流水线 CPU 工程化设计方法总结

计组实验的 P5，P6，其设计难度和工作规模都相当的大，尤其是调试工作的艰巨，也是困扰大部分同学的难点。就笔者的观察，许多同学之所以在处理 P6 的过程中遇到困难，很大一部分原因是设计结构上出现了一些问题，导致添加指令的工作变得繁琐，调试也变得十分困难。因此，笔者总结了自己运用高小鹏老师的工程化设计方法完成 P5，P6，以及帮助同学解答在 P5，P6 中遇到的问题的经验，写下这篇流水线工程化设计方法总结，希望能帮到在流水线设计中遇到困难的同学。

食用须知：本总结推荐给已经基本理解单周期基本指令原理、流水线工作原理和数据冒险基本概念的同学食用。同时注意考虑自身病情，可以分次食用。否则可能出现消化不良、头晕脑胀等不适症状，笔者概不负责。

PS：为了给计组实验留点悬念，本文不细讲 J 类指令和乘除法类指令，利用工程化分析方法类推即可！

一、 前期准备与指导原则

就笔者的理解，工程化方法，是在牺牲一定的设计灵活度的基础上，将设计工作模式化、机械化、自动化以求减少开发难度，提高开发速度的一种方法理念。因而，不可避免地在开发过程中会出现一些看似“麻烦”的行为。但是从全局来看，所谓磨刀不误砍柴工，前期准备过程中的麻烦，可以大大减少我们后期的工作量。所以首先，我们谈一些指导性的原则。

1、先写实验报告！先写实验报告！先写实验报告！撰写实验报告的过程，实际上就是你思考这个实验任务，整理明确自己的设计思路，把自己的设计结构完全定型的过程。不可否认的是，在实验做完了之后再撰写实验报告，确实可以轻松的完成任务。但是，我们的任务并不是写实验报告，而是完成这个计组实验！在思路没有定型的情况下就贸然开始写代码，非常容

易造成一个问题就是，写着写着忘记了，或者记混了某个设计细节，尤其容易记混的就是各个控制信号的意义和数据端口的设计。这样细小的问题一旦出现就很难再找出，之后往往要付出大量的时间 DEBUG，得不偿失。预先把模块功能、结构设计、端口定义、控制信号都整理清楚写在报告里，则工作的时候思路会非常清晰，大大减少无谓的 BUG。

2、规范化命名。在流水线 CPU 的设计中，不可避免的会出现大量的端口和用来在端口中传输数据的 wire 变量，如何给它们命名是一个非常重要的问题。一旦端口的名字混淆，很容易产生 BUG 而且十分难以发现。笔者所使用的命名方式是：对于 wire 型变量，统一使用如下的“信号名_阶段名_方向”来命名，如”ALUout_E_out”（E 阶段 ALU 的计算结果的输出端口，传送到 EM 级寄存器），”ALUout_EM_in”（输入 EM 级流水线寄存器的 ALU 计算结果）；对于控制变量，用”模块名_端口名”来明明，如 ALU_op；对于多选器的选择变量，统一用“多选器名+sel”，如” MUX_ALUAsel”。总之，无论采取何种命名原则，务求意义明确、避免歧义，而且从一而终。为此多花费的编码时间是完全值得的。

3、模块化结构。把每一个流水级作为一个模块，每一级流水线寄存器作为一个模块，数据只能在相邻模块之间传递。有的同学把所有模块都统一放在 MIPS 顶层文件里，这固然是没有错的，但是由于各级的数据线路和信号混杂在一起，尤其是分布式译码的五个完全一样的译码器混杂在一起，给代码的修改带来的极大的风险。因为修改或者增减某个端口，其要连带修改的端口是非常多的，如果混杂在一起，则非常容易漏改端口造成 BUG。把每个流水级独立构建成模块，可以一定程度上解决这个问题。

4、分布式译码。推荐各位把指令传递到每一级，然后在每一级都用一个译码器译码。这样不仅减少了流水级之间的数据传输端口数量，也省去了判断哪些控制信号需要传递的功夫。实现起来非常简单，编写一个全能译码器，然后在不同的流水级分别实例化一次，然后只需要

连接本级需要的端口即可。

二、 数据通路的构建

在 P6 的工作过程中，工作量最庞大的一部分，无疑是那 50 条指令的添加了。在 P5 只有 10 条指令的情况下，有的同学还可以完全依靠脑补的方式构建数据通路和结构，或者依靠实验指导书上的参考结构图（不得不吐槽那个参考结构图有错误……至少很容易让人发生误解）。而到了 P6，指令条数大大增加，纯粹依靠自己自由发挥的做法无疑会带来一定的困难。而工程化的数据通路构建，可以大大简化这一过程到无脑的地步（笔者增加 P6 的全部指令只花了不到一个小时）。而这一方法的核心，就是数据通路标准表格。

