

# 架构实验1

支持RISC - V的流水线CPU  
RVI32指令集

## 任务

- 理解RISC - V RV32I指令
- 掌握执行RV32I指令的流水线CPU的设计方法
- 掌握流水线前递检测和旁路单元的设计方法
- 掌握预测不跳转分支单周期停顿的设计方法
- 掌握执行RV32I的流水线CPU程序验证方法

## 指令

Chenlu Miao 09/2022

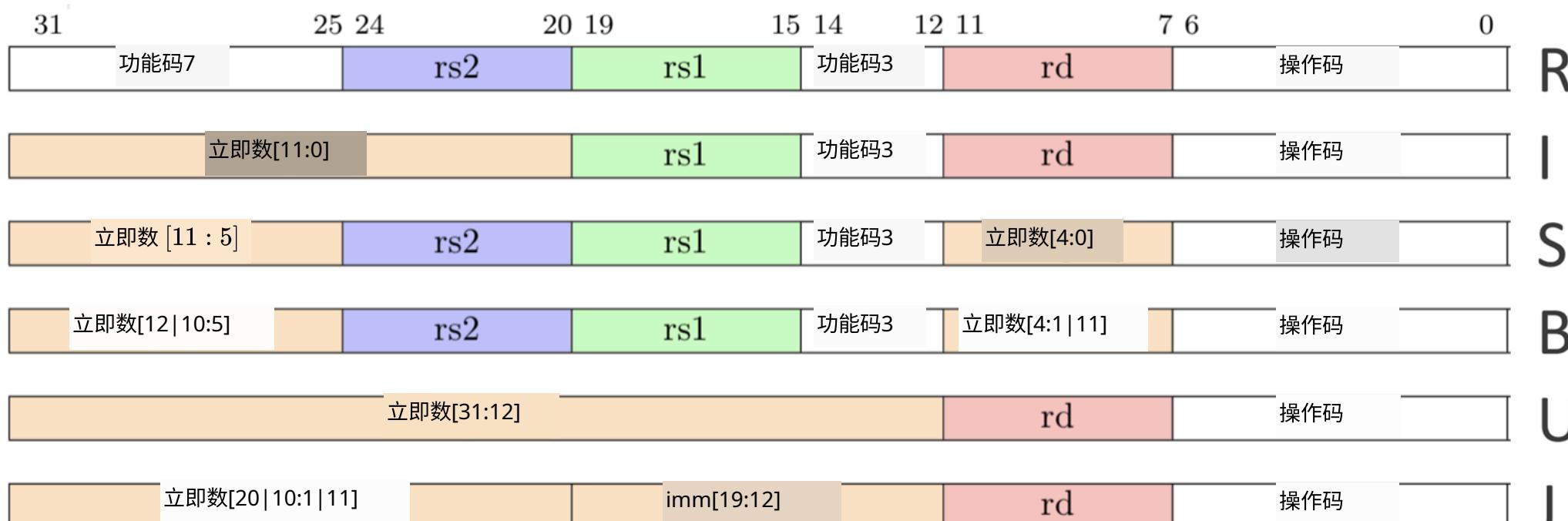
## 代码概述

## 概述

- RV32I指令集
- 架构概述
- 数据冒险与转发
- 控制冒险与分支预测
- 冒险检测单元

# RV32I指令集

## RV32I指令集



## RV32I指令集 - R类型

31	25 24	20 19	15 14	12 11	7 6	0
功能码7	rs2	rs1	功能码3	rd	操作码	

00000000	rs2	rs1	000	rd	0110011	ADD
01000000	rs2	rs1	000	rd	0110011	SUB
00000000	rs2	rs1	001	rd	0110011	SLL
00000000	rs2	rs1	010	rd	0110011	SLT
00000000	rs2	rs1	011	rd	0110011	SLTU
00000000	rs2	rs1	100	rd	0110011	XOR
00000000	rs2	rs1	101	rd	0110011	SRL
01000000	rs2	rs1	101	rd	0110011	SRA
00000000	rs2	rs1	110	rd	0110011	OR
00000000	rs2	rs1	111	rd	0110011	AND

## RV32I指令集 - B类型

31	25 24	20 19	15 14	12 11	7 6	0
立即数[12 10:5]	rs2	rs1	功能码3	立即数[4:1 11]	操作码	

立即数[12 10:5]	rs2	rs1	000	立即数[4:1 11]	1100011	BEQ
立即数[12 10:5]	rs2	rs1	001	立即数[4:1 11]	1100011	BNE
立即数[12 10:5]	rs2	rs1	100	立即数[4:1 11]	1100011	BLT
立即数[12 10:5]	rs2	rs1	101	立即数[4:1 11]	1100011	BGE
立即数[12 10:5]	rs2	rs1	110	立即数[4:1 11]	1100011	BLTU
立即数[12 10:5]	rs2	rs1	111	立即数[4:1 11]	1100011	BGEU

## RV32I指令——S类型

31	25:24	20:19	15:14	12:11	7:6	0
立即数[11:5]	rs2	rs1	功能码3	立即数[4:0]	操作码	

立即数[11:5]	rs2	rs1	000	立即数[4:0]	0100011	SB
立即数[11:5]	rs2	rs1	001	立即数[4:0]	0100011	SH
立即数[11:5]	rs2	rs1	010	立即数[4:0]	0100011	SW

## RV32I指令——I类型

31	25:24	20:19	15:14	12:11	7:6	0
立即数[11:0]	rs1	功能码3	rd		操作码	

立即数[11:0]	rs1	000	rd	0010011	
立即数[11:0]	rs1	010	rd	0010011	
立即数[11:0]	rs1	011	rd	0010011	
立即数[11:0]	rs1	100	rd	0010011	
立即数[11:0]	rs1	110	rd	0010011	
立即数[11:0]	rs1	111	rd	0010011	
0000000	移位量	rs1	001	rd	0010011
0000000	移位量	rs1	101	rd	0010011
0100000	移位量	rs1	101	rd	0010011

## RV32I指令——I类型

31	25:24	20:19	15:14	12:11	7:6	0
立即数[11:0]	rs1	功能码3	rd		操作码	

立即数[11:0]	rs1	000	rd	0000011	LB
立即数[11:0]	rs1	001	rd	0000011	LH
立即数[11:0]	rs1	010	rd	0000011	LW
立即数[11:0]	rs1	100	rd	0000011	LBU
立即数[11:0]	rs1	101	rd	0000011	LHU

立即数[11:0]	rs1	000	rd	1100111	JALR
-----------	-----	-----	----	---------	------

## RV32I指令——U类型

31	25:24	20:19	15:14	12:11	7:6	0
立即数[31:12]	rd		操作码			

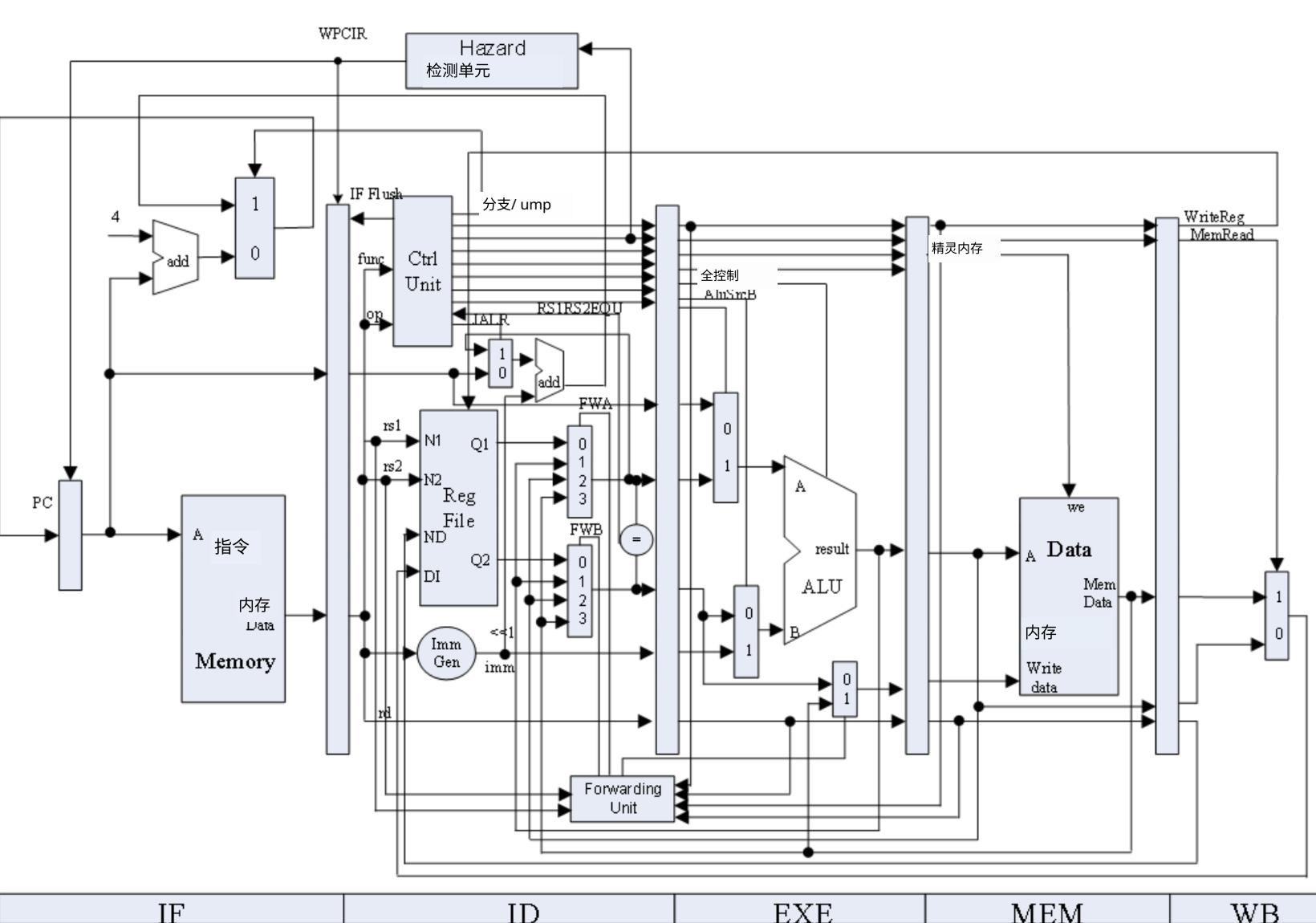
立即数[31:12]	rd	0110111	LUI
imm[31:12]	rd	0010111	AUIPC

## RV32I指令——J类型

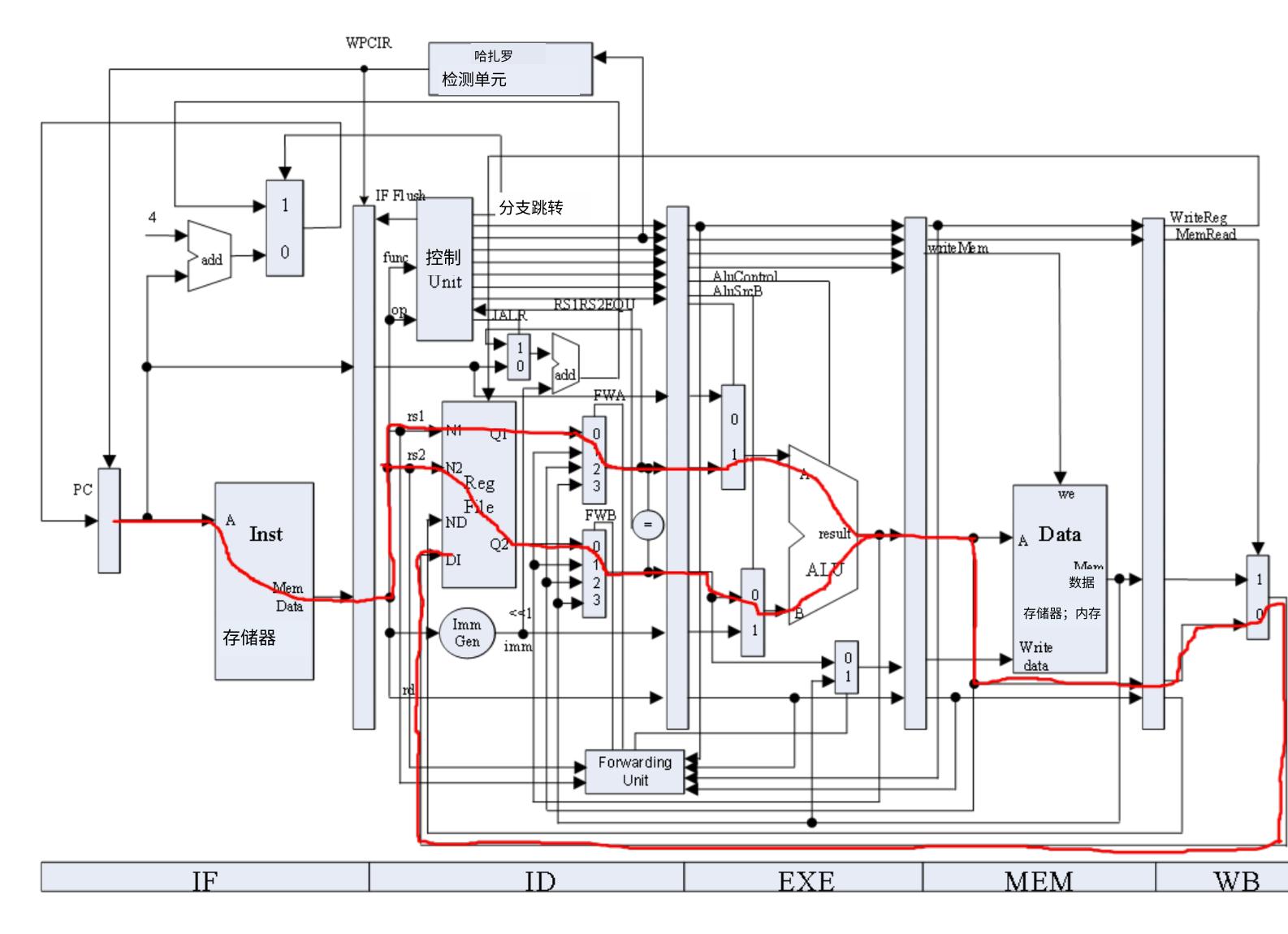
31	25:24	20:19	15:14	12:11	7:6	0
立即数[20:10:1 11]	立即数[19:12]	rd		操作码		

imm[20:10:1 11 19:12]	rd	1101111	JAL
-----------------------	----	---------	-----

