برای شروع میخوام یک مثال بزنم.

فرض کنید یک شرکت وجود داره که ۵ میز داره که شما باید وارد اون شرکت شده و به ترتیب از میز ها امضا بگیرید و خارج بشید و وقتی خارج شدید نفر بعدی حق ورود داره یعنی در هر لحظه فقط یک نفر میتونه در شرکت باشه.

دقت کنین که شما در هر لحظه فقط میتونین روی یک میز باشید و بقیه میز ها خالی و بدون کار هستند.

ایده پایپلاین اینه که میگه وقتی نفر اول وارد شد و امضا رو از میز اول گرفت و رفت سراغ میز دوم , حالا نفر دوم به جای اینکه صبر کنه تا نفر اول خارج بشه , وارد بشه و کارش رو بر روی میز اول که الان خالی هست شروع کنه.

مرحله بعد نفر اول میره میز سوم , نفر دوم میره میز دوم , و نفر سوم وارد میشه و میره سراغ میز اول.

در این حالت بعد از ۵ مرحله که نفر اول به میز پنجم میرسه دیگه همیشه تمام میز ها مشغول هستند و بیکار نیستند.

ما هم اینجا کامپیوتری که داریم ۵ تا مرحله داره که به ترتیب اسمشون رو میگم و توضیحاتی راجبش میگم.

مرحله اول Fetch Instruction هست یعنی گرفتن دستور که در این مرحله ما یک حافظه داریم که دستورات درون اون نوشته شده و ما میریم دستوری که باید اجرا بشه رو میخونیم که به مرحله FI معروف است.

مرحله دوم Decode و Fetch Operand هست که یعنی اینکه ببینیم دستور چیه و همزمان ثبات هایی که واسه کارمون نیاز داریم رو بخونیم . در مرحله اول دستوری که میخونین یک عدد باینری هستش پس نیازه که بدونیم منظور از این عدد چیه که پس عمل decode انجام میشه و درون همین عدد باینری علاوه بر اینکه اعلام کرده دستور چیه همچنین گفته که به چه ثبات هایی نیاز داره که عمل fetch operand انجام میشه و اون ثبات ها (که دوتا ثبات هستند) از بانک ثباتی گرفته میشه (بانک ثباتی بعد از حافظه قرار داره که تمامی ثبات ها اونجا قرار دارند) که به مرحله DO یا d,FO معروف میباشد.

در مرحله بعد یک ALU وجود داره که عملیاته محاسباتی ای که نیاز داریم رو انجام میده که این مرحله به اسم EXE معروف است.

مرحله بعد یک حافظه وجود دارد که این حافظه فقط برای داده است مثل حافظه مرحله اول که فقط واسه دستور بود یعنی در این مرحله حافظه ما فقط داده (عدد) درونش وجود داره که به مرحله MEM معروف است.

در مرحله آخر یک عملیات نوشتن وجود داره که نتیجه عملیات در یک ثبات مقصد ممکنه نوشته بشه که بهش میگن Write Back که نتیجه برمیگرده به همون مرحله دوم که بانک ثباتی بود و در یکی از ثبات ها می نویسه که البته قاطی نکنین اینجا تداخلی پیش نمیاد و شما این مرحله رو یک مرحله جدا فرض کنید که به مرحله WB معروف است.

یک توضیح کلی که در این مراحل هر عمل چگونه اتفاق میفته : ابتدا دستور به صورت یک عدد باینری از حافظه خونده میشه سپس می بینیم دستور چیه و ثبات هاشو میخونیم مرحله بعد که فهمیدیم دستور پیه) مثلا دستور ADD باشه (این عمل رو بر روی ثبات ها اعمال میکنیم در ALU سپس اگر نیاز باشه در حافظه داده ها می نویسیم سپس اگر لازم باشه در یکی از ثبات ها هم می نویسیم.

حالا به ترتیب اول دستور اول رو میخونیم (FI) بعد میفرستیم به مرحله DO و همزمان دستور دوم رو میخونیم و به همین ترتیب هر مرحله یک دستور میره جلو دستور پشت سریش هم جاشو مگیره و میاد جلو.

حالا یه نکته : مرحله اول فرض کنید ۳۰۰ نانو ثانیه طول بکشه , مرحله دوم ۱۵۰ نانو ثانیه , مرحله سوم ۲۵۰ نانو ثانیه , مرحله چهارم ۳۰۰ نانو ثانیه , مرحله پنجم هم ۱۵۰ نانو ثانیه.

خب یه سوال : یه دستور در مرحله دومه و همزمان با اون یک دستور در مرحله سوم , بعد از ۱۵۰ نانو ثانیه دستوری که در مرحله دوم بوده کارش تموم میشه اما دستور حاضر در مرحله سوم هنوز زمان نیاز داره , خب چیکار کنیم ؟

این مشکل رو در مراحل چگونه برطرف کنیم ؟

میان یک مانع بین هر دو تا مرحله قرار میدن که اجازه ندن کسی بدون اجازه از مرحله ای عبور کنه یعنی حتی اگه کسی کارش تموم شد تا ما اجازه ندیم نمیتونه بره مرحله بعد.

حالا ما چه زمانی باید اجازه بدیم که رد بشن ؟

میایم بین این مراحل , زمان اون مرحله ای که بیشترین زمان رو داره انتخاب میکنیم و میگیم هرکسی در هر مرحله ای باشه باید ۳۰۰ نانو ثانیه در اون مرحله باشه) در اینجا ۳۰۰ نانو ثانیه

شده ممکنه زمان ها تغییر کنه در جاهای مختلف پس منظور ما همون max گرفتن بین زمان های مراحل هست (و بعد از ۳۰۰ نانو ثانیه باید بره مرحله بعد.