第一步：构造表头。

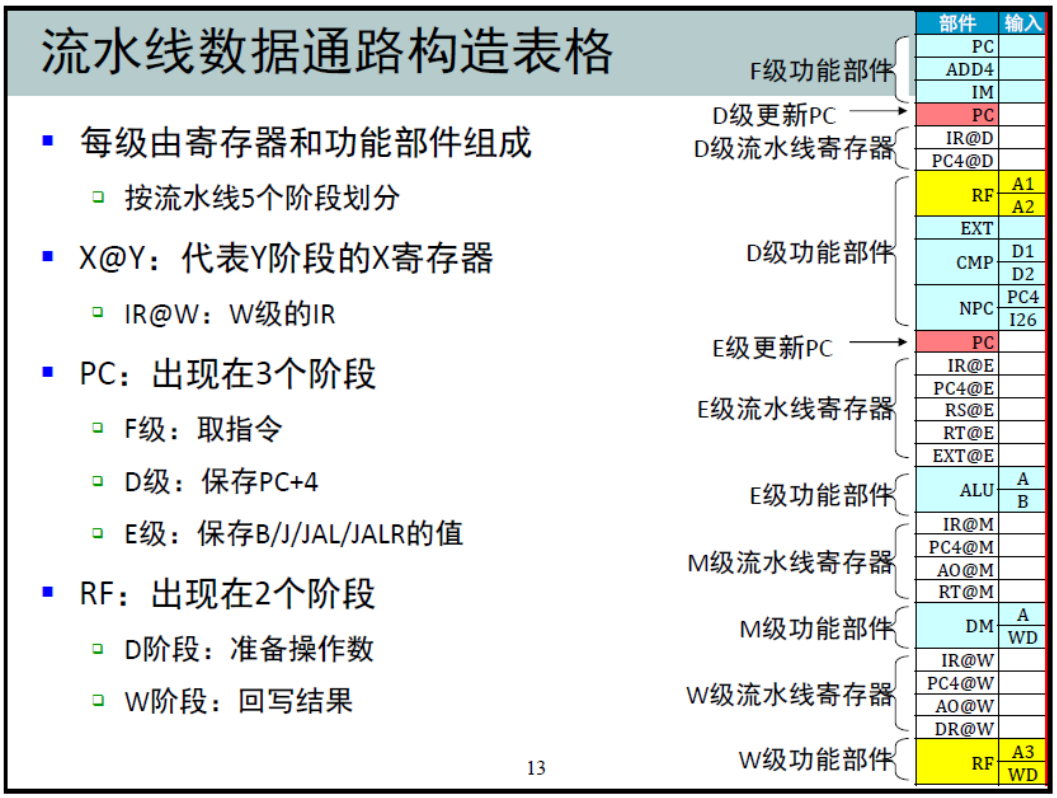


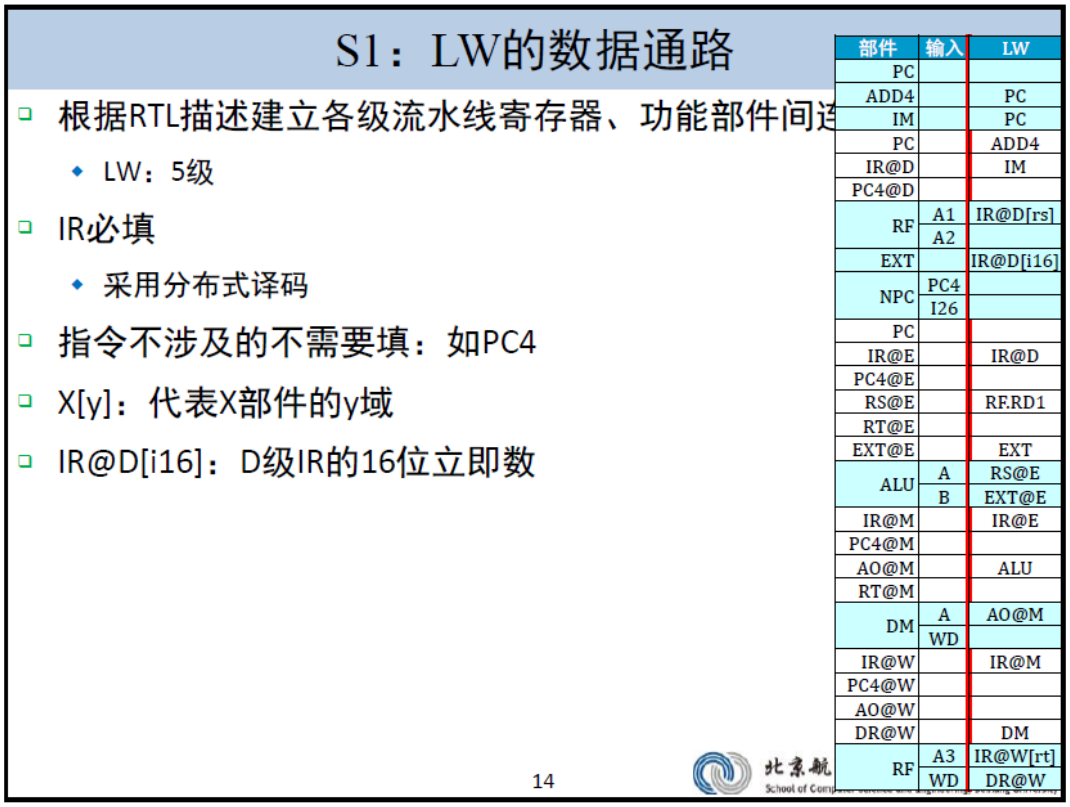
图-1 表头，来自高小鹏老师讲义

在表头中，我们需要写出每一个功能模块，以及这些功能模块的每个数据输入。注意，表

头中不包括控制信号输入（如 op,en），也不包括输出。如图 1，CMP 有两个数据输入，ALU 也有两个数据输入，而 RF 我们把读功能和写功能分别写在 D 级和 W 级（在空间上这两个流水级处于同一个地方，但是在时间上，指令先经过 D 级再经过 W 级），因此有四个数据输入。

第二步，分析指令，并填写表格。

在数据通路构建步骤中，我们暂且忽略流水线带来的各种冒险，纯粹按照单周期的方式来思考每一个指令是如何运转，即数据从哪个部件流动到哪个部件，最后结果去向何处，最终写在表格中。如下图为 LW 指令的数据通路构造图。



部件	输入	LW
PC		
ADD4		PC
IM		PC
PC		ADD4
IR@D		IM
PC4@D		
RF	A1	IR@D[rs]
	A2	
EXT		IR@D[i16]
NPC	PC4	
	I26	
PC		
IR@E		IR@D
PC4@E		
RS@E		RF.RD1
RT@E		
EXT@E		EXT
ALU	A	RS@E
	B	EXT@E
IR@M		IR@E
PC4@M		
AO@M		ALU
RT@M		
DM	A	AO@M
	WD	
IR@W		IR@M
PC4@W		
AO@W		
DR@W		DM
RF	A3	IR@W[rt]
	WD	DR@W

图 2，数据表格填写实例

如图我们可以看到，对于 LW 指令，首先是 PC 的输入来源为 ADD4，指令数据线的来源为 IM(指令存储器)，相应的从 RF 中读取 rs 寄存器的值，从 EXT 中扩展指令中立即数的值，依次传递到 ALU 的部分，计算出内存地址。DM 的地址端口从 ALU 的 OUTPUT（图中简写为 AO）得到读取地址，把读出的数据传递给 RF 的写入数据端口，RF 的写入地址端口则来自指令的 rt 字段，最终完成这一列的填写。

需要注意的是，数据不能跨越流水级寄存器进行传递，所以在 M 级引用 ALU 输出必须经过 E/M 级流水线的一次传递。而某个端口如果在这条指令的执行过程中不需要使用，则留空。

以此类推，每条指令占据一列，我们依次完成所有指令的填写。如下图：

部件	输入	LW	SW	ADDU	SUBU	ORI	BEQ	J	JAL	JALR
PC										
ADD4		PC	PC	PC	PC	PC	PC	PC	PC	PC
IM		PC	PC	PC	PC	PC	PC	PC	PC	PC
PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4
IR@D		IM	IM	IM	IM	IM	IM	IM	IM	IM
PC4@D							ADD4	ADD4	ADD4	ADD4
RF	A1	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]	IR@D[rs]		IR@D[rs]
	A2			IR@D[rt]	IR@D[rt]		IR@D[rt]	IR@D[rt]		
EXT		IR@D[i16]	IR@D[i16]			IR@D[i16]				
CMP	D1						RF.RD1			
	D2						RF.RD2			
NPC	PC4						PC4@D	PC4@D	PC4@D	
	I26						IR@D[i16]	IR@D[i26]	IR@D[i26]	
PC							NPC	NPC	NPC	RF.RD1
IR@E		IR@D	IR@D	IR@D	IR@D	IR@D			IR@D	IR@D
PC4@E									PC4@D	PC4@D
RS@E		RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1				
RT@E			RF.RD2	RF.RD2	RF.RD2					
EXT@E		EXT	EXT			EXT				
ALU	A	RS@E	RS@E	RS@E	RS@E	RS@E				
	B	EXT@E	EXT@E	RT@E	RT@E	EXT@E				
IR@M		IR@E	IR@E	IR@E	IR@E	IR@E			IR@E	IR@E
PC4@M									PC4@E	PC4@E
AO@M		ALU	ALU	ALU	ALU	ALU				
RT@M			RT@E							
DM	A	AO@M	AO@M							
	WD		RT@M							
IR@W		IR@M		IR@M	IR@M	IR@M			IR@M	IR@M
PC4@W									PC4@M	PC4@M
AO@W				AO@M	AO@M	AO@M				
DR@W		DM								
RF	A3	IR@W[rt]		IR@W[rd]	IR@W[rd]	IR@W[rt]			0x1F	IR@W[rd]
	WD	DR@W		AO@W	AO@W	AO@W			PC4@W	PC4@W

图 3，P5.2 的所有指令数据通路

这一部分是工作量最大的部分，也是最强烈建议同学们自己完成的部分。这 9 条指令基本包括了各种指令类型。完成了这一部分，P6 剩余的四十条指令绝大部分都是简单的复制粘贴加微小的修改即可。

