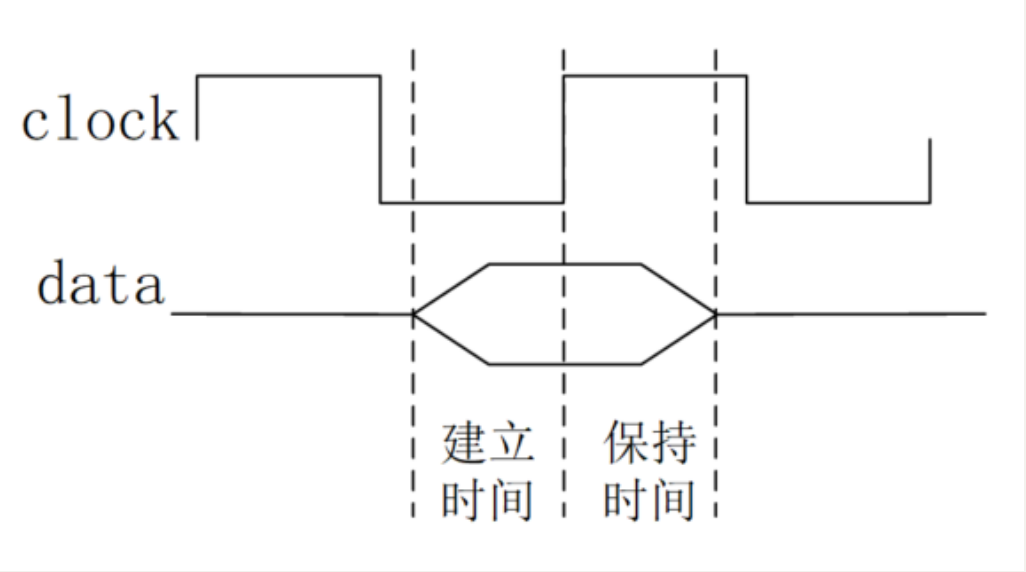
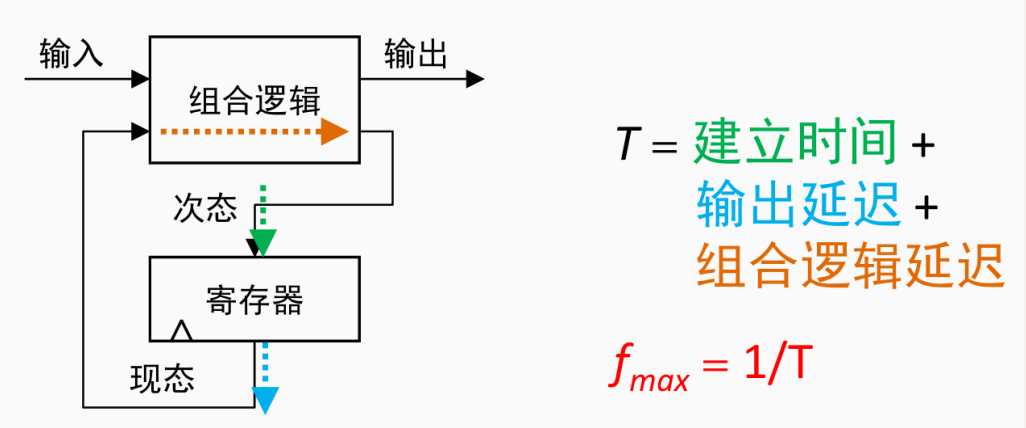
时钟频率

术语

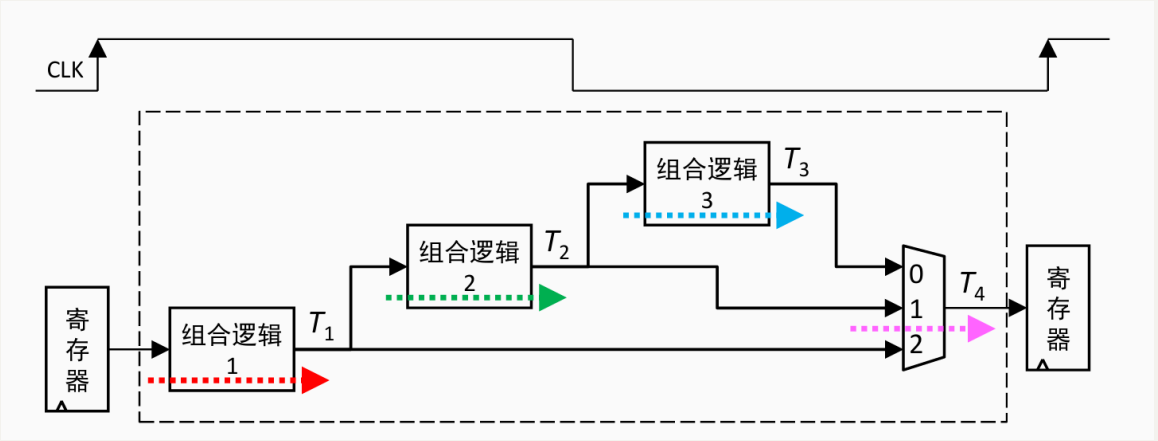
- 建立时间：输入信号在时钟上升沿之前必须有效的时间
- 保持时间：输入信号在时钟上升沿之后必须保持有效的时间
  - 