خب الان مطمعن هستیم که هرکسی در هر مرحله ای باشه میتونه در این زمان تعیین شده کارش رو تموم کنه چون ما بیشترین زمان ممکن رو در نظر گرفتیم که یک مرحله میتونه طول بکشه و همچنین میدونیم که دیگه تداخل پیش نمیاد چون همه با هم از یک مرحله میرن به مرحله بعدی.

خب اینکه می یک مانع گذاشتیم تا کسی رد نشه بعد از اینکه اجازه بدیم تا اینکه دستور بخواد عبور کنه و برسه به مرحله بعد یک زمان خیلی کوتاه طول میکشه که بهش میگن delay که با d نشونش میدن.

به زمانی که یک دستور در یک مرحله شروع به کار میکنه تا اینکه در مرحله بعد آماده به کار باشه) یعنی این max زمان رو صبر کنه و از اون مانع با زمان d عبور کنه (بهش میگن) τتاو ( ) (tao)همون T خودمونه یکم کج و کوله تر ☺ که برای راحتی میتونین ح بنویسین!!! که ما از این به بعد با t نشونش میدیم) پس t یعنی بیشترین زمان بین مراحل بعلاوه. delay

تا قبل از پایپلاین یک دستور وارد سیستم میشد هر مرحله رو انجام میداد و T ثانیه طول میکشید) این واقعا T هستش یعنی جمع زمان کل مراحل که البته اینجا دیگه پایپلاین نیست و مرحله ای وجود نداره و همه با هم یکی اند (تا اولین دستور خارج شود و بعد دستور دوم میومد و T ثانیه بعد خارج میشد یعنی یه دستور میتونست در هر لحظه در سیستم باشه پس اگر ما n تا دستور داشته و کدوم T ثانیه طول بکشه تا تموم بشن پس برای اجرای این دستورات ما به n * T ثانیه زمان نیاز داریم.

حالا که پایپلاین داریم اگر فرض کنیم این سیستم به K مرحله (اینجا ۵ مرحله) تقسیم شده باشه , ما از وقتی که دستور اول در مرحله آخر باشه , بعد از زمان t دستور اول خارج میشه و دستور دوم میاد مرحله آخر , t ثانیه بعد دستور دوم خارج میشه و دستور سوم میاد مرحله آخر , به همین ترتیب میشه گفت اینجا ما هر t ثانیه یک خروجی داریم در حالی که قبلا هر T ثانیه یک خروجی داشتیم , فقط باید این رو هم در نظر بگیرین که در ابتدا باید دستور اول وارد بشه و به مرحله آخر برسه بعد از اون دیگه هر t ثانیه یک خروجی داریم که برای اینکه دستور اول به مرحله آخر برسه باید K مرحله رو رد کنه که هر مرحله t ثانیه براش زمان میبره پس t * K زمان لازمه دستور اول به مرحله آخر رو رد کنه و بره بیرون (یعنی تمام مراحل

رو رد کنه بره بیرون و دستور دوم به مرحله آخر رسیده باشه) حالا دیگه n-1 دستور بعدی هرکدوم بعد از t ثانیه خارج میشن.

حالا اگه بخوایم زمان اجرای n دستور در پایپلاین K طبقه رو حساب کنیم میشه:

زمان اجرا $t * (n+k-1) = (n-1) * t + (t * K)$ =

یعنی دستور اول بره بیرون بعلاوه n-1 دستور باقیمانده هم بعدش برن بیرون.

یک فرمول داریم به اسم speed up که میگه زمان اجرای بدون پایپلاین رو بر زمان اجرای با وجود پایپلاین تقسیم کنید پس داریم:

$$\text{Speed up} = (n * k) / (n+k-1) * t$$

برای مثال فرض کنید ما یک سیستم داریم که هر دستوری که وارد میشه ۲۵ ثانیه بعد خارج میشه یعنی , T=25 حالا فرض کنید این سیستم رو تبدیل کنیم به یک پایپلاین ۵ طبقه (k=5)و با delay برابر ۱ ببینیم چه وضعیتی براش پیش میاد:

اول اینکه کلش ۲۵ ثانیه است و چون نگفته هر مرحله چقدر زمان باید داشته باشه پس ما تمام زمان هارو برابر فرض میکنیم یعنی ۵ مرحله ک چون کل سیستم ۲۵ ثانیه است پس هر مرحله ۵ ثانیه داره پس max مراحل همون میشه ۵ به همراه ۱ که از delay داریم پس داریم. t= 6

$$\text{Speed up} = n * 25 / (n+4) * 6$$

اگر فرض کنیم که تعداد دستورات یعنی n برابر با بینهایت باشد میتوانیم speed up رو به صورت زیر در نظر بگیریم:

$$\text{Speed up} = 25 / 6$$

حالا مثال دیگری ببینیم که در آن یک پایپلاین ۵ طبقه (k=5) داریم با زمان های ۸ , ۶ , ۳ , ۶ , ۲ و با delay برابر ۱ حالا speed up رو حساب کنیم:

ابتدا باید t رو محاسبه کنیم که max این مراحل برابر ۸ است که با ۱ از delay پس داریم t=9 :

حالا دقت کنید فرمول speed up میشه n*T تقسیم بر t*(n+k-1) که T برابر زمان کل و t همون تاو است که گفته شده و ما الان T رو نداریم که زمان کل است و بدیهی است که زمان کل یعنی جمع زمان مراحل که اینجا ۲+۶+۳+۶+۸=۲۵ پس داریم:

$$\text{Speed up} = n * 25 / (n+4) * ۹$$

که باز اگر n را به بینهایت میل بدهیم:

Speed up = T / t = 25 / 9

خب پس نکته ای هم که اینجا متوجه شدیم اینه که وقتی n زیاد باشه دیگه اون دستور اول که باید به مرحله آخر برسه زمانش به چشم نمیاد و speed up برابر میشه با. T / t

حالا میخوام در مورد چالش های موجود در پایپلاین صحبت کنم یعنی اینکه شما نمیتونین به این راحتی ای که گفته شد کاراتون رو انجام بدین و همه پشت هم کارشونو انجام بدن و برن.

