花开一季 叶落一地

一、流水线CPU概述

1、前言

单周期CPU上所有指令都在一个时钟周期内完成，所以其时钟周期一般较长（能够完成最慢的指令），吞吐量不高。出于增大吞吐量的考虑，引入了流水线CPU，同一时刻有多条指令在其上运行，因此理论上五段流水CPU的吞吐量是单周期的五倍。

在单周期CPU的实验中，我们学习了CPU的主要搭建方法，并且实际动手搭建了一个简单的CPU，这些经验和方法在流水线CPU的搭建中同样适用。

讲述五段式流水线CPU的搭建方法之前，对文中用词做出一些说明：文中出现的“模块”一般是指CPU的和指令执行五个阶段一一对应的顶层模块（不包含控制器和冲突处理）；文中出现的“阶段”可以理解是CPU的顶层模块，也可以理解成是指令的执行阶段，因为二者是一一对应的。

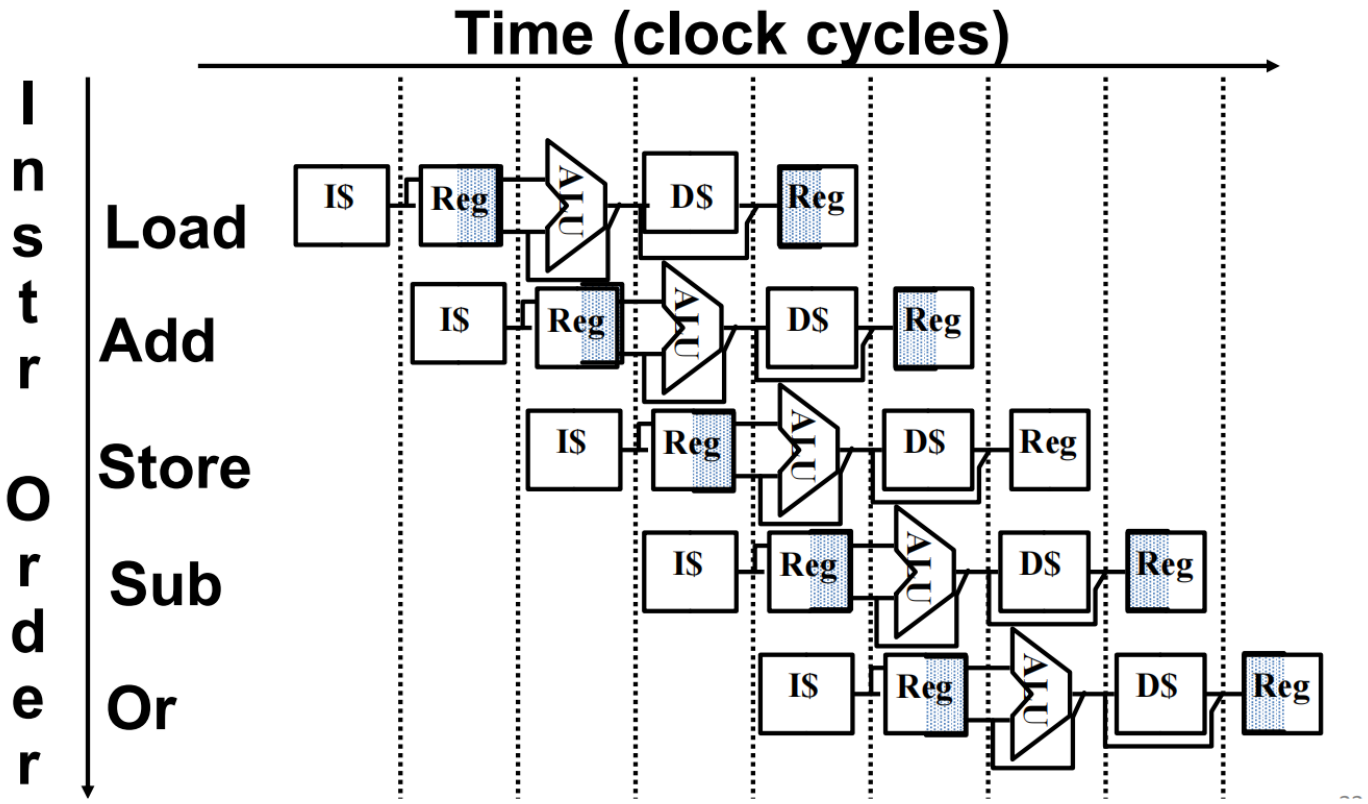
2、基本原理

流水线CPU提高吞吐量的关键就在于同一时刻有多条指令在执行。相比单周期CPU，流水线CPU的硬件利用率更高，大部分时间，每个顶层模块都在连续工作。

五级流水线CPU按照指令运行的五个阶段对应地分成了五个顶层模块：IF（instruction fetch）、DEC（decode）、EXE（execute）、ME（memory）、WB（write back），每个模块执行一条指令的一个阶段。

指令运行的**“五个阶段”**在此更加明显的体现：一条指令在一个时钟周期只完成其一个阶段，一条指令需要经过多个时钟周期完成。因为一个时钟周期内完成一条指令的一个阶段，因此**流水线CPU的时钟周期就是耗时最长的模块/阶段所需要的时间**。

如下图所示：第一个时钟周期（T），IF模块运行load的取指令阶段（简称I阶段）；第二个T，IF模块运行 add指令的取指阶段，DEC（译码）模块继续运行load的译码阶段（简称D阶段）；第三个T，IF模块运行store的取指阶段，DEC模块继续运行add的译码阶段，EXE（执行）模块继续运行load的执行阶段（简称E阶段）.....