- 输出延迟：输出信号在时钟上升沿输出有效值的时间。
- 最大频率：取决于关键路径的最大延迟越小
  - 最大延迟 = 建立时间 + 输出延迟 + 组合逻辑延迟



单周期的性能

- 关键路径：任意两个寄存器之间的最大延迟

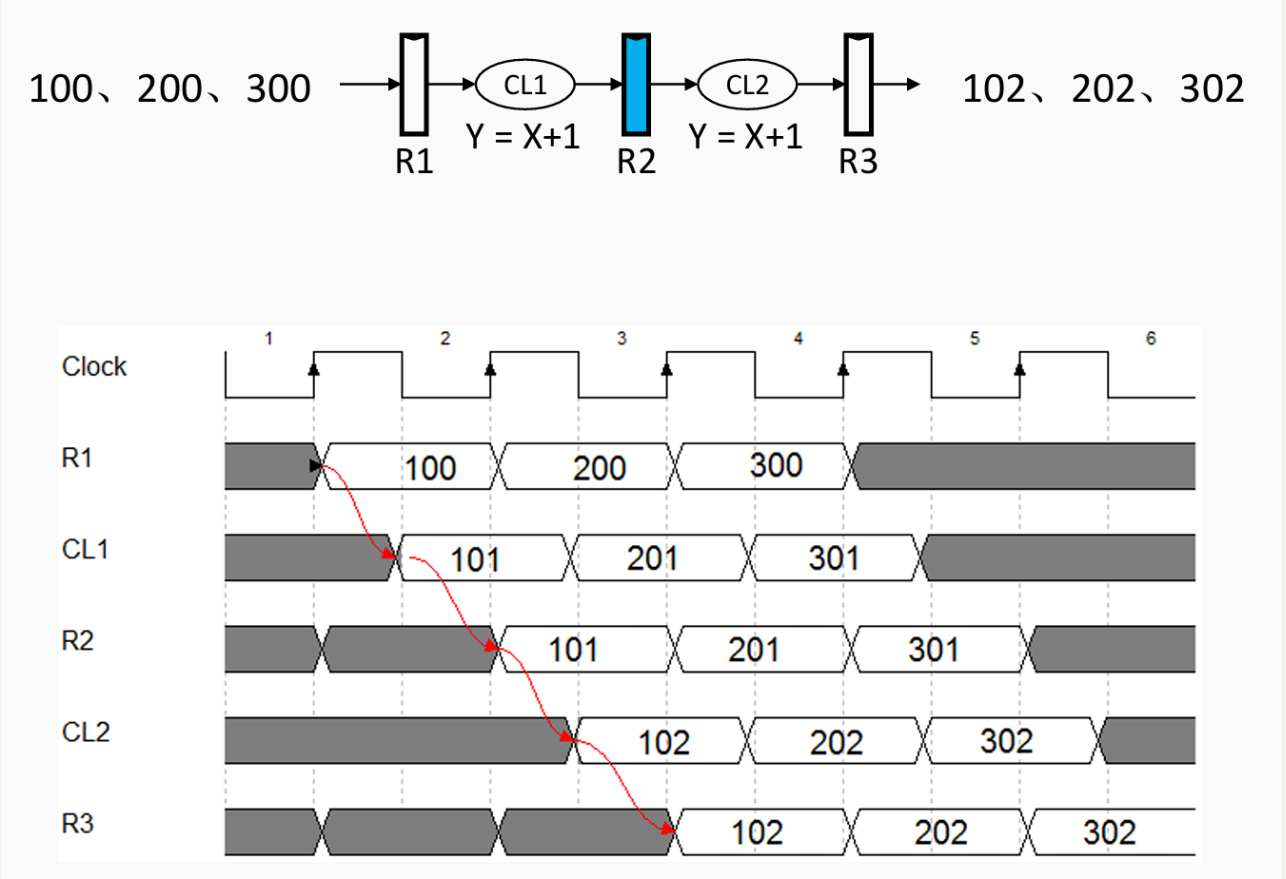


  - 电路中的时间周期：大于关键路径， 否则无法正常传递数据。
- 最大频率，由路径最长的指令决定，即为**lw**指令
- 单周期的局限性：最大频率取决于延迟最长的指令
  - 切成若干份，降低延迟。