چالش اول , چالش داده ای است یعنی دستوری در یک مرحله وجود داره که به داده دستوری نیاز داره که هنوز در پایپلاین هست و کارش تموم نشده و اون داده رو به ما نداده.

البته ما ۴ نوع وابستگی داده ای داریم که نوعی که برای ما دردسر درست میکنه همین چزیه که گفتم و اسمش هم هست read after write یعنی بخواین چیزی رو بخونین که دستور قبلی (کسی که از شما زودتر اجرا شده) میخواد تازه بنویسیه اون رو و چون در پایپلاین هستیم هنوز ننوشته و این مشکل سازه.

نوع دیگه write after write هست که یکی میخواد داخل x بنویسه و دستور بعدی هم میخواد در x بنویسه که مشکلی نداره خب بنویسن چون دارن به ترتیب می نویسن مشکلی به وجود نمیاد.

بعدی read after read است که یکی میخواد x رو بخونه بعدی هم میخواد x رو بخونه , خب بخونن چیکار به کار هم دارن.

بعدی write after read است که یکی x رو میخونه بعدش یکی میخواد در x بنویسه که چون اون زودتر چیزی که میخواسته رو خونده پس مشکلی نیست.

پس مشکل زمانی پیش میاد که یکی میخواد در x چیزی بنویسه بعد ما برش داریم اما چون در پایپلان بعد از وارد شدن اون منم وارد میشم پس قبل از نوشتن اون در x من میخوام x رو بخونم و چون هنوز در x چیزی نوشته نشده به مشکل بر میخورم که این وابستگی داده ای از نوع read after write است.

حالا برای رفع این مشکل باید چیکار کرد ؟

پاسخ اول اینه که باید سوخت و ساخت!

بیایم ببینیم اصلا مشکل کجا پیش میاد وقتی همچین چالش داده ای به وجود میاد ؟

مشکل read after write است پس مشکل زمانی هست که میخوای read کنی یعنی چیزی رو بخونی که هنوز نوشته نشده.

همونطور که گفتم شما در مرحله دوم مقدار مورد نظرتون رو از ثبات مورد نظرتون میخونین پس یعنی وقتی این مشکل باشه شما در مرحله دوم وقتی میخوای مقدار x رو بخونی دچار مشکل میشی.

مشکلت اینه که شما باید مقدار جدید x رو بخونین اما دستوری که مقدار جدید x رو داره هنوز از پایپلاین خارج نشده و مقدار جدید x رو جایی قرار نداده که شما بتونین بخونینش.

راه حل مدارا کردنمون اینه که خب باید در پایپلاین صبر کنین و جلو نرین تا اینکه اون دستور جلوییه مقدار جدید x رو در یک ثبات بنویسه یعنی مرحله write back که عمل نوشتن در ثبات اتفاق میفته , و بعد از اینکه نوشت شما برین بخونینش و ازش استفاده کنین.

در این مواقع اگر بخوایم در شکل نشون بدیم مثل زیر میشه:

۱) ADD s1,s2,s3

۲) ADD s1,s4,s2

که دستور اول s2 , s3 رو جمع میکنه و حاصل رو میریزه در s1 و دستور دوم s4 , s2 رو جمع میکنه و میریزه در s1 که در این دو دستور وابستگی read after write وجود نداره.

این وابستگی زمانی پیش میاد که وقتی دستور اول در یک ثبات چیزی بنویسه) که اینجا در s1نوشته (و دستور دوم بخواد اون ثبات رو بخونه) که اینجا دستور دوم s4 , s2 رو میخونه و s1رو نمیخونه بلکه مقدار نهایی رو درونش قرار میده که همون ابستگی write after write هست که مشکلی نداره. (

WB					۱	۲
MEM					۲	۱
EXE				۱	۲	
DO			۱	۲		
FI		۱	۲			

شکل بالا بدون وابستگی و چالش هست و بدون مشکل دو دستور ۱ و ۲ تونستن اجرا بشن به این ترتیب که از چپ شروع میشه که اول ۱ وارد مرحله اول یعنی FI میشه و بعد در کلاک بعدی میره به DO و دستور ۲ وارد مرحله اول میشه همزمان باهاش.

به همین ترتیب چون وابستگی ندارن تا آخر ادامه میدن.

حالا به کد و شکل زیر توجه کنید:

۱) ADD s1 , s2 , s3

۲) SUB s4,s5,s1

اینجا دستور اول در s1 می نویسه و دستور دوم) که sub یعنی منها کردن (داره s1 رو میخونه که اینجا وابستگی مورد نظر وجود داره.