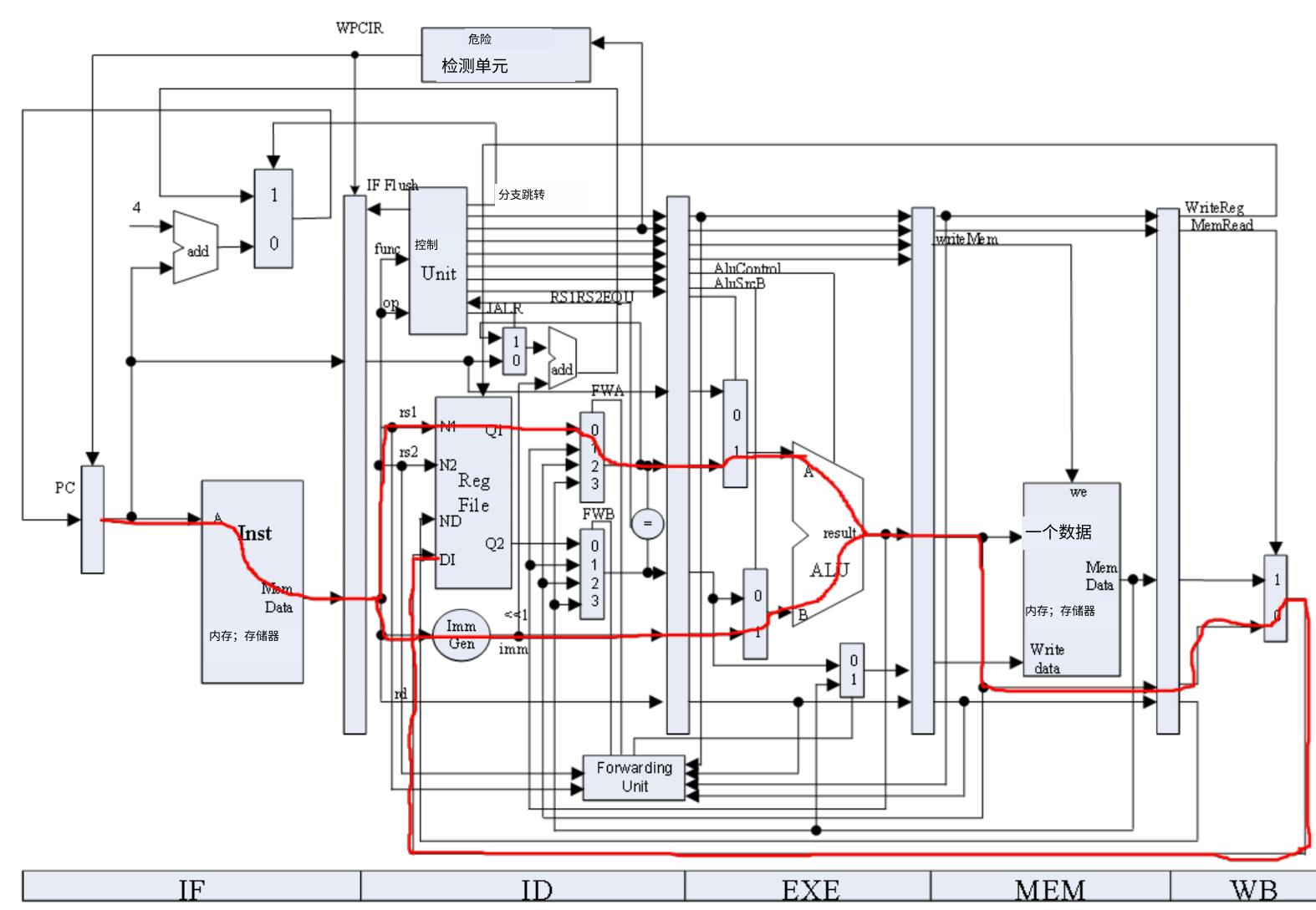
## 架构概述



R型

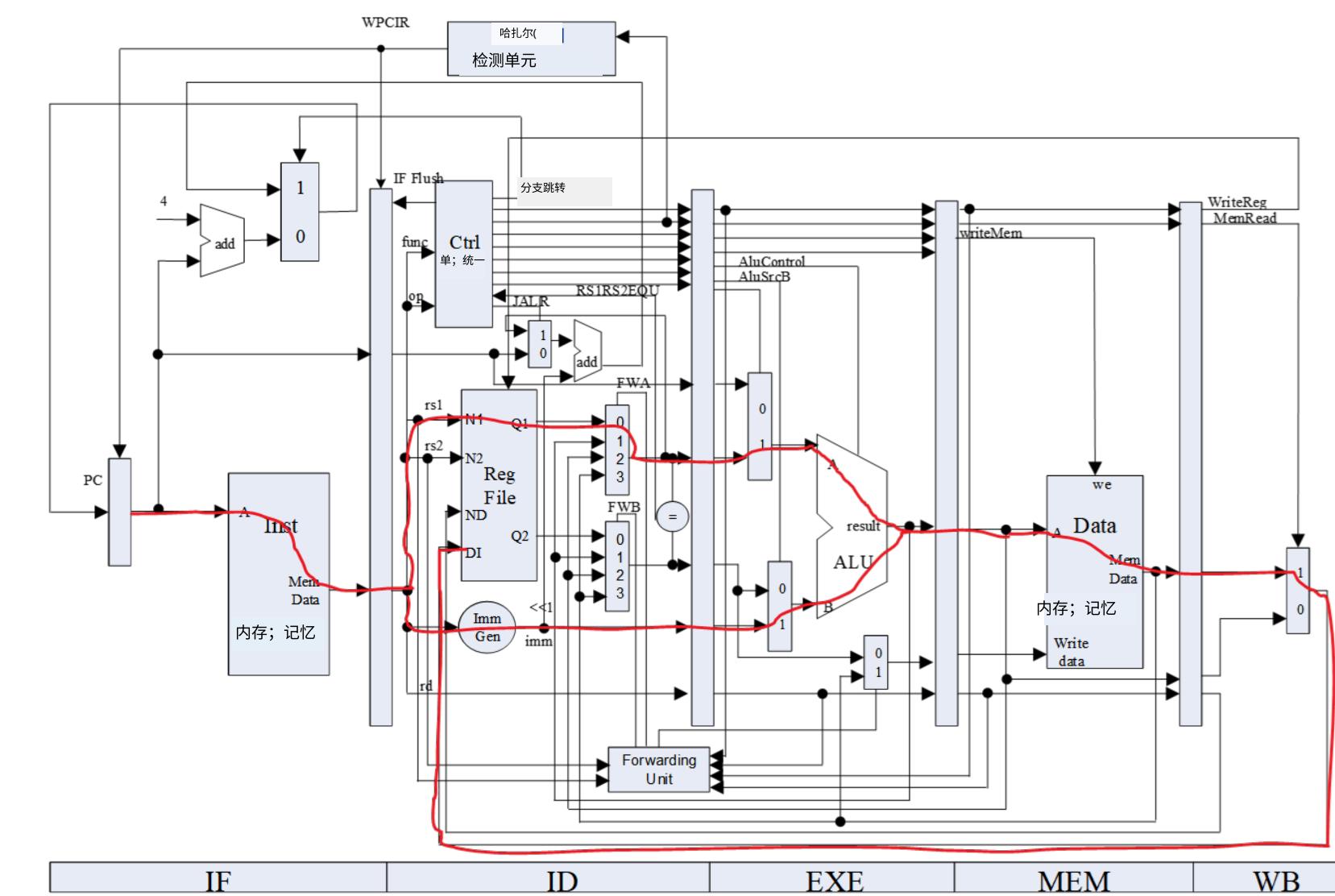


# I型 算术运算 与逻辑



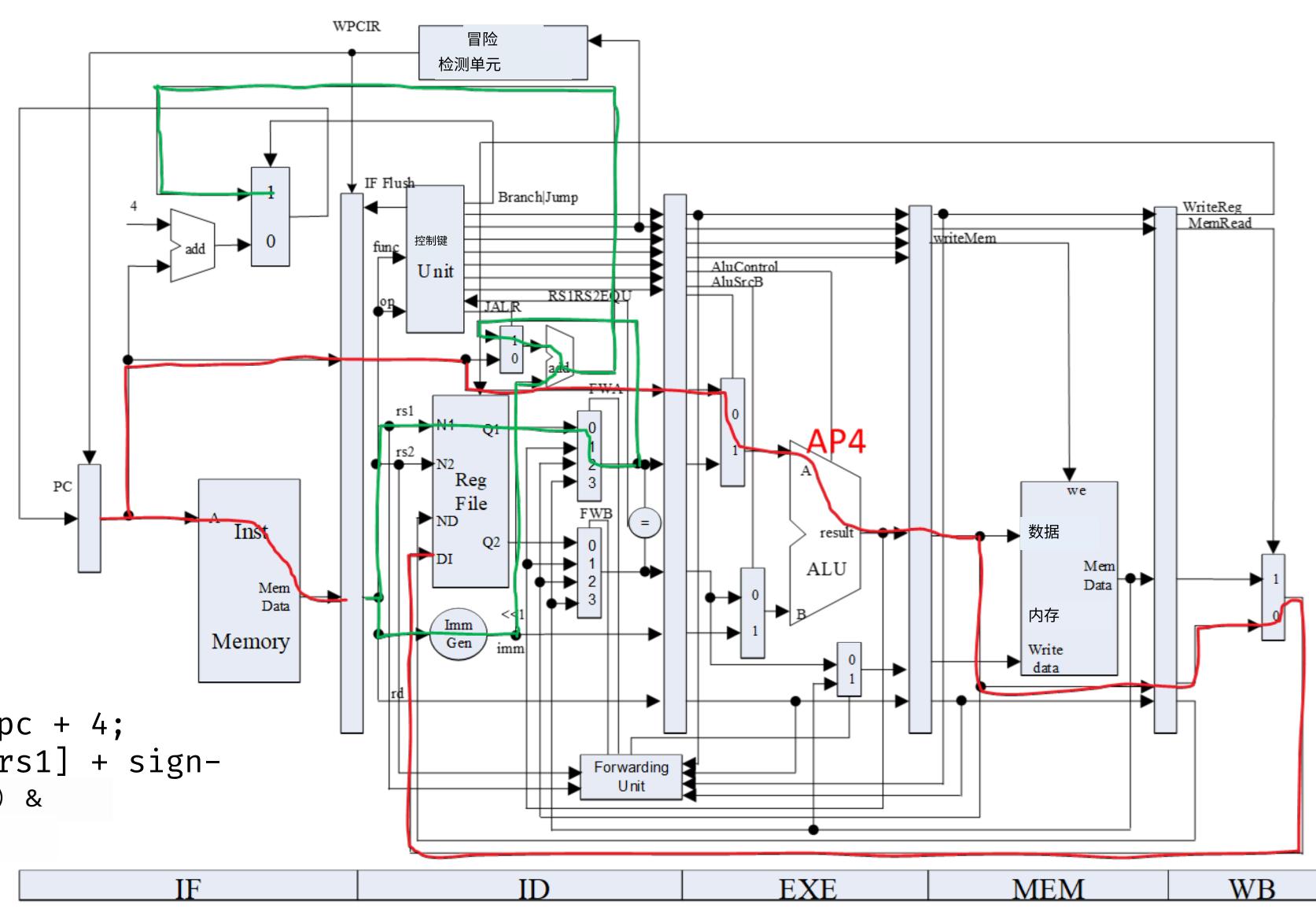
I型

加载；负载



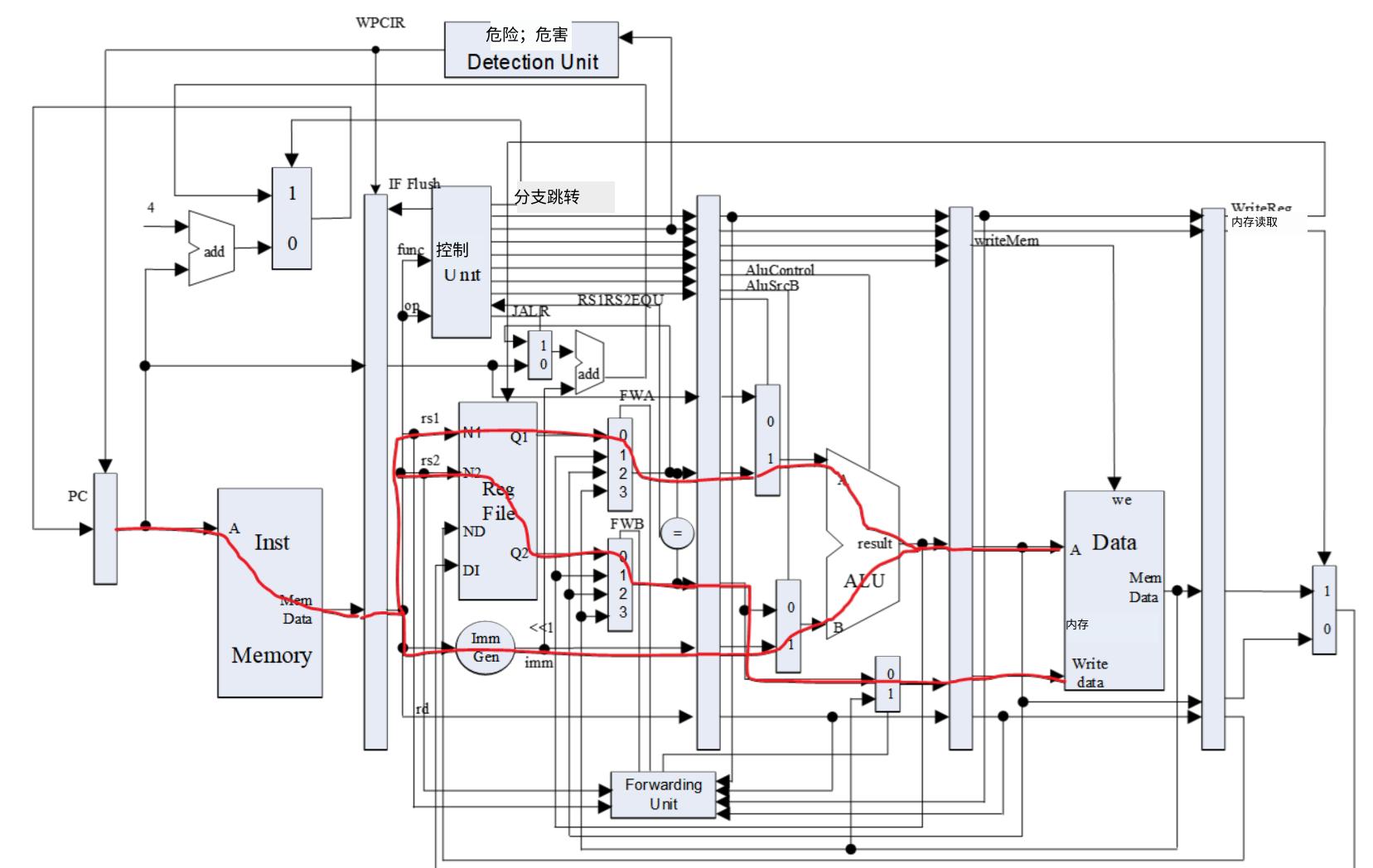
I型

跳转并链接寄存器

