架构实验6

使用计分板的动态调度流水线

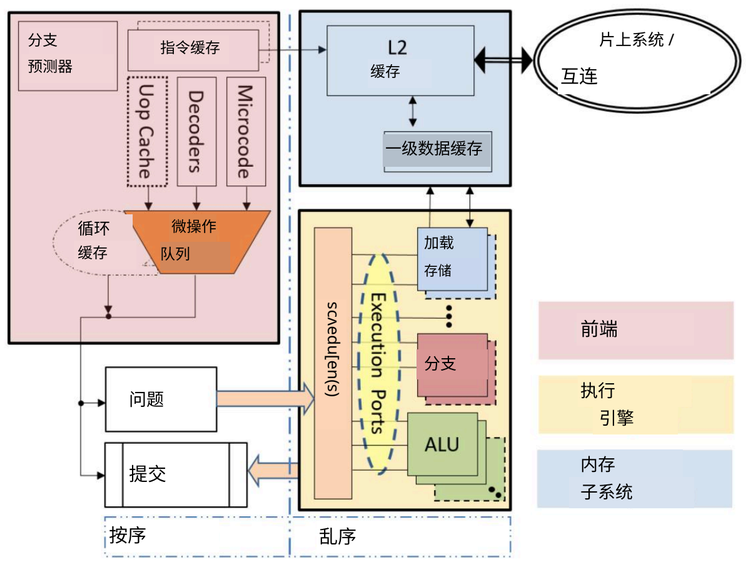
Chenlu Miao 12/2022

任务

•重新设计具有取指（IF）/指令调度（IS）/寄存器操作（RO）/功能单元执行（FU）/写回（WB）阶段的流水线，并支持多周期操作。

•设计一个计分板并将其集成到CPU中。

处理器核心微架构



https://dl.acm.org/doi/pdf/10.1145/3369383

计分板概述

指令状态 (c)

指令	发射	操作数	执行	写入
	1	2	3	4
	5	6	7	
	6			
	7			
除法 F10 F0 F6				

寄存器状态 ⑦

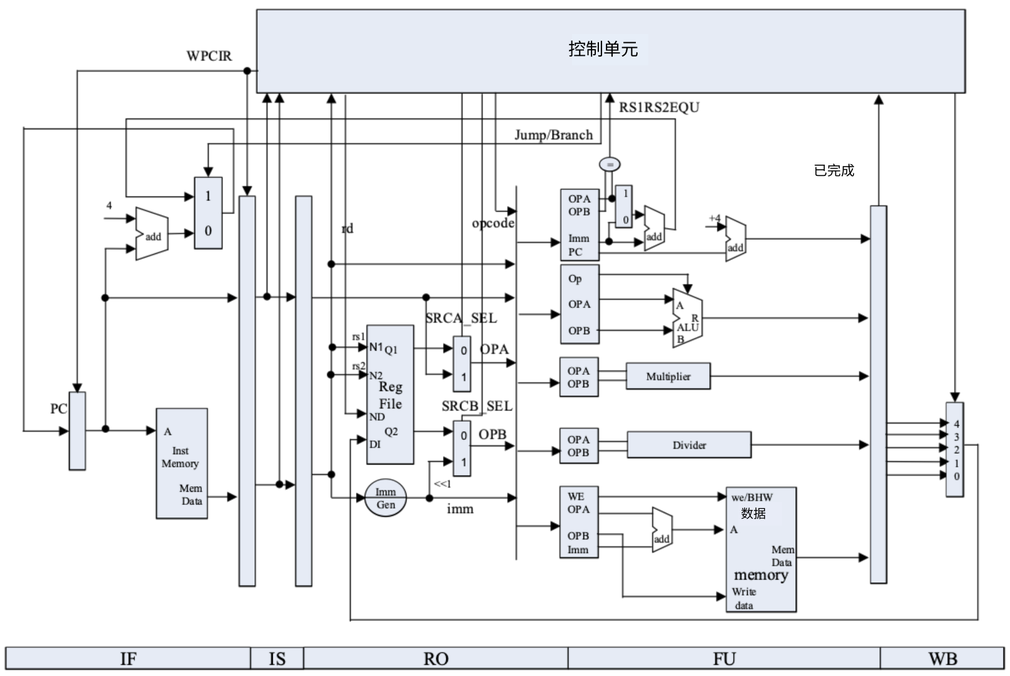
R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
矩阵		整数						加法							
F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31