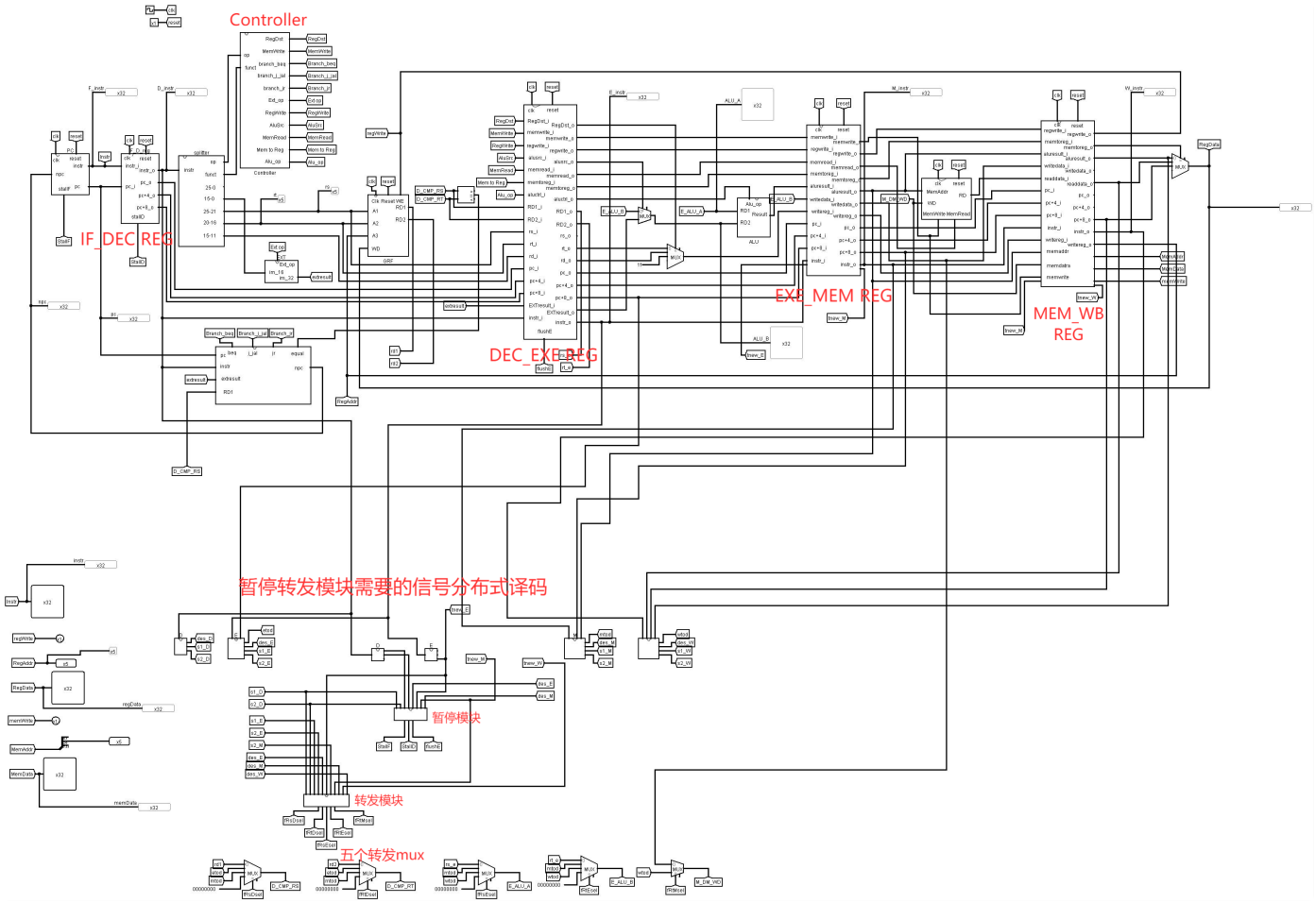
3、流水线寄存器

对流水线CPU的前四个顶层模块中任一模块来看：**当前时钟周期**下某模块执行某指令计算或取出的结果，需要在**下一周期**供下一模块利用并执行该指令的下一阶段，但按照单周期CPU的设计是不可能的，因为下一周期时，该顶层模块的数据已经被下一条指令在该顶层模块的执行产生数据所替代。

为了某模块执行指令产生的数据能够正确进入下一模块在下一时钟周期继续执行，在该模块后需要添加**流水线寄存器**以保存该模块的运行数据并提供给下一模块使用。

写回模块后面不需要流水线寄存器的原因是：其一，它运行的是指令执行的最后一个阶段，它的运行数据并不需要给对应指令的下一阶段使用；其二，写回部分是寄存器堆，本身可以看作“第五个流水线寄存器”。

在五段式CPU中，我们需要额外添加4个流水线寄存器：IF/DEC、DEC/EXE、EXE/MEM、MEM/WB。



注意：上图仅作解释流水线寄存器之用，可以借鉴顶层模块的排布，切忌完全照搬，因为图不包含模块细节。

4、冲突和解决

- **硬件冲突**

  硬件冲突是说同一时刻需要对同一互斥访问硬件（同一时刻只允许一次读或写）进行访问，举例来说，译码段需要从存储器中取出指令，访存段需要对存储器写入数据，这两个操作同时进行就会带来冲突。

  在我们的设计中，IM和DM是独立的两个存储器，因此**不必考虑**。

- **控制冲突**

  控制冲突是分支指令和跳转指令带来的冲突。分支指令的分支信号在下一周期到来前未能及时计算出，导致下一条指令的地址不能及时决定；跳转指令的跳转地址计算也有类似的问题。

  例如指令序列：

```
    beq $1, $2, offset
    addu $1, $2, $4
    nop
offset:
    subu $1, $2, $4
    nop
```

上述指令序列的问题在于：如果像单周期CPU一样由ALU计算出zero信号，即在EXE阶段计算出结果，那么IF模块取出的地址将是可能不正确的，导致执行序列有误。假设EXE阶段计算出需要分支，但由于这个结果来得太晚，IF模块可能顺序取出指令码，导致beq后两条指令开始执行，出现错误。

可见分支地址和分支信号的计算越早越好，在我们的实验中将分支地址和分支信号的计算放在**译码阶段**，因为需要通过译码才能知道是否是跳转/分支指令。

解决这样的冲突主要是通过**假设不跳转或者延迟槽**。假设不跳转是说先假设不跳转和分支，正常执行下一条指令，当计算出要跳转时清除已执行的结果（通过对流水线寄存器添加同步复位功能）。延迟槽是说跳转指令后面加空指令nop或**不会引起跳转的指令**，即等待跳转地址计算出来再决定是继续下一条指令还是跳转，无需清除结果。