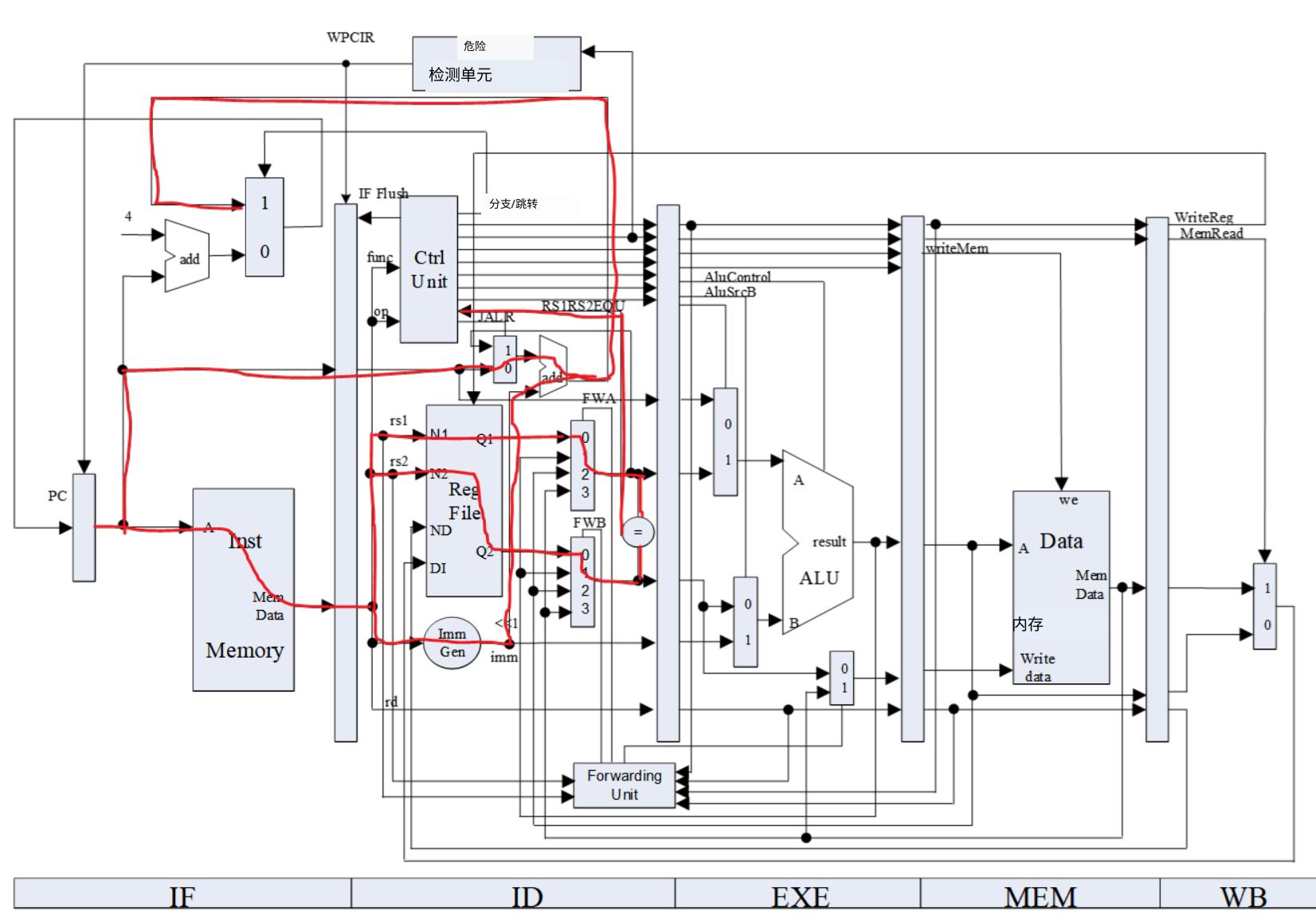


1.  $x[rd] = pc + 4;$
2.  $pc = (x[rs1] + \text{sign-扩展(偏移量)}) \& 0xffff\_ffff$

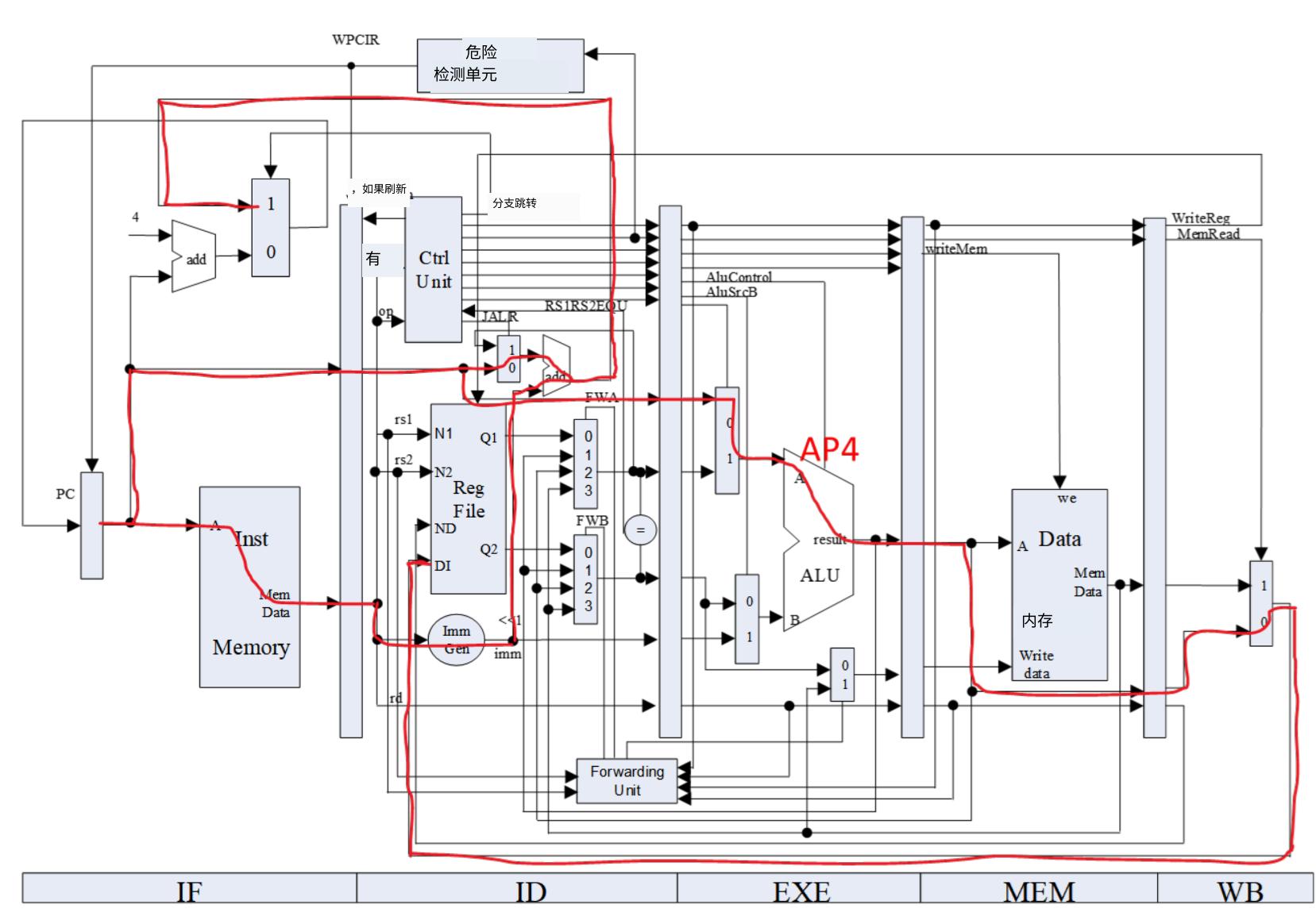
S型



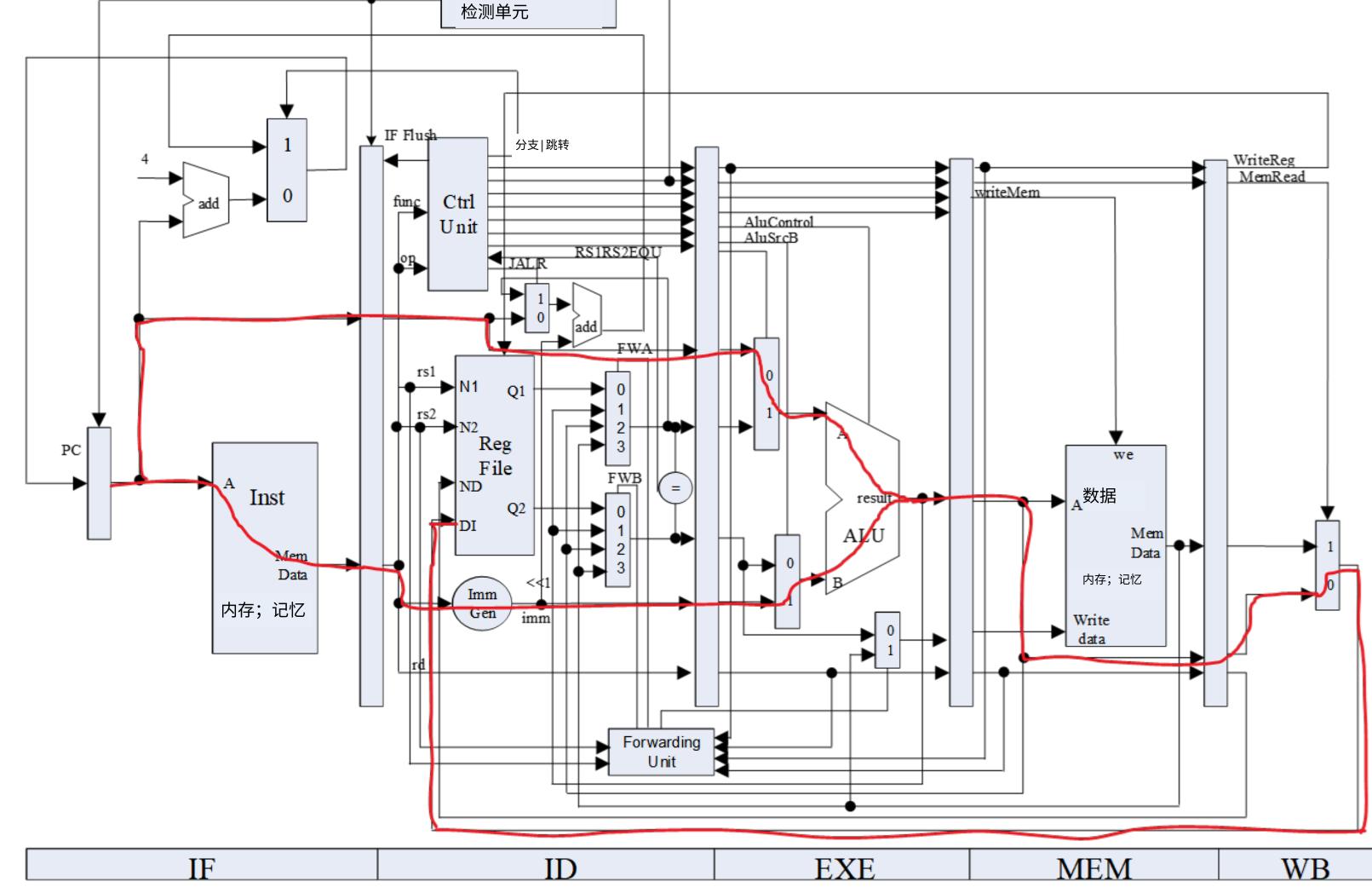
B型



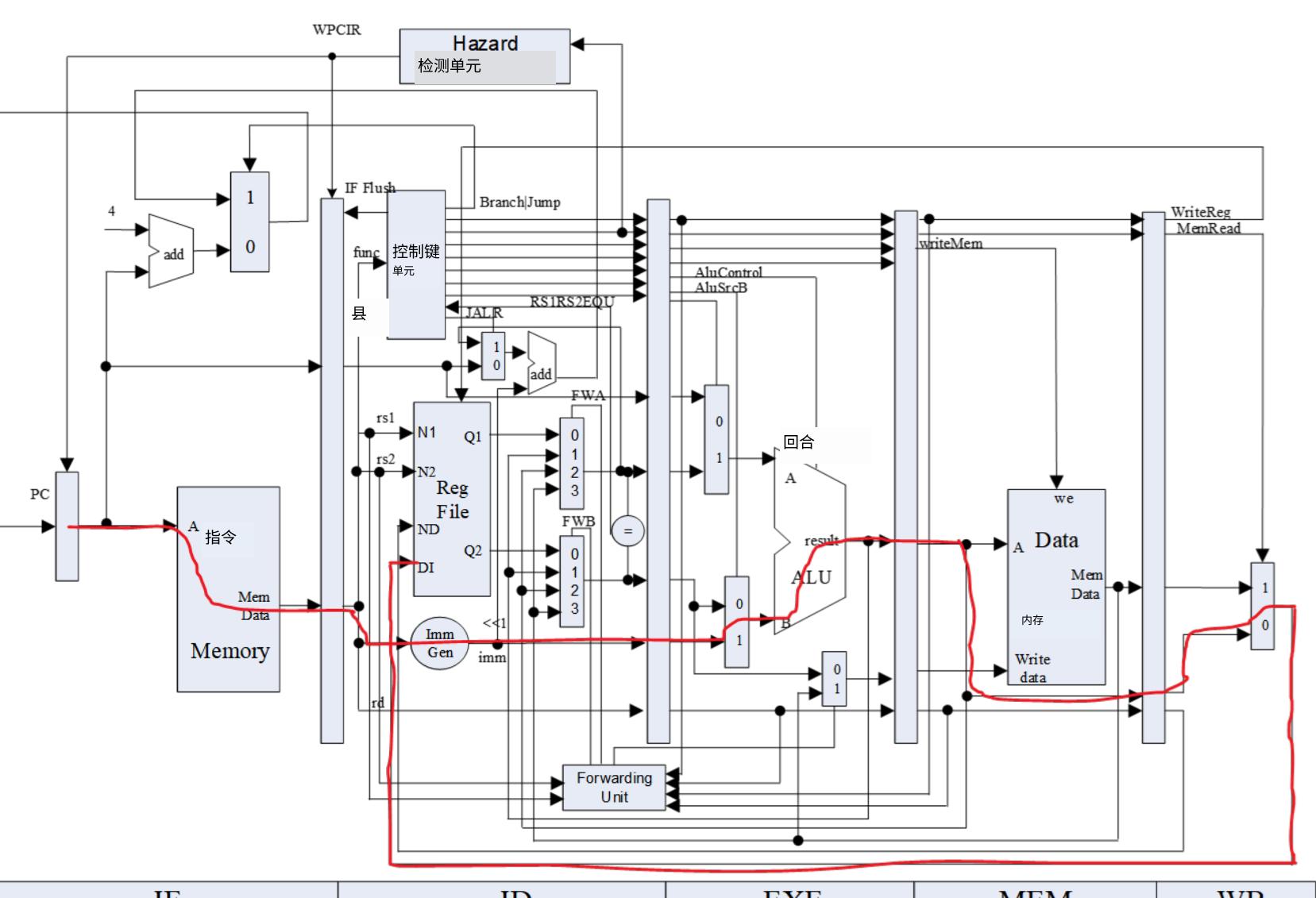
J型



