dependency:
زمانی اتفاق می‌افتد که دو دستورالعمل بخواهند در یک جا عمل نوشتن انجام دهند.

Anti Dependency:
اگر دستوری بخواهد یک عملوند جایی بخواند که دستور دیگری می‌خواهد در آن بنویسد.

طبیعت Output Dependency و Anti Dependency
این دو وابستگی ذاتی داده نمی‌باشند بلکه تداخل در استفاده از محل ذخیره‌سازی هستند.
این وابستگی فقط نتیجه مدیریت برنامه و استفاده از ثبات‌ها توسط کامپایلر هستند.

سیاست‌های اجرای صوازی دستورات:
توان‌یابی را با SC توسط دو معیار زیر سنجیده می‌شود

۱- تعداد خطوط لوله موازی
۲- مکانیزم‌هایی که پردازنده برای بیه‌سازی وابستگی دستورات وابسته استفاده می‌کند.

سیاست‌های اجرای دستورات توسط دو فاکتور تعیین می‌شوند

۱- ترتیب صدور دستورات برای اجرا
۲- ترتیب کامل شدن دستورات

- ساده‌ترین سیاست اجرای و کامل شدن دستورات براساس ترتیب آن‌ها می‌باشد
- سیاست دیگر تغییر ترتیب دستورات برای بیه‌سازی وابستگی‌ها و اجرا به صورت صوازی می‌باشد.

سیاست‌های اجرا:

۱- به همان ترتیب که صادر می‌شوند کامل کردند
۲- به ترتیب صادر شده و خارج از نوبت کامل شوند
۳- خارج از نوبت صادر شده و خارج از نوبت نیز کامل کردند.

فصل نهم کتاب سورپس مانو :

پیوازش سولزی :

پردازش سولزی به معنی بکارگیری تکنیک های متنوعی در پردازش همزمان داده ها است که به منظور افزایش سرعت محاسبات سیستم های کامپیوتری مورد استفاده قرار گرفته اند. یک سیستم پردازش سولزی به جای پردازش متوالی دستورالعمل ها قادر پردازش همزمان داده ها را برای رسیدن به سرعت پردازش بیشتر انجام دهد. مثلاً فرض اجرای دستوری در ALU ، دستور بعدی قابل خواندن از حافظه می باشد. یک سیستم ممکن است دارای دو یا چند ALU باشد ولذا قادر خواهد بود دو یا چند دستور را به طور همزمان اجرا نماید.

بعلاوه سیستم ممکن دو یا چند پردازنده داشته باشد که همزمان با هم عمل می نمایند. در هر صورت هدف از پردازش سولزی بالا بردن سرعت پردازش کامپیوتر و افزایش دفعات پردازش در طول بازه معینی از زمان است. (توان عملیاتی Throughput)

با بکارگیری تکنیک های پردازش سولزی حجم کمتر افزاری افزایش می یابد و همراه با آن قیمت سیستم نیز بالا می رود. با این وجود توسعه تکنولوژی برای کمتر افزاری را تا حدی پایین آورده است که در پردازش سولزی کاملاً محسوس است.

پردازش سولزی را می توان از زوایای مختلفی بررسی کرد. ساده ترین سطح، پردازش سولزی دستوری را از نظم ثبات ها به کار رفته بررسی می کنیم. ثبت های شیفت دهنده به طور سری و به مقدار یک بیت در هر بار عمل سر و کار دارند (انجام می دهند)، در حالیکه ثبات ها با بارشدن سولزی در هم لحظه قادرند روی تمام بیت های کلمه عمل کنند. در سطح بالاتر، پردازش سولزی دارای واحد های عملیاتی (functional unit) چندگانه است که اعمال یک یا متفاوتی را بطور همزمان انجام می دهند. در این حال پردازش سولزی با توزیع داده ها در میان این واحد ها صورت می گیرد. مثلاً واحد حساب، منطق و شیفت قابل تفکیک به سه واحد بوده و عملوند ها تحت نظم واحد کنترل به هم وارد می شوند.

شکل (1-9) یک راه تفکیک واحد اجرایی به هشت واحد تابعی (عملیاتی) که به طور سولزی با یکدیگر کار می کنند را نشان می دهد. عملوند در ثبات ها جسته به دستور العمل به یکی از واحد ها تحویل می گردد. عملی که در هر واحد انجام می شود در داخل چهارگوشه های دیاگرام نشان داده شده اند. جمع کننده و ضرب کننده مقدر صحیح روی اعداد صحیح عمل می کنند. اعمال منطقی، شیفت و افزایش در روی داده های مختلف بطور همزمان عمل می نمایند. کلیه واحد ها لزوماً یکه مستقل بوده ولذا یک محدود، شیفت و دیکری افزایش داده می شود. سازمان چنین تابعی معمولاً به یک واحد کنترل پیچیده نیاز دارد تا کلیه فعالیت ها را در میان قسمت های مختلف هماهنگ نماید.