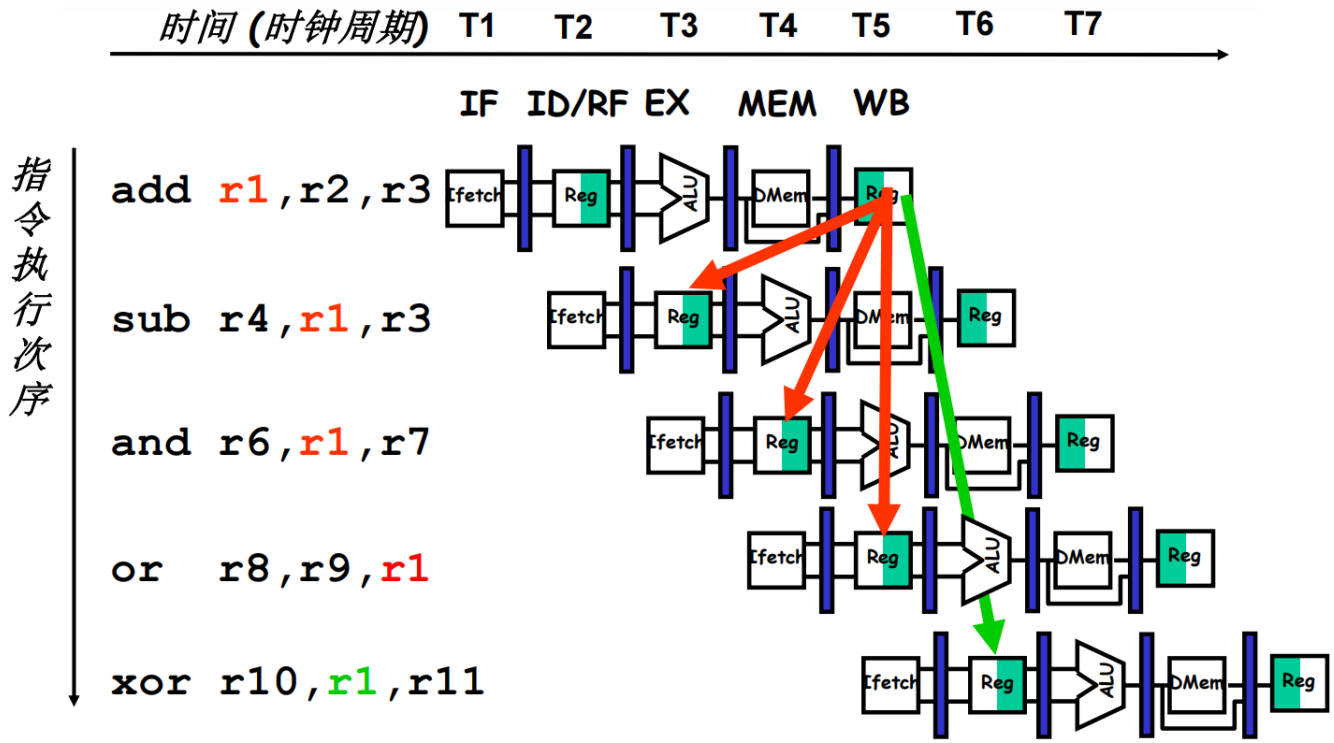
实验中，我们采取**延迟槽**解决，即编写程序时在每个可能跳转的指令后加延迟槽指令（最简单的就是nop）。

- **数据冲突**

数据冲突是关于数据“新鲜性”的冲突。指令的执行离不开寄存器，有的指令会利用寄存器数据，有的指令会写回寄存器，有的指令两者皆有，当前序执行的指令的操作寄存器和后序执行的指令的操作寄存器有重叠时，就存在**数据关联**。当前序指令的关联数据还未写入目的寄存器，后序指令就要用到该关联数据时，就会产生错误（冲突）。

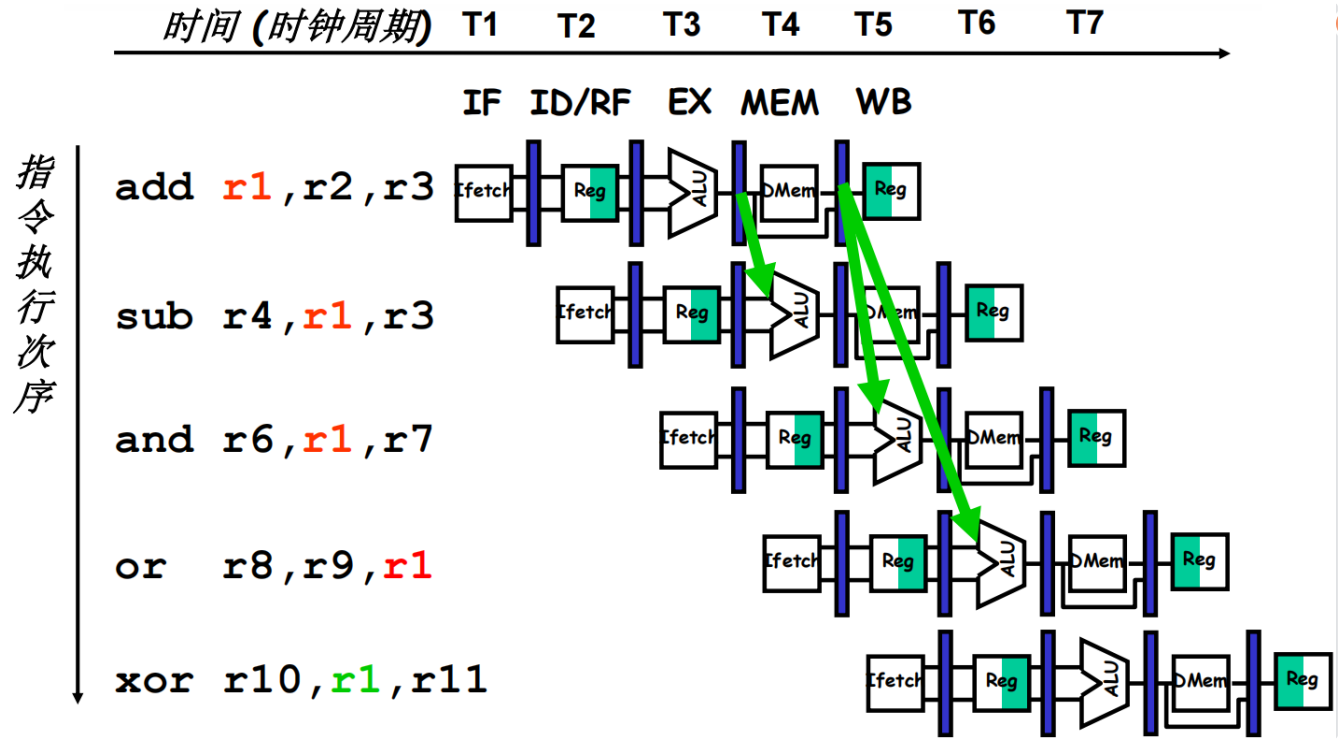
```
addu $1, $2, $3
subu $2, $4, $5 #存在数据关联

addu $1, $2, $3
subu $1, $4, $5 #存在数据关联

addu $1, $2, $3
subu $4, $1, $5 #存在数据冲突
```

下图红色箭头存在数据冲突，绿色箭头则不存在。



解决这种冲突需要**暂停或者转发**：当前序指令的将要写回寄存器的数据（计算）赶不及后序指令使用其数据时就需要**暂停后序指令**，直到前序指令的“新”数据准备好；当前序指令的“新”数据能赶上后序指令的使用，当前序指令的“新”数据准备好后，**立刻转发给后序指令**当前所处阶段的数据源头，以达到更新数据的效果。