功能单元 ⊕

时间	名称	伯里	Op	Fi	药	Fk	Qj	Qk	则	Rk
0	整数	真	LD	F2		R3			真	真
	多	真	MULT	F0	F2	F4	整数		假	真
	乘法2	疑似读写错误，可能涉及"假"							真	真
	添加	真	SUBD	F8	F6	F2		整数	真	制表符
	除	假							真	真

https://jasonren0403.github.io/scoreboard/

架构概述

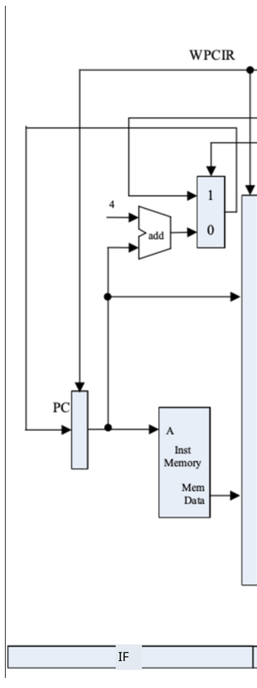


架构概述 - 指令获取阶段（IF）

// IF

赋值 PC_EN_IF = IS_EN | FU_jump_finish & is_jump_FU; 32位寄存器

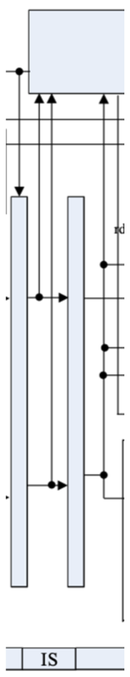
32位寄存器 REG_PC（.clk(调试时钟),.rst(复位信号),.CE(PC_EN_IF),.D(下一阶段指令获取阶段的PC值),.Q(指令获取阶段的PC值)）；32位加法器 add_IF（.a(指令获取阶段的PC值),.b(32'd4),.c(指令获取阶段PC值加4)）；32位二路选择器 mux_IF（.I0(指令获取阶段PC值加4),.I1(功能单元跳转完成后的PC值),.s(功能单元跳转完成 & 是功能单元跳转),.o(下一阶段指令获取阶段的PC值)）；只读存储器 inst_rom（.a(指令获取阶段PC值[8:2]),.spo(指令获取阶段的指令)）；



架构概述 – IS//IssueREG_IF_IS

reg_IF_IS(.clk(调试时钟), .rst(复位), .EN(IS使能),.flush(1'b0),.PCOUT(IF阶段的程序计数器),.IR(IF阶段的指令寄存器),.IR_IS(IS阶段的指令寄存器), .PCurrent_IS(IS阶段的当前程序计数器));

ImmGen模块 imm_gen(.ImmSel(ImmSel_ctrl), .inst_field(inst_IS), .Imm_out(Imm_out_IS));控制单元模块 ctrl(.clk(debug_clk), .rst(rst), .PC(PC_IS), .inst(inst_IS), .imm(Imm_out_IS), .ALU_done(FU_ALU_finish), .MEM_done(FU_mem_finish), .MUL_done(FU_mul_finish), .DIV_done(FU_div_finish), .JUMP_done(FU_jump_finish), .is_jump(is_jump_FU), .IS_en(IS_EN), .ImmSel(ImmSel_ctrl), .ALU_en(FU_ALU_EN), .MEM_en(FU_mem_EN), .MUL_en(FU_mul_EN), .DIV_en(FU_div_EN), .JUMP_en(FU_jump_EN), .PC_ctrl(PC_ctrl), .imm_ctrl(imm_ctrl), .rs1_ctrl(rs1_addr_ctrl), .rs2_ctrl(rs2_addr_ctrl), .JUMP_0p(Jump_ctrl), .ALU_op(ALUControl_ctrl), .ALU_use_PC(ALUSrcA_ctrl), .ALU_use_imm(ALUSrcB_ctrl), .MEM_we(mem_w_ctrl), .MEM_bhw(bhw_ctrl), .MUL_op( ), .DIV_op( ), .write_sel(DatatoReg_ctrl), .reg_write(RegWrite_ctrl), .rd_ctrl(rd_ctrl));



架构概述 – RO

// RO

Regs寄存器(.clk(调试时钟), .rst(复位),

.R_addr_A(rs1_addr_ctrl), .rdata_A(rs1_data_RO),.R_addr_B(rs2_addr_ctrl), .rdata_B(rs2_data_RO),