به شکل زیر توجه کنید:

WB				۱					۲
MEM					۱			۲	
EXE						۱			
DO			۲	۲	۲	۲	۱		
FI	۱	۲							

در اینجا چون دستور دوم به اول وابستگی داشت پس باید صبر میکرد تا دستور اول عمل Write Backرو انجام بده و سپس در کلاک بعدی بره مقداری که میخواد رو بخونه پس دستور دوم وقتی وارد مرحله DO شد و فهمید که وابستگی داره همونجا صبر کرد تا دستور اول به wbبرسه . وقتی wb دستور اول تموم شد یعنی مقدار جدید وارد s1 شده و دستور دوم میتونه بگیرتش که همونطور که می بینین دستور ۲ بعد از اتمام دستور اول یک مرحله دیگه در DO مونده که در اون مرحله تازه داره میخونه یعنی بعد از اینکه دستور ۱ مرحله wb رو تموم کرد , و سپس به راه خودش ادامه میده و میره.

این نکته هم در نظر بگیرید که اگر دستور سومی هم وجود داشت و به هیچ کس هم وابسته نبود چون دستور ۲ صبر کرده دستور های قبلی هم باید پشتش صبر کنن یعنی در این مراحل که دستور ۲ به خاطر وابستگیش در مرحله DO بوده , دستور ۳ هم باید در مرحله FI می موند و پس از حرکت دستور دوم , دستور سوم میتونست به راهش ادامه بده.

حالا سوال اینجاست تا چندتا دستور ممکنه به یک دستور وابسته باشن ؟

به کد زیر دقت کنید:

۱) ADD s1 , s2 , s3

۲) SUB s4 , s5 , s6

۳) ADD s6 , s2 , s3

۴) SUB s5 , s1 , s2

اگر دقت کنید فقط دستور ۴ به دستور ۱ وابستگی داره زمانی که داره s1 رو میخونه.

به شکل زیر دقت کنید:

wb						۱	۲	۳	۴
mem					۱	۲	۳	۴	
exe				۱	۲	۳	۴		
do			۱	۲	۳	۴			
fi		۱	۲	۳	۴				

دقت کنید که دستور ۴ یک مرحله در فاز do صبر کرده , چرا ؟

چون دستور ۴ به ۱ وابستس و زمانی میتونه از فاز do استفاده کنه که دستور ۱ از فاز wb عبور کرده باشه , دوباره تکرار میکنم باید عبور کرده باشه نه اینکه در اون فاز باشه , پس اینجا وقتی ۴ به do رسید , دید که ۱ هنوز wb رو تموم نکرده پس مجبور شد صبر کنه تا ۱ فاز wb رو تموم کنه بعد ۴ میتونه از do استفاده کنه که اینجا یه بار وایساد , که همین جا هم بگم به همین وایسادن که اینجا مثلا یه بار وایساد میگن حباب که اینجا یک حباب داریم یعنی یه بار وایساد یعنی یک مرحله پایپلاین بدون کار می مونه چون یکی بدونه عقب میفته و فاز ها بدونه به ترتیب خالی می مونن.

حالا نتیجه ای که از این مثال میگیریم اینه که باید تا جایی وابستگی رو با دستورات بالا سری چک کنیم که wb نشده اند که راهش اینه وقتی وارد do شدی در شکل به بالا سرت نگاه کن چه کسایی در پایپلاین هستن و هنوز بیرون نرفتن , اگر به اون ها وابستگی داری باید صبر کنی تا اونا خارج بشن بعد تازه میتونی از do استفاده کنی که حداکثر وابستگی ها ۳ تا دستور بالاتر هستش که میتونن بهت وابسته باشن.

خب این بود روش سوختن و ساختن!

روش های دیگه رو توضیح میدم که کمی بهمون کمک میکنه که وضعیت بهتر بشه.

اول اینکه میگه شما دستوراتی که به هم وابسته هستن رو از هم دور کنین یعنی اگر دستور دوم به دستور اول وابسته هست شما دستور دوم رو ببر عقب تر و مثلا در دستور پنجم اجراش کن یعنی بعد از دستور اول دستور سوم بیاد بعد چهارم بعد پنجم و بعد دوم در ایین حالت دیگه فاصله از ۳ تا بیشتر شده و حباب به وجود نمیاد یعنی کسی گیر نمیکنه و همه اجرا میشن.

این روش دردسر های خودشو داره چون نمیشه سرخود هر دستوری رو جابه جا کرد بلکه باید ترتیب وابستگی ها رعابت بشه یعنی مثلا دستور ۵ به ۲ وابسته است در همین مثالی که ۲ به ۱ وابسته بود , حالا دیگه شما نمیتونین اول دستور ۵ رو اجرا کنین بعد دستور ۲ را , یا اینکه وابستگی ای وجود نداره اما بعد از اینکه ۲ اومد بعد از ۵ یک وابستگی از ۲ به ۵ ایجاد شد یعنی ۵ داره در چیزی می نویسه که ۲ داره میخونه , چون ۲ زودتر داره میخونه پس مشکلی نیست اما اگر ۲ بعد از ۵ بیاد حالا مشکل به وجود میاد چون اول ۵ می نویسه و بعد ۲ می خونه.

پس در جا به جایی باید کاملا هوشیار باشین که ترتیب وابستگی ها عوض نشه و وابستگی اشتباهی به جود نیاد که این وابستگی جدید محدود به ۳ خط نیست بلکه باید مراقب باشین اون دستوری که داره جا به جا میشه چیز هایی که داره میخونه نباید جزو ثبات هایی باشن که در دستورات زیریش دارن نوشته میشن وگرنه نمیشه جابه جا کرد چون مقدارش عوض میشه.

بیشتر از این توضیح نمیدم این قضیه رو که باعث گیج شدن و دور شدن از مبحث اصلیمون نشه.