下图是通过转发解决数据冲突。



二、流水线CPU搭建的方法

0、基本步骤

流水线CPU数据通路搭建的方法和单周期大致相同：列出指令集，找硬件，构建数据通路，构造控制器，**设计冲突处理单元**。除了最后一步，前面步骤和单周期CPU类似（需要额外加入四个流水线寄存器），在此不加赘述，着重讨论如何处理冲突。

注意：下面介绍的冲突处理方法只是一种思路。

1、控制冲突

实验中采用延迟槽解决控制冲突且将分支指令计算放在译码阶段，也就是当DEC模块运行分支指令或跳转指令（**设地址为pc**）时，IF模块执行延迟槽指令（一般是nop，但实验测试不保证只有nop），而该周期内DEC模块决定下一条指令地址。这里需要注意如果顺序执行指令，一定要注意是执行**延迟槽指令的下一条指令（pc+8），这里的pc是分支/跳转指令在F段时的地址**。

2、数据冲突

处理数据冲突依靠暂停和转发，介绍如何操作前，先说明几个概念。

2.0 几个概念

- **几个定义**

  对任意一条通用指令（自创定义：只涉及通用寄存器的指令），有**源头寄存器和目的寄存器**：当一条通用指令i需要寄存器s1、寄存器s2的数据时，s1和s2就是i的源头寄存器（不需要用到其数据时，s1和s2为0）；当一条通用指令i需要写入寄存器des时，des就是i的目的寄存器（不需要写寄存器时，des为0）。

  对于指令的源寄存器s有时间tuse ***D（/E/M）***，*意思是在从**D（/E/M）** 段开始，过几个时钟周期需要使用s里的数据。对于指令的目的寄存器des有数据tnew* ***D(/E/M)**，意思是从**D（/E/M）*** 段开始，过几个时钟周期将要写入des的数据产生。

上述定义可结合第三章第6节的表格一起理解。

- **一些技巧**

  tnew每经过一个阶段会减一，减到0之后，后面阶段的tnew都是0。

  tuse也有一样的规律。

2.1 暂停

- 何时暂停

  暂停是因为将要使用的数据来不及转发过来，如果不暂停就会使用错误的数据进行计算。由于分支指令最早在译码阶段就需要使用寄存器数据计算，暂停阶段放在译码阶段，即DEC模块。

  结合开始介绍的概念，每条指令i在译码段时（DEC模块运行指令i），要和它前面运行的指令j（前面模块运行的指令j，一般是前面的所有模块）对照判断，看是否需要暂停：当**i**的源寄存器和**j**的目的寄存器**相同**时（设为寄存器k），i和j存在**数据关联**，这时如果**i的tuse小于j的tnew**，代表k中数据还没被j准备好（甚至不能转发过来），这时需要**暂停D段指令**i，否则i会使用k中旧数据里错误运行。

- 如何暂停

  暂停就是将当前指令和其后面的指令全部“冻结”，而其前面的指令正常运行，以等待数据准备好。D段指令冻结也就是IF/DEC寄存器仍然保持当前状态，而此时F段中的pc已经取出下一条指令地址，为防止丢失，pc也保持当前状态。再者，为了D段需要暂停的指令无法向后进行，需要在下一上升沿在E段插入气泡。

  暂停需要**将DEC/EXE流水线寄存器同步复位（插入气泡，使当前指令无法向下传递），且IF/DEC流水线寄存器和PC锁定（维持当前状态，即禁止使能）**，直到暂停的条件被打破。

2.2 转发

- 何时转发

  转发是当前序指令和后序指令存在数据冲突时，在关联数据准备好时**立刻转发**到后序指令当前所在模块。

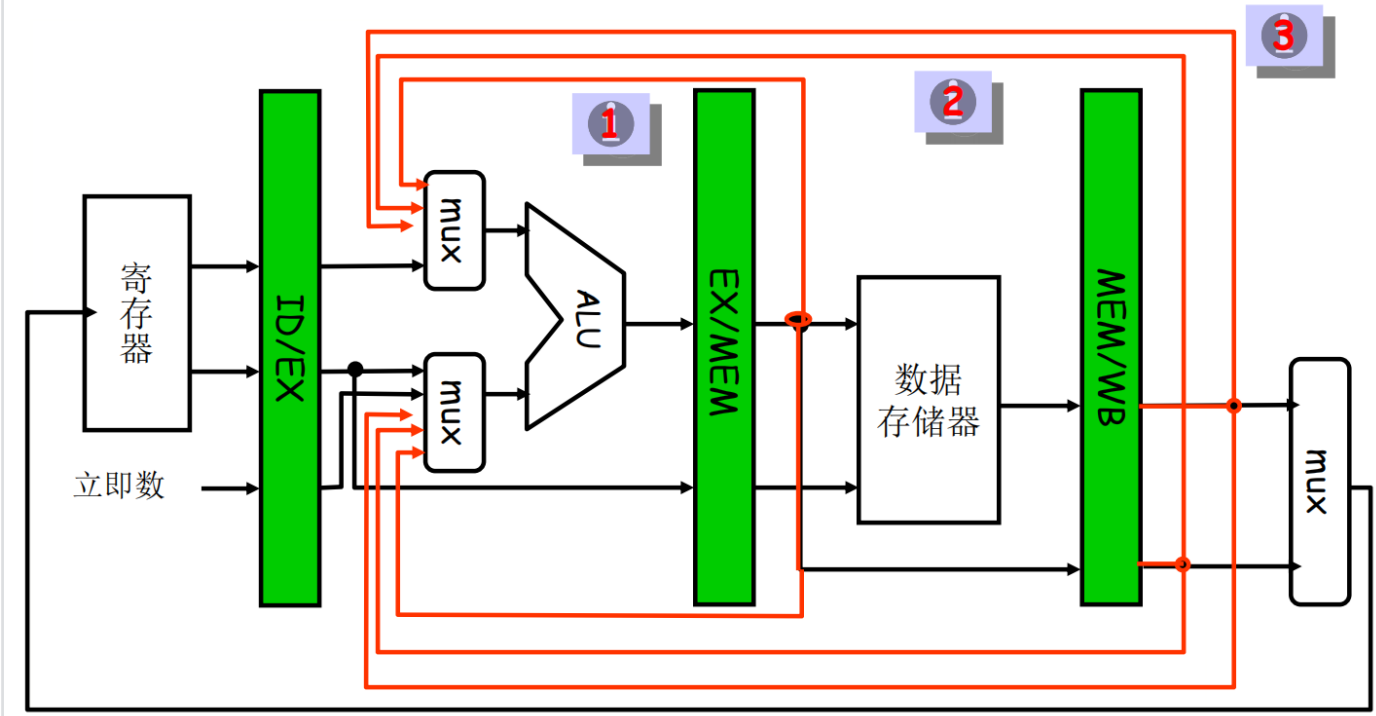
  前序指令的**tnew为0**时代表新数据已经准备好。

- 转发去哪

  转发的目的是后序指令的所在模块（阶段）的**数据源头**，以保证后序指令之后执行都是依据最新数据。这里需要在源头处加多选器，希望达到：当不需要转发时自然使用原始数据，需要转发时选择正确的转发源头。