روش های مختلفی برای طبقه بندی پردازش سولزی وجود دارد. می توان از دیدگاه سازمان داخلی یا از نظم اتصالات درون بین پردازنده ها و یا از نظم جریان اطلاعات در درون سیستم آن ها را طبقه بندی کرد. یکی از این دسته بندی ها توسط Flynn پیشنهاد گردیده که طی آن سازمان یک کامپیوتر را با توجه به تعداد دستورات و داده های دستکاری شده در یک زمان مورد بررسی قرار می دهد.

روال معمول یک کامپیوتر برداشت دستورات از حافظه و اجرای آن در پردازنده است. دستورات خوانده شده متوالی از حافظه رشته دستورات را تشکیل می دهند. داده هایی نیز که رشته اعمال روی آن ها انجام می شود رشته داده ها را تشکیل می دهند. پردازش سولزی ممکن است روی رشته دستورات یا روی رشته داده ها یا هر دو باشد.

دسته بندی Flynn کامپیوتر را به 4 گروه مجزا مطابق زیر تقسیم می کند :

SISD : رشته تک دستوری، رشته تک داده ای

SIMD : رشته تک دستوری، رشته چند داده ای

MISD : رشته چند دستوری، رشته تک داده ای

MIMD : رشته چند دستوری، رشته چند داده ای

منظور از SISD سازمانی از یک کامپیوتر است که در آن یک واحد کنترل، یک واحد پردازنده و یک واحد حافظه وجود دارد. دستورات به ترتیب اجرا می شوند و سیستم ممکن است توانایی پردازش موازی را داشته باشد یا نداشته باشد. در این حالت پردازش موازی می تواند توسط واحدهای چندتایی یا پردازش خط لوله صورت گیرد.

SIMD سازمانی را نشان می دهد که دارای چند واحد پردازش تحت یک واحد کنترل مشترک است. همهٔ پردازنده ها دستور الحمل واحدی را توسط واحد کنترل دریافت می کنند ولی روی داده های مختلفی عمل می نمایند. واحد مشترک حافظه باید از چند ماژول (بخش) تشکیل شده باشد تا بتواند با تمام پردازنده ها ارتباط برقرار کند.

MISD: فقط از نظر تئوری حائز اهمیت است چون عملاً هیچ سیستمی با چنین سازمانی ساخته نشده است.

MIMD به کامپیوترهایی اختصاص دارد که توان پردازش چندین برنامه را در یک زمان دارند. اغلب سیستم های چند پردازنده ای و سیستم های چند کامپیوتری در این دسته طبقه بندی می شوند.

دسته بندی flynn بستگی به اختلافات بین عمل واحد کنترل و واحد پردازش داده دارد. این دسته بندی بیشتر به مشخصات رفتاری سیستم کامپیوتری می پردازد تا به ارتباطات درونی آن. یک نوع پردازش موازی که منطبق بر تقسیم بندی flynn نیست روش خط لوله است. دو نمونه موجود را این طبقه بندی یکی پردازنده های آرایه ای SIMD بحث شده و دیگری چند پردازنده MIMD می باشد.

انواع پردازش های موازی:

۱- پردازش خط لوله

۲- پردازش برداری

۳- پردازنده های آرایه ای

پردازش خط لوله یک تکنیک پیاده سازی است که در آن جزء (ریز) اعمال محاسبه ها یا فازهای سیکل دستورالعمل بطور هم پوش اجرا می شوند. پردازش برداری درباره محاسبات بردارها و ماتریس های بزرگ است. پردازنده های آرایه ای هم بر روی آرایه های بزرگی از داده ها عمل می نمایند.

خط لوله: خط لوله تکنیکی است که یک پردازش سری را به عملیات جزیی تکنیک می نماید و هر عمل جزئی در مقطع خاص همزمان با سایر مقاطع اجرا می گردد. خط لوله اغلب توان به عنوان مجموعه ای از مقطع پردازش کننده هایی تصور کرد که در آن ها اطلاعات دودویی جریان دارد. هر قطعه بخشی از پردازش را که به آن دیکته شده است انجام می دهد. نتیجه حاصل از محاسبات در هر قطعه به قطعه بعدی در خط لوله منتقل می گردد. کلمه خط لوله برجریانی از اطلاعات در سیستم ها مشابه با خط تولید در کارخانجات اتلاق می شود. از مشخصه های خط لوله این است که چندین محاسبه در یک زمان در قطعات مختلف در حال پیشروی است. هم پوشی یا هم اصطلاح همزمانی اجرا به این ترتیب تحقق می یابد که به هر قطعه در خط لوله ثباتی تعلق می گیرد. این ثبات ها نوعی جداسازی را در هر قطعه ایجاد می نمایند بطوری که هر قطعه می تواند روی داده های جداگانه ای بصورت همزمان عمل نماید.