流水线电路

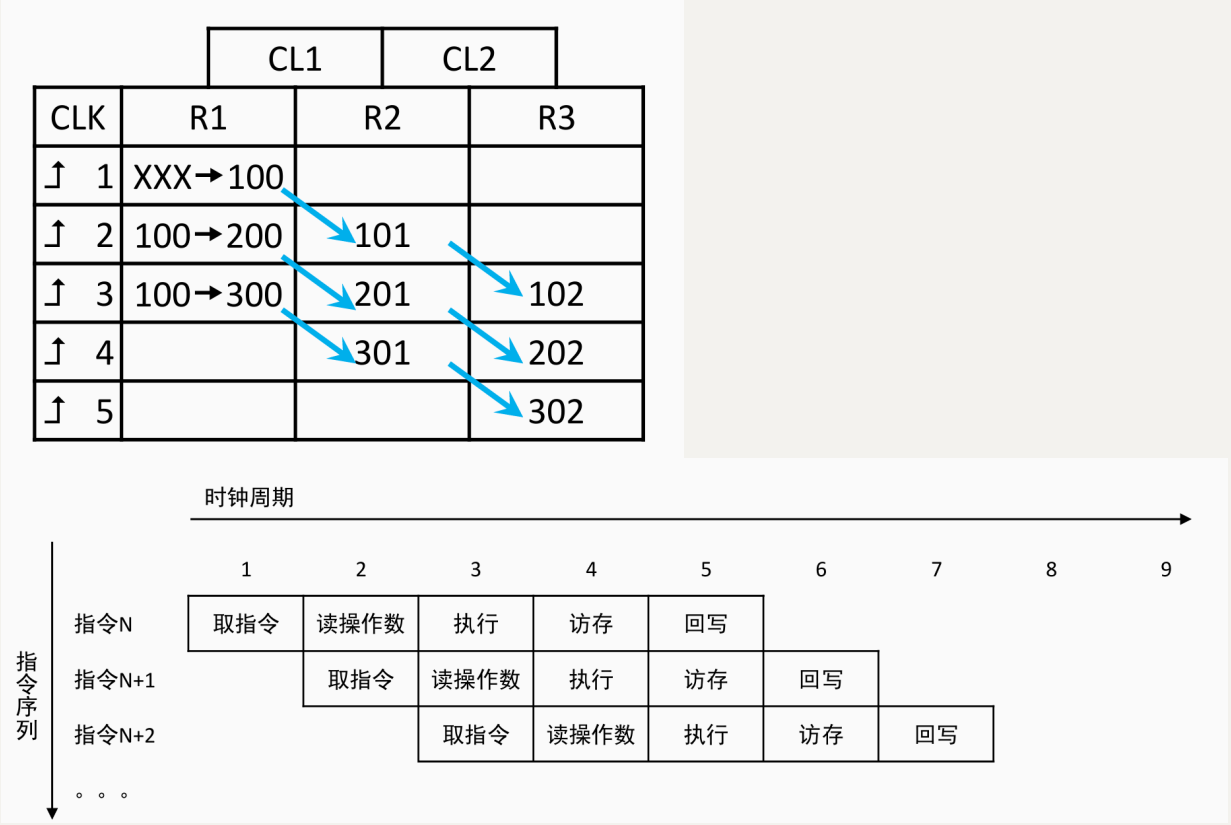
- 我们能不能在每一个环节之间都加入一个寄存器？

实例：简单的流水线电路



- 从左向右，依次推演
- 如果把`100` `200` `300`依次当成一个指令，那么就会交错执行，提升效率

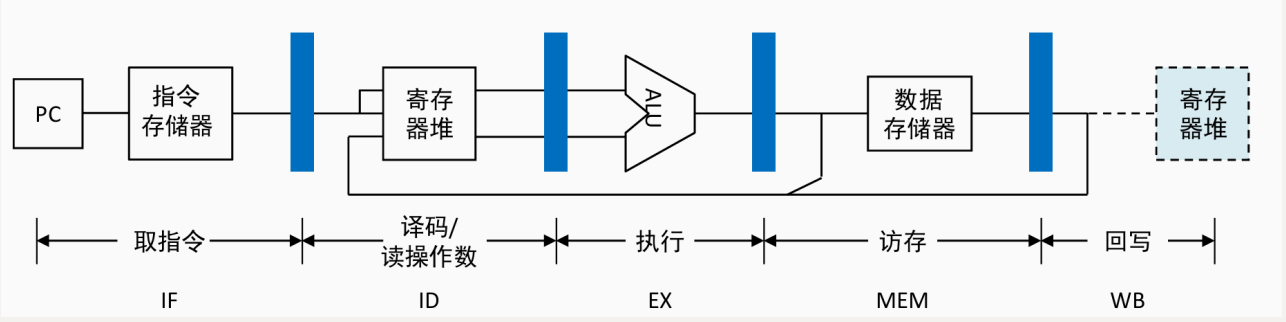
流水线的形式表



我们在流水线中，关心的是指令之间的关系，而非关心组合逻辑。

- 左侧表示第几个上升沿
- 水平方向表示流水线执行布局
- 竖直方向表示每一个寄存器在相应时钟上升沿后的值

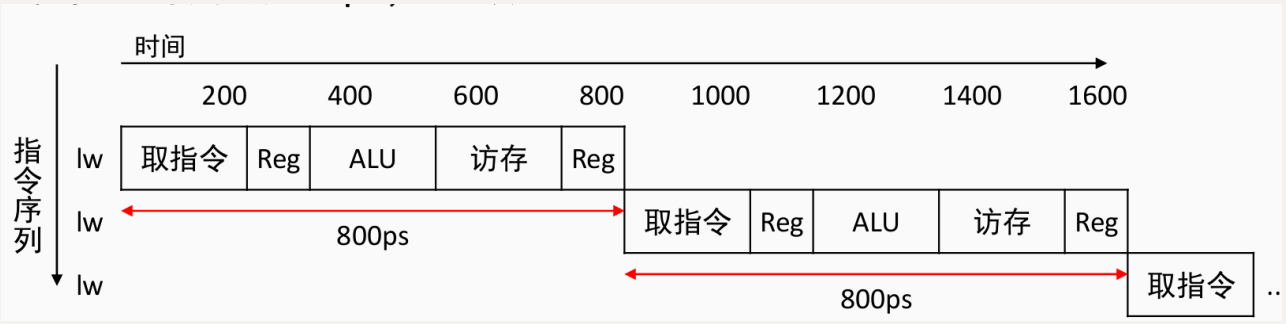
流水线性能



- 均衡性，因为总体频率取决于最慢的段。
- 对于大部分指令，完成周期均为N （N为流水线级数）
- 流水线充满：一组指令序列全部进入之后，一个时钟周期就可以完成一条指令

性能比较

单周期：



流水线

- 比较1：单条周期没有加速（反而会变长），因为需要按照各段最慢的时间来定义周期
- 比较2：流水线改善的是吞吐率：整体多条指令会变快

流水线加速比

$$流水线加速比 = \frac{单周期的程序执行时间}{流水线的程序执行时间}$$

- 理想加速比（完全均分） = 流水线级数
- 如果各个流水线执行时间不均，降低加速比
- 填充和排放流水线，会降低加速比
- BEQ / JAL / J，会改变指令顺序，中断流水线，

流水线CPI

- CPI: cycles per instruction
- 理想情况下，每个周期完成一个指令 $CPI = 1$
- 实际上，$CPI > 1$
  建模计算：期望（加权平均）

  - 某程序指令分布如下：load占25%，store为10%，分支指令为11%，R型计算类指令为54%
  - 假设：①load导致暂停概率为40%，且暂停代价为1个时钟周期。②分支指令预测成功率为75%，但预测失败就需要暂停1个时钟周期

  解

  - load：无数据相关时，load的CPI为1；有数据相关时，CPI为2

    $$CPI_{load}=1\times(1-40\%)+2\times40\%=1.4$$

  - store：CPI_{store}为1
  - 分支：预测成功，分支的CPI为1；预测失败，分支的CPI为2

    $$CPI_{分支}=1\times75\%+2\times(1-75\%)=1.25$$

  - R型：$CPI_{R型}$为1

    $$CPI = CPI_{load}\times25\% + CPI_{store}\times10\% + CPI_{分支}\times11\% + CPI_{R型}\times54\% = 1.1275$$

指令级并行（ILP）

Instruction level Parallelism

单核CPU

- 单条流水线
- 超标量多条流水线：共用同一个寄存器堆

多核CPU

- 任务级并行

MIPS指令集与流水线效率

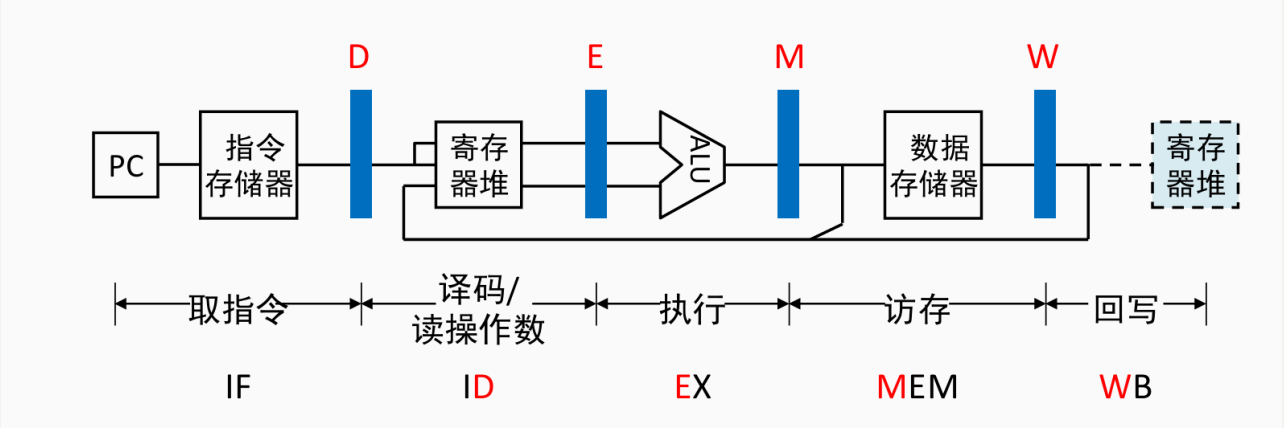
- 指令定长，都是32位
- 指令格式少，格式规整，两个源寄存器（rs,rt）位置保持不变（control 和 读寄存器并行)
- 存储器操作只有 load/store，访存的周期数固定（1个周期）
- 面向流水线的设计。