U型  
auipc

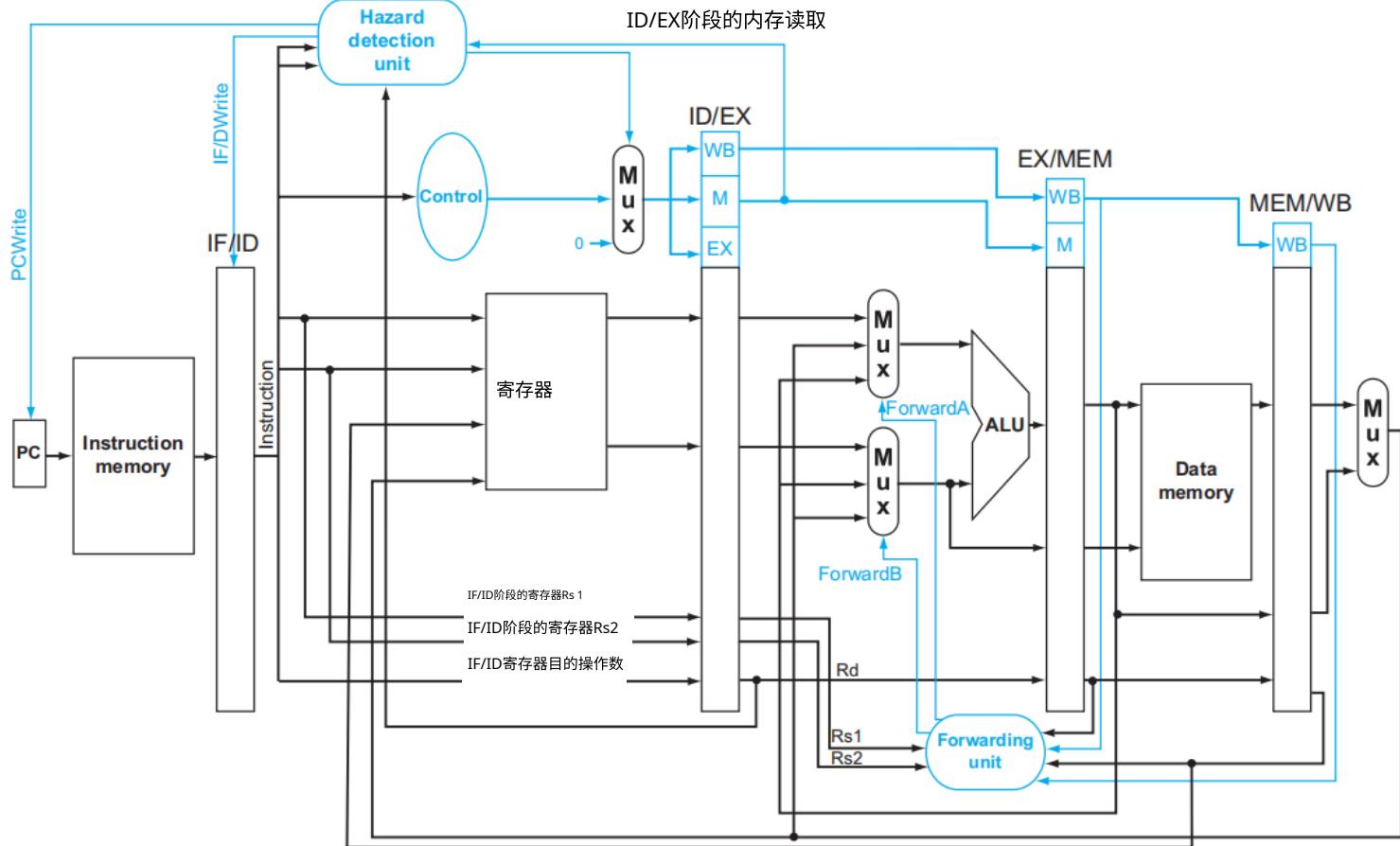


U型  
他



## 数据冒险与转发

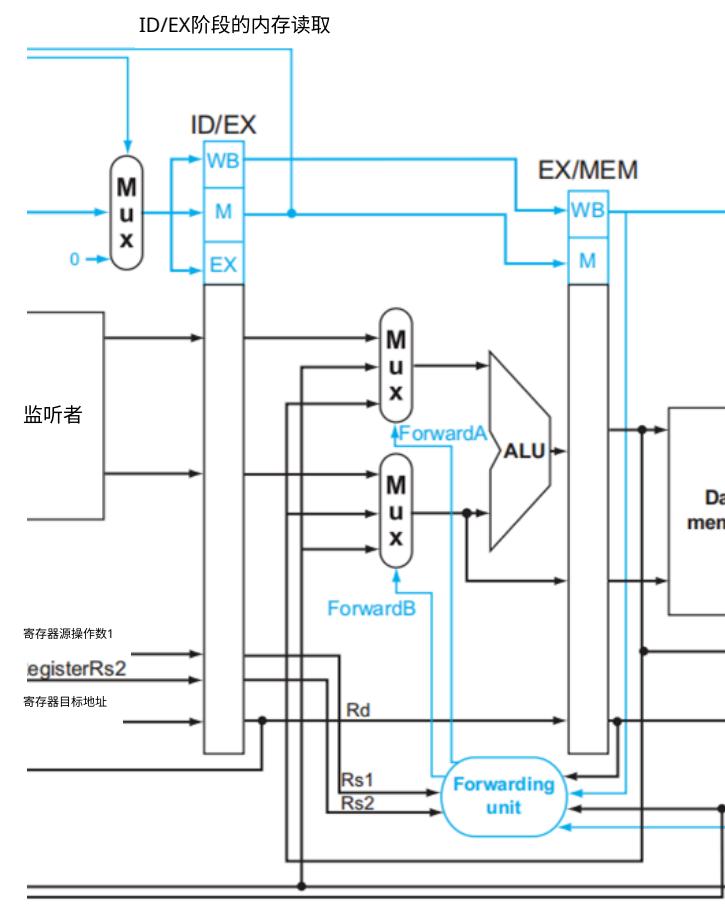
# 数据冒险与转发



## 数据冒险与转发

如果 (EX/MEM 寄存器写使能  
并且 (EX/MEM. 寄存器目标地址 ≠ 0)  
并且 (EX/MEM 寄存器目标地址 = ID/EX 寄存器源地址 1))  
前向信号 A = 10