روش بعدی ای که یکم کمکمون میکنه یا به عبارتی یک حباب ازمون کم میکنه اینه که میگن زمان خوندن ثبات و یا نوشتن در ثبات یعنی فاز do و فاز wb هر کدومشون نصف t هستن یعنی نصف زمانشون خالیه پس میان میگن به جای اینکه وقتی یکی وابستگی داره صبر کنه تا فاز wb طرف مقابل تموم بشه میگن که همزمان با wb اون دستور , این دستور هم عمل خوندن در فاز doرو انجام بده به این صورت که در نصف زمان عمل wb انجام میشه و در نصف بعدی عمل readانجام میشه و میتونه مقدار جدید رو بخونه یعنی دیگه وقتی ۲ به ۱ ابسته باشه باید تا زمانی که ۱ به wb میرسه صبر کنه و وقتی به wb رسید دیگه ۲ میتونه از do استفاده کنه و وقتی ۱ خارج شد دیگه ۲ میره مرحله بعدی یعنی exe که در این حالت یه مرحله جلو افتادیم.

میشه دوتا راه حل قبلی رو نیز با هم ادغام کرد تا جواب بهتری بگیریم.

راه حل بعدی و راه اصلی ای که باید بیان کنم forwarding است که در زیر به توضیح اون می پردازم.

ما وقتی وابستگی داشتیم مشکلمون چی بود ؟

بله ما به مقداری نیاز داشتیم که هنوز آماده گرفتن نشده , اینجا ثبات بهونست یعنی جایی واسه نگهداریه مقداره یعنی ما به مقدار نیاز داریم نه به ثبات.

خب این مقداری که داریم کجاست چطوری داره آماده میشه ؟

معمولا این مقدار باید در ALU مجاسبه بشه مثلا در دستور ADD s1 , s2 , s3 باید s2 با s3 در ALU جمع بشن تا اون مقدار به دست بیاد.

یعنی این مقدار بعد از اجرای فاز exe آمادست ؟

پس دوتا فاز بعدی چی ؟

بله معمولا این مقدار بعد از فاز exe آمادست و در فاز mem معمولا کاری نمیکنن و سپس wbانجام میشه و اون مقدار در ثباتی نوشته میشه و سپس ما میتونیم ازش استفاده کنیم.

حالا این مقداری که ما نیاز داریم و هنوز wb نشده که بگیریم رو کجا بهش نیاز داریم چرا اصلا داریم این مقدار رو میگریم ؟

معمولا میخوایم بدیمش به ALU تا برامون با این مقدار محاسبات انجام بده.

یعنی قراره (معمولا) چیزی که در خروجی ALU قرار داره از mem عبور کنه بعد wb بشه بعد من بگیرمش بعد بدمش به ورودی ALU ؟؟؟

خب بیا بعد از اینکه در ALU اون مقدار رو محاسبه کردی , حالا که من بهش نیاز دارم یه کپی ازش بده به من کارم راه بیفته بعد تو خودت بیا بنویسش تو ثبات که نفرات بعدی که اومدن مقدارشو بخونن , فعلا مقدارشو یه کپی ازش بده به من که الکی ۳ تا مرحله منتظرت نمونم.

پس میان از خروجی ALU یک کپی میگیرن میدن به ورودی ALU حالا اگر دستور قبلیت به مقداری که قراره wb کنی نیاز داره میتونه بدون اینکه صبر کنه بگیرتش.

در اینجا دستور دوم این مقدار رو باید بده به ALU پس از ثبات چیزی نمیگیره به جاش این کپی از خروجی ALU رو میگیره.

حالا اگر دستور سوم بیاد و به دستور اول وابسته باشه چی ؟

الان که خروجی ALU تغییر کرده و دیگه مقدار قبلیش نیست الان یه مرحله رفتیم جلو.

خب خودتون دارین میگین یه مرحله رفتن جلو پس خروجی مرحله بعد ALU یعنی خروجی memرو بده به ورودی ALU تا دستور سوم منتظر خروجی دستور اول نباشه.

دستور چهارم اومد و به دستور اول وابسته بود هم با همون روش که در نصف کلاک write انجام بشه و در نصف بعدی read انجام بشه مشکل رو حل میکنن.

Forwardingجزییات بیشتری داره که همون طور که در اول مقاله گفتم ما در این مقاله وارد جزییات نمیشیم ولی در کل بگم که lw یعنی خوندن از mem و دستور st یعنی نوشتن در mem که با حافظه دوم در ارتباطن.

اگر دستور اول lw باشه و دستوری دوم st باشه و بخواد همون چیزی که از حافظه load میشه رو در حافظه store کنه به جای اینکه صبر کنه load بشه بعد wb بشه بعد بخونه بعد بره alu بعد store کنه میان میگن خروجی mem رو یه کپی ازش بدین به ورودی mem که اگر خواستیم lwکنیم و بعد همونو st کنیم حباب به وجود نیاد.

تنها جایی که پایپلاین نتونست کاری کنه این بود که شما بخواین چیزی رو load کنین و در مرحله بعد در فاز ALU ازش استفاده کنین دیگه مجبورین یه مرحله در ALU صبرکنین که memخروجی بده بعد از خروجیش کپی بگرین پس اگر دستوری lw بود و دستور بعدی خواست از مقدارش استفاده کنه باید یه مرحله در فاز ALU صبر کنه.

مثل:

۱) lw s0 , 0(s2)

2) add s1 , s0 , s3

که اینجا یک مقداری در s0 نوشته میشه که در دستور بعدیی قراره اون مقدار خونده بشه و در فاز ALU مثلا اینجا جمع بشه که پس یک حباب خواهیم داشت.