شاید ساده ترین راه در شناخت یک ساختار خط لوله این باشد که تصور کنیم هر قطعه از یک ثبات ورودی و به دنبال آن یک مدار ترکیبی ساخته شده است. ثبات داده را نگهدارد و مدار ترکیبی جزء عملی را در ارتباط با قطعه انجام می دهد. خروجی مدار ترکیبی در یک قطعه متصل به ثبات ورودی قطعه دیگر تحویل می گردد. به تمام ثبات ها پالس ساعتی اعمال می شود تا زمان کافی برای فعالیت تمام قطعات تأمین گردد. سازمان خط لوله توسط مثال ساده ای بیان می گردد. فرض کنید که می خواهیم یک عمل ترکیبی ضرب و جمع را با دنباله ای از اعداد انجام دهیم. $A_i * B_i + C_i$ for $i = 1, 2, 3, 4, \ldots, 7$

هر جزء (زیرعمل) در یک قطعه از خط لوله رخ می دهد. هر قطعه دارای یک یا دو ثبات و یک مدار ترکیبی می باشد. R_1 تا R_5 ثبات هایی هستند که داده های جدید را با هر پالس ساعت دریافت می کنند. ضرب کننده و جمع کننده مدارات ترکیبی هستند. زیرعملی که در هر قطعه از خط لوله اجرا می شود مطابق زیر است:

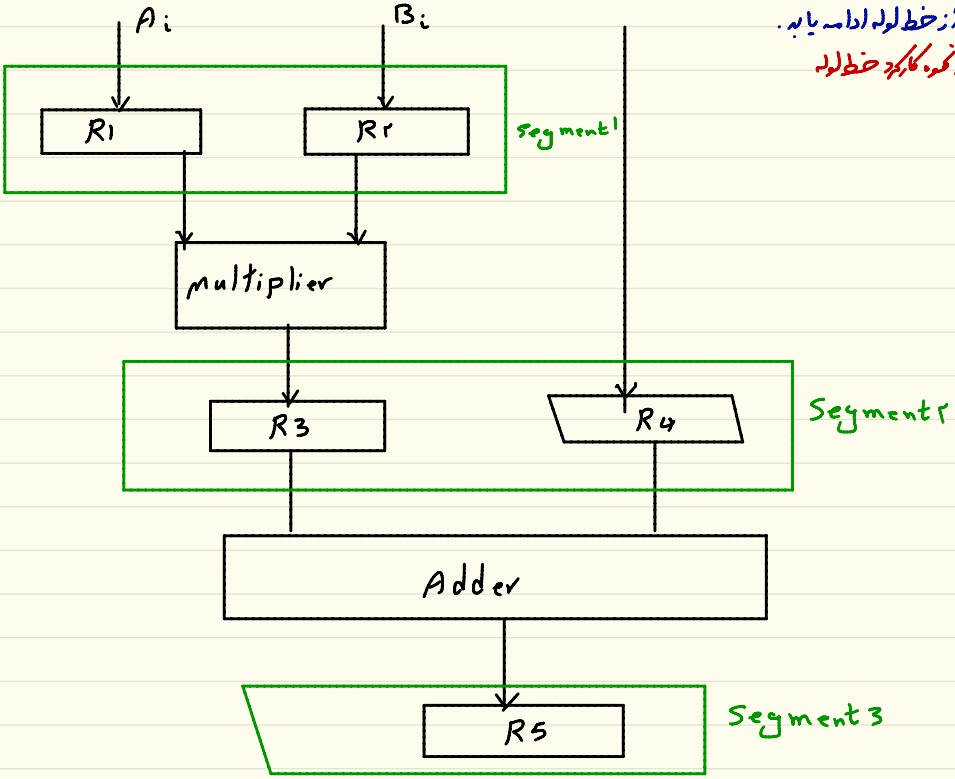
$R1 \leftarrow A_i$ و $R2 \leftarrow B_i$ $\quad$ دریافت A_i و B_i

$R3 \leftarrow R1 * R2$ ، $R4 \leftarrow C_i$ $\quad$ ضرب و دریافت C_i

$R3 \leftarrow R3 + R4$ $\quad$ جمع C_i با حاصلضرب

پنج ثبات به ازای هر پالس با داده های جدیدی پر می شوند. از هر پالس در جدول ۱-۹ دیده می شود. اولین پالس ساعت A_1 و B_1 را به داخل $R1$ و $R2$ انتقال می دهند. پالس ساعت دوم حاصلضرب $R1$ در $R2$ را به $R3$ و C_i را به $R4$ منتقل می کند. همان پالس ساعت A_2 و B_2 را به $R1$ و $R2$ انتقال می دهد. سومین پالس ساعت به طور همزمان پردازی هر سه قطعه عمل می کند. بدین ترتیب که A_3 و B_3 را در $R1$ و $R2$ می گذارد، حاصلضرب $R2$ و $R1$ را به $R3$ منتقل و C_2 را در $R4$ و به علاوه حاصل جمع $R3$ و $R4$ را در $R5$ قرار می دهد. بنابراین برای پرکردن خط لوله و دریافت اولین خروجی از $R5$ سه پالس ساعت زمان لازم است. از این پس یک خروجی تولید و داده را در یک قدم به پایین خط لوله می راند. این اعمال ادامه دارد تا که داده های ورودی جدید به داخل سیستم جریان داشته باشد. وقتی دیگر داده ای وجود نداشته باشد، اعمال پالس باید تا آخرین خروجی از خط لوله ادامه یابد.