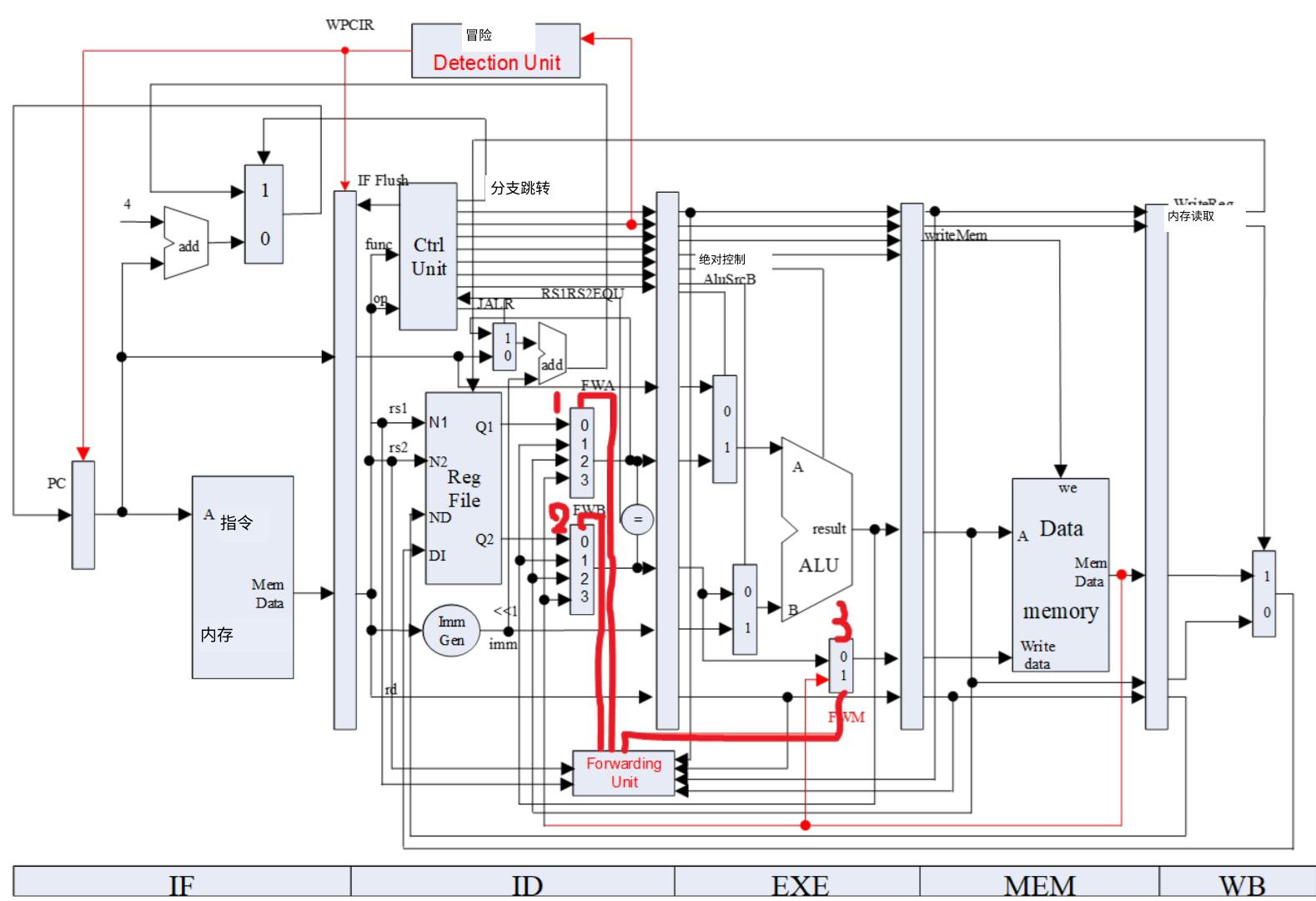
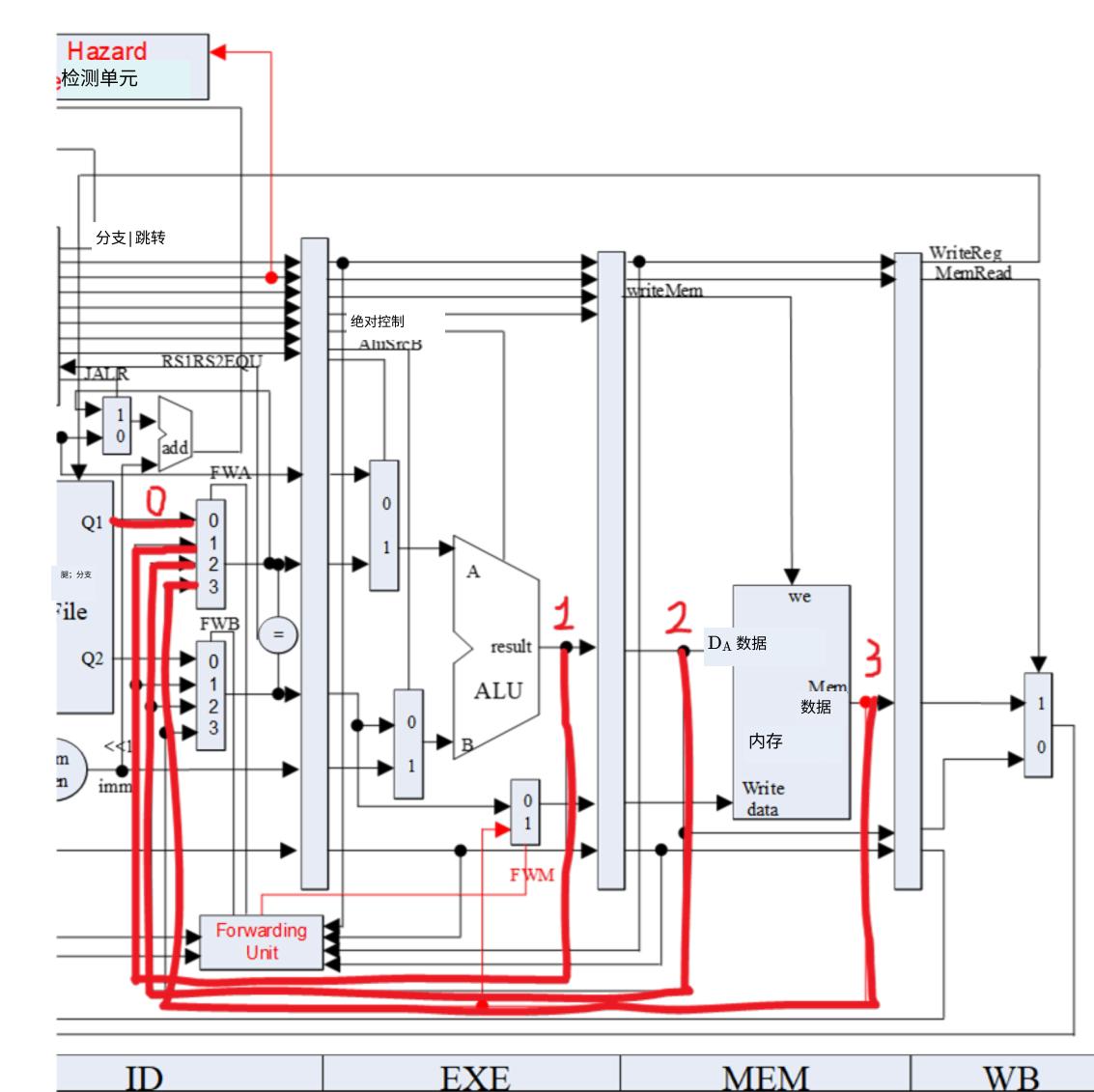
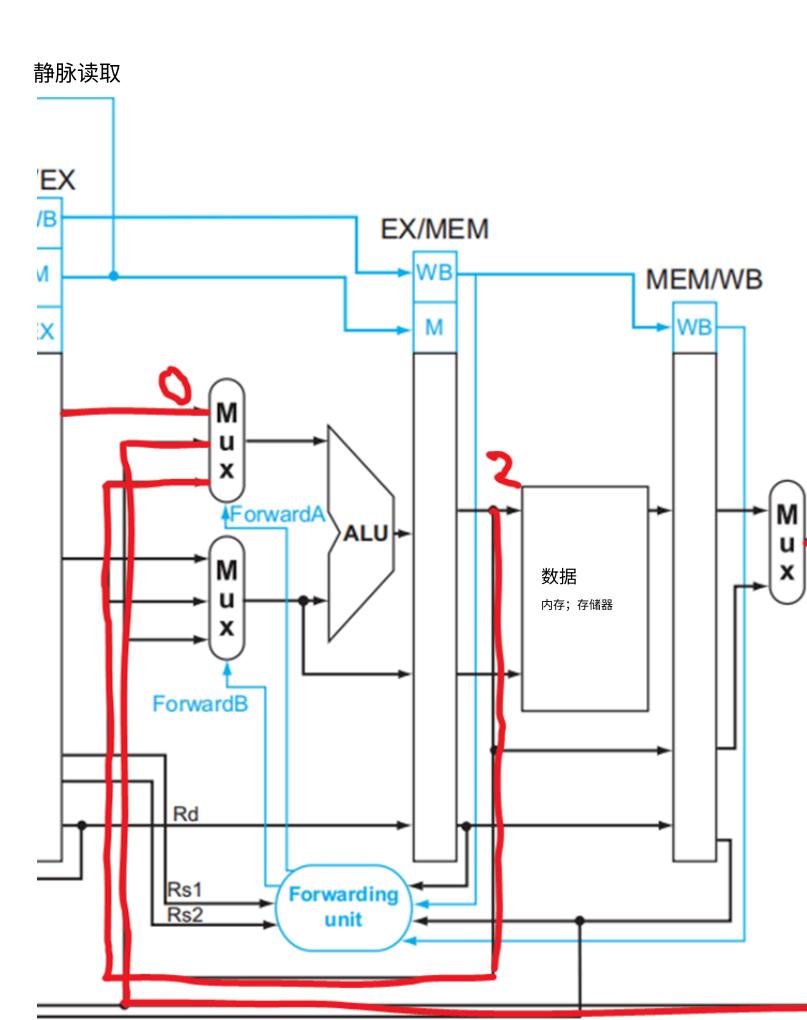
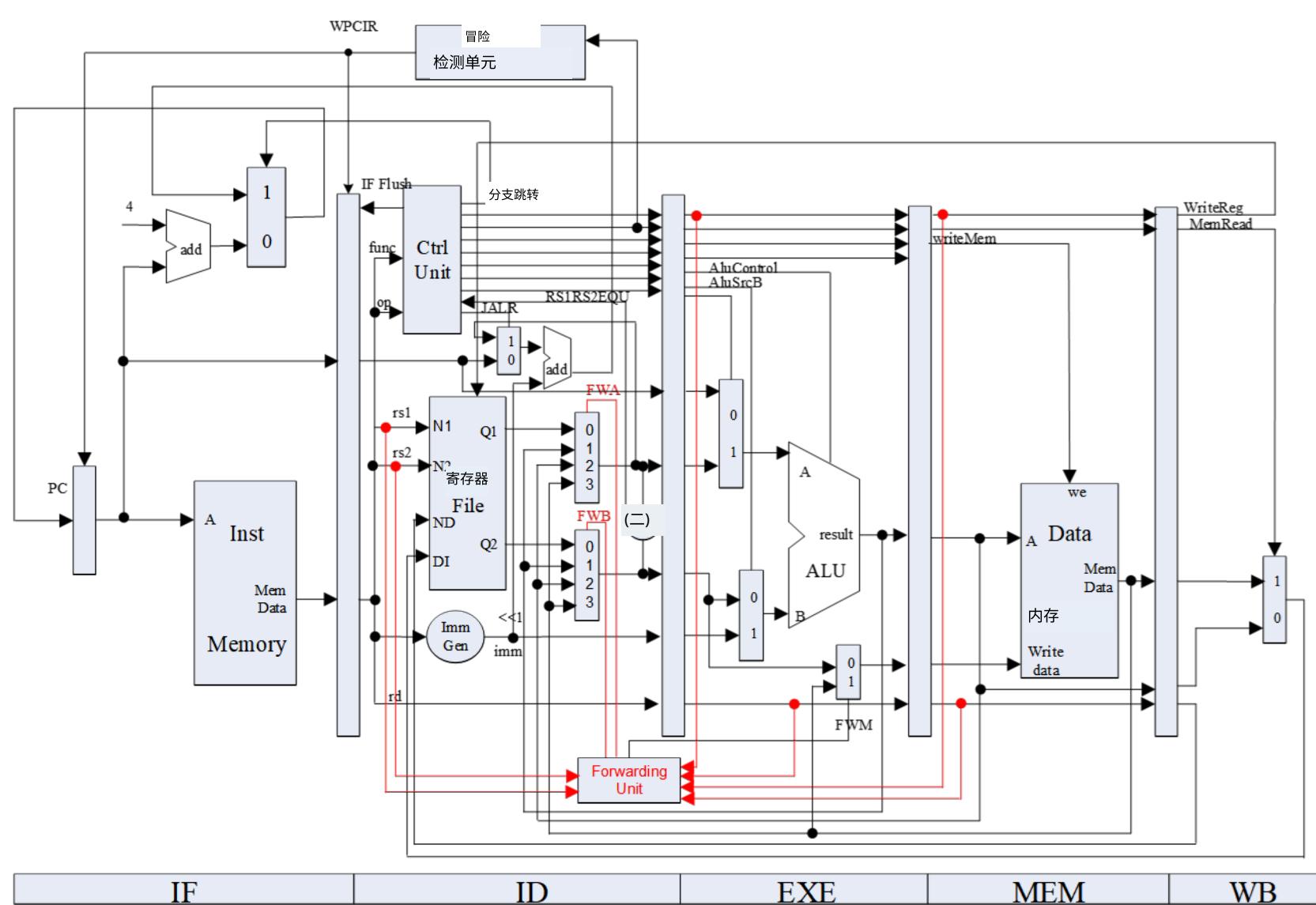
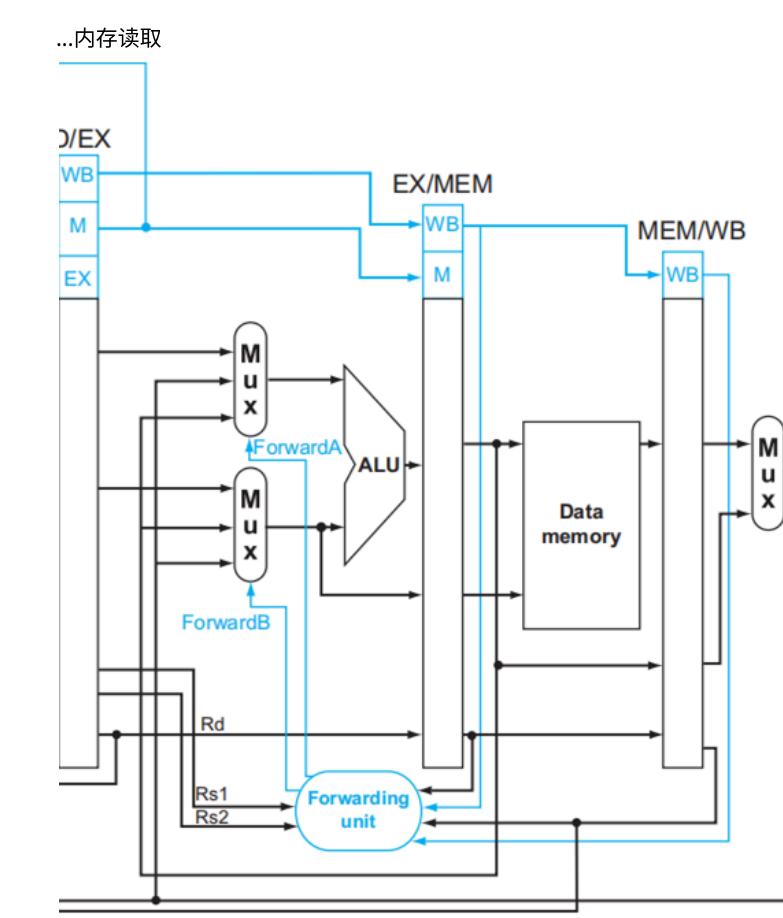
如果 (EX/MEM 寄存器写使能  
并且 (EX/MEM 寄存器目标地址 ≠ 0)  
并且 (EX/MEM 寄存器目标地址 = ID/EX 寄存器源地址 2))  
前向信号 B = 10



## 数据冒险与转发

如果 (MEM/WB. 寄存器写使能  
且 (MEM/WB. 寄存器目标地址 ≠ 0)  
且 非(EX/MEM. 寄存器写使能 且 (EX/MEM. 寄存器目标地址 ≠ 0))  
且 (EX/MEM. 寄存器目标地址 = ID/EX. 源寄存器1地址))  
并且 (MEM/WB 寄存器目标地址 = ID/EX 寄存器源地址)  
1) 前向信号 A = 01

如果 (MEM/WB. 寄存器写使能  
并且 (MEM/WB 寄存器目标地址 ≠ 0)  
并且 非(EX/MEM 寄存器写使能 并且 (EX/MEM 寄存器目标地址 ≠ 0))  
并且 (EX/MEM 寄存器目标地址 = ID/EX 寄存器源地址 2))  
并且 (MEM/WB 寄存器目标地址 = ID/EX 寄存器源地址 2)) 前向信号 B = 01