流水线规范

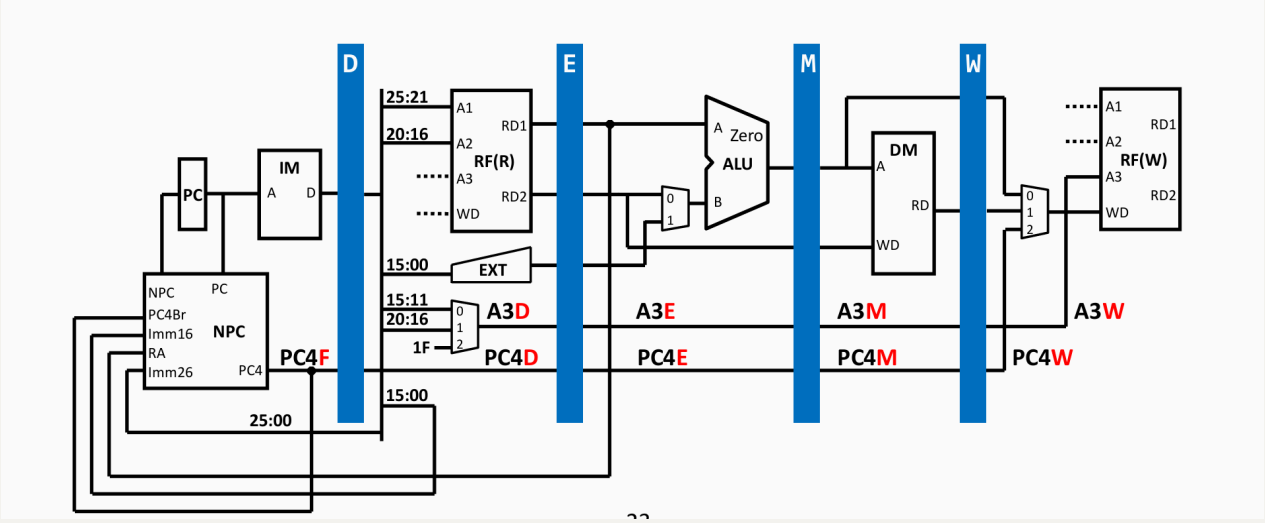
功能分区与寄存器命名

用各级后面的功能的某个字母命名

- 信号命名：信号_对应寄存器
- **IF**：Instruction Fetch（取指令）
- **ID**：Instruction Decode（指令译码）
- **EX**：Execute（执行）
- **MEM**：Memory Access（访存）
- **WB**：Write Back（回写）



###



流水线时空图

指令 + 周期 + 寄存器

相对PC的地址偏移	指令			IF级	ID级	EX级	MEM级	WB级	
		CLK	PC	IM	D	E	M	W	RF
0	Instr 1	↑ 1	0→4	Instr 1	Instr 1				
4	Instr 2	↑ 2	4→8	Instr 2	Instr 2	Instr 1			
8	Instr 3	↑ 3	12→12	Instr 3	Instr 3	Instr 2	Instr 1		
12	Instr 4	↑ 4	12→16	Instr 4	Instr 4	Instr 3	Instr 2	Instr 1	
16	Instr 5	↑ 5	16→20	Instr 5	Instr 5	Instr 4	Instr 3	Instr 2	Instr1
20	Instr 6	↑ 6	16→20	Instr6	Instr6	Instr5	Instr4	Instr3	Instr2