第三步，对于可能有多个数据来源的输入端口，构造转发。

观察图 3 我们可以发现，有些端口有着多个可能的数据来源，如 ALU 的 B 端口，输入根据指令不同可能是 EXT@E 或者 RT@E，因此我们需要对这个输入端口前加入一个多选器。在表格里面，我们用如下方式来构建多选：

部件	输入	输入来源			MUX	控制
PC						
ADD4		PC				
IM		PC				
PC		ADD4	NPC	RF.RD1	M1	PCSel
IR@D		IM				
PC4@D		ADD4				
RF	A1	IR@D[rs]				
	A2	IR@D[rt]				
EXT		IR@D[i16]				
CMP	D1	RF.RD1				
	D2	RF.RD2				
NPC	PC4	PC4@D				
	I26	IR@D[i26]				
IR@E		IR@D				
PC4@E		PC4@D				
RS@E		RF.RD1				
RT@E		RF.RD2				
EXT@E		EXT				
ALU	A	RS@E				
	B	EXT@E	RT@E		M2	BSel
IR@M		IR@E				
PC4@M		PC4@E				
AO@M		ALU				
RT@M		RT@E				
DM	A	AO@M				
	WD	RT@M				
IR@W		IR@M				
PC4@W		PC4@M				
AO@W		AO@M				
DR@W		DM				
RF	A3	IR@W[rt]	IR@W[rd]	0x1F	M3	WRSel
	WD	DR@W	AO@W	PC4@W	M4	WDSel

图 4，构造多选器

如图，我们构造输入来源这一个大列，和后面的 MUX（多选器）列以及对应的多选器控制信号列，如果某个数据端口只有一个输入，则这若干列中只需要填一项。如果某个数据端口有多个可能输入，则分别写到数据来源这若干列里，并且在 MUX 列中新建一个多选器和相应的控制信号。

P.S.不推荐高老板图中这样的命名方式，M1234 什么的……推荐用 MUX_ALU_A 命名多选器，用 MUX_ALU_Asel 来命名多选信号。多敲几个字母的时间比起弄混多选器造成的灾难性后果而言简直不值一提。

级别	部件	输入	输入来源				MUX	MUX控制信号	lw	lb
IF级部件	PC									
	ADD4		PC						PC	PC
	IM		PC						PC	PC
D级对PC更新	PC		ADD4						ADD4	ADD4
F/D级流水线寄存器	IR_D		IM						IM	IM
	PC4_D		ADD4							
	PC8_D		ADD4+4							
D级部件	RefFile	A1	IR_D[rs]						IR_D[rs]	IR_D[rs]
		A2	IR_D[rt]							
	CMP	D1	RF.RD1							
		D2	RF.RD2							
	EXT		IR_D[i16]						IR_D[i16]	IR_D[i16]
	NPC	PC4	PC4_D							
		I16	IR_D[i16]							
E级对PC更新	PC		ADD4	NPC	RF.RD1		MUX_PC	PCsel		
D/E级流水线寄存器	IR_E		IR_D						IR_D	IR_D
	PC4_E		PC4_D							
	PC8_E		PC8_D							
	RS_E		RF.RD1						RF.RD1	RF.RD1
	RT_E		RF.RD2							
	EXT_E		EXT						EXT	EXT
E级部件	ALU	A	RS_E	IR_E[10:6]			MUX_ALUa	ALU_asel	RS_E	RS_E
		B	EXT_E	RT_E			MUX_ALUb	ALU_bsel	EXT_E	EXT_E
	XALU	D1	RS_E							
		D2	RT_E							
E/M级流水线寄存器	IR_M		IR_E						IR_E	IR_E
	PC4_M		PC4_E							
	PC8_M		PC8_E							
	ALUout_M		ALUout						ALUout	ALUout
	XALUout_M		XALU_HI	XALU_LO			MUX_XALUout	XALU_outsel		
	RT_M		RT_E							
M级部件	DM	A	ALUout_M						ALUout_M	ALUout_M
		WD	RT_M							
M/W级流水线寄存器	IR_W		IR_M						IR_M	IR_M
	PC4_W									
	PC8_W									
	ALUout_W		ALUout_M							
	XALUout_W		XALUout_M							
	DM_W		DM						DM	DM
W级功能部件	EXT_DM	A	IR_W[1:0]							IR_W[1:0]
		Din	DM_W							DM_W
	RF	A3	IR_W[rt]	IR_W[rd]	31		MUX_GPRwa	RF_WAsel	IR_W[rt]	IR_W[rt]
		WD	ALUout_W	DM_W	PC8_W	XALUout_W	MUX_GPRwd	RF_WDsel	DM_W	DM_W

图 5,笔者构造的 P6 数据通路（后面的指令省略）

完成以上的三个步骤之后，P6 剩余的添加指令部分其实就相当简单了，绝大部分同类指令的数据通路都是完全相同的。只需要在对应的计算模块添加计算功能，以及在译码器模块里构建相应的控制信号即可。实测在 ALU 功能已经事先完备的情况下，笔者和笔者指导的同学都只需要 1~2 分钟即可添加一条指令。非常高效而且不会犯下功能上的设计性错误。

三、 数据冒险的分析

之前我们已经讨论了不考虑冒险的情况下，数据通路的构造方法，相当的简单无脑。然而流水线中最麻烦、最难理解、最容易犯错的部分，就是处理数据冒险的部分。可以说，这部分也是 P5 和 P6 的核心所在，也是同学们棘手的部分。接下来两节，笔者将介绍如何用工程化方法来轻松地解决暂停和转发机制的处理问题。

几乎每一条指令，都需要获取一定的数据输入然后某些指令还会产生数据输出。流水线之所以会产生冒险，就是因为后面指令需求的数据，正是前面指令供给的数据，而后面指令在需要使用数据时，前面供给的数据还没有存入寄存器堆。因此我们从需求数据和供给数据的行为来入手分析暂停、转发情况。

需求者：我们从原理上分析，对于某条指令，实际上需求寄存器数据的是某些硬件部件。如，对于 addu 指令，需要数据的是 EX 级的 ALU，对于 BEQ 指令，需要数据的是 ID 级的比较器 CMP。而对于 SW 指令，需要数据的有 EX 级的 ALU（这个数据用来计算存储地址），还有 MEM 级的 DM（这里需要存入的具体的值）。

供给者：所有的供给者，都是存储了上一级传来的各种数据的流水级寄存器。至于为什么一定要由流水级寄存器来提供数据而不是由 ALU 或者 DM 来提供数据，超出本文讨论范围，各位不妨暂时理解为这是一项硬性的设计要求，具体原因有兴趣可以查阅相关资料或者与老师同学讨论。

分析清楚了数据的需求者和供给者，我们就可以理清处理数据冒险的策略了。假设当前我需要的数据，其实已经计算出来，只是还没有进入寄存器堆，那么我们可以用转发(Forwarding)来解决，即不引用寄存器堆的值，而是直接从后面的流水级的供给者把计算结果发送到前面流水级的需求者来引用。如果我们需要的数据还没有算出来。则我们就只能暂停(Stall)，让流水线停止工作，等到我们需要的数据计算完毕，再开始下面的工作。