پس اگر در جایی forwarding پیاده سازی شده بود دیگر وابستگی های داده ای برامون مهم نیست (به جز همین یک مورد خاص) و همه دستورات بدون صبر کردن عبور میکنن (که هرجا این دو دستور پشت سر هم بودن یک حباب خواهیم داشت).

پس اگر در سوالی آمد که دو دستور lw , add پست هم قرار دارند , اگر forwarding نباشه که کلا باید صبر کنه و ۳ حباب به وجود میاد به ازای هر دو دستور ولی اگر forwarding باشه دیگه دقت کنید که فقط در این مدل خاص یک حباب داریم به ازای این دو دستور) اگر دقیقا پشت سر هم بیان و add بعد از lw اومده باشه. (

خب به نظرم در همین حد کافیه که در مورد چالش های داده ای صحبت کنیم الان باید بریم سراغ چالش بعدی یعنی چالش دستوری.

قبلش بگم که شما وقتی کد می نویسی ممکنه بخوای از یک خطی بپری بری چند خط اونورتر یعنی اون ما بین چند خط اجرا نشن یا بخوای برگردی عقب و حلقه درست کنی مثل for و یا به هر دلیلی بخوای پرش داشته باشی اونم از نوع شرطی یعنی با یه شرطی بپر وگرنه نپر مثلا میگی اگر n=10 شد بپر برو فلان خط رو اجرا کن در غیر این صورت برو خط بعد , خب در سیستم از کجا بفهمه شرط برقرار هست یا نه ؟

شما می تونین دوتا ثبات رو باهم مقایسه کنین , به کد زیر دقت کنید:

Beq s1 , s2 , lable

این کد میگه اگر s1 , s2 با هم برابر بودن بپر به خط (lable که اینجا lable اسم یک خط است مثلا بپر به خط ۱۰۰.)

حالا سوال اینجاست که در سیستم چطوری می فهمیم s1 , s2 با هم برابرند یا نه ؟

راهش اینه که اینه که اولا باید دسور رو بخونیم تا بفهمیم دستور اینه , بعد بریم این دوتا ثبات رو از بانک ثباتی مقدارشونو بگیریم , بعد بدیمشون به alu تا این دوتارو منها کنه و اگر جوابشون صفر شد پس با هم برابرند و بعد می فهمیم که قراره پرش انجام بدیم یا باید بریم خط بعد.

پس دیدم که تا بخوایم بفهمیم پرش اتفاق میافته یا نه ۳ مرحله طول میکشه حالا سوال اینجاست که وقتی در پایپلاین یک دستور پرشی شرطی مثل این داشتیم و وارد پایپلانش کردیم , وقتی این دستور میره از مرحله اول به مرحله دوم , حالا چه دستوری باید وارد مرحله او بشه ؟

ما گفتیم باید دستور بعدی در پایپلاین وارد بشه اما اینجا دستور بعدی کیه ؟ آیا خط بعدیشه یا خطی که باید بهش بپره ؟

وابستگی دستوری یعنی این و راه حل هایی براش داریم:

اولی همون جا به جایی دستوره با همون قوانین پیچیده ای که گفتیم یعنی اگر دستور پرشی شمارش ۱۰ هست این دستور رو نفر ۸ ام وارد FI کن (دستور شماره ۱۰ زودتر وارد شد) بعد که رفت , DO دستور شماره ۸ رو وارد FI کن و بعد که دستور پرسی رفت در ALU دستور شماره ۹ رو وارد کن و مرحله بعد دیگه معلوم شده پرش اتفاق میافته یا نه که دیگه هر کروم که درسته وارد میکنی.

این جابه جایی دستورات به شرط نبودن وابستگی بین این ۳ دستور قابل انجا هست.

مثال:

) add s1 , s2 , s3۸

9) sub s1 , s2 , s3

10) beq s4 , s5 , 20

11) .....

WB					10
MEM				10	8
EXE			10	8	9
DO		10	8	9	20 or 11
FI	10	8	9	20 or 11	

که ۲۰ یا ۱۱ بودن در این جدول بستگی به مساوی بودن دو ثبات s5 , s4 داره که تا اون موقع مشخص میشه.

روش بعدی hardware interlock هست و اینو میگه که وقتی دستور پرشی وارد شد بعدش هیچکس رو وارد نکن بذار خالی بمونه تا وقتی که این دستور پرشی فاز ALU رو ردکنه و بفهمیم که قراره کدوم دستور اجرا بشه و اونوقت همون دستوری که باید اجرا بشه رو وارد کن.

WB					۱۰
MEM				۱۰	
EXE			۱۰		
DO		۱۰			
FI	۱۰			11 or 20	

طبق همون کد بالا:

که در نظر داشته باشین که دیگه این جا ما دستورات را جابه جا نکردیم و اول ۸ و بعد ۹ اجرا شدن بعد ۱۰ وارد شد که این اتفاق براش افتاد که می بیند ۳ مرحله طول کشید تا بفهمیم

پرش اتفاق میفته یا نه و ۲ حباب داریم چون یک مرحله که در FI بود که دستوری نمیتونست وارد بشه و دو مرحله بعدی که باید یکی وارد میشد و ما وارد نکردیم باعث ایجاد دو حباب شده .