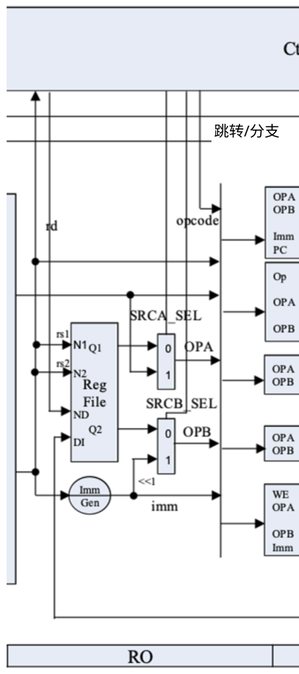
.L_S(寄存器写控制), .Wt_addr(rd_ctrl), .Wt_data(wt_data_WB),.Debug_addr(调试地址[4:0]),.Debug_regs(调试寄存器));

MUX2T1_32

mux_imm_ALU_RO_A(.I0(rs1_data_RO),.I1(PC控制),.s(ALU源A控制),.o(ALUA_RO));

32位二选一多路选择器

mux_imm_ALU_RO_B(.I0(rs2_data_RO),.I1(立即数控制),.s(ALU源B控制),.o(ALU B端输入_RO));



架构概述 - FU

// FU

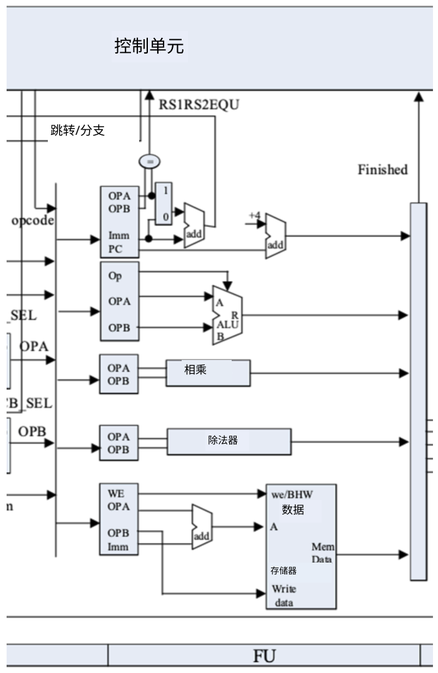
算术逻辑单元功能单元alu（...）；

内存功能单元mem（...）；

乘法功能单元mu（...）；

除法功能单元du（...）；

跳转功能单元ju（...）；



架构概述 – 写回阶段（WB）

// WB

32位寄存器reg_WB_ALU(.clk(调试时钟), .rst(复位信号), .CE(功能单元ALU完成信号), .D(功能单元ALU输出), .Q(写回阶段ALU输出));

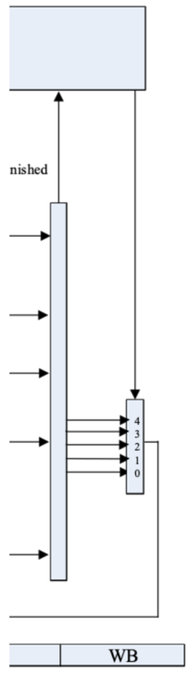
REG32

reg_WB_mem(.clk(调试时钟),.rst(复位信号),.CE(功能单元内存访问完成信号),.D(功能单元内存数据),.Q(写回阶段内存数据));

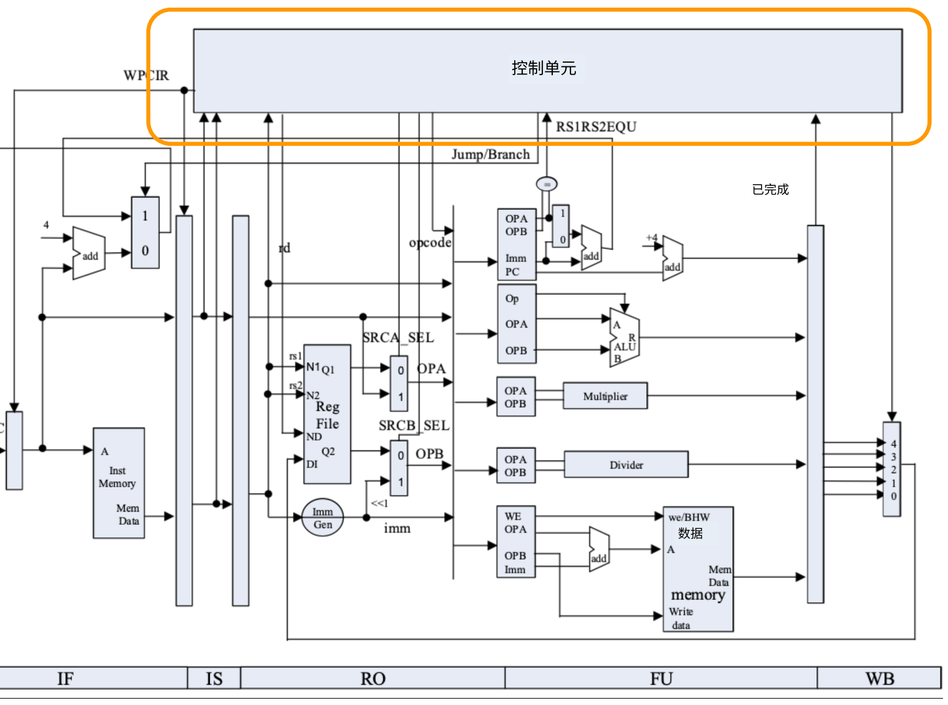
32位寄存器reg_WB_mul(.clk(调试时钟), .rst(复位信号), .CE(功能单元乘法完成信号), .D(功能单元乘法结果), .Q(写回阶段乘法结果));