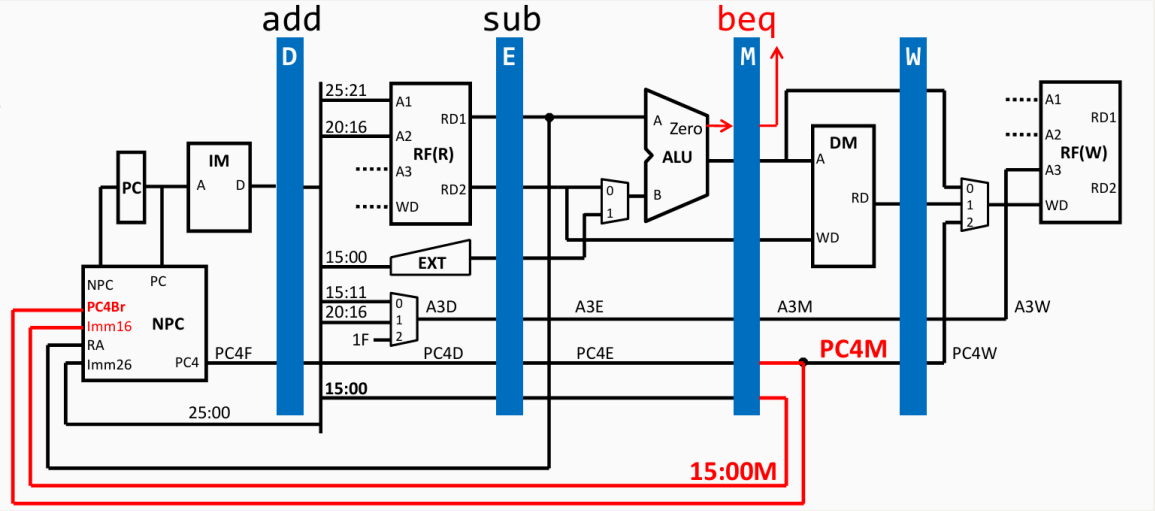
流水线数据通路

Q1: D级能否读出W级写入RF的值？

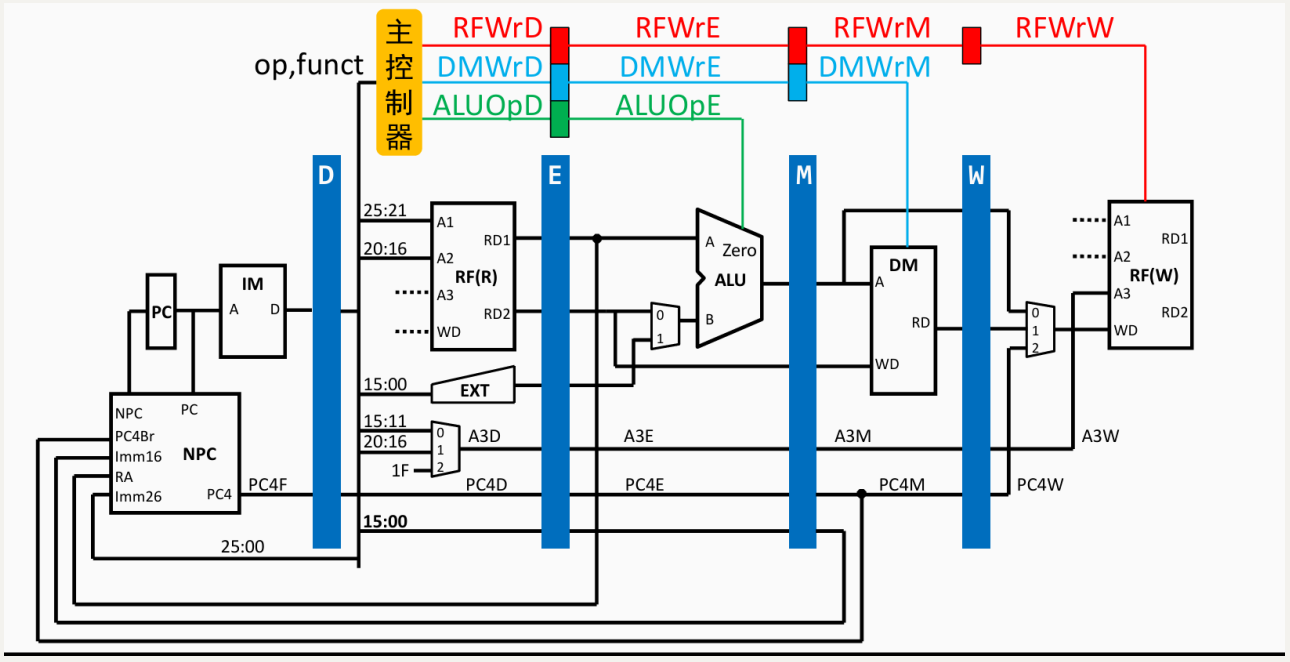
- 多条指令同时读写同一个寄存器时，会产生一致性问题
- 出错原因： 后发**Read**操作，在先发**Write**操作之前完成
- 解决方案：转发

Q2 信息同步

- 例子：多条连续指令用到了三种NPC的信息，同时进入NPC中
- 出错原因：指令的多个相关信息没有同步传递，需要在执行时，再传入NPC中



- 流水线的控制信号：也必须要同步传递



冒险

生产与消费的同步

结构冒险

征用同一个资源

- 取指令与访存(IM cache/DM cache)要分开，否则会产生资源冲突
- **RF** 读数据与写数据要分开， 读端口与写端口要分开，否则也会冲突

已经在单周期内解决了!

数据冒险

指令之间的数据依赖，例如前面的**Q1**

基于寄存器的数据相关

- 想法 写是一个比较有破坏性的操作
  对于同一个寄存器来说，有W-W R-W W-R**三种读写组合会残生冲突
- 原则： 程序员的逻辑，后续指令需要获得之前指令的结果
- 实例：`add $1, $2, $3` `sub $4, $1, $2`后一个sub需要用到add的结果`$1`

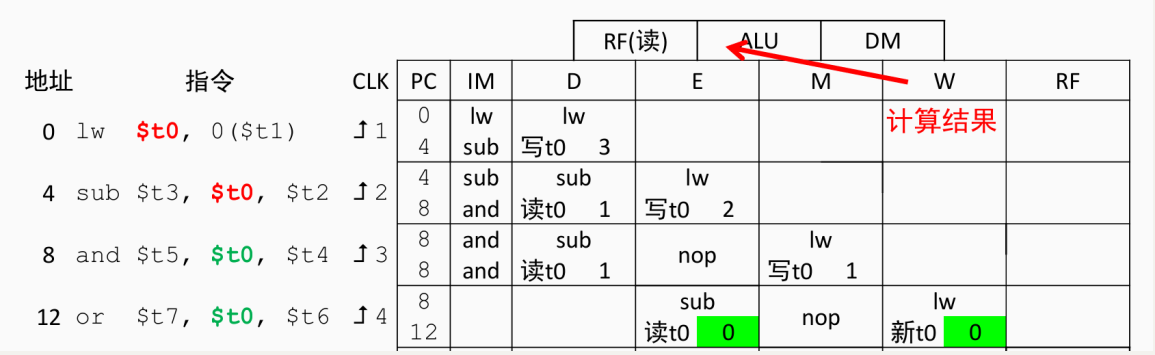
其他指令

-

load 导致的数据冒险

`lw`指令产生结果太晚，在W级才可以得到

- 因此需要先暂停一拍（`nop`指令）。

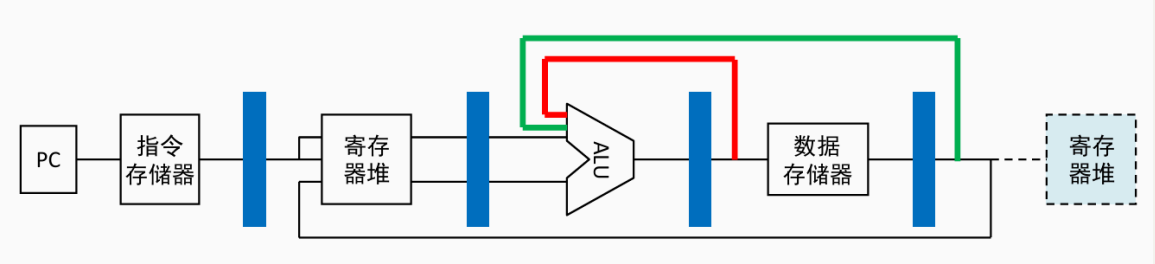


旁路 **bypast** 转发 **forward**

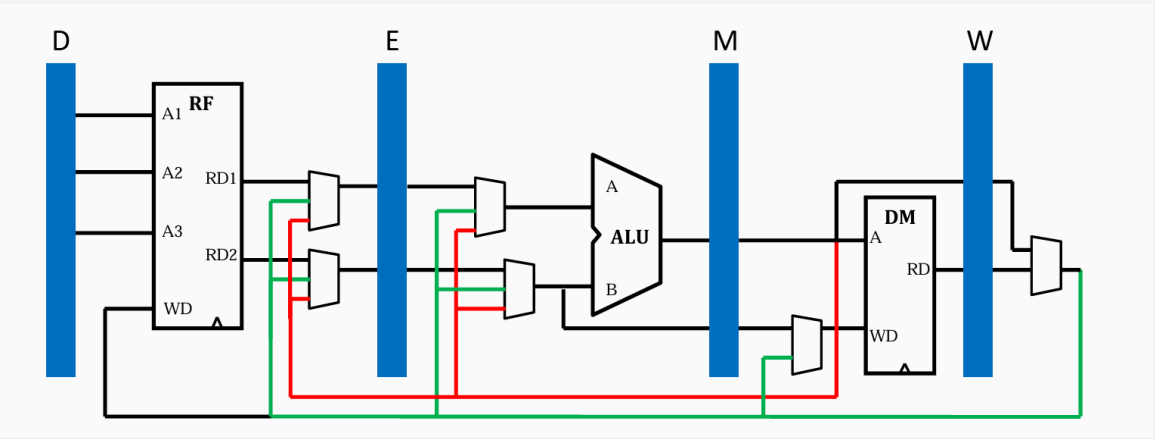
将**M**、**W**级保存的计算结果向前级**E**级的**ALU**传递。