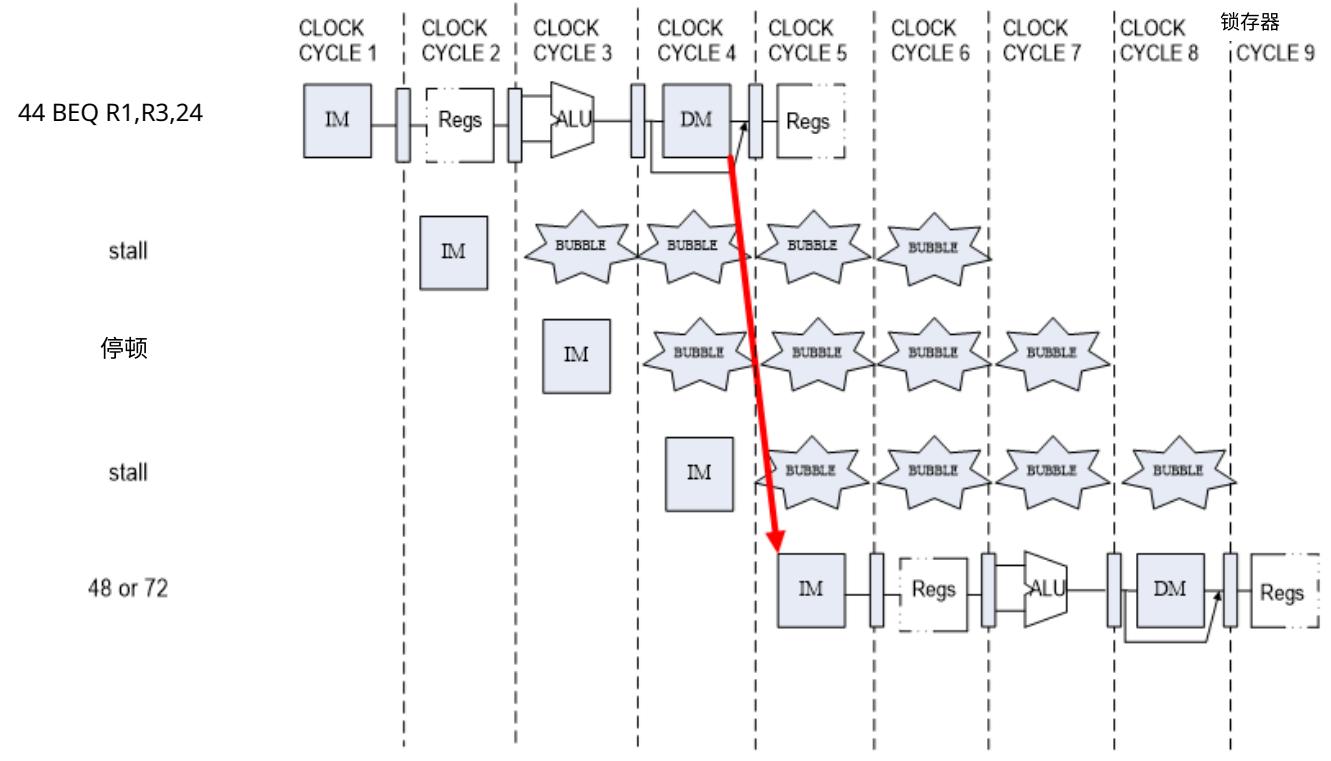
## 数据冒险与转发 - 停顿

- 程序计数器使能 -> 禁用
- 指令获取/指令译码停顿
- 指令译码/执行阶段刷新

## 控制冒险

# 控制冒险 & 分支预测

•冻结或刷新流水线 → 效率低下

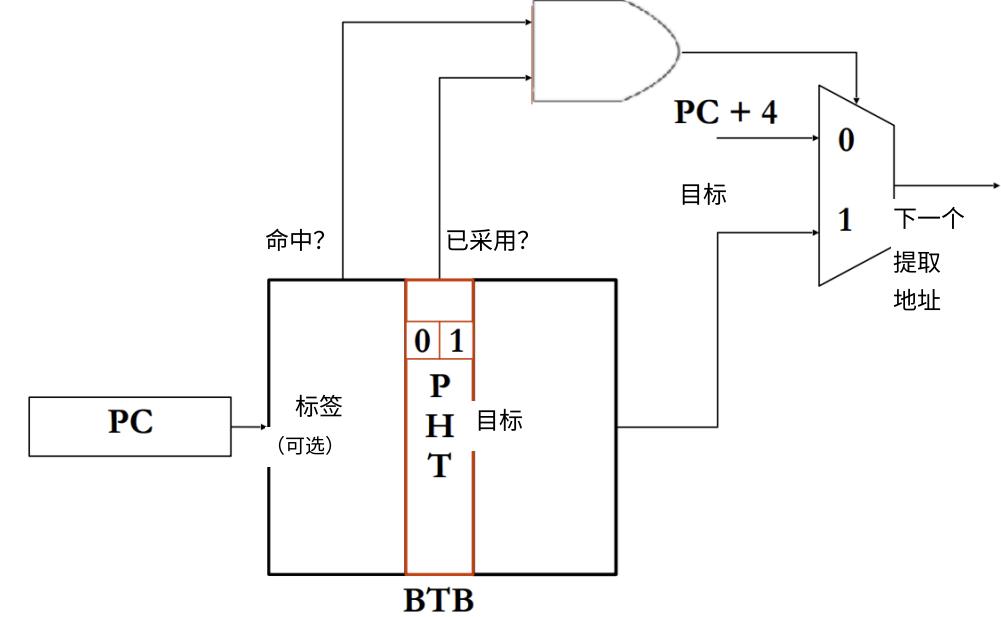


## 控制冒险

分支预测!

预测方向：是否被采用

预测目标：若执行，目标地址是？

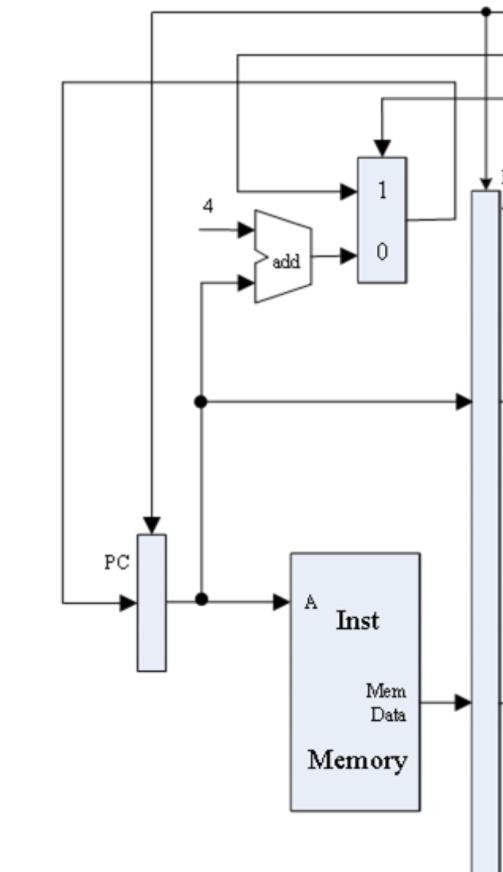


## 分支预测

● 预测不跳转

○ 预测方向：不跳转

○ 预测目标

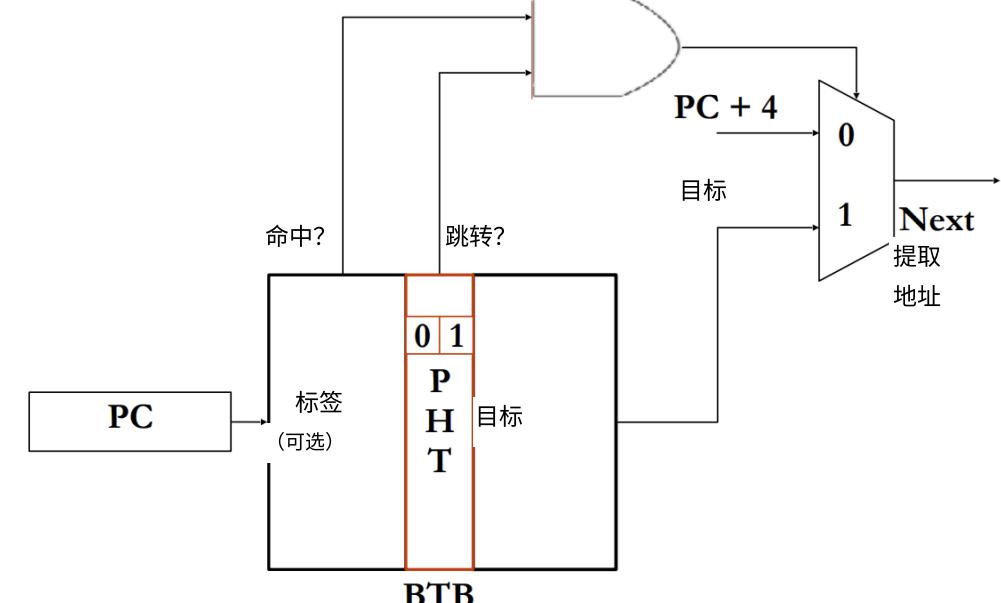


## 分支预测

• 预测是否跳转？

• 预测方向：跳转

• 预测目标：跳转地址

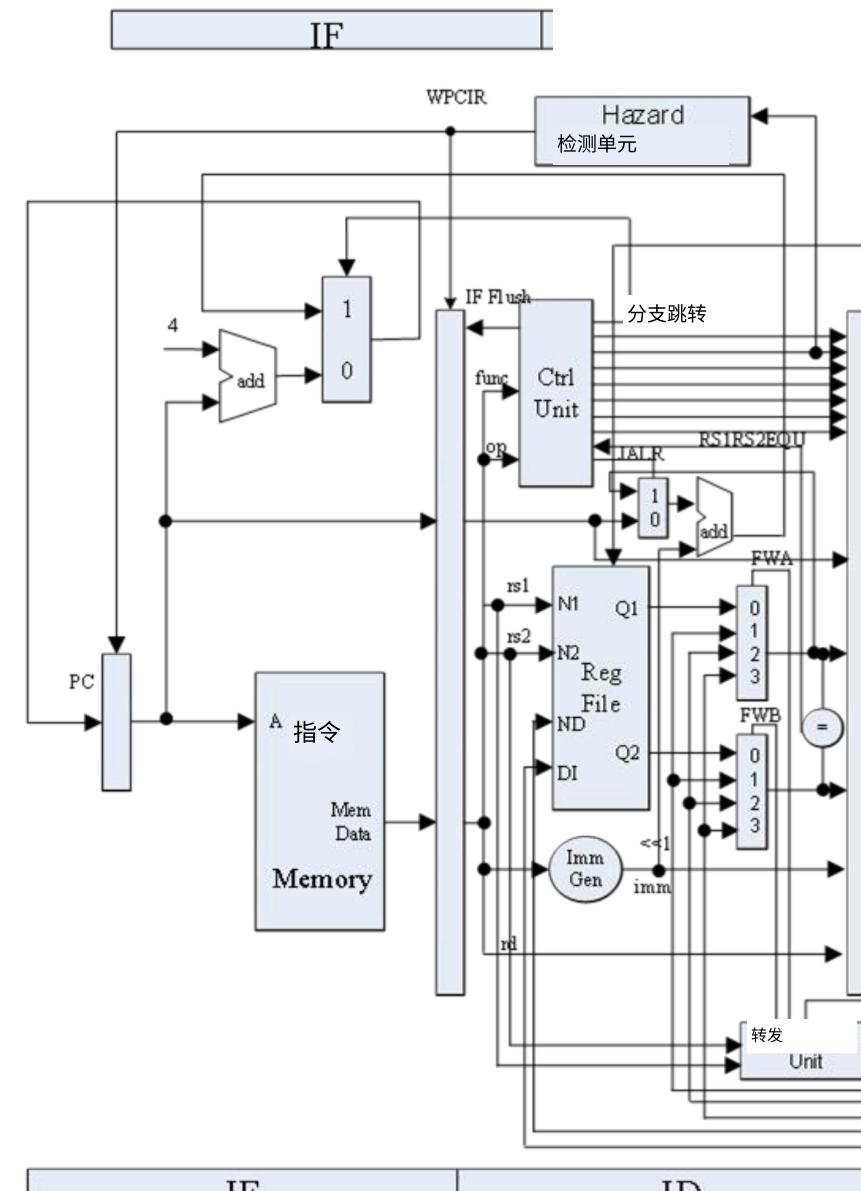


## 分支预测

● 预测不跳转

• 回滚 → 下一条指令不正确

刷新指令获取/指令译码 (IF/ID) 流水线寄存器



## 冒险检测单元

模块 冒险检测单元(

输入 时钟信号,

输入Branch ID、rsluse ID、rs2use\_ID

| 输入[1:0] hazard optype ID, ID阶段: ALU/加载/存储?