那么，如何判断我们需要的数据是否算出来了呢？我们介绍一个简单高效的暂停/转发判定机制：**需求时间——供给时间模型**。

对于某一个指令的某一个数据需求，我们定义 Tuse 为：这条指令位于 ID 级的时候，再经过多少个时钟周期就必须要使用相应的数据。

例如，对于 BEQ 指令，立刻就要使用数据，所以 Tuse=0。

对于 addu 指令，等待下一个时钟周期它进入 EX 级才要使用数据，所以 Tuse=1。

而对于 sw 指令，在 EX 级它需要 GPR[rs]的数据来计算地址，在 MEM 级需要 GPR[rt]来存入值，所以对于 rs 数据，它的 Tuse_rs=1,对于 rt 数据，它的 Tuse_rt=2。

特点 1：是一个定值，每个指令的 Tuse 是一定的

特点 2：一个指令可以有两个 Tuse 值

对于某个指令的数据产出，我们定义 Tnew 为：位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。

例如，对于 addu 指令，当它处于 EX 级，此时结果还没有存储到流水级寄存器里，所以此时它的 Tnew=1，而当它处于 MEM 或者 WB 级，此时结果已经写入了流水级寄存器，所以此时 Tnew=0。

特点 1：是一个动态值，每个指令处于流水线不同阶段有不同的 Tnew 值

特点 2：一个指令在一个时刻只会有一个 Tnew 值（一个指令只有一个结果）

那么，当两条指令发生数据冲突（前面指令的写入寄存器，等于后面指令的读取寄存器），我们就可以根据 Tnew 和 Tuse 值来判断策略。

1、 Tnew=0，说明结果已经算出，如果指令处于 WB 级，则可以通过寄存器的内部转发设计解决（关于内部转发请自行查阅资料），不需要任何操作。如果指令不处于 WB 级，则可以通过转发结果来解决。

2、 Tnew<=Tuse，说明需要的数据可以及时算出，可以通过转发结果来解决。

3、 Tnew>Tuse，说明需要的数据不能及时算出，必须暂停流水线解决。

我们发现，对于暂停，只需要把处在 ID 级的指令与后续指令进行比较即可判断，并且简单的处理。而转发，我们需要比较各级指令，并且在各级之间加入转发数据通路，比较复杂。

因此，我们先讨论暂停机制。

四、 暂停机制的构建

由上一节的讨论，我们知道，暂停机制的判定是相当轻松的。也就是比较当前处在 ID 级的指令的 Tuse 与处在后续流水级的指令的 Tnew 比较，如果二者读写同一寄存器且对应的 Tnew>Tuse，则暂停流水线。

同样的，我们采用工程化的分析、构造方法，通过暂停机制表格来简单明了的分析各个情况，避免了脑补过程中产生的各种错误。

首先，我们把各种指令根据它们的行为进行分类，同一类指令的工作流程是相似的。

Cal_r 类（R-R，寄存器与寄存器进行计算）：add,addu,or,等。

Cal_i 类（R_I，寄存器与常数进行计算）：addi,lui,ori 等。

Beq 类（通过 CMP 比较器判断跳转）：beq,bne,bgez 等。

Load 类（读取内存的值）：lw,lb,lbu,lh 等。

Save 类（保存值到内存）：sw,sb,sh 等。

J 类的四条指令各自都有微妙差别，推荐四条指令分开处理。

其中，只有 cal_r,cal_i,load 三类指令和 jal,jalr 会产生结果。（jal,jalr 在接下来的分析中省略，请自行思考！）

接下来，我们用横行表示处在 ID 级，需求数据的各个指令；用纵列表示后面的各个流水级，能够提供数据的各个指令。构造一个表格框架：

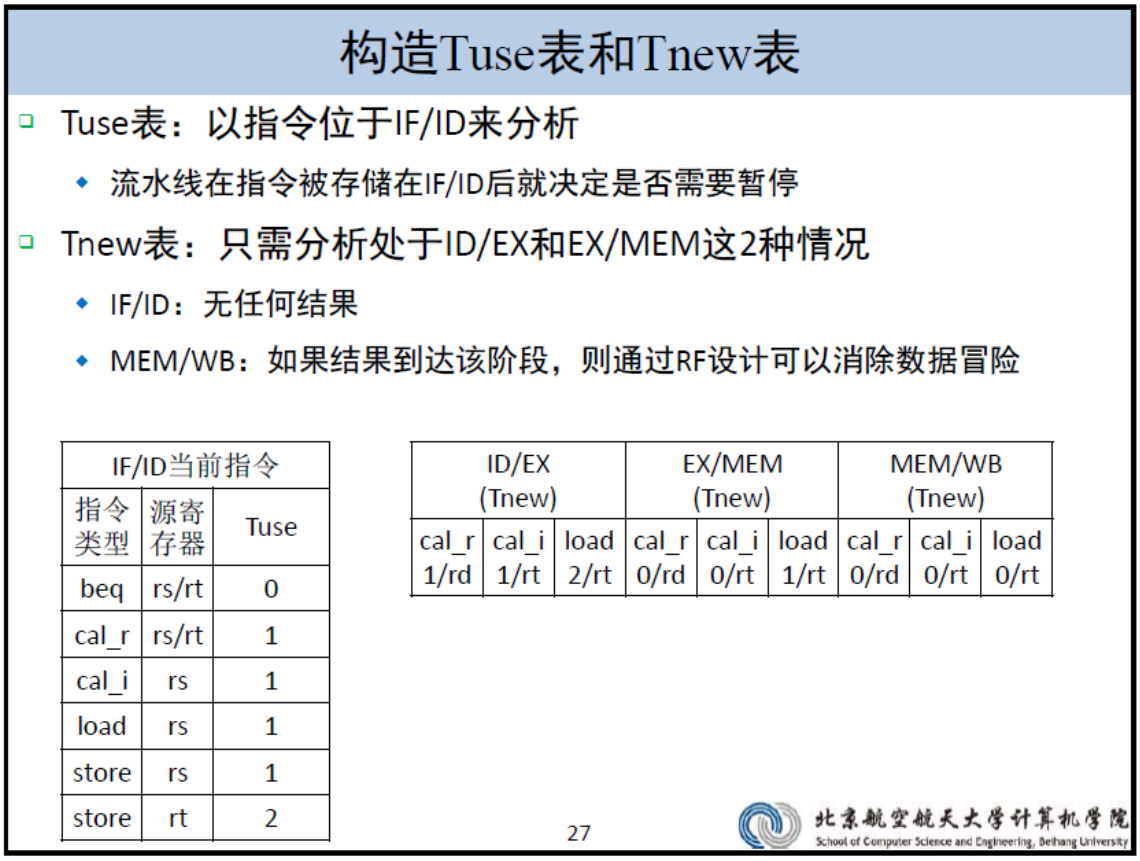


图 6：暂停分析表格的框架

如图所示，对于每一横行，我们写明，是哪一类指令，这一类指令读取的寄存器对应的是指令的哪个字段，同时这个数据读取的 Tuse 是多少。对于每一纵列，我们写明，是哪一个流水级（图中 ID/EX 表示指令处于 ID/EX 级流水线寄存器中，等效于处在 EX 级），每个流水级里面再写明是哪一类指令，这一类指令在这个流水级的 Tnew 是多少，它产生的数据存储的寄存器对应的是指令的哪个字段。

那么，对于每一横行每一纵列交错形成的一个小格，就对应了一个潜在的冲突情况，我们只需要比较这个小格对应的 Tuse,Tnew 谁大谁小，即可得出这种情况是否需要暂停的结论。