  何为数据源头，以DEC模块举例：DEC模块中的寄存器堆的两个输出就是该模块内两寄存器数据的源头。

  下图为转发到EXE模块：

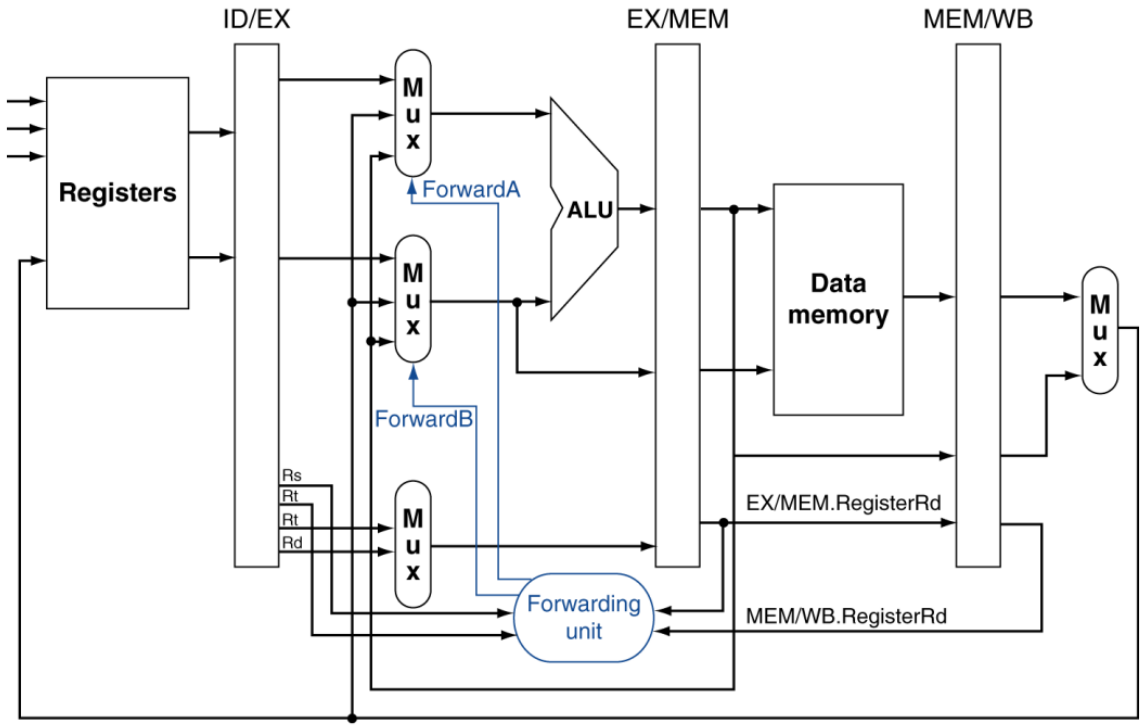


- 转发什么

  转发的是前序指令已经准备好的数据：每个模块内，tnew为0的一类指令的“新”数据对应某内部模块的输出。我们希望达到：对于每个模块，依据其当前运行指令（类别），就可以给出**可能转发的数据**。

  对于有多个转发来源时，应当选择最接近转发目的所在阶段的源头数据。

```
1 ......
2 addu $1, $2, $3
3 subu $1, $4, $5
4 ori $6, $1, $7 #应转发第二条指令的计算结果
5 ......
```



特别的对于**写回模块到译码模块（W到D）的转发**，由于写回阶段写入和译码阶段读出都是对于寄存器堆，我们可以采用**内部转发**：通过寄存器堆的结构使得，在同一个时钟上升沿，若有数据写入**寄存器r**，也有从**寄存器r**读出数据的请求，把将写入数据直接读出。这样一来，写回模块转发到译码模块的数据可以不通过外部转发，**当然外部转发也可**。

内部转发逻辑 寄存器堆的RD1输出：是0当且仅当A1输入为0； 是**将要写入**寄存器A3的数据当且仅当将要写入的寄存器编号A3和读取寄存器编号A1相等； 是寄存器A1的数据当且仅当将要写入的寄存器编号A3和读取寄存器编号A1不相等。 寄存器堆的RD2输出：是0当且仅当A2输入为0； 是**将要写入**寄存器A3的数据当且仅当将要写入的寄存器编号A3和读取寄存器编号A2相等； 是寄存器A2的数据当且仅当将要写入的寄存器编号A3和读取寄存器编号A2不相等。

- 转发“到底”

  转发是将前一条指令A的“新”寄存器数据转发到后面指令B，我们希望达到：该寄存器的数据在被更新前，可以一直跟随B在CPU上的运行，以达到“该数据从取出开始就正确的错觉”。这样做的原因是为了当该数据在**后面阶段可能被再次需要**时，依然是“最新的”。

三、流水线CPU构建的例子

以下只是一个实践的样例，仅仅为了说明方法之用。

1、分类指令集

列出支持指令并将其按照执行的硬件次序分类。

从增量开发的角度，建议同学们以**指令类**为对象来设计cpu的控制逻辑和冲突逻辑，因为同一类指令涉及到的**功能硬件相同**。

str	ld	cal_r	cal_i	lui	b_type	j	jal	nop
sw	lw	addu	ori		beq			
		subu						

2、分析功能硬件和流水线寄存器

2.1 流水线寄存器

像实验三说的方法一样，通过遍历指令来决定每个阶段（大模块）需要从上一阶段得到的信号，依次设计流水线寄存器。
下图为流水线寄存器需要传递的数据，D代表IF/DEC，E代表DEC/EXE，M代表EXE/MEM，W代表MEM/WB。
当然，这只是暂时需要传递的数据，之后添加指令/处理冲突还需要**添加传递的信号**。

D	E	M	W
IR	IR	IR	IR
			RD
		AO	AO
	RD1		
	EXTD	EXTD	EXTD
	RD2	RD2	
PC4			
PC8	PC8	PC8	PC8
PC	PC	PC	PC

2.2 功能硬件

依据指令集给出所需硬件，期间注意**模块化**的设计。

阶段	module	input	output	功能描述
IF	MFPC	PCSEL	D	选择下一个pc值
		PC4		
		NEXTPC		
	PC	D	Q	
	ADD4	PC	PC4	
	ADD8	PC	PC8	
	IM	IA	IR	指令存储器
DE	RF	A1	RD1	从寄存器堆读出寄存器数据
		A2	RD2	
	EXT	I16	EXTD	选择输出SIMM,LIMM（供lui）,UIMM
		EXTOP		
	CMP	D1	RES	输出比较结果
		D2		
	NPC	PC4	NEXTPC	选择输出BPC,JPC
		I26		
		NPCOP		
EX	MFB	BSEL	B	选择alu的b输入
		EXT_E		
		RT_E		
	ALU	A	AO	执行不同操作
		B		
		ALUOP		
ME	DM	DA[11:2]	RD	支持写入字节、半字、字和读出字
		WD		
		WM		
WB	MFA3	A3SEL	A3	选择a3
		IR_W[RT]		
		IR_W[RD]		
		31		
	MFWD3	WD3SEL	WD3	选择wd3
		AO_W		
		EXT_W		
		PC8_W		
	RF	A3		
		WD3		
		WR		