روش بعدی که میگن اینه که چرا حبابا ایجاد کنیم میگن چرا کسی رو وارد نکنیم, بیاین شانسی یکی رو وارد کنیم خدا رو چه دیدی شاید درست دراومد و وقتی دستور پرشی ALU رو رد کرد و فهمیدیم درست وارد کردیم که دیگه حباب نداریم و به کارمون ادامه میدیم ولی اگه اون موقع فهمیدیم اشتباه وارد کردیم خب اونی که وارد کردیم رو حذف میکنیم و دستوری که باید وارد بشه رو از اول وارد میکنیم که در این شرایط اگر همیشه اشتباه حدس زده باشیم تعداد حباب ها میشه مثل) hardware interlock البته با یکم کار کشیدنه بیشتر از سیستم که بیهوده بوده ولی معمولا حدس هامون بیشتر درسته. (

دو مدل داره این روش که یکیش static هست و دیگری. dynamic

روش استاتیک میگه قبل از شروع برنامه یه حدس بزن و بگو که مثلا تمام دستورات پرشی اتفاق میفتد یا نمیفتد , مثلا شما میگید حدس میزنم تمام پرش ها اتفاق میفتد , برنامه به هر دستور پرشی که برسه دیگه صبر نمیکنه که به alu برسه و بعدش دستور درست رو وارد کنه بلکه وقتی دستور پرشی وارد DO شد دستوری که باید به آن پرش کنیم رو وارد میکنه و تا وقتی دستور پرشی به alu برسه صبر میکنه بعدش میبینه درست وارد کرده یا نه.

طبق همون کد قبلی به مثال زیر توجه کنید:

اینجا فرض میکنیم حدسی که زدیم درست بوده و وقتی دستور ۱۰ به ALU رسیده فهمیده پرش باید اتفاق بیفته:

WB					۱۰
MEM				۱۰	۲۰
EXE			۱۰	۲۰	۲۱
DO		۱۰	۲۰	۲۱	۲۲
FI	۱۰	۲۰	۲۱	۲۲	۲۳

حالا فرض کنید وقتی دستور ۱۰ به ALU میرسه مفهمه که پرش نباید اتفاق بیفته و باید دستور ۱۱ اجرا بشه:

WB					۱۰
MEM				۱۰	
EXE			۱۰		
DO		۱۰	۲۰		۱۱
FI	۱۰	۲۰	۲۱	۱۱	۱۲

می بینید که بعد از اینکه ۱۰ فاز EXE را رد کرد فهمید که باید دستور ۱۱ وارد بشه و به اشتباه از ابتدا دستور ۲۰ را وارد کرده است پس بیخیاله ۲۰ شد و ۱۱ رو وارد کرد.

خب این بود روش استاتیک ولی حالا ببینیم روش داینامیک چی میگه:

اینجا هم دقیقا مثل قبله که باید یک حدس بزنی اما دیگه نه از قبل شروع برنامه , اینجا در هنگامی که دستور پرشی رو میبینی حدس میزنی و همواره میشه حدس رو عوض کرد.

الگوریتم اینجوریه که میگه وقتی به یک دستور پرشی رسیدی برو ببین دستور پرشی قبلی چی شده , یعنی اگر اون پریده اینجا حدس بزن که این هم می پره و اگر اون نپریده اینجا هم حدس بزن که اون هم نمی پره.

دقت کنید که تفاوت استاتیک و داینامیک فقط در حدس است و روش اجرا دقیقا یکی است که در استاتیک از قبل یه حدس میزنی واسه پرش ها اما اینجا آخرین پرش رو نگاه میکنی و طبق تجربه تصمیم میگیری.

در این روش داینامیک یک بیت دارین که اگر ۱ باشه یعنی آخرین پرش اتفاق افتاده و اگر ۰ باشه یعنی آخرین پرش اتفاق نیفتاده.

روش دیگری از داینامیک هم هست که به شما ۲ بیت میده تا تجربه بیشتری کسب کنین و حدس درست تری بزنین که شما میتونین با این دو بیت ۴ حالته ۰۰ و ۰۱ و ۱۰ و ۱۱ در نظر بگیرین که پشت سر هم هستن به ترتیب کوچیک به بزرگ.

حالا اگر این دوبیت در ۰۰ باشد یعنی حدس شما نپریدن است , حال اگر پرش اتفاق بیفتد این دو بیت به ۰۱ تغییر میکند اما همچنان حدس نپریدن است , ۱۰ حدس پریدن است و ۱۱

هم حدس پریدن است و با هر پرش به این دو بیت اضافه می شود و با هر عدم پرش از این دو بیت کم میشود , حال اگر یک دستور پرشی ببینید نگاه میکنید این دو بیت چند است و تصمیم میگرید.

مباحثی که نیاز بود در این مقاله گفته بشه به پایان رسید حالا میخوایم با یک مثال کارمون رو تموم کنیم:

فرض کنید کد زیر داده شده و از شما CPI , SPEED UP رو میخواد و همچنین قوانین پایپلاین هم باید بیان بشه از جمله تعداد مراحل و اسمشون که طبق مثال هامون عمل میکنیم و نکته مهم اینه که باید بگن FORWARDING وجود داره یا نه و اینکه بگن از چه روشی برای مقابله با دستورات پرشی استفاده میشه (روش های مقابله با وابستگی داده ای و دستوری باید مشخص بشه) که اینجا ما فرض میکنیم FORWARDING نداریم و روش HARDWARE INTERLOCK ( )جلوگیری از واکشی دستور بعدی) رو هم قراره استفاده کنیم:

۱) add s1 , s2 , s3

۲) sub s2 , s3 , s4

۳) add s4 , s3 , s1

۴) beq s4 , s2 , 1

۵) exit

خب در صورت سوال نیز باید اشاره بشه که دستور پرشی اتفاق میفته یا نه که اینجا میگیم فرض کنید پرش اتفاق نمیفته , این جمله به این معنی نیست که شما از الان میدونین پرش اتفاق نمیفته , یعنی وقتی دستور شماره ۴ به فاز exe برسه بعدش مفهمین که پرش اتفاق نمیفته نه اینکه از قبل بدونین.