省略掉 Tnew=0 的纵列（此时一定不用暂停），我们可以构造出暂停情况的表格：

构造阻塞矩阵

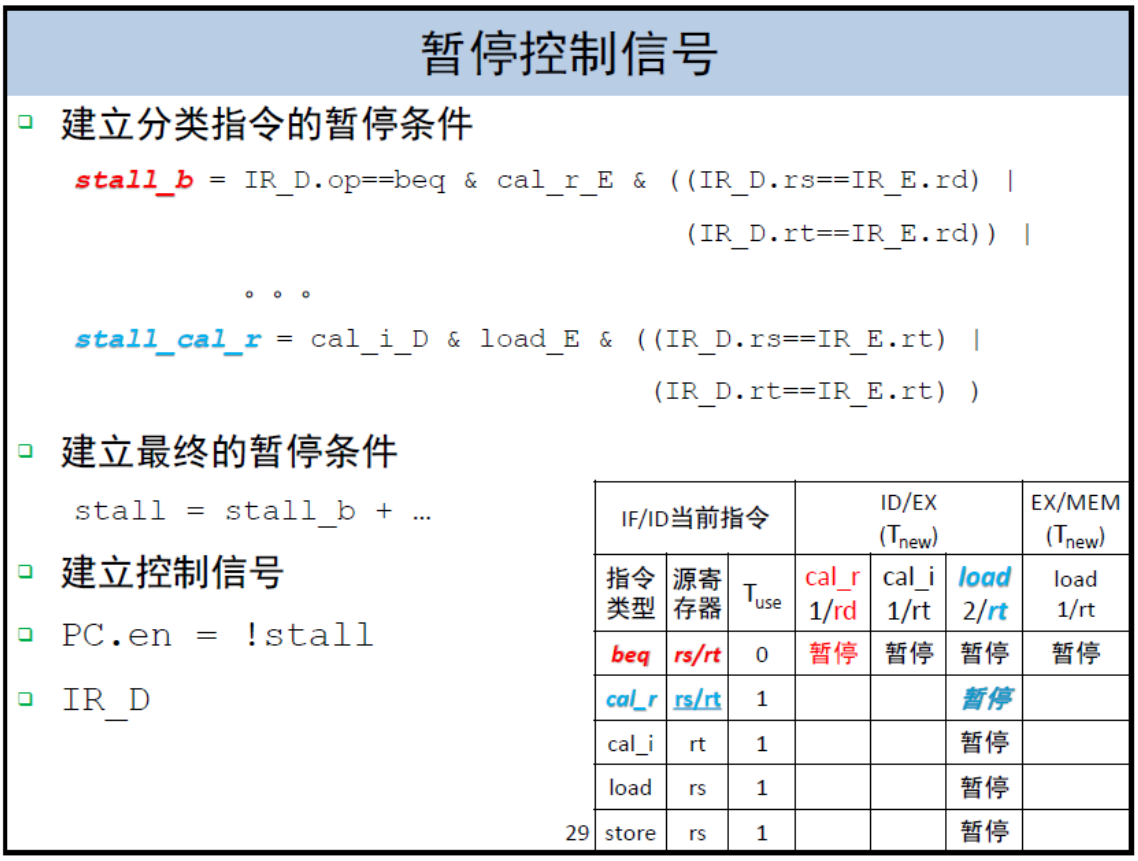
- 凡是$T_{new} > T_{use}$的指令序列，都需要阻塞
- 示例
  - **序列1 cal_r – beq**：由于cal_r需要1个cycle后才能得到结果，而beq现在就需要读取寄存器，因此只能暂停
  - **序列2 load – store**：store要读取的rs将在1个cycle后必须使用，而位于ID/EX的load必须经过2个cycle后才能读出DM的数据，因此只能暂停

IF/ID当前指令			ID/EX (T_{new})			EX/MEM (T_{new})
指令类型	源寄存器	T_{use}	cal_r 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
beq	rs/rt	0	暂停	暂停	暂停	暂停
cal_r	rs/rt	1			暂停	
cal_i	rt	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	

图 7：暂停表格实例，不包括 j 类和乘除法类

可以看到，通过这种方式，我们简单明了的分析出了什么情况需要暂停。当我们需要新加指令的时候，如果指令属于已经分析过的指令类，则此表格完全不用修正。如果指令属于新的指令类，则我们可以依照工程化的分析法：如果指令有输出，则加入新的纵列；如果指令有输入，则加入新的横行；然后完善 Tnew,Tuse 和目标寄存器信息；接下来即可非常简单的对这个表格做出补充，得出加入新指令之后的暂停表格。

那么对于暂停的处理，我们只需要计算分类指令的暂停条件，然后取这些暂停条件的或即可。



IF/ID当前指令			ID/EX (T_{new})			EX/MEM (T_{new})
指令类型	源寄存器	T_{use}	cal_r 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
beq	rs/rt	0	暂停	暂停	暂停	暂停
cal_r	rs/rt	1			暂停	
cal_i	rt	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	

图 8：暂停控制信号构建实例，stall_b 表示指令为 beq 这一行的整个判断情况

当判定需要暂停的时候，我们需要执行三个操作：

① 冻结 PC，不让 PC 的值改变

② 让 ID/EX 流水级寄存器清零，等于插入了一个 NOP 指令

③ 冻结 IF/ID 流水级寄存器，不让它的值改变。

如此，我们就完成了暂停机制的构建。当我们插入一个新的指令时，只需要按照表格的格式，分析 Tuse,Tnew，然后就可以轻松的构造它对应的暂停判断信号。

五、 转发机制的构建

处理完了暂停，接下来我们要处理的就是相对复杂的转发机制。

首先，让我们来梳理一下转发机制的具体原理。转发的原理，是后面的指令需要用到前面指令的运算结果，而后面指令在执行时，前面指令的运算结果还没有来得及写入寄存器堆。则此时，我们增加若干条数据通路，把前面指令的运算结果直接传递回来，达到不用暂停也能规

避风险的目的。

转发的情况和机制，相比于暂停要更加的复杂，很多同学都是在分析转发的时候分析漏了或者错了，从而导致 BUG 的产生。而通过工程化的方法，我们可以非常简便而清楚地构造出所有的转发情况，从而避免这些问题。

根据我们之前的数据需求——供给模型，当两条指令发生数据冒险，而且 Tnew<=Tuse，则我们可以通过把前面指令的结果转发到后面的指令来解决问题。因此，我们的首要任务就是通过分析每一种指令组合情况，得出每一种情况的转发结果。这一过程，我们可以通过构造转发分析表格来进行。

与暂停分析表格十分类似，我们依然用横行来表示需要使用寄存器的指令，只不过与暂停不同，暂停只需要考虑 IF/ID 级的指令；而转发则需要考虑每一个流水级的需求，如此一来整个表格会变得非常臃肿，我们可以通过一些策略来简化这个表格的构建过程。

第一步：分析转发情况，构造转发多选器

对于任何一个指令，引用数据本质都是引用寄存器堆的值，而通过对指令集的解析，我们知道引用寄存器的值，必然是引用指令的 rs 字段或者 rt 字段对应的寄存器。所以对于每个阶段，我们需求的数据其实只有两种，即需求 rs 寄存器，或者需求 rt 寄存器。而对于 MEM 阶段则不会再需求 rs 寄存器，所以对数据的需求一共只有五种情况：ID 阶段对 rs,rt 寄存器的需求，EX 阶段对 rs,rt 寄存器的需求，MEM 阶段对 rt 寄存器的需求。只不过，指令只对应其中的一个或两个需求而已。如 beq 指令需求 ID 阶段的 rs,rt 寄存器，addu 指令需求 EX 阶段的 rs,rt 寄存器。