上图中有些项的样式是“A/B”，意思是该处原信号为A，由于处理数据冲突而更改为转发到某模块（IF、DEC等模块）的新信号B。例如：MFPC模块的输入之一写为“RD1/FRSD”，意思是该输入原本为RD1，即rs号寄存器的数据，但由于转发，现在输入是FRSD，即DEC模块内**选择转发数据和原数据的多选器**的输出。

上表可以看出，有的模块的输入端口名，就是其他模块的输出端口名，这样命名是为了表达这两个端口相连接的意思，便于连线。

3、构建基本数据通路

依据指令和上面的硬件，构建数据通路。

模块	INPUT	NOP	LD	STR	CAL_R	CAL_I	LUI	J	JAL	B_TYPE
PC	D									
ADD4	PC	Q	Q	Q	Q	Q	Q	Q	Q	Q
ADD8	PC								Q	
IM	IA	Q	Q	Q	Q	Q	Q	Q	Q	Q
PC	D	PC4	PC4	PC4	PC4	PC4	PC4			PC4
REG_D	IR_D	IR	IR	IR	IR	IR	IR	IR	IR	IR
	PC4_D							PC4	PC4	PC4
	PC8_D								PC8	
RF	A1		IR_D[RS]	IR_D[RS]	IR_D[RS]	IR_D[RS]				IR_D[RS]
	A2				IR_D[RT]					IR_D[RT]
EXT	I16		IR_D[I16]	IR_D[I16]		IR_D[I16]	IR_D[I16]			
CMP	D1									RD1
	D2									RD2
NPC	PC4							PC4_D	PC4_D	PC4_D
	I26							IR_D[I26]	IR_D[I26]	IR_D[I16]
PC	D							NEXTPC	NEXTPC	NEXTPC/ PC4
REG_E	IR_E	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D
	RS_E		RD1	RD1	RD1	RD1				
	EXT_E		EXTD	EXTD		EXTD	EXTD			
	RT_E			RD2	RD2					
	PC8_E								PC8_D	
ALU	A		RS_E	RS_E	RS_E	RS_E				
	B		EXT_E	EXT_E	RT_E	EXT_E				
	SHAMT									
REG_M	IR_M	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E
	AO_M		AO	AO	AO	AO				
	RT_M			RT_E						
	PC8_M								PC8_E	
	EXT_M						EXT_E			
DM	DA		AO_M	AO_M						
	WD			RT_M						
REG_W	IR_W	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M
	RD_W		RD							
	AO_W		AO_M		AO_M	AO_M				
	PC8_W								PC8_M	
	EXT_W						EXT_M			
RF	A3		IR_W[RT]		IR_W[RD]	IR_W[RT]	IR_W[RT]		$31	
	WD3		RDEXTD		AO_W	AO_W	EXT_W		PC8_W	

4、整合基本数据通路

和实验三一样，将上面的数据通路整合，将”支流“的汇入处添加多选器。

	模块	端口						复用器	选择信号
IFU	PC	Q	PC4	NEXTPC				MFPC	PCSEL
	ADD4	PC	Q						
	ADD8	PC	Q						
	IM	IA	Q						
		nowstate							
REG_D		IR_D	IR						
		PC4_D	PC4						
		PC8_D	PC8						
RFU	RF	A1	IR_D[RS]						
		A2	IR_D[RT]						
		A3	IR_W[RT]	IR_W[RD]	$31			MFA3	A3SEL
		WD3	AO_W	EXT_W	PC8_W			MFWD3	WD3SEL
IDU	EXT	I16	IR_D[I16]						
	CMP	D1	RD1						
		D2	RD2						
	NPC	PC4	PC4_D						
		I26	IR_D[I26]						
REG_E		IR_E	IR_D						
		RS_E	RD1						
		EXT_E	EXTD						
		RT_E	RD2						
		PC8_E	PC8_D						
EXU	ALU	A	RS_E						
		B	EXT_E	RT_E				MFB	BSEL
REG_M		IR_M	IR_E						
		AO_M	AO						
		RT_M	RT_E						
		PC8_M	PC8_E						
		EXT_M	EXT_E						
MEU	DM	DA	AO_M						
		WD	RT_M						
REG_W		IR_W	IR_M						
		RD_W	RD						
		AO_W	AO_M						
		PC8_W	PC8_M						
		EXT_W	EXT_M						

5、构建控制器

和实验三一样，构建控制器。

TYPE	指令	INPUT			OUTPUT									
		OP	FUNCT	RT	PCSEL	EXTOP	NPCOP	ALUCTRL	ALUOP	BSEL	WM	A3SEL	WD3SEL	WR
LD	LW	100011	X	X	0	0	X	0	0(+)	0	0(不写)	0	0	1
STR	SW	101011	X	X	0	0	X	0	0(+)	0	1	X	X	0
CAL_R	ADDU	000000	100001	X	0	X	X	15	0	1	0	1	1	1
	SUBU	000000	100011	X	0	X	X	15	1	1	0	1	1	1
CAL_I	ORI	001101	X	X	0	2	X	2	2	0	0	0	1	1
LUI		001111	X	X	0	1	X	X	X	X	0	0	2	1
B_TYPE	BEQ	000100	X	X	RES	X	0	X	X	X	0	X	X	0
J		000010	X	X	1	X	1	X	X	X	0	X	X	0
JAL		000011	X	X	1	X	1	X	X	X	0	2	3	1
NOP		000000	000000	X	0	X	X	X	X	X	0	X	X	0