WB					۱		۲			۳			۴		۵
MEM				۱	۲			۳			۴			۵	
EXE			۱	۲		۳			۴			۵			
DO		۱	۲	۳	۳	۳		۴	۴	۴			۵		

FI	۱	۲	۳	۴	۴	۴	۴					۵				

همونطور که تا الان متوجه شدید دستور ۲ به یک وابسته نیست و به راهش ادامه داده اما دستور ۳ به دستور ۱ وابسته هستش پس در DO صبر کرده تا ۱ تموم بشه بعد کارش رو انجام داده دقت کنین صبر کرده تا ۱ تموم بشه بعد یک مرحله در DO مونده ثباتی که میخواسته مقدارشو که الان جدید شده گرفته بعد رفته مرحله بعد.

دستور ۴ که پرشی است به ناچار باید صبر میکرد تا ۳ بره بعد بیاد بالا بعدشم که اومد بالا دید به ۲ و ۳ وابستگی داره که چون ۲ دیگه تموم شده وابستگی ای وجود نداره و چون ۳ هنوز در پایپلاین هست پس صبر کرده تا ۳ تموم مبشه بعد به کارش رسیده.

حالا دقت کنین که دستور ۴ پرسی هست و وقتی وارد شده دیگه کسی بعدش وارد نشده طبق HARDWARE INTERLOCK بودنه روشمون و بعد از اینکه EXE رو رد کرد فهمیدیم پرش اتفاق نمیفته دستور ۵ رو وارد کرد.

حالا اول ببینیم SPEED UP چطوری به دست میاد:

تعداد مراحل $K = 5$:

تعداد دستورات اجرا شده (اجرا شده مهمه ممکنه چند تا دستور باشن که از روشون بپریم و اجرا نشن) $N = 5$:

خب t رو که نداریم مهم نیست اینجا اما باید T رو داشته باشم که زمان اجرای یک دستور بدون پایپلاین است.

برای به دست آوردن T در اینجاها باید بگیم ما K مرحله داریم که باید رد کنه هرکدوم با زمان t پس میشه گفت T حدودا برابر است با $T = K * t$: پس در این جاها در فرمو به جای T که نداریم به جاش $K * t$ میذاریم.

حالا برای به دست آوردن زمان دقیق پایپلاین ما این جدول رو کشیدیم.

زمان ایده آل پایپلاین که برابر $(N+K-1) * t$ می باشد اما اینجا می بینید که پایپلاین ایده آل نیست و حباب هایی وجود دارد که اینجا میایم نگاه میکنیم چند مرحله (کلاک) کار انجام دادیم یعنی میایم تعداد ستون های شکل رو میشماریم تا ببینیم چند مرحله کار کردیم که اینجا برابر با ۱۶ است که همانطور که میدانید هر مرحله زمانش برابر با t میباشد پس زمان اجرای پایپلاین برابر با $16 * t$ می باشد.

پس داریم:

SPEED UP = N * K * t / ۱۶ * t = N*K / 16 = 25 / 16

در مورد CPI باید بگم که یعنی CLOCK PER INSTRUCTION به این معنی که به ازای هر دستور چند کلاک نیاز است ؟ که از روی اسمشم میشه فهمید که خودش بنده خدا داره میگه تعداد کلا هارو تقسیم کن (PER) بر تعداد دستورات پس اینجا ببین چند کلاک داری که شمردیم و ۱۶ تا بود و بعد ببین چندتا دستور اجرا شدن که دیدیم ۵ تا بود بعد تقسیم به هم کن یعنی CPI = : ۱۶ / ۵ = ۳,۱ که به این معنی است که به ازای هر یک دستور ما اینجا ۳,۱ کلاک زمان (جدا) صرف کردیم.

اینم از مثال , برای تمرین هم سعی کنین روش ها رو عوض کنید مثلا با FORWARDIN انجام بدین و ببینین چی میشه (که در اینجا دیگه وابستگی داده ای نخواهیم داشت).

چک کنید با روش های استاتیک و داینامیک چه بلایی سر اون دستور پرشی میاد.

یه نکته دیگه بگم اینکه فرض کنید در همین مثال وقتی که گفتم پرش اتفاق نمیفته , اگر بگم این دستور ۱۰۰ بار پرشش اتفاق میفته و بعد اتفاق نمیفته چی ؟

هیچی شما همین شک رو بکشین بعد ببینین دستور ۱ تا ۴ که ۱۰۰ بار قراره تکرار بشن چند کلاک زمان صرف کردن که اینجا از شروع دستور ۱ تا پایان دستور ۴ ما ۱۳ کلاک داشتیم و بعدشم ۳ کلاک تا پایان حالا قراره این ۱۳ کلاک ۱۰۰ بار اجرا بشه پس میشه گفت تبدیل میشه به ۱۳۰۰ کلاک و بعدشم که ۳ کلاک و برنامه تموم میشه (البته این روش حدودی حساب میکنه و شما میتونین دقیق تر حساب کنین) از طرفی هم دیگه N = 5 نیست بلکه ۴ دستور دارن ۱۰۰ بار اجرا میشن و بعدشم یک دستور داریم پس N = 401 است در این فرمول.