- 当E级指令与M级指令写的寄存器是同一个时，则选择转发的结果，否则是E级传递的寄存器值
  例子：

- M-E转发：`add $1, $2, $3 sub $4, $1, $6`
- W-E转发: `add $1, $2, $3  xxx  or $5, $1, $2`



1. 最新结果： 可能产生在**M**级和**W**级
2. 需要使用结果： 可能产生在 **D,E,M**级



3. 最终策略：
    - D级：分析与E/M/W的相关性。若转发无法解决则暂停，直至转发可以解决相关
    - E级：无暂停问题，只需要分析与M/W的相关性，决定是否转发即可
    - M级：无暂停问题，只需要分析与W的相关性，决定是否转发即可

D级：加载延迟槽

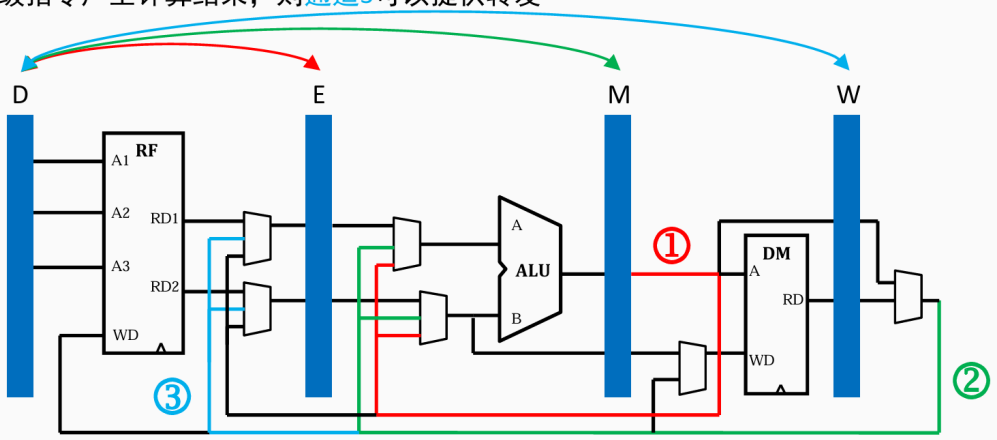
D级大门，要有一个看大门的，一旦之前指令有冲突，就需要暂停。

1. 计算类指令：

如果由E级指令产生计算结果，则通道①可以提供转发

如果由M级指令产生计算结果，则通道②可以提供转发

如果由W级指令产生计算结果，则通道3可以提供转发



2. 暂停类指令：LW

- 条件1：D级的指令要读取RS寄存器
- 条件2：E级为load类指令
- 条件3：D级读取的寄存器和E级写的寄存器相同
- `Stall = (add + sub + … + lw) & lwE & (A1==A3E)`

3. 暂停操作

- 冻结D级：使能信号为0
- 清除E级：clr = 1
- 禁止PC计数：使能信号为0

编译优化【编译原理】

- 将与**load**无关的指令放置在**load**类指令后

控制冒险

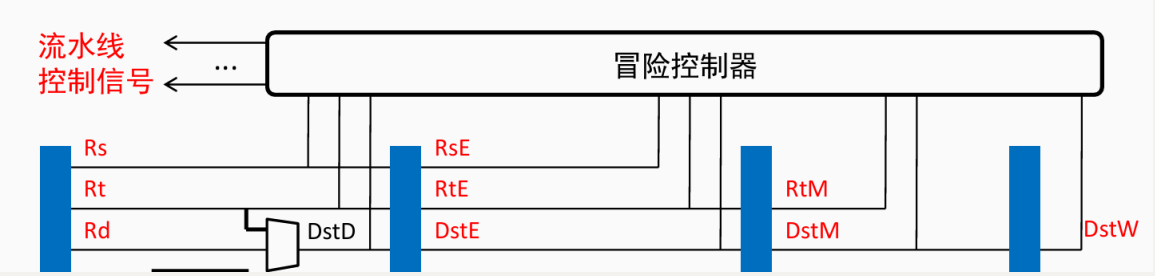
产生原因：分支指令影响控制流

- 当分支指令结果出现前，D级取指令无法确保指令是正确的！
- 方案1：直接暂停，会大大降低效率
- 方案2：假设不进入分支，若Zero为1，再清除前两条指令(D,E)
    - 方案**3**：**BEQ**判断前置：因此使用数据提前了
    - 和计算类指令的冲突：
    - 和lw指令的冲突；
- 方案4：分支延迟槽

冒险控制器

检测分析各类冒险，控制流水线的运行

- 传入每一级的数据



流水线工程方法

目的：对冲突进行建模，进行覆盖性分析，不遗漏任何冲突

- 核心：建模两个时间，利用策略矩阵，反向构造所有暂停和转发案例的表达式！
- 实践-认识-建模实践-再认识

全速流水线

1. 构造无转发的基础流水线：插入几级寄存器
2. 构造转发电路
    a. 对于每一个功能部件，询问它是否需要用到寄存器的值，谁在功能部件后，谁就要连过来
    b. 需要注意D级判断Zero的也是一个功能部件！
3. 添加暂停（延迟槽）：利用生产消费模型
4. 构造暂停/转发策略矩阵，根据矩阵生成暂停和转发控制表达式

流水线命名

例如：**rt**就有二义性，可能代表源寄存器编号，也可能是写寄存器编号

为消除歧义，对指令字段进行重命名：

- RS → A1（源操作数1编号）
- RT → A2（源操作数2编号）或 A3（目的操作数编号）
- 对流水线中传递的值统一命名，如V1（A1对应寄存器值）、V2（A2对应寄存器值）、AO（ALU输出）、DR（数据存储器读出值）、PC+4等，确保跨级传递的一致性。