مثالی از نحوه کارکرد خط لوله



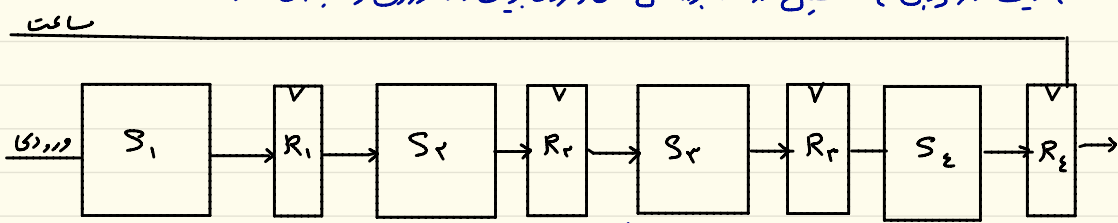
جدول محتوی ثبات ها برای مثال قبل

Clock Pulse Number	Segment1		Segment2		Segment3
	R1	R2	R3	R4	R5
۱	A_1	B_1	—	—	—
۲	A_2	B_2	$A_1 \times B_1$	C_1	—
۳	A_3	B_3	$A_2 \times B_2$	C_2	$A_1 \times B_1 + C_1$
۴	A_4	B_4	$A_3 \times B_3$	C_3	$A_2 \times B_2 + C_2$
۵	A_5	B_5	$A_4 \times B_4$	C_4	$A_3 \times B_3 + C_3$
۶	A_6	B_6	$A_5 \times B_5$	C_5	$A_4 \times B_4 + C_4$
۷	A_7	B_7	$A_6 \times B_6$	C_6	$A_5 \times B_5 + C_5$
۸	—	—	$A_7 \times B_7$	C_7	$A_6 \times B_6 + C_6$
۹	—	—	—	—	$A_7 \times B_7 + C_7$

بررسی های کلی :

هر عملی که بتوانه به‌صورت‌های از جزء عمل ها با پیچیدگی یک تابی تجزیه شود می تواند توسط پردازنده خط لوله پیاده سازی شود. این تکنیک برای آن دسته از کار بردها مفید است که نیاز به تکرار عملیات یکسانی بر روی مجموعه‌ای از داده های مختلف دارند.

ساختار عمومی یک خط لوله ۴ قطعه‌ای در شکل زیر نشان داده شده است. عملوندها از تمام قطعه ها با ترتیب خاصی عبور می کنند. هر قطعه از یک مدار ترکیبی S_i تشکیل شده که جزء عمل خاصی را روی جریان داده درون لوله انجام می دهد.



قطعات توسط ثبات های R_i که نتایج میانی قطعات را نگه می دارند از هم جدا شده اند. اطلاعات بین قطعات توسط پالس ساعت مشترکی که به تمام ثبات ها به طور همزمان اعمال می شود جریان می یابد. ما یک تکلیف را به عنوان کل یک عمل که با طی شدن تمام قطعات خط لوله انجام می شود تعریف می کنیم. رفتار خط لوله را می توان توسط یک نمودار فاصله - زمان تشریح کرد. این نمودار استفاده از قطعه را به صورت تابعی از زمان نشان می دهد. نمودار فاصله - زمان یک خط لوله ۴ قطعه‌ای در شکل صفحه بعد نشان داده شده است. محور افقی نمودار زمان را برحسب پالس های ساعت و محور عمودی شماره قطعه را در آن نشان می دهد، نمودار شش تکلیف T_1 الی T_6 را نشان می دهد که در چهار قطعه اجرا شده است.

در ابتدا تکلیف T_1 توسط قطعه ۱ انجام می شود. پس از پالس ساعت اول، قطعه ۲ مشغول اجرای T_1 است و در همان زمان قطعه ۱، T_2 را انجام می دهد. به همین ترتیب، تکلیف T_1 پس از طی ۴ پالس ساعت تکمیل شده است. از این به بعد خط لوله در هر پالس