接下来，每一个数据需求，它的正确数据，可能是之前从寄存器堆中读取来的值，也有可能是其他流水级尚未来得及写入寄存器堆的结果，所以我们需要对这五种数据需求分别建立一个转发多选器，用来控制这些数据是应该选择原始的寄存器读取结果作为数据，还是选择后面

流水级的计算结果作为数据。

图 9，转发表格的表头部分

对表格的解释：一共有五种数据需求，分别对应表格的五行。每一个数据需求都对应了若干类指令。当我们后续添加指令的时候，不会产生新的情况，只需要根据指令的输入数据情况把指令添加在涉及指令这部分框里即可。

而转发 MUX 的名字，其意义为：M(MUX 多选器)F(转发)RS(对寄存器 RS 的需求)D(位于 D 阶段)

相应的对每种情况的转发 MUX，设定一个专门的控制信号，当控制信号输入 0，也就是不转发的时候，多选器输出之前直接从寄存器堆读取的数据。

而与暂停表格的构建相同，用纵列表示后面的各个流水级，能够提供数据的各个指令，注意，当 Tnew<=Tuse，仅仅表示当前情况可以通过转发解决，当 Tnew=0 时，才说明结果已经得出，此时才能真正进行转发。所以，我们省掉纵列中，Tnew≠0 的项。

根据Tuse和Tnew构造每个转发MUX

- 用Tnew中剔除非0后的表项，来分析转发MUX的后续输入
  - 注意：并非有N个0项就有N个后续输入

ID/EX (Tnew)			EX/MEM (Tnew)			MEM/WB (Tnew)		
cal_r 1/rd	cal_i 1/rt	load 2/rt	cal_r 0/rd	cal_i 0/rt	load 1/rt	cal_r 0/rd	cal_i 0/rt	load 0/rt

流水级	源寄存器	涉及指令				EX/MEM (Tnew)		MEM/WB (Tnew)		
						cal_r 0/rd	cal_i 0/rt	cal_r 0/rd	cal_i 0/rt	load 0/rt
IF/ID	rs	beq	MFRSD	ForwardRSD	RF.RD1					
	rt	beq	MFRTD	ForwardRTD	RF.RD2					
ID/EX	rs	cal_r, cal_i, ld, st	MFRSE	ForwardRSE	RS@E					
	rt	cal_r	MFRTE	ForwardRTE	RT@E					
EX/MEM	rt	store	MFRTM	ForwardRTM	RT@M					
			转发MUX	控制信号	输入0					

图 10，转发表格纵列的构造

此时，我们就基本构造完了这个转发表格，下一步，就是分析表格每一个空所对应横行纵列的情况，并且在格子中填入这种情况下，这个 MUX 应该选择哪个数据作为最终的读取数据。

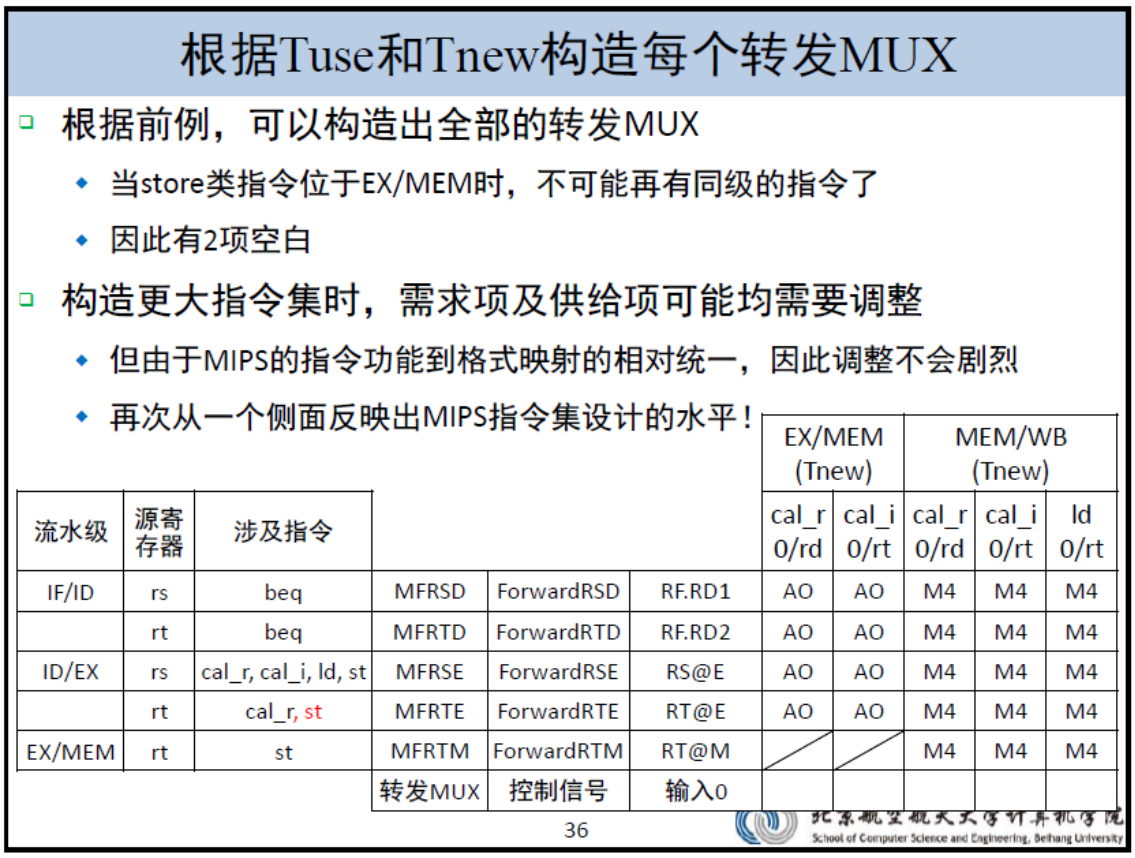
根据Tuse和Tnew构造每个转发MUX

- 构造每个转发MUX的后续输入
- 示例：MFRSD
  - EX/MEM：cal_r和cal_i指令都是计算类，结果必然由ALU产生，因此均填入AO。即代表MFRSD的输入来自EX/MEM中的AO寄存器
    - AO：代表ALUOut
  - MEM/WB：由于这是最后一级，即所有指令的结果都通过M4(MUX)回写，因此均填入M4。

流水级	源寄存器	涉及指令				EX/MEM (Tnew)		MEM/WB (Tnew)		
						cal_r 0/rd	cal_i 0/rt	cal_r 0/rd	cal_i 0/rt	load 0/rt
IF/ID	rs	beq	MFRSD	ForwardRSD	RF.RD1	AO	AO	M4	M4	M4
	rt	beq	MFRTD	ForwardRTD	RF.RD2					
ID/EX	rs	cal_r, cal_i, ld, st	MFRSE	ForwardRSE	RS@E					
	rt	cal_r	MFRTE	ForwardRTE	RT@E					
EX/MEM	rt	store	MFRTM	ForwardRTM	RT@M					
			转发MUX	控制信号	输入0					

图 11，转发表格的填写

由此，我们可以按照规律分析每一格的情况，并且把表格填满。



根据Tuse和Tnew构造每个转发MUX

- 根据前例，可以构造出全部的转发MUX
  - 当store类指令位于EX/MEM时，不可能再有同级的指令了
  - 因此有2项空白
- 构造更大指令集时，需求项及供给项可能均需要调整
  - 但由于MIPS的指令功能到格式映射的相对统一，因此调整不会剧烈
  - 再次从一个侧面反映出MIPS指令集设计的水平！