名字	宽度	描述	对应的指令域	命名考虑
A1	5位	第1个源寄存器编号	当前只有rs	与RF设计一致
A2	5位	第2个源寄存器编号	当前只有rt	A3需要由rd/rt/+31转换得到（后续介绍）
A3	5位	目的寄存器编号	rd或rt或+31	
V1	32位	RF的第1个寄存器输出值		Value的首字母
V2	32位	RF的第2个寄存器输出值		
E32	32位	EXT的32位扩展结果		扩展的英文缩写
AO	32位	ALU计算结果		沿用多周期命名
DR	32位	DM输出值		
PC4	32位	下一条指令地址		

流水线数据通路

注：

1. RF：将读写分离，便于理解第五阶段。

寄存器的功能：

1. 保存前一级功能部件产生的信息 【所以关注信息产生的来源】
2. 传递信息，一路向下传即可

以**LW**指令，说明数据通路

步骤	**RTL**	控制信号
回写	`RF[A3@W] <- DR@W`	RFWr: 1
访存	`DR@W <- DM[AO@M] A3@W <- A3@M`	
计算	`AO@M<- ALU(V1@E,E32@E) A3@M <- A3@E`	ALUOp:ADD
读数	`V1@E<-RF[IR[rs]@D]; E32@E<-EXT(IR[i16]@D); A3@E<-IR[rt]@D; A1@E<-IR[rs]@D`	EXTOp: Sign
读指令	`IR@D <- IM[PC]; PC <- PC + 4`	

综合无转发数据通路

1. 合并每一个指令数据通路，
2. 对于输入源在2个以上者，必须部署相应功能**MUX**

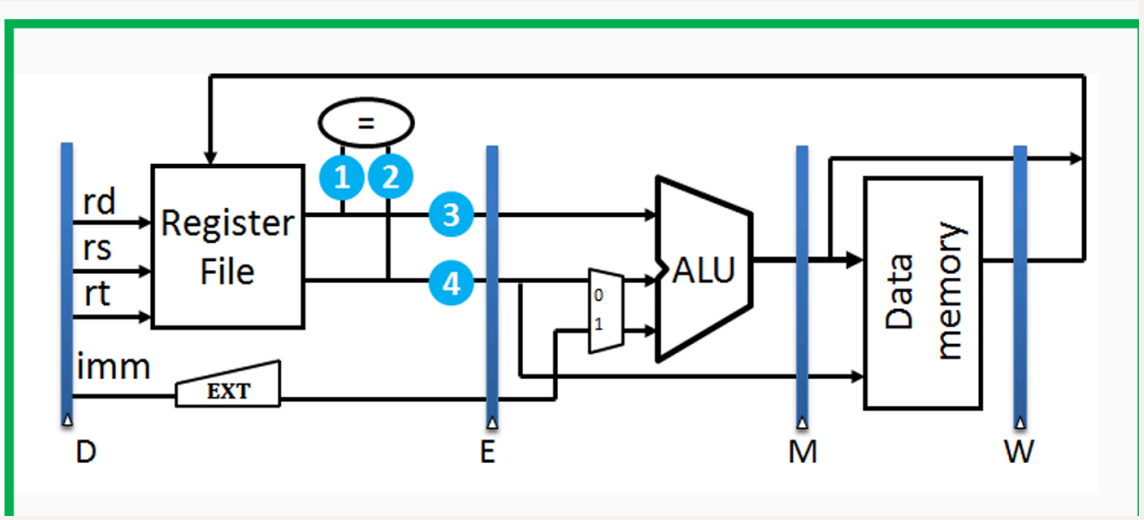
数据通路表

转发设计

- 生产消费模型

找到所有与寄存器堆值有关的位置

- 尤其注意`RF.RD1(V1) RF.RD2(V2)`的关系，标有这四个的需求者必须要加转发电路
- beq比较电路前移后，如图，1、3；2、4分别也需要一个转发MUX



在消费者处增加转发**MUX**

1. 转发MUX的路数等于N+1 N：后级转发数据的数量，1：前级传递寄存器数据
2. 建议规划转发MUX端口时，端口数字越大则其优先级越高(选择语句中靠前)
3. 命名：

MFALUBE	
MF	MUX Forward
ALUB	ALU的B输入
E	E级

4. 注意，离需求者最近的生产者的转发信息，越优先

例子：MALUB的输入0(来自V2@E)，需要替换为MFALUBE

1. 接收来自AO@M AO@W DR@W 的转发信息，以及他自己V2@E的信息
2.

	0	1	2	3
MFALUBE	V2@E	DR@W	AO@W	AO@M
优先级	低	中		高

3. verilog 中，优先级越高，越放在靠前冒号的左侧。

表格

MUX	0	1	2
MPC	ADD4	NPC	MFPCF
PC			
PC			
IM			
ADD4			
IR[rs]@D			
IR[rt]@D			
IR[i16]@D			
PC4@D			
IR[i26]@D			
MFCMP1D			
MFCMP2D			
IR[rs]@D			
IR[rt]@D			
MA3E	IR[rt]@D	IR[rd]@D	
EXT			
PC4@D			
V1@E			
MALUB	MFALUBE	E32@E	
MFV2M			
A2@E			
ALU			
A3@E			
PC4@E			
AO@M			
MFWDM			
A3@M			
PC4@M			
AO@M			
DM			
A3@W			
MRFWD	AO@W	DR@W	PC4@W

生产消费模型

供给者：保存有reg新结果的流水线寄存器

- 例1：所有运算类指令的供给者是M、W级
- 例2：Load指令的供给者是W级

需求者：使用或传递寄存器值的功能部件和流水线寄存器

- 例1：add/sub/ori 的需求在E级的ALU
- 例2：j指令不需要读任何寄存器的值，因此无所谓

数据冒险的本质为供需的匹配

对于一条指令来说，需要考虑

时间**1**：什么时候用到寄存器的值？【消费】T_{use}

- T_{use} 是静态值，只关注每条指令的操作语义。
- 同一条指令可以有两个不同的T_{use} 例如Store类指令。
- Beq 指令的 $T_{use} = 0$ (前移后)

	T_{use}	
	rs	rt
add	1	1
sub	1	1
andi	1	
ori	1	
lw	1	
sw	1	2
beq	0	0
jr	0	
	{0,1}	{0,1,2}

- 用变量表示Tuse
  - `Tuse_RS0 = beq+ jr`
  - `Tuse_RS1 = add + sub + andi+ … + sw`

时间**2**：什么时候产生寄存器的值？【生产】T_{new}

- **E**级及之后的指令需要多少周期产生写入寄存器的结果
- T_{new}是动态值，每走一拍**-1**
- 例如：`lw $1, 0($0)` 要到W级才可产生。
    - $T_{new} = 2$ E级
    - $T_{new} = 1$ M级
    - $T_{new} = 0$ W级