32位寄存器reg_WB_div(.clk(调试时钟),.rst(复位信号),.CE(功能单元除法完成信号),.D(功能单元除法结果),.Q(写回阶段除法结果));

32位寄存器reg_WB_jump（.clk（调试时钟），.rst（复位信号），.CE（功能单元跳转完成信号），.D（写回阶段功能单元的程序计数器），.Q（写回阶段写回的程序计数器））；32位8选1多路复用器mux_DtR（.s（数据到寄存器控制信号），I0（写回阶段算术逻辑单元输出），I1（写回阶段内存数据），I2（写回阶段乘法结果），I3（写回阶段除法结果），.I4（写回阶段写回的程序计数器），.I5（32位常数0），.I6（32位常数0），.I7（32位常数0），.o（写回阶段写入数据））；



控制单元



控制单元

指令	指令状态 发布	读取操作数	执行完成	写入结果
L.D F6,34(R2)	✓	✓	✓	✓
L.D F2,45(R3)	✓	✓	✓	✓
MUL.D F0、F2、F4	✓	✓	✓	
SUB.D F8、F6、F2	✓	✓	✓	✓
DIV.D F10、F0、F6	✓			
ADD.D F6，F8，F2	✓	✓	✓	

功能单元状态

名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
整数	No								
乘法1	是	乘法	F0	F2	F4			No	No
乘法2	No								
添加	是	添加	F6	F8	F2			No	No
除	是	除	F10	F0	F6	乘法1		No	是

寄存器结果状态

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	乘法1			加法		除法			

控制单元 - 功能单元状态

功能单元状态

名称	忙碌	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
整数	是	加载	F2	R3				No	
乘法1	是	乘法	F0	F2	F4	整数		No	是
乘法2	No								
加法	是	减法	F8	F6	F2		整数	是	No
除	是	除法	F10	F0	F6	乘法1		No	是

忙碌 - 指示该单元是否忙碌。

Op - 单元中要执行的操作（例如，加法或减法）。

Fi - 目标寄存器。

Fj, Fk - 源寄存器编号。

Qj, Qk - 产生源寄存器 Fj, Fk 的功能单元

Rj, Rk - 标志，指示 Fj, Fk 何时就绪且尚未读取。操作数处理后设置为 No 。

+ 功能单元完成

控制单元 - 功能单元状态

忙碌 - 指示该单元是否忙碌。操作 - 该单元要执行的操作

Fi - 目标寄存器。

Fj, Fk - 源寄存器编号。

Qj, Qk - 产生源寄存器 Fj, Fk 的功能单元

Rj, Rk - 指示 Fj, Fk 何时就绪且尚未读取的标志。

操作数后设置为否。

+功能单元完成

寄存器[31:0] FUS[1:5];

例如 FUS[`FU_MEM][`SRC1_H:`SRC1_L]

```
//功能单元
    定义 FU_BLANK        3'd0
    定义 FU_ALU          3 'd1
    定义FU_MEM           3 'd2
    定义FU_MUL           3 'd3
    `定义FU_DIV          3 'd4
    定义FU_JUMP          3 'd5

// 功能单元状态（FUS）中的位
    定义 BUSY            )
    `定义 OP_L 为 1
    `定义 OP_H 为 5
    `定义 DST_L 为 6
    定义目标高位为10
    定义源1低位为11
    定义源1高位为1       5
    定义源2低位为16
    定义SRC2_H为20
    定义FU1_L为21
    定义FU1_H为23
    定义FU2_L为24
    定义 FU2_H 为 26
    定义 RDY1 为 27
    定义 RDY2 为 28
    定义 FU_DONE 为 29
```