输入[4:0] rd\_EXE、rd\_MEM、rs1\_ID、rs2\_ID、rs2\_EXE, 输出

PC\_EN\_IF、reg\_FD\_EN、reg\_FD\_stall、reg\_FD\_flush、

reg\_DE\_EN、reg\_DE\_flush、reg\_EM\_EN、reg\_EM\_flush、

reg\_MW\_EN, 输出forward\_ctrl\_ls, 输出[1:0] forward\_ctrl\_A、

forward\_ctrl\_B;

参数hazard\_optype\_ALU = 2'd1;

参数hazard\_optype\_LOAD = 2'd2;

参数hazard\_optype\_STORE = 2'd3

# 冒险检测单元

## 危险检测单元 - 转发

```

寄存器[1:0] 危险操作类型_EXE, 危险操作类型_MEM;
始终@(时钟信号的上升沿) 开始
    危险操作类型_MEM <= 危险操作类型_EXE & {2{~寄存器_EM_刷新}};
    危险操作类型_EXE <= 危险操作类型_ID & {2{~寄存器_DE_刷新}};
结束

参数hazard_optype_ALU = 2'd1;
参数hazard_optype_LOAD = 2'd2;
参数hazard_optype_STORE = 2'd3;

线网rs1_forward_1 = ... && 执行阶段的危险操作类型 == 算术逻辑单元操作类型;
线网rs1_forward_stall = ... && 执行阶段的危险操作类型 == 加载操作类型 &&
指令译码阶段的危险操作类型 != 存储操作类型;
线 rs1_forward_2 = ... 且 危险操作类型_MEM 等于 危险操作类型_ALU;
线 rs1_forward_3 = ... 且 危险操作类型_MEM 等于 危险操作类型_LOAD;

赋值 前向控制_A = ...
赋值 前向控制_B = ...
赋值 前向控制_ls ...

```

NO.	指令	地址	标签	ASM	注释
0	00000013	0	——开始:	将立即数0加到x0寄存器, 结果存于x0	
1	00402103	4		从x0寄存器的值加4的地址处加载字数据到x2寄存器	
2	00802203	8		将内存地址为x0加8处的数据加载到寄存器x4	
3	004100b3	C		将寄存器x2和x4的值相加, 结果存入寄存器x1	
4	fffo8093	10		将寄存器x1的值减1, 结果存入寄存器x1	
5	00c02283	14		将内存地址为x0加12处的数据加载到寄存器x5	
6	01002303	18		将内存地址为x0加16处的数据加载到寄存器x6	
7	01402383	1C		将内存地址为x0加20处的数据加载到x7寄存器	
8	402200b3	20		用x4寄存器的值减去x2寄存器的值, 结果存入x1寄存器	
9	002270b3	24		对x4和x2寄存器的值进行按位或运算, 结果存入x1寄存器	
10	002260b3	28		或 x1, x4, x2	
11	002240b3	2C		对x4和x2寄存器的值进行按位异或运算, 结果存入x1寄存器	
12	002210b3	30		将x4寄存器的值逻辑左移x2寄存器指定的位数, 结果存入x1寄存器	
13	002220b3	34		x1 = (x4 < x2) ? 1 : 0	
14	004120b3	38		x1 = (x2 < x4) ? 1 : 0	

## 冒险检测单元 - 冒险

```

赋值 PC_EN_IF = ...;
赋值 reg_FD_stall = ...;
赋值 reg_FD_flush = ...;
赋值 reg_DE_flush = ...

```

NO.	指令	地址	标签	ASM	注释
15	002350b3	3C		逻辑右移, 将x0寄存器的值右移x2位后存入x1	
16	402350b3	40		将x6逻辑右移x2位后存入x1	
17	4023d0b3	44		将x7逻辑右移x2位后存入x1	
18	007330b3	48		若x6无符号小于x7, 则将1存入x1, 否则存入0	
19	0063b0b3	4C		若x7无符号小于x6, 则将1存入x1, 否则存入0	
20	00000033	50		将x0与x0相加, 结果存入x0	
21	ffd50093	54		将x10的值减3后存入x1	
22	00f27093	58		将x4与15按位与的结果存入x1	
23	00f26093	5C		将x4与15按位或的结果存入x1	
24	00f24093	60		将x4与15按位异或的结果存入x1	
25	00f22093	64		若x4小于15, 则将1存入x1, 否则存入0	
26	00121093	68		将x4左移1位的结果存入x1	
27	00225093	6C		将x4逻辑右移2位存入x1	
28	40c35093	70		将x6算术右移12位存入x1	
29	fff33093	74		若x6无符号小于-1, 则将1存入x1, 否则存入0	

NO.	指令	地址	标签	ASM	注释
30	fff3b093	78		无符号小于立即数比较, 将x7与-1比较, 结果存于x1	
31	00520863	7C		若x4等于x5, 则跳转到标签label0	
32	00420663	80		若x4等于x4, 则跳转到标签label0	
33	00000013	84		将x0加上立即数0, 结果存于x0	
34	00000013	88		将x0加上立即数0, 结果存于x0	
35	00421863	8C	标签0:	若x4不等于x4, 则跳转到标签1	
36	00521663	90		若x4不等于x5, 则跳转到标签1	
37	00000013	94		将x0的值加0后存回x0	
38	00000013	98		将x0的值加0后存回x0	
39	0042c863	9C	标签1:	如果x5小于x4, 则跳转到标签label2	
40	00524663	A0		如果x4小于x5, 则跳转到标签label2	
41	00000013	A4		将x0寄存器的值加0	
42	00000013	A8		将x0寄存器的值加0	
43	00736863	AC	标签label2:	如果无符号的x6小于x7, 则跳转到标签label3	
44	0063e663	B0		分支小于无符号数 x7,x6,标签3	

NO.	指令	地址	标签	ASM	注释
45	00000013	B4		将立即数0加到x0寄存器, 结果存于x0	
46	00000013	B8		将立即数0加到x0寄存器, 结果存于x0	
47	00525863	BC	标签3:	若x4寄存器的值大于等于x5寄存器的值, 则跳转到标签4	
48	0042d663	C0		如果x4大于或等于x4, 则跳转到标签label4	
49	00000013	C4		将x0的值加0	
50	00000013	C8		将x0的值加0	
51	0063f863	CC	标签label4:	如果无符号x7大于或等于x6, 则跳转到标签label5	
52	00737663	D0		如果无符号x6大于或等于x7, 则跳转到标签label5	
53	00000013	D4		将立即数0加到x0寄存器, 源操作数为x0寄存器和立即数0	
54	00000013	D8		将立即数0加到x0寄存器, 源操作数为x0寄存器和立即数0	
55	00425663	DC	标签5:	若x4寄存器的值大于或等于x4寄存器的值, 则跳转到标签6	
56	00000013	E0		将立即数0加到x0寄存器, 源操作数为x0寄存器和立即数0	
57	00000013	E4		将立即数0加到x0寄存器, 源操作数为x0寄存器和立即数0	
58	000040b7	E8	标签6:	lui x1,4	
59	00c000ef	EC		跳转并链接 x1,12	

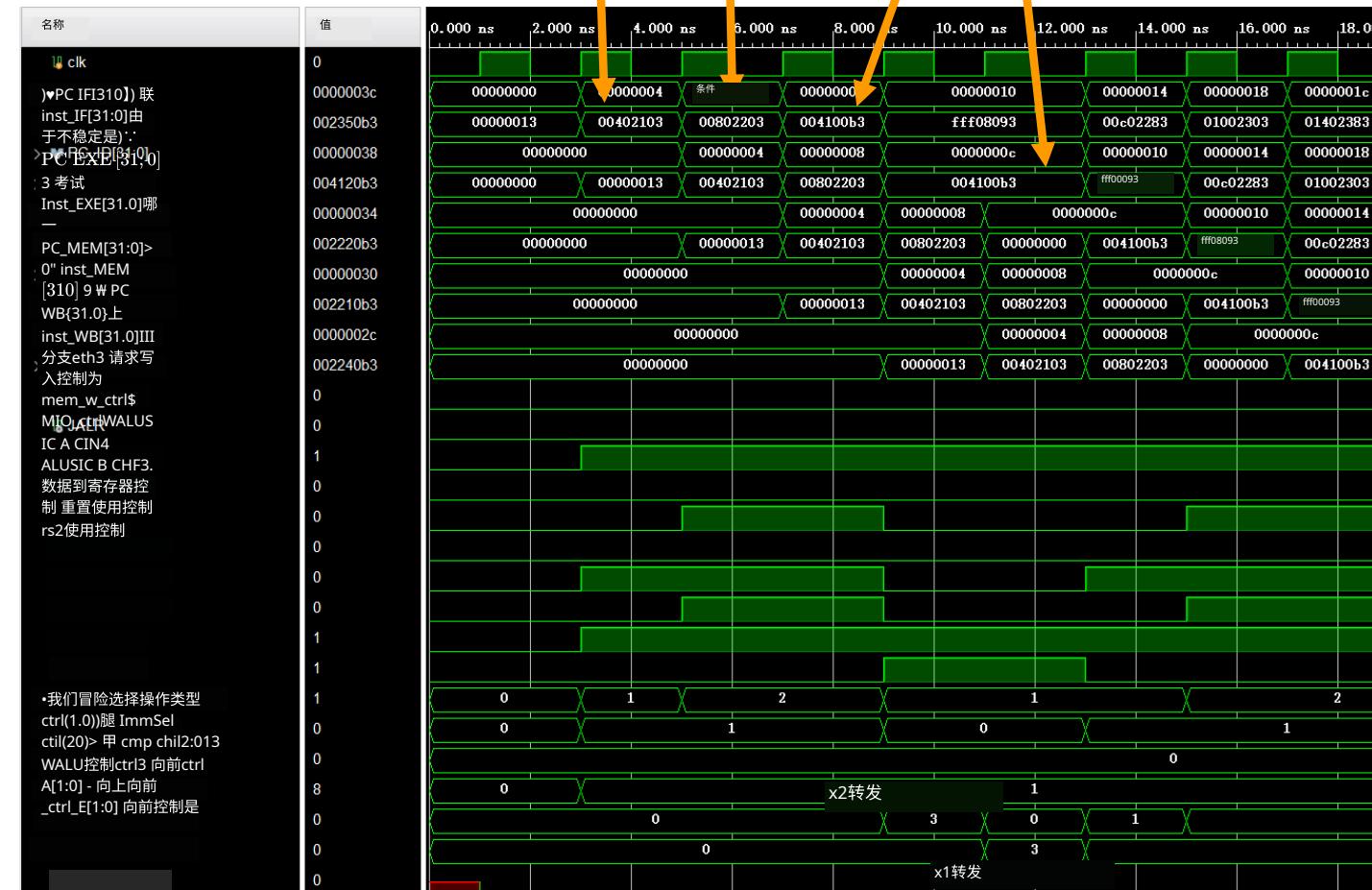
NO.	指令	地址	标签	ASM	注释
60	00000013	F0		立即数加 x0,x0,0	
61	00000013	F4		将立即数0加到x0寄存器, 结果存于x0	
62	01802403	F8		从x0寄存器的值加24的地址处加载一个字到x8寄存器	
63	00802e23	FC		sw x8, 28(x0)	
64	01c02083	100		lw x1, 28(x0)	
65	02801023	104		sh x8, 32(x0)	
66	02002083	108		lw x1, 32(x0)	
67	02800223	10C		sb x8, 36(x0)	
68	02402083	110		lw x1, 36(x0)	
69	01a01083	114		lh x1, 26(x0)	
70	01a05083	118		加载无符号半字 x1, 26(x0)	
71	01b00083	11C		lb x1, 27(x0)	
72	01b04083	120		从x0寄存器的值加27的地址处加载一个无符号字节到x1寄存器	
73	ffff0097	124		将0xffff0左移12位后加到当前PC值, 结果存于x1寄存器	
74	000000e7	128		跳转到链接寄存器, 偏移量为0 (零寄存器)	

NO.	数据	地址	
0	000080BF	0	
1	00000008	4	
2	00000010	8	
3	00000014	C	
4	FFFFF0000	10	
5	0FFF0000	14	
6	FF000F0F	18	
7	F0F0F0F0	1C	
8	00000000	20	
9	00000000	24	
10	00000000	28	
11	00000000	2C	
12	00000000	30	
13	00000000	34	
14	00000000	38	
15	00000000	3C	