指令	功能部件	T_{new}		
		E	M	W
add	ALU	1	0	0
sub	ALU	1	0	0
andi	ALU	1	0	0
ori	ALU	1	0	0
lw	DM	2	1	0
sw				
beq				
jal	PC	0	0	0

- 可以归类为**3**类产生寄存器值的指令：**ALU**、**DM**、**PC**类
- W级所有指令已经产生新值
- 建模T_{new}计数器
    - 在D级根据指令产生编码值

指令类别	T_{new}初值
运算类	1
读存储类	2
函数调用类	0

    - 逐级-1（注意代码中两个always模块合并）

```
reg [1:0] Tnew_E, Tnew_M ;

always @(……)
  if (add | sub | andi | ori)
    Tnew_E <= `T_ALU ;
  else if (lw)
    Tnew_E <= `T_DM ;
  ……

always @(…)
  if (Tnew_E != 0)
    Tnew_M <= Tnew_E - 1 ;
```

暂停条件：结果产生的时间晚于需要的时间，$T_{use} < T_{new}$

转发条件：结果产生的时间早于或者等于所用时间，$T_{use} \geq T_{new}$

如果有多个供给者：离需求者最近的生产者的转发信息，越优先

基于生产消费模型的构造暂停、转发

根据指令的 T_{use}、T_{new} 分别构造**rs/rt**的策略矩阵

- 例子：ADD

add的rs暂停转发分析矩阵

F 转发
S 暂停

T^{rs}_{use} \ T_{new}	E级			M级		W级
	0	1	2	0	1	0
1	F	F	S	F	F	F

各级可能的取值范围

将T_{new}划分为**ALU**、**DM**、**PC**三类指令！ 从而得到一般性的策略矩阵！

rs策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

暂停表达式的构造

- 每一个暂停的条件：$T_{use} \wedge T_{new}$ & 寄存器相等 & 该级寄存器的写使能

```
`define A1 IR[25:21]

Stall_RS0_E1 = Tuse_RS0 & (Tnew_E==2'b01) & (`A1==A3_E) & W_E
Stall_RS0_E2 = Tuse_RS0 & (Tnew_E==2'b10) & (`A1==A3_E) & W_E
Stall_RS0_M1 = Tuse_RS0 & (Tnew_M==2'b01) & (`A1==A3_M) & W_M
Stall_RS1_E2 = Tuse_RS1 & (Tnew_E==2'b10) & (`A1==A3_M) & W_E
```

- `Stall_RS = Stall_RS0_E1 | Stall_RS0_E2 ...`
- `Stall_RT = ...`
- `Stall = Stall_RS | Stall_RT`

转发表达式的构造

转发的条件：`A2_X = A3_Y  & A3_Y产生数据 & Y级写使能`

- 无需关注该域是否有效，无效时被功能**MUX**筛走了

优先级：近的优先级高，远的低：

- 例子：ALU B端的转发MUX控制信号 FALUBE

```
`define M2E_ALU 2   //M向E转发ALU结果
`define W2E_WD  1   //W向E转发回写结果
......

FALUBE =
    ((A2_E==A3_M) & (Tnew_M==2'b00) & W_M ? `M2E_ALU :
    ((A2_E==A3_W)                   & W_W ? `W2E_WD  :
                                           0
```

NOTE
书写表达式时，尽量多使用宏，增加可读性。

高 ↓ 低

一定注意顺序！！ 表示信号优先级

控制冒险的处理

实现延迟槽、比较前移

实现延迟槽

- 硬件解决PC+8
- 注意jal及jalr指令应保存PC+8(单独加一个**adder**，或者在**NPC**里对两个指令特殊处理）
- 编译调度指令，不归我们管
- `JAL JALR` 正常回写即可

模块	LW	SW	ADD	SUB	ORI	BEQ	J	JAL	JR	MUX	0	1	2
PC	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	NPC	NPC	RF.RD1	MPC	ADD4	NPC	RF.RD1
IM	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC			
ADD4	PC	PC	PC	PC	PC	PC	PC	PC		ADD4			

	模块	LW	SW	ADD	SUB	ORI	BEQ	J	JAL	JR	MUX	0	1	2
D 级	IR	IM	IM	IM	IM	IM	IM	IM	IM	IM	IR[rs]@D			
D 级	PC4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4		IR[rt]@D			
RF	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D	IR[i16]@D			
RF	A2		IR[rt]@D	IR[rt]@D	IR[rt]@D		IR[rt]@D				PC4@D			
EXT		IR[i16]@D	IR[i16]@D			IR[i16]@D					IR[i26]@D			
NPC	PC4	PC4					PC4@D	PC4@D	PC4@D		RF.RD1			
NPC	I26						IR[i16]@D	IR[i26]@D	IR[i26]@D		RF.RD2			
CMP	D1						RF.RD1				RF.RD1			
CMP	D2						RF.RD2				RF.RD2			
E 级	V1	RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1					IR[rs]@D			
E 级	V2		RF.RD2	RF.RD2	RF.RD2						IR[rt]@D			
E 级	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D					MA3E	IR[rt]@D	IR[rd]@D	
E 级	A2		IR[rt]@D	IR[rt]@D	IR[rt]@D						EXT			
E 级	A3	IR[rt]@D		IR[rd]@D	IR[rd]@D	IR[rt]@D			31		PC4@D			
E 级	E32	EXT	EXT								V1@E			
E 级	PC4								PC4@D		MALUB	V2@E	E32@E	
ALU	A	V1@E	V1@E	V1@E	V1@E	V1@E					V2@E			
ALU	B	E32@E	E32@E	V2@E	V2@E	E32@E					A2@E			
M 级	V2		V2@E								ALU			
M 级	A2		RD2@E								A3@E			
M 级	AO	ALU	ALU	ALU	ALU	ALU					PC4@E			
M 级	A3	A3@E		A3@E	A3@E	A3@E					AO@M			
M 级	PC4								PC4@E		V2@M			
DM	A	AO@M	AO@M	AO@M	AO@M	AO@M					A3@M			
DM	WD		RD2@M								PC4@M			
W 级	A3	A3@M		A3@M	A3@M	A3@M					AO@M			
W 级	PC4								PC4@M		AO@M			
W 级	AO	AO@M		AO@M	AO@M	AO@M					DM			
W 级	DR	DM									A3@W			
RF	A3	A3@W		A3@W	A3@W	A3@W					MRFWD	AO@W	DR@W	PC4@W
RF	WD	DR@W		AO@W	AO@W	AO@W			PC4@W					