流水级	源寄存器	涉及指令				EX/MEM (Tnew)		MEM/WB (Tnew)		
						cal_r 0/rd	cal_i 0/rt	cal_r 0/rd	cal_i 0/rt	ld 0/rt
IF/ID	rs	beq	MFRSD	ForwardRSD	RF.RD1	AO	AO	M4	M4	M4
	rt	beq	MFRTD	ForwardRTD	RF.RD2	AO	AO	M4	M4	M4
ID/EX	rs	cal_r, cal_i, ld, st	MFRSE	ForwardRSE	RS@E	AO	AO	M4	M4	M4
	rt	cal_r, st	MFRTE	ForwardRTE	RT@E	AO	AO	M4	M4	M4
EX/MEM	rt	st	MFRTM	ForwardRTM	RT@M			M4	M4	M4
			转发MUX	控制信号	输入0					

图 12，初步完成的转发表格

可以看到，虽然表格较大，但是构造起来并不难。此时如果我们需要添加指令，则首先考虑：它需不需要使用数据？如果需要，根据使用寄存器的情况，在横行里添加涉及指令。

其次，它是否提供数据结果？如果提供，则需要从它能计算出结果的那一级开始，在每一级都增加一个新的纵列（如 jal，jal 指令，从 ID 级就产生结果，则表格需要增加一个大列），同时在对应的新空格里，填写这种情况下的转发数据来源。

流水级	源寄存器	涉及指令	转发MUX	控制信号	默认输入	D/E(Tnew)		EX/MEM(Tnew)				MEM/WB(Tnew)			
						jal0/$31	jalr0/rd	cal_r0/rd	cal_i0/rt	jal0/$31	jalr0/rd	cal_r0/rd	cal_i0/rt	load0/rt	jal0/$31
F/D	rs	beq, jr, jalr	MFRSD	ForwardRSD	RF.RD1	DE_PC8	DE_PC8	ALUout	ALUout	EM_PC8	EM_PC8	RF内部转发			
	rt	beq	MFRTD	ForwardRTD	RF.RD2	DE_PC8	DE_PC8	ALUout	ALUout	EM_PC8	EM_PC8				
D/E	rs	cal_r, cal_i, ld, st, movto, mtdv	MFRSE	ForwardRSE	RS_E			ALUout	ALUout	EM_PC8	EM_PC8	MUX_GPRwd	MUX_GPRwd	MUX_GPRwd	MUX_GPRwd
	rt	cal_r, st, mtdv	MFRTE	ForwardRTE	RT_E			ALUout	ALUout	EM_PC8	EM_PC8	MUX_GPRwd	MUX_GPRwd	MUX_GPRwd	MUX_GPRwd
E/M	rt	st	MFRTM	ForwardRTM	RT_M			none	none	none		MUX_GPRwd	MUX_GPRwd	MUX_GPRwd	MUX_GPRwd
						sel=3		sel=1		sel=4				sel=2	

图 13，添加了新指令类别的转发表格示例

当我们添加新指令类别时，只需要依照上面的操作方式，进行表格填写即可，非常方便，

最多只有少量的微调，不会有较大的改动。

在完成了转发多选器的构造之后，我们就需要把转发多选器加入数据通路中。我们只需要牢记一点：数据通路表格中任何引用寄存器数据的地方，都要改成引用对应的转发多选器输出数据！

数据通路增加转发MUX

- 遍历数据通路的功能部件，找到所有出现rs和rt的需求点
- 注意ALU.B和RT@M，这两个rt需求是相同的！
  - 这意味着它们应该来自同一个转发MUX

部件	输入	输入来源			MUX	控制
PC						
ADD4		PC				
IM		PC				
PC		ADD4	NPC	RF.RD1	M1	PCSel
IR@D		IM				
PC4@D		ADD4				
RF	A1	IR@D[rs]				
	A2	IR@D[rt]				
EXT		IR@D[i16]				
CMP	D1	RF.RD1				
	D2	RF.RD2				
NPC	PC4	PC4@D				
	I26	IR@D[i26]				
IR@E		IR@D				
PC4@E		PC4@D				
RS@E		RF.RD1				
RT@E		RF.RD2				
EXT@E		EXT				
ALU	A	RS@E				
	B	EXT@E	RT@E		M2	BSel
IR@M		IR@E				
PC4@M		PC4@E				
AO@M		ALU				
RT@M		RT@E				
DM	A	AO@M				
	WD	RT@M				
IR@W		IR@M				
PC4@W		PC4@M				
AO@W		AO@M				
DR@W		DM				
RF	A3	IR@W[rt]	IR@W[rd]	0x1F	M3	WRSel
	WD	DR@W	AO@W	PC4@W	M4	WDSel

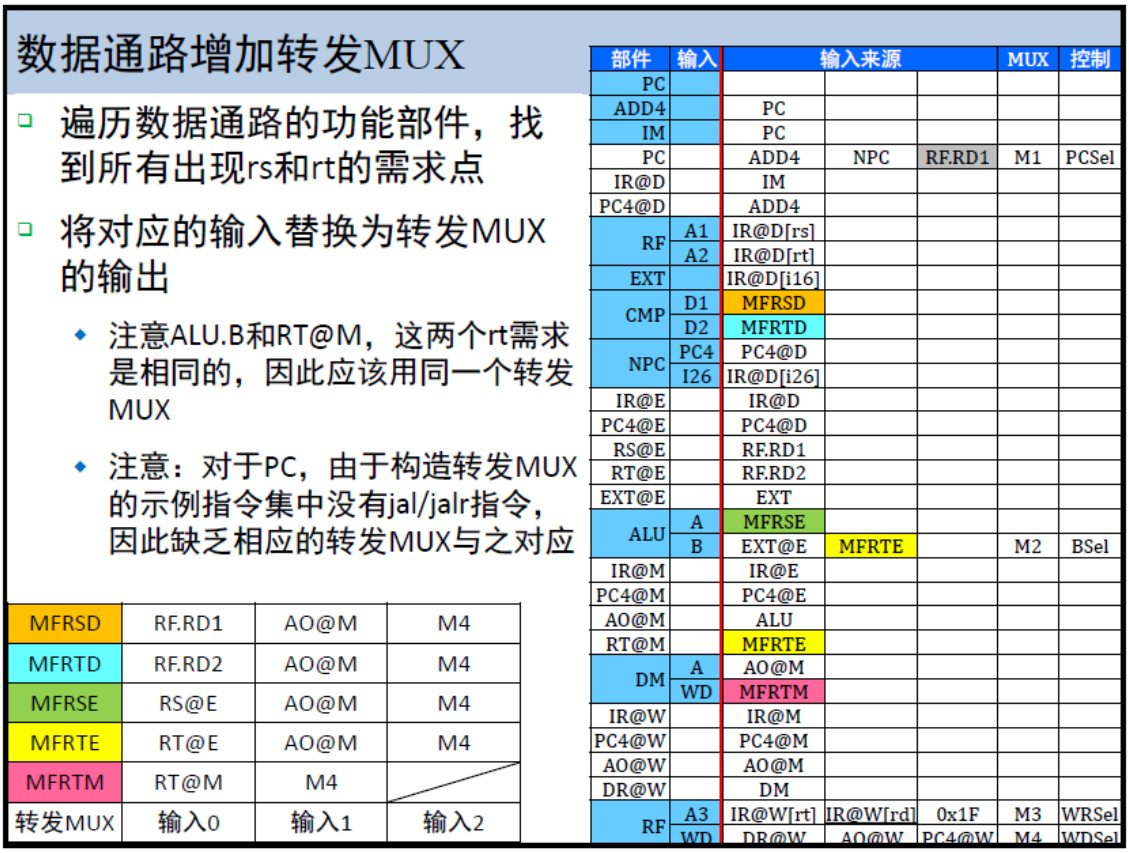


图 14，在数据通路中加入转发 MUX

第二步，根据完成的转发表格，构造每个转发多选器的控制信号

控制信号的构造非常简单！依次分析表格的每一行，从左到右的每一格，如果这一格的条件成立（前后指令符合表格，且指令的对应目标寄存器相同），则设定这一个多选 MUX 的多选信号为当前格你所设定的控制信号值。同时注意，如果当前流水级的指令同时与两个流水级的指令发生冲突，且都需要转发来解决，则我们选择转发与当前指令**更靠近**的指令的计算结果！（想一想，为什么？）