یک تکلیف را کامل می‌کند. تعداد قطعات در سیستم اهمیتی ندارد، به معنی این که خط لوله پر شود در هر پالس ساعت یک خروجی به دست خواهد آمد. حال خط لوله‌ای متشکل از k قطعه را در نظر بگیرید که در آن پالس ساعت t_p برای انجام n تکلیف بکار رفته است. اولین تکلیف را در زمانی برابر با kt_p را نیاز دارد تا پایان یابد زیرا در لوله تعداد k قطعه وجود دارد. $(n-1)$ تکلیف باقیمانده در لوله با سرعت یک تکلیف در هر پالس ساعت خارج می‌شوند و پس از زمانی برابر با $(n-1)t_p$ کلیه تکلیف‌ها پایان می‌یابد. بنابراین برای تکمیل n تکلیف با k قطعه در لوله، $k+(n-1)$ پالس ساعت لازم است. به عنوان مثال در یک برنامه صفحه بعد چهار قطعه و شش تکلیف را نشان می‌دهد. می‌بینید که زمان لازم برای تکمیل تمام عملیات $9=(1-6)+4$ پالس ساعت است.

حال یک واحد غیر خط لوله را در نظر بگیرید که همان اعمال را انجام دهد. اگر زمان تکلیف، t_n باشد زمان کل لازم برای n تکلیف برابر با nt_n خواهد بود. افزایش سرعت پردازش خط لوله نسبت به یک پردازش غیر خط لوله با رابطه زیر تعریف می‌شود.

$$S=\frac{nt_p}{(k+n-1)t_p}$$ رابطه

دیاگرام فاصله-زمان برای خط لوله

قطعه	۱	۲	۳	۴	۵	۶	۷	۸	۹
۱	T_1	T_2	T_3	T_4	T_5	T_6			
۲		T_1	T_2	T_3	T_4	T_5	T_6		
۳			T_1	T_2	T_3	T_4	T_5	T_6	
۴				T_1	T_2	T_3	T_4	T_5	T_6

جریان‌های وقت ←

هر چه تعداد تکلیف‌ها افزایش یابد، n از $k-1$ بیشتر می‌شود و $k+n-1$ به سمت n میل می‌کند. تحت این شرط رابطه فوق برابر است با $S=\frac{t_n}{t_p}$

اگر فرض کنیم که t_p زمان لازم برای غیر خط لوله همان زمان در خط لوله باشد داریم $t_n=kt_p$. لذا رابطه نسبت سرعت برابر است با:

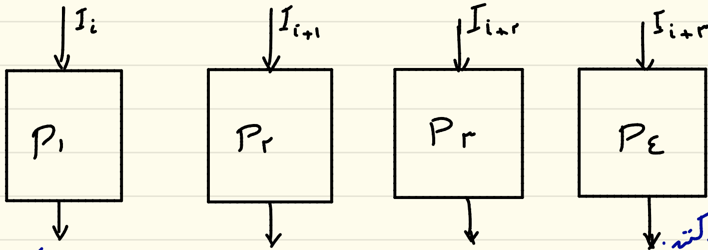
$$S=\frac{kt_p}{t_p}=k$$

رابطه فوق نشان می‌دهد که حداکثر نسبت سرعت تئوری در خط لوله می‌تواند k باشد که در آن k تعداد قطعه در خط لوله است.

برای درک بهتر نسبت افزایش سرعت مثال عددی زیر را در نظر بگیرید. فرض کنید زمان پردازش در هر قطعه $t_p=20ns$ باشد. اگر خط لوله دارای $k=4$ قطعه باشد و کلاً ۱۰۰ تکلیف پردازش گردد، سیستم خط لوله زمانی برابر با $(k+n-1)t_p=(4+99)\times 20=2060ns$ برای تکمیل پردازش نیاز دارد. با فرض این که سیستم یک سیستم غیر خطی لوله‌ای زمانی برابر با $nkt_p=100\times 80=8000ns$ برای تکمیل ۱۰۰ تکلیف نیاز خواهد داشت. نسبت سرعت برابر است با $8000/2060=3.88$. با افزایش تعداد تکلیف‌ها، افزایش سرعت به سمت ۴ میل می‌کند که با تعداد قطعات در خط لوله برابر است. اگر $t_p=60ns$ در نظر گرفته شود، نسبت افزایش سرعت $60/20=3$ خواهد شد.

برای آنکه بتوانیم به سرعت تئوری یک پردازش خط لوله توسط یک طرح چند تابعی، لازم است که k واحد مشابه را ساخت تا به صورت موازات عمل کنند. به این ترتیب که می‌توان انتظار داشت تا k قطعه خط لوله، همانند k مدار همسان غیر خط لوله‌ای تحت شرایط یکسان به صورت موازات بسته شده‌اند. هر مدار p معادل با یک مدار خط لوله کار می‌کند. در بعضی در یافت داده‌های متوالی، همچون مدار خط

خط لوله، مدارات موازی چهار داده همزمان را دریافت می کنند و چهار کار را در یک زمان انجام می دهند. از نظر سرعت این عملکرد معادل با عملکرد یک لوله ۴ قطعه ای است. توجه داشته باشید چهار مدار زیر یک سازمان SIMD را تشکیل می دهند چون دستورات یکسانی برای عمل روی داده های مختلف بطور موازی عمل می کنند.