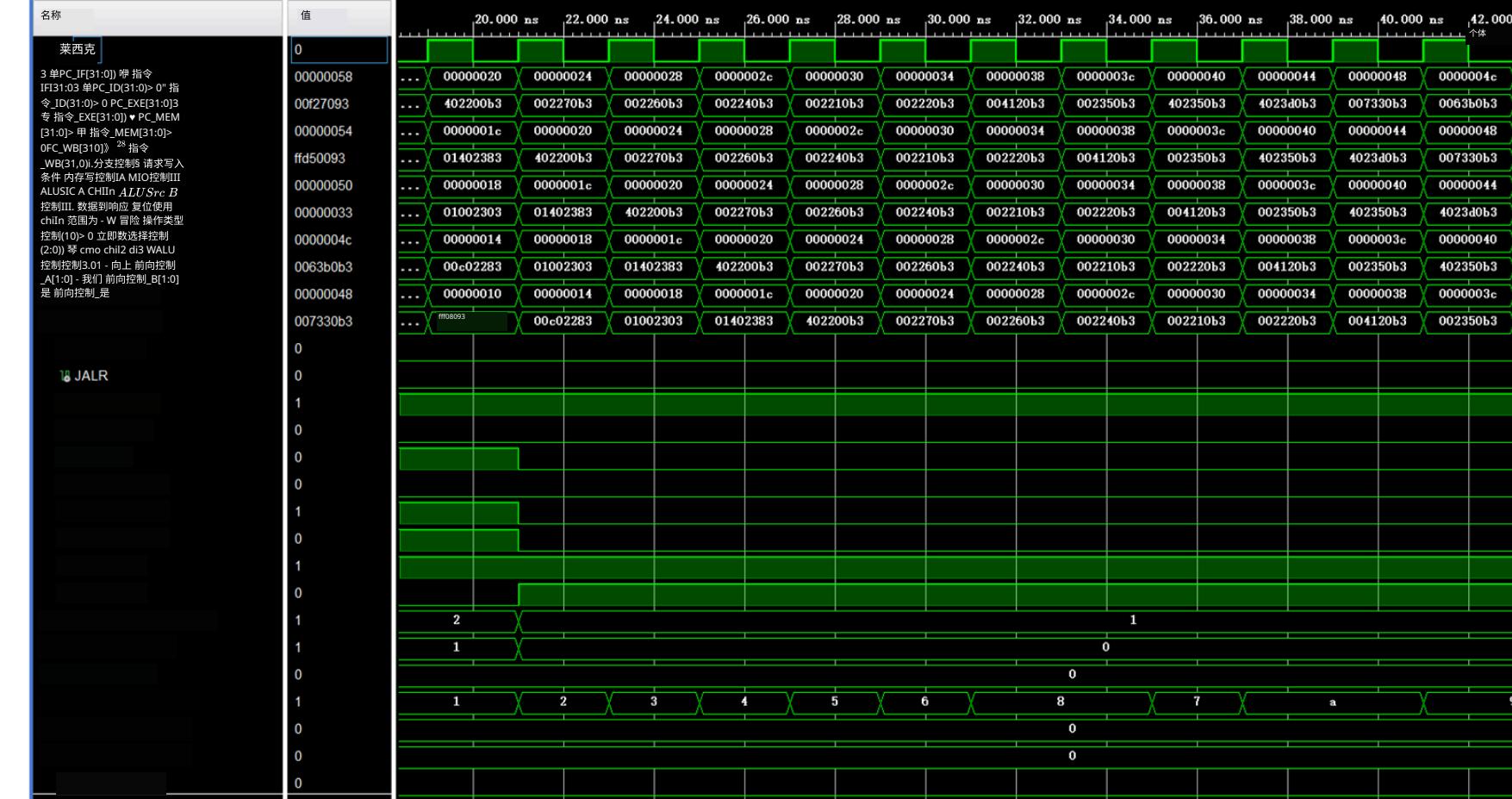
将内存地址为x0加4处的字加载到x4寄存器

将内存地址为x0加4处的字存储到x2寄存器指向的物理和逻辑的地址,并刷新x4寄存器

## 模拟 (1)



## 模拟 (2)

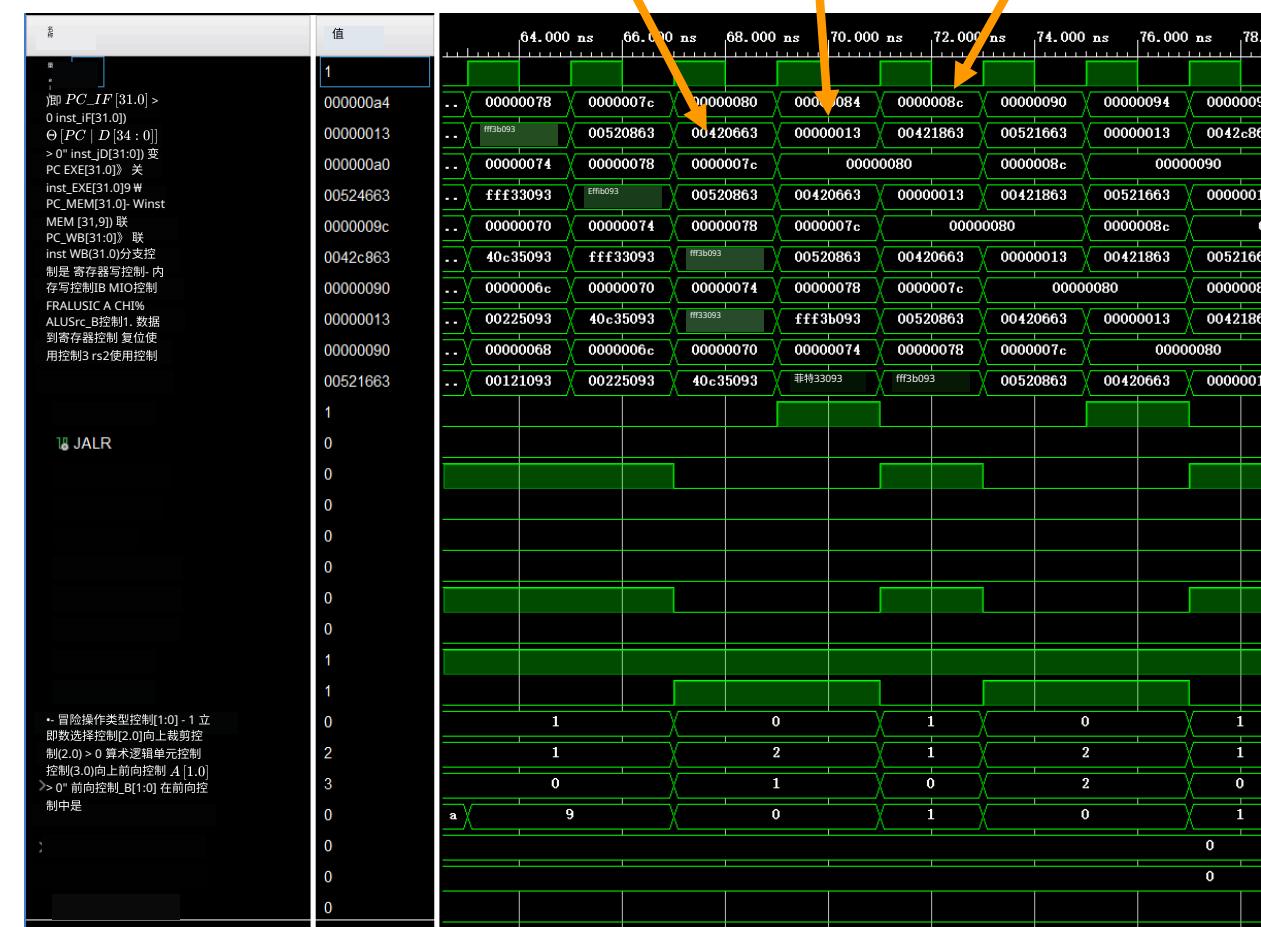


若x4等于x4，则跳转到标签0 跳转到标签0并刷新IF/ID

## 模拟 (3)

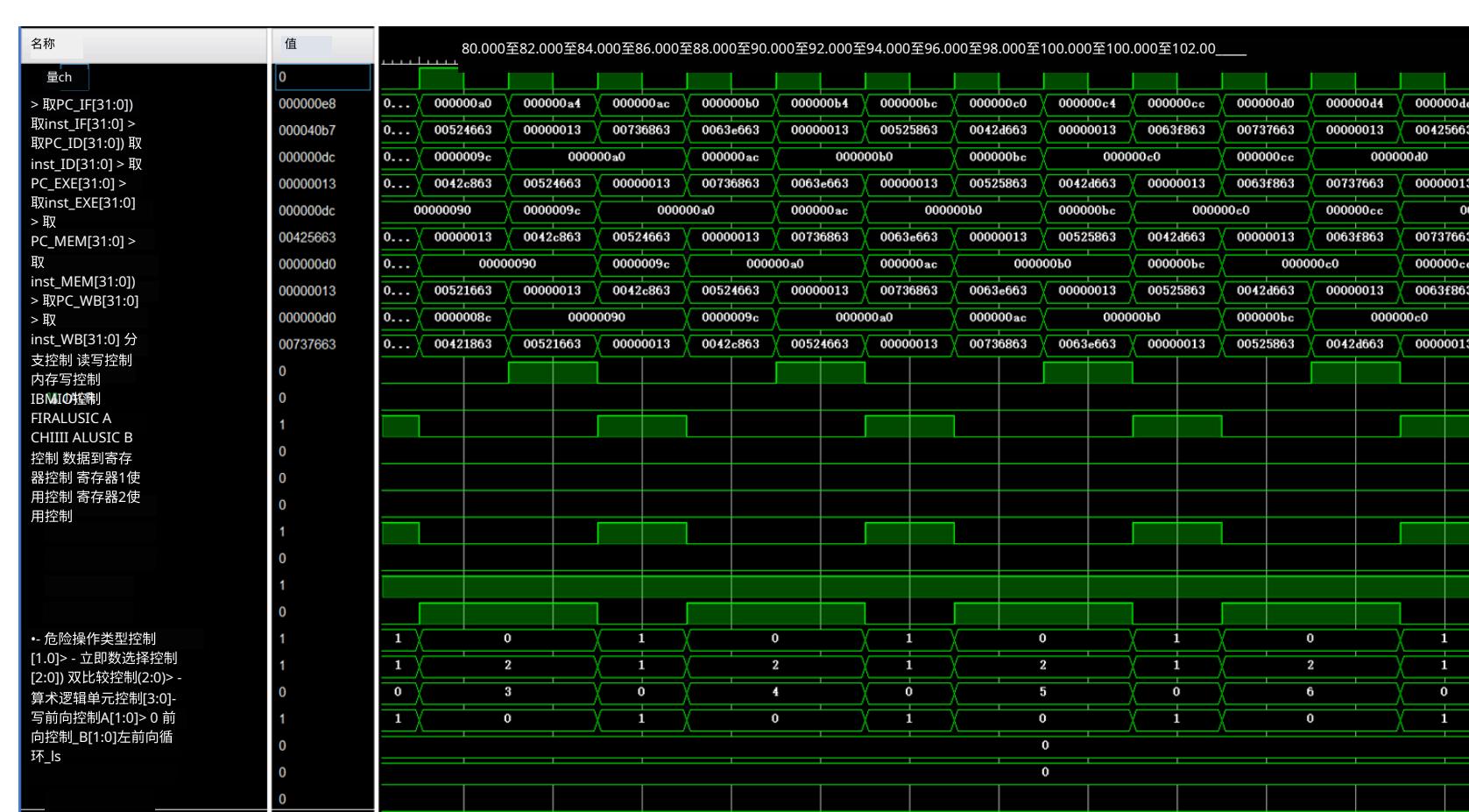


## 模拟 (4)

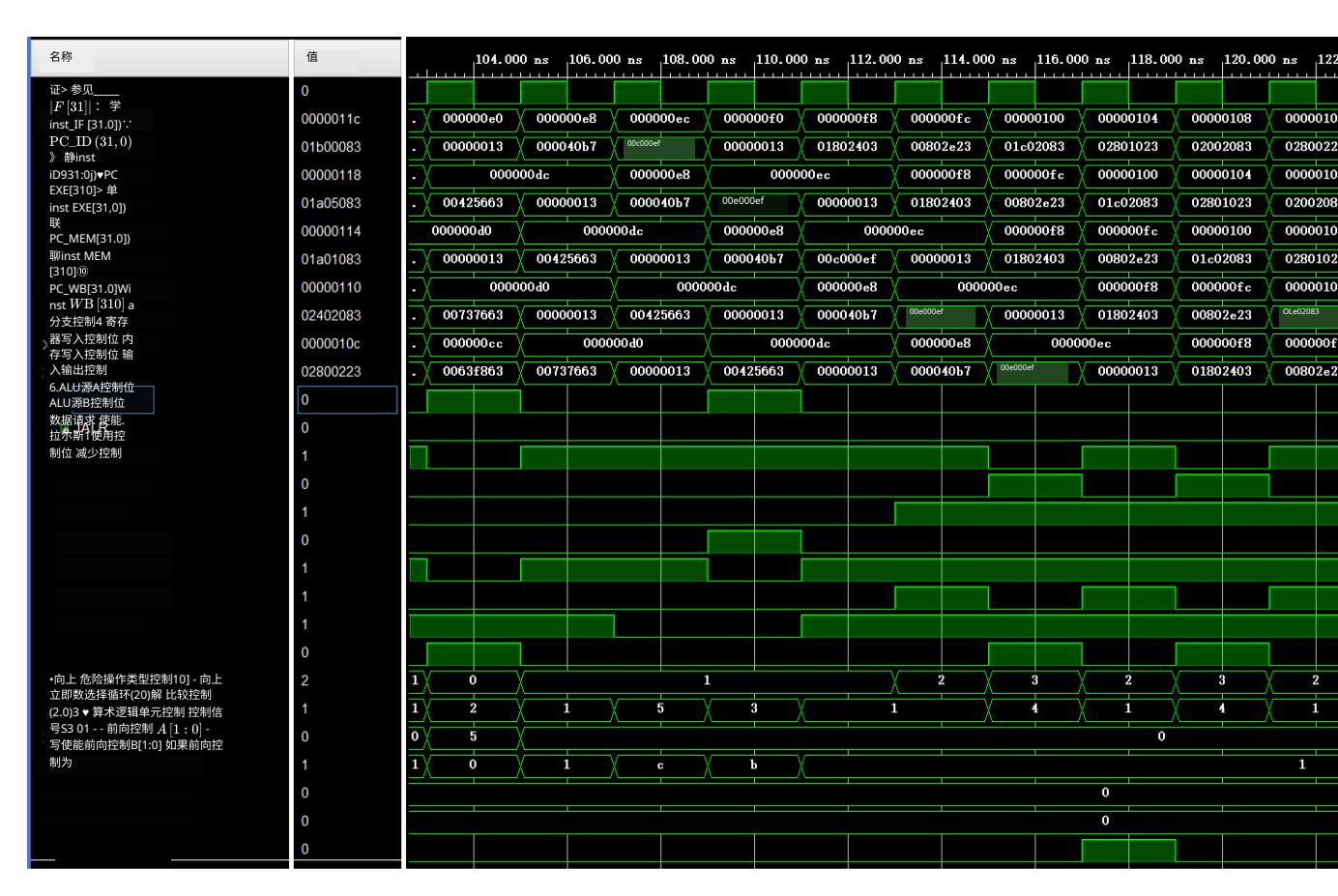


预测不跳转

## 模拟 (5)



## 模拟 (6)



## 仿真 (7)



•单步调试： - SW[0]  
= 1- 按下BTNX4Y0-  
非法指令

•完成Code2Inst.v

## 参考文献

- <https://riscv.org/wp-content/uploads/2018/05/13.15-13-50-Talk-riscv-base-isa-20180507.pdf>

•计算机组织与设计