6、构建冲突处理模块

在每个模块内，分析所有类别的源寄存器、目的寄存器、tuse、tnew，以便写出冲突控制逻辑。根据写出的控制逻辑，构造处理单元。

这里不论是每个模块都添加TUSE、TNEW等参量计算单元，还是统一在冲突处理单元内部添加都可行。

下表（AT表）里的S1_X和S2_X表示X阶段的源寄存器，DES_X表示X阶段的目的寄存器，S1X和S2X表示X阶段的RS/RT源寄存器数据。

				以下信号为每个阶段对应的转发阻塞控制模块的输出信号，X代表不需要该信号，$0代表不是源寄存器或者目的寄存器												
				S1_D/E/M	TUSE1_	S1D	MUX		S2_D/E/M	TUSE2_	S2D	MUX		DES_	TNEW_	DESD_
D	右侧为模块输入信号IR_D	STR	第一个源寄存器编号和tuse	IR_D[RS]	1	RD1	MFRSD	第二个源寄存器编号和tuse	IR_D[RT]	2	RD2	MFRTD		X	X	X
		LD		IR_D[RS]	1	RD1	MFRSD		$0	X				X	X	X
		CAL_R		IR_D[RS]	1	RD1	MFRSD		IR_D[RT]	1	RD2	MFRTD		X	X	X
		CAL_I		IR_D[RS]	1	RD1	MFRSD		$0	X				X	X	X
		LUI		$0	X				$0	X				X	X	X
		B_TYPE		IR_D[RS]	0	RD1	MFRSD		IR_D[RT]	0	RD2	MFRTD		X	X	X
		J		$0	X				$0	X				X	X	X
		JAL		$0	X				$0	X				X	X	X
		NOP		$0	X				$0	X				X	X	X
E	右侧为模块输入信号IR_E	STR	第一个源寄存器编号和tuse	IR_E[RS]	0	RS_E	MFRSE	第二个源寄存器编号和tuse	IR_E[RT]	1	RT_E	MFRTE	目的寄存器编号和tnew	$0	X	X
		LD		IR_E[RS]	0	RS_E	MFRSE		$0	X				IR_E[RT]	2	X
		CAL_R		IR_E[RS]	0	RS_E	MFRSE		IR_E[RT]	0	RT_E	MFRTE		IR_E[RD]	1	X
		CAL_I		IR_E[RS]	0	RS_E	MFRSE		$0	X				IR_E[RT]	1	X
		LUI		$0	X				$0	X				IR_E[RT]	0	EXT_E
		B_TYPE		$0	X				$0	X				$0	X	X
		J		$0	X				$0	X				$0	X	X
		JAL		$0	X				$0	X				$31	0	PC8_E
		NOP		$0	X				$0	X				X	X	X
M	右侧为模块输入信号IR_M	STR	第一个源寄存器编号和tuse	$0	X			第二个源寄存器编号和tuse	IR_M[RT]	0	RT_M	MFRTM	目的寄存器编号和tnew	$0	X	X
		LD		$0	X				$0	X				IR_M[RT]	1	X
		CAL_R		$0	X				$0	X				IR_M[RD]	0	AO_M
		CAL_I		$0	X				$0	X				IR_M[RT]	0	AO_M
		LUI		$0	X				$0	X				IR_M[RT]	0	EXT_M
		B_TYPE		$0	X				$0	X				$0	X	X
		J		$0	X				$0	X				$0	X	X
		JAL		$0	X				$0	X				$31	0	PC8_M
		NOP		$0	X				$0	X				X	X	X
W	右侧为模	STR		X	X				X	X			目的寄存器编号和tnew	$0	X	X	可以靠rf内部转发解决到D段的冲突
		LD		X	X				X	X				IR_W[RT]	0	RD_W	
		CAL_R		X	X				X	X				IR_W[RD]	0	AO_W	
		CAL_I		X	X				X	X				IR_W[RT]	0	AO_W	
		LUI		X	X				X	X				IR_W[RT]	0	EXT_W	
		B_TYPE		X	X				X	X				$0	X	X	
		J		X	X				X	X				$0	X	X	
		JAL		X	X				X	X				$31	0	PC8_W	

注意：关于上表中lui的行信息，和笔者将lui的计算放到D段是绑定的也就是说，若将lui并入ALU，上表中的lui信息不正确。W到D的转发笔者采取寄存器堆内部转发因此第7节的MFRSD模块没有WTOD信号，即没有W到D的外部转发。

7、整合数据通路

前面整合过的数据通路时，没有考虑转发，现在分析转发之后，在原有数据通路上添加多选器并**修改模块内数据源头**，以支持转发。

下表中的XTOY型信号（$X \in E, M, W, Y \in D, E, M$）表示从阶段X到阶段Y可能转发的数据。

四、实验相关

1、实验要求

使用logism自主搭建出一个支持所给指令集的32位五级流水线CPU，并通过课下正确性测试。
课上会通过新增指令的方式当场修改CPU，来考核课下设计的CPU。

- 指令集：

  lw,sw,addu,subu,lui,ori,jal,j,beq,nop。

- 内存大小、指令容量

  内存32字，要求能执行1024条指令。

- 控制冲突采取添加**延迟槽指令**解决，请尽量通过转发解决冲突。
- 需要在适当的地方添加输出信号以便检验结果正确性，详情参见提交说明。

2、提供硬件

为了降低工作量，提供冲突处理的核心逻辑模块和五个阶段的流水线寄存器。在理解该模块接口的基础上，给模块适当的输入，并利用该模块的输出，使用该模块。

- 冲突处理模块接口
  - 转发模块

    forwardUnit.circ

    接口定义如下：

信号	方向	长度
s1_d：D段rs寄存器或0（取决于D段指令是否把rs作为源寄存器）	input	[4:0]
s2_d：D段rt寄存器或0（取决于D段指令是否把rt作为源寄存器）	input	[4:0]
des_e：E段目的寄存器编号	input	[4:0]
tnew_e：E段tnew	input	[3:0]
des_m：M段目的寄存器编号	input	[4:0]
tnew_m：M段tnew	input	[3:0]
s1_e：E段rs寄存器或0（取决于E段指令是否把rs作为源寄存器）	input	[4:0]
s2_e：E段rt寄存器或0（取决于E段指令是否把rt作为源寄存器）	input	[4:0]
des_w：W段目的寄存器编号	input	[4:0]
tnew_w：W段tnew	input	[3:0]
s2_m：E段rt寄存器或0（取决于M段指令是否把rt作为源寄存器）	input	[4:0]
frsdsel：D段中rs寄存器接收转发数据的**多选器**（选择来自E（frsdsel为1）、M（frsdsel为2）段数据或者使用本段原数据（frsdsel为0））	output	[1:0]
frtdsel：D段中rt寄存器接收转发数据的**多选器**（选择来自E（frtdsel为1）、M（frtdsel为2）段数据或者使用本段原数据（frtdsel为0））	output	[1:0]
frsesel：E段中rs寄存器接收转发数据的**多选器**（选择来自M（frsesel为1）、W段数据（frsesel为2）或者使用本段原数据（frsesel为0））	output	[1:0]
frtesel：E段中rt寄存器接收转发数据的**多选器**（选择来自M（frtesel为1）、W（frtesel为2）段数据或者使用本段原数据（frtesel为0））	output	
frtmsel：M段中rt寄存器接收转发数据的**多选器**（选择来自W（frtmsel为1）段数据或者使用本段原数据（frtmsel为0））	output	