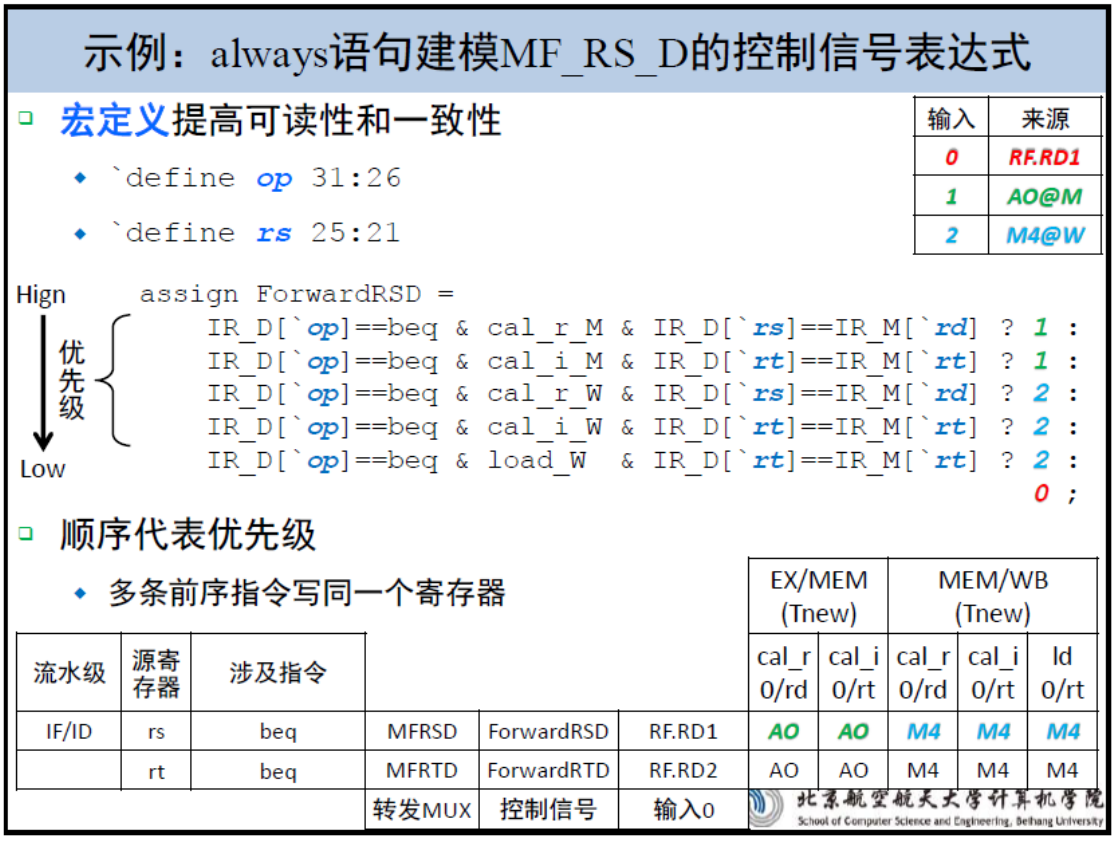


图 15，转发控制信号构建示例

六、 工程化调试

之前讲解了我们如何利用工程化方法来帮助我们设计流水线 CPU，但是设计仅仅是我们工作的一部分，更重要的是在我们的设计出现问题的时候如何去调试它。而调试绝不是等着 ISE 的输出窗口发呆，也不是碰运气的瞎找，调试也是可以有系统化的策略的。

1、问题再现：我们发现一个 BUG，肯定是我们的 ISE 模拟结果和 MARS 的汇编程序运行结果有了出入。那么此时，需要我们耐心的一步步运行（一开始也可以一次跑 10 步或者更多来快速粗略找到问题出现的位置），直到我们发现某个寄存器或内存的值与 MARS 的运行结果不同。也就是再现这个 BUG 的“案发现场”

2、错误追溯与定位：当我们发现一个写入数据有误，无非就是几种可能：要么这个数据计算无误，是我们连接错了线路；要么是这个数据有误。此时我们需要记录当前的时刻，然后重新开始模拟，运行到上一个时钟周期处，检查上一时刻的数据是否有误，如果无误，则说明是连线问题；如果有误，说明可能是计算错误，或者控制信号错误，那么进一步检查上一时刻

这些对应控制信号和数据。直到某一时刻，它之前的状态都准确无误，只在这个时刻，指令执行的结果与预想出了问题。在这个基础上，我们就可以检查各个部分的数据，看是哪个部件对哪个数据的计算出现了错误，从而修正错误。

以上两条只是总纲性质的原则，说起来比较空泛，各位同学在调试的时候一定要切记，避免看着 BUG 陷入崩溃或者烦躁。一定要有条理 ,首先找到是什么时候开始最先表现出了问题，然后一步一步倒退追踪错误的信号 ,直到错误信号第一次出现 ,这样就能比较高效的排查 BUG。笔者在 P6 的检测中 ,因为键盘误操作原本设计好的转发信号被弄出了 BUG。最后用时两小时，对一个 300 行的测试程序，追踪运行了近 400 个时钟周期，成功定位了这个微小的错误并修正，运用的就是这个“问题再现——错误追溯与定位”的方法。

七、一些小 TIPS 和经验

1、千万要重视 ISE 的 warning！大量的误连线和漏连线都可以通过 warning 轻易的找出来！

2、想好了再动手！一旦写到一半发现思路出了问题需要修正，哪怕只是极小的修正，都很容易引发崩盘！

3、多想多问多讨论！mooc 上的讨论区有很多前辈们留下的宝贵财富！

4、除非你对自己有绝对的自信，否则千万不要拖延到 deadline 再干活，这是作死。

5、临场考试心态很重要！发现 BUG 了不要慌，慢慢追踪定位，这看似慢的方法其实是最有效率的。

6、如果你认真按照本文的工程化方法设计，并且通过 ISE 的 warning 解决了连线问题，那么你绝大多数的问题都会出在译码的 ctrl 模块和控制冒险的 hazard 模块，可能是误操作，也可能是加指令的时候对应的控制信号没有全部加上，一定要多加小心。

7、绝对不要抄袭！

写在最后：

感谢高小鹏老师留下的优秀课件！

感谢计组助教学长学姐们的辛劳与答疑解惑！

感谢在学习过程中与我讨论，给我帮助的同学们！

以上，谢谢你们！

同时，祝 15 级的各位，能够顺利度过计组这大学生涯中第一个重课！

你们一定行！

1406 级实验班 黄宏鸣