دلایل متعددی برای علت عمل نکردن خط لوله در وضعیت حداکثر سرعت تئوری اش وجود دارد. قطعات مختلف ممکن است دارای زمان های اجرای مختلفی باشد. در نتیجه سایر قطعات مجبورند وقت خود را تا پالس ساعت بعدی بیهوده تلف کنند.
بعلاوه صحیح نیست تصور کنیم که خط لوله دارای زمان تأخیر مشابه با مدار غیر خط لوله است. بسیاری از ثبات های واسط در مدار تک واحدی که مداری ترکیبی است لازم نیستند. با این وجود تکنیک خط لوله امکان پردازش سریعتری را در مقایسه با یک پردازش سری فراهم می آورد، حتی اگر سرعت ماکزیمم تئوری هرگز بدست نیاید.

در طراحی کامپیوتر دو زمینه برای کاربرد خط لوله وجود دارد. خط لوله حسابی اعمال حسابی را بصورت جزء عمل ها برای اجرا در قطعات خط لوله تقسیم می کند. خط لوله دستورالعمل روی رشته ای از دستورات عمل می کند و ضمن آن فازهای برداشت، دیکد و اجرای سیکل دستورالعمل را تواماً اجرا می نماید. دو نوع خط لوله فوق در زیر توضیح داده شده اند:

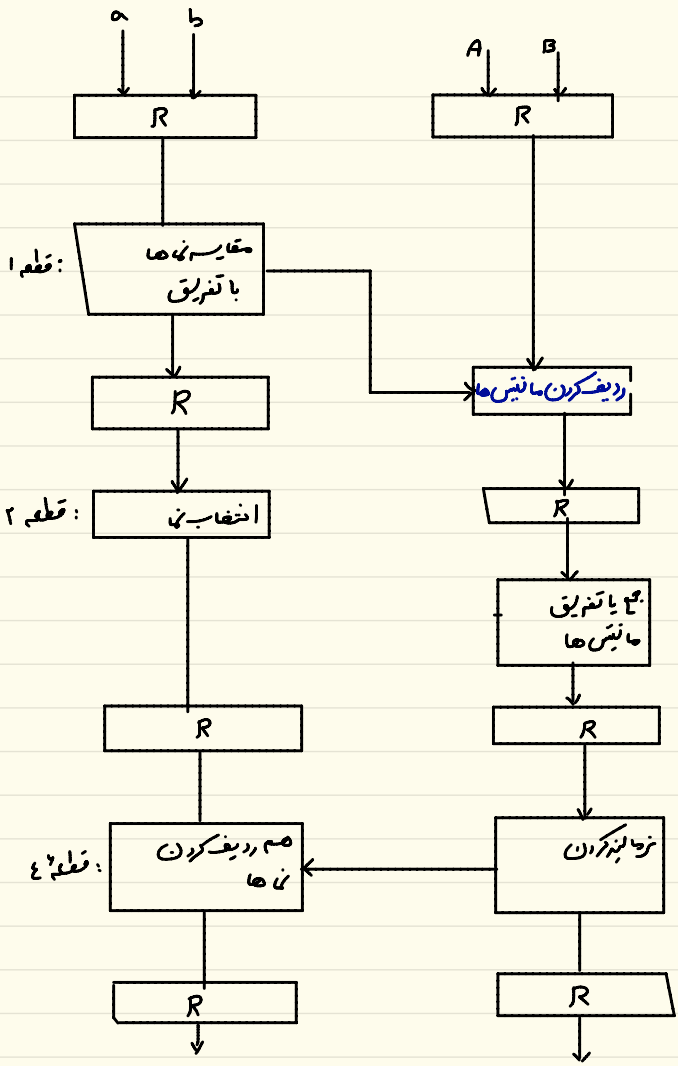
خط لوله حسابی:

خط لوله حسابی معمولاً در کامپیوترهای بسیار سریع یافت می شوند و برای پیاده سازی اعمال ممیز شناور، ضرب اعداد با ممیز ثابت و سایر محاسباتی از این قبیل که در مسائل علمی یافت می شوند به کار می روند. یک ضرب کننده خط لوله ای در اصل یک ضرب کننده آرایه ای مانند شکل صفحه بعد خواهد بود. که در آن جمع کننده های خاصی برای کاهش زمان تأخیر انتشار در محاسبه حاصلضرب های جزئی طراحی شده اند. اعمال ممیز شناور بسادگی به جزء عمل هایی مانند آن چه در بخش ۵-۱۰ آمده تفکیک می گردد. در این جا مثالی از یک خط لوله برای جمع و تفریق شناور را نشان می دهیم. ورودی های خط لوله جمع کننده ممیز شناور دو عدد دودویی ممیز شناور نرمالیزه هستند:

$$X = A \times 2^a \qquad Y = B \times 2^b$$

A و B دو عدد کوچکتر از یک هستند که مانتیس را تشکیل می دهند و a و b نماهای آن ها می باشند. جمع و تفریق ممیز شناور را می توان در چهار قطعه انجام داد. شکل ۶-۹. ثبات هایی که با R مشخص شده اند بین قطعات قرار گرفته اند تا نتایج میانی را ذخیره کنند. جزء عمل های اجرا شده در قطعات عبارتند از:

۱- مقایسه نماها
۲- هم ردیف کردن مانتیس ها
۳- جمع یا تفریق مانتیس ها
۴- نرمالیزه کردن نتیجه



این اعمال در شکل بالا با کمی تغییر به منظور کاهش زمان اجرای زیر عمل ها آورده شده است. نما ها توسط تفریق مقایسه شده تا اختلاف آن ها پیدا شود. نمای بزرگتر به عنوان نمای نتیجه انتخاب می شود. تفاضل نما ها نشان می دهد که چند بار مانتیس مربوط به نمای کوچکتر باید به راست شیفت داده شود. به این ترتیب دو مانتیس هم ردیف می شوند. باید توجه داشت که برای کاهش زمان شیفت مدار شیفت دهنده به صورت ترکیبی تهیه می شود. دو مانتیس در قطعه سوم جمع یا تفریق می شوند. نتیجه در قطعه چهار نرمالیزه می شود. وقتی که یک سرریز رخ دهد تعداد صفرهای جلوی مانتیس بیانگر تعداد شیفت به چپ در مانتیس بوده و عددی که باید از نما کسر شود را معین می کند.

مثال عددی زیر می تواند ریز عمل های انجام شده در هر قطعه را مشخص نماید. به منظور سادگی، هر چند که در شکل ۶-۹ اعداد دودویی به کار رفته اند، از اعداد دهدهی استفاده می کنیم. دو عدد هم ارز شناور نرمالیزه زیر را در نظر بگیرید.

$X = 0.9504 \times 10^3$ $\qquad$ $Y = 0.8200 \times 10^2$

دو نما در اولین قطعه از یکدیگر کسر می شوند تا $3 - 2 = 1$ بدست آید. نمای بزرگتر یعنی 3، به عنوان نمای نتیجه انتخاب شده است. قطعه بعدی مانتیس Y را به راست شیفت می دهد تا روابط زیر بدست آیند:

$$X = 0.9504 \times 10^3 \qquad Y = 0.0820 \times 10^3$$

بدین ترتیب دو نما هم ردیف و دارای نمای یکسان می شوند. جمع دو مانتیس در قطعه ۳ حاصل جمع زیر را تولید می نماید:

$$Z = 1.0324 \times 10^3$$

حاصل جمع با نرمالیزه کردن آن تنظیم می گردد بطوری که دارای اولین رقم غیر صفر در قسمت اعشاری باشد این عمل توسط شیفت به راست و افزایش نما بدست می آید. $Z = 0.10324 \times 10^4$

مقایسه گر، شیفت دهنده، جمع کننده، تفریق کننده، افزایش دهنده و کاهش دهنده در خط لوله ممیز شناور بوسیله مدارات ترکیبی ساخته می شوند. فرض کنید که تأخیر در چهار قطعه $t_1 = 60ns$، $t_2 = 70ns$، $t_3 = 100ns$، $t_4 = 80ns$ و ثبات مدار واسط $t_r = 10ns$ باشد. پالس ساعت $t_p = t_3 + t_r = 110$ انتخاب می شود. یک مدار غیر خط لوله ممیز شناور در جمع و تفریق معادل، زمان تأخیری برابر $t_n = t_1 + t_2 + t_3 + t_4 + t_r = 320$ دارد. در این حالت جمع کننده خط لوله سرعتی برابر با $320/110 = 2.9$ برابر نسبت به جمع کننده غیر خط لوله خواهد داشت.

very large Instruction word : VLIW

افزایش کارایی پردازنده از دو جهت است : ۱- تکنولوژی تیمه‌های سریعتر ۲- پردازش موازی

cpu های قدرتمند امروزی برای دسترسی به کارایی بیشتر براساس Instruction level parallism یا ILP پیاده سازی می‌شوند.

- VLIW یکی از روش های طراحی است که تلاش به سطح بالای موازی سازی با استفاده از اجرای دستورالعمل های طولانی دست یابد.

super scalar : پردازنده هایی هستند که دارای چند واحد عملیاتی می‌باشند، به طوریکه تصمیم گیری برای اجرای دستورالعمل ها برعهده سخت افزار است.