因此使用该模块，一般来说你还需要**自己添加五个多选器**，第三章第七节的表格就出现过。

- 阻塞模块

  stallUnit.circ

  接口定义如下：

信号	方向	长度
s1_d：D段rs寄存器或0（取决于D段指令是否把rs作为源寄存器）	input	[4:0]
tuse1_d：D段rs寄存器的tuse	input	[1:0]
s2_d：D段rt寄存器或0（取决于D段指令是否把rt作为源寄存器）	input	[4:0]
tuse2_d：D段rt寄存器的tuse	input	[1:0]
des_e：E段目的寄存器编号	input	[4:0]
tnew_e：E段tnew	input	[3:0]
des_m：M段目的寄存器编号	input	[4:0]
tnew_m：M段tnew	input	[3:0]
des_w：W段目的寄存器编号	input	[4:0]
tnew_w：W段tnew	input	[3:0]
en_pc：pc使能信号	output	
en_regd：D段流水线寄存器使能信号	output	
clr_e：E段流水线清空信号	output	

- 流水线寄存器

  提供流水线寄存器的模块，根据你的具体实现**可能需要自行修改**。接口说明见模块。

  流水线寄存器.zip

- **特别说明**

  提供模块是基于**寄存器实现内部转发的假设，并没有支持W段到D段的转发**。请同学们注意。

  是否内部转发，请自行决定，可适当修改提供模块。

  除了转发模块和阻塞模块可以不加修改（**如果实现了内部转发**），流水线寄存器很可能**需要添加流水信号**。

3、测试相关

建议同学们多测试指令的组合，以便测试CPU是否能够正确处理转发和暂停。

测试思路：

0、先检查单独的模块的行为是否符合预期。

1、先单独测试每个指令的行为：时钟仿真来观察流水线各个阶段的状态是否合理，如PC跳变、寄存器写入、存储器写入等，不要只关注最后输出的信号，而是关注每个阶段流水线的信号。

2、再测试指令集合的执行，期间相邻两条指令添加3个nop，来避免冲突的情况，从而只测试cpu的功能。

3、最后尽可能遍历冲突情况来编写指令进行测试。

在Mar进行观察时，注意勾选延迟槽选项。

4、添加指令概述

- **基本流程**

  理解指令的**执行过程** -> 判断是否已经实现同类指令 -> 是否需要新增**功能硬件** -> **控制信号**添加逻辑 -> **冲突处理**模块需要的信号（tuse，tnew，des_data，s1，s2，des）的译码逻辑 -> 检查新增的信号是否需要**流水**如果有新增信号。

- **jr实例**

编码	31 26	25 21	20 11	10 6	5 0
	special 000000	rs	0 00 0000 0000	0 00000	jr 001000
	6	5	10	5	6
格式	jr rs				
描述	PC ← GPR[rs]				
操作	PC ← GPR[rs]				
示例	jr $31				
其他	jr 与 jal/jalr 配套使用。jal/jalr 用于调用函数，jr 用于函数返回。				

步骤	结果
指令执行分析	jr指令的行为是跳转至rs寄存器
判断同类	不能归于现有的类别，继续向下（若能归于现有的指令类，则只需增加分类的逻辑把其归于同类，**前提**是你的设计全部都是按照**指令类**为单位进行）
功能硬件分析	只需要给pc前的多选器mfpc添加一个来自GPR[rs]的输入
控制信号分析	mfpc的控制信号需要修改，以便能够选择新增的输入
冲突处理分析	jr指令不需要**写入**寄存器（等价于写入0号寄存器），因此tnew、des_data的译码逻辑不必修改，而des需要修改译码逻辑。但其**需要**rs寄存器的信息，因此tuse1_d和s1_d需要增加逻辑。
流水分析	后面模块并不需要jr产生的信息，因此并没有新的信息需要流水。

5、一些建议

在**理解流水线cpu的原理**基础上，将提供的冲突处理模块充分理解后，你要自己搭建的电路就是**产生冲突处理模块所需要信号**的译码器。

请大家设计的电路产生信号满足冲突处理模块的接口要求，如果可以，尽量看看提供模块的逻辑，并不复杂。

关于控制信号表格、AT表格的使用：这些表格和具体实现有着很大的勾连关系，第三章的所有表格是一个整体，也就是说使用自己的设计直接套用AT表格不一定正确。

建议同学们关于自己的设计构造AT表格。

五、扩展指令

除了基本的指令，还有一些特别的指令，在这里做出介绍。

第一类指令像是lw、sw的扩展，它们是lh（读取半字并进行符号扩展）、lhu（读取半字并进行无符号扩展）、lb（读取字节并进行符号扩展）、lbu（读取字节并进行无符号扩展）、sh（存储半字），sb（存储字节）。

对于sh和sb，需要修改DM的内部结构，因为需要写入半字和字节。我们给出的建议是：给DM增加BE[3:0]输入端口（Byte Enable），以决定写入地址的字的哪些字节需要修改。BE信号由BEEXT模块产生，具体实现就任君发挥了。

对于lh、lhu、lb和lbu，选择添加**新的字处理模块**RDEXT，它可以将从**DM中取出的字**进行适当的提取和扩展，产生字写回寄存器堆。这里并没有把该模块的功能并入DM，因为DM已经足够复杂，进一步的耦合不利于电路设计和查找bug，也会增加DM模块的耗时，降低吞吐量。

注意：下面涉及指令一定不会在课上考察。

第二类指令是乘除相关指令，它们是multu（无符号乘法），mult（有符号乘法），divu（无符号除），div（有符号除）。

32位数的乘法一般是64位结果，因此乘法结果保存在[HI:LO]中，HI是存储高32位的寄存器，LO是存储低32位的寄存器，内置在乘除单元中（MDU）。对于除法，将两个32位数的商存储在LO寄存器，余数存储在HI寄存器中。

MDU是执行乘除法的单元，一般乘除法的耗时比一般的加减逻辑运算慢得多，因此将其**独立于ALU**，以免拖累ALU模块的执行速度。举例来说：不考虑乘除指令，ALU中运算需要一个时钟周期，而乘法需要十个时钟周期，除法需要二十个时钟周期。如果把乘除法并入ALU，ALU的速度会降低二十倍，进而延长时钟周期，降低吞吐量。但如果把乘除法独立出来，在原来的时钟周期基础上，MDU执行乘除法，而其余指令正常通过ALU计算，从而MDU不会拖累ALU。

六、致谢

感谢同学们的阅读，希望同学们能学会处理冲突的方法，并顺利完成实验四。