Dynamic scheduling : وقتی که پردازنده در زمان اجرا تصمیم بگیرد که کدام دستورالعمل روی کدام واحد عملیاتی می‌بایست اجرا گردد

out of order : اغلب برای کارایی بیشتر اجازه داده می‌شود دستورات مستقل بعدی زودتر اجرا شوند.

speculative prediction : وقتی که شرطی اتفاق می‌افتد برای اجتناب از تاخیر شرط ها پردازنده ها تلاش می‌کنند خروجی شرط را پیش بینی کنند و دستورات را ادامه دهند. اگر خروجی شرط به اشتباه پیش بینی شده عملیات انجام شده undo می‌شود.

Data Dependency : اگر یک دستور محتاج است از نتیجه دستور قبل استفاده نماید این دو دستور به هم وابستگی دارند پس :

- نمی‌توانند با هم اجرا شوند
- دومی نمی‌تواند قبل از اولی اجرا شود

مزایا و معایب super scalar ها :

مزایا : سخت افزار موازی سازی بین دستورات را شناسایی می‌کند

تغییر نام رجیسترها را هم سخت افزار انجام می‌دهد

سازگاری باینری : اگر واحد جدیدی به cpu اضافه شود و یا معماری cpu تغییر کند برنامه های قدیمی می‌توانند با cpu جدید کار کنند.

معایب : معماری پیچیده برای یافتن دستورات

محدودیت پنجره اجرا

مزایا و معایب VLIW :

مزایا :

- سخت افزار ساده : واحدهای جدید بدون نیاز به سخت افزار اضافی برای شناسایی دستورات اضافه می‌شوند
- کامپایلر می‌تواند شناسایی را بین کل دستورات انجام دهد.

معایب :

- تعداد زیادی رجیستر برای نگهداری عملوندها و نتایج لازم است.
- حجم تبادل اطلاعات بالا : بین FU ها و رجیسترها و ...
- پهنای باند بالا بین واحد fetch و Instruction cache
- بزرگ بودن کد دستورات و جاهای خالی بودن opcode ها
- ناسازگاری باینری

در VLIW این که کدام دستورات می‌بایست به صورت موازی اجرا شوند به عهده کامپایلر می‌باشد.

کامپایلر برنامه را آنالیز کرده و عملیات را برای اجرا در موازی سازی کشف می‌نماید، سپس عملیات درون یک دستورالعمل بزرگ یک می‌شود.

Issue packet : مجموعه از دستورالعمل ها که در یک packet قرار می‌گیرند و در یک کلاک صادر می‌شوند

تفاوت های super scalar و VLIW :

روش کار در SC داینامیک و در زمان اجرا و توسط تحت افزاری باشد ، در VLIW روش کار بصورت ایستا و تحت کنترل کامپایلر است .
در SC تعداد واحدهای اجرایی در مجموعه دستورات تأثیری ندارد و هر دستور تنها یک عمل را بیان می کند و در VLIW هر دستور ، بیانگر چندین عملیات می باشد .

تکنیک loop unrolling :

با داشتن منابع موجود و یک حلقه تعداد تکرارهایی که می توان unroll کرد محدودیت دارد
این روش فضای حافظه مورد نیاز را افزایش می دهد
یک کامپایلر خوب باید بتواند یک مقدار بهینه برای هر حلقه پیدا کند
تعداد رجیستر های زیادی لازم است و ترافیک بالایی دارد .

Trace scheduling :

تکنیک کامپایلری است که برای اجرای دستورات انشعابی که بر مبنای پیشگویی است کار می کند .
مراحل کار :

- Trace selection
- Instruction scheduling
- Replacement and compensation

وقتی که یک پرش به وجود آید کامپایلر براساس نتایج آمارها حدس می زند که کدام پرش احتمال رخ داد بیشتری دارد سپس دستورالعمل ها را برای آن مسیر اجرا می کند اگر حدس پرش اشتباه باشد دستورات قسمت دوم پرش ، علت دستورات اجباری به طور کند تر اجرا میشود .
- این روش یک بهینه سازی کامپایلری است به طوریکه طوری که بیشتر معتبر بر اتفاق افتادن است زودتر اجرا می شود .
- اجرای دستورات اضافی در حالت با احتمال کمتر ممکن است .
- مسیر صحیح همیشه اتفاق خواهد افتاد هر چند اگر این پیشگویی توسط کامپایلر صحیح نباشد اجرا به خاطر کد جبرانی کند تر می گردد
- همه موارد در زمان کامپایل انجام می گیرد .

Branch prediction :

- یک تکنیک کامپایلر بسیار سریع برای تولید کد دستورالعمل در سطح موازی می باشد .
ایده : اجازه دهیم دستورات از هر دو انشعاب به حالت موازی بروند قبل از این که وضعیت انشعاب ارزیابی شود .
- زمان از دست رفته ای برای شکست پیش بینی وجود دارد .

Branch prediction :

فرض برای یک انشعاب و سپس همراهی کردن آن یکی اگر فرض اشتباه بود تمام کار را خنثی می کنیم .
Branch predication ، هر دو انشعاب شروع می شوند وقتی رجیستر خبری آمد دستورات درست اجرا شده و بقیه دور ریخته می شوند .