控制单元 - 寄存器结果状态

指示如果一条活跃指令将某个寄存器作为其目标寄存器，哪个功能单元将写入该寄存器

当没有待处理的指令会写入该寄存器时，该字段设置为空。

每当没有待处理的指令会写入该寄存器时，此字段设置为空。

寄存器结果状态

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	乘法器1	整数			加法	除法			

// 记录哪个功能单元将在 WB reg[2:0]
RRS[0:31]处写入相应的寄存器;

```
// 功能单元 定义
FU_BLANK 定义 FU_ALU    3 ' d0
定义 FU_MEM             3 ' d1
                        3′d2

定义 FU_MUL             3 'd3
定义 FU_DIV             3′d4
定义 FU_JUMP 3'         3′d5
```

ISSUE

指令状态②

指令	Issue	操作数	执行	写入
LD F6 34 R2	1			

寄存器状态©

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
						Integer									
F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31

功能单元 ⊕

时间	名称	忙碌	Op	Fi	行	Fk	Qj	Qk	则	Rk
	整数	真	LD	F6		R2			真	真
	多	假							真	真
	乘法2	假							真	
	加法	假							真	真
	除法	假							真	真

ISSUE

```
// 正常停顿：结构冒险或写后写冲突
赋值正常停顿 = ...;

将 IS_en 赋值为 IS_flush 或非正常停顿且非控制停顿;
将 RO_en 赋值为非 IS_flush 且非正常停顿且非控制停顿;

始终在时钟上升沿或复位上升沿时开始
    if (rst) begin
        控制停顿信号赋值为0;
    结束
    否则开始
        // Is
        如果 (RO_en 且 (use_FU 等于 `FU_JUMP)) 开始
            ctrl_stall 小于等于 1;
        结束
        否则如果 (JUMP_done) 开始
            控制停顿（ctrl_stall） <= 0;
        结束
    结束
结束
```

```
// IS
如果 (RO_en) 开始
    // 不忙，无写后写冲突，将信息写入 FUS 和 RRS
    如果 (|dst)
        RRS[dst] <= 使用功能单元;

    FUS[使用功能单元][`忙碌] 赋值为 1'b1;
    FUS[使用功能单元][`操作码高位:`操作码低位] 赋值为 ...
    FUS[使用功能单元][`目标地址高位:`目标地址低位] 赋值为 ...
    功能单元选择器（FUS）[使用的功能单元][源操作数1高位:源操作数1低位]赋值为...
    功能单元选择器（FUS）[使用的功能单元][源操作数2高位:源操作数2低位]赋值为...
    功能单元选择器（FUS）[使用的功能单元][功能单元1高位:功能单元1低位]赋值为...
    FUS[use_FU][`FU2_H:`FU2_L] 小于等于 ...
    FUS[use_FU][`RDY1] 小于等于 ...
    功能单元状态（FUS）[使用功能单元（use_FU）][`就绪2（RDY2）] <= ...
    功能单元状态（FUS）[使用功能单元（use_FU）][`功能单元完成（FU_DONE）] <= ...

    IMM[使用功能单元] <= 立即数;
    PCR[使用功能单元] <= 程序计数器;
结束
```

读取操作数

```
// JUMP
如果 (FUS[`FU_JUMP][`RDY1] 与 FUS[`FU_JUMP][`RDY2])
开始
    ALU使能 = 1'b0;
    MEM使能 = 1'b0;
    MUL使能 = 1'b0;
    除法使能 = 1'b0;
    跳转使能 = 1'b1;

    跳转操作码 = FUS[`FU_JUMP][`OP_H:`OP_L];
    源寄存器1控制 = FUS[`FU_JUMP][`SRC1_H:`SRC1_L];
    源寄存器2控制 = FUS[`FU_JUMP][`SRC2_H:`SRC2_L];
    PC控制信号 = PCR[ 跳转功能单元 ];
    立即数控制信号 = 立即数[`跳转功能单元];
结束
```

读取操作数

```
// RO
如果 (功能单元状态[`跳转功能单元][`就绪1] & 功能单元状态[`跳转功能单元][`就绪2]) 开始
    // JUMP
    功能单元状态[`跳转功能单元][`就绪1] <= 1'b0;
    FUS[`FU_JUMP][`RDY2] 赋值为 1'b0;
结束
否则如果 (...) 开始                    // 在此处填充内容。
    // ALU
    ...                                 // 在此处填充内容。
结束
否则如果 (..) 开始                     // 在此处填充某物。
    // MEM
    ...                                 // 在此处填充某物。
结束
否则如果 (...) 开始                     // 在此处填充某物
    // MUL
    ...                                 // 在此处填充内容。
结束
否则如果 (..) 开始                     // 在此处填充内容。
    // DIV
    ...                                 // 在此处填充内容。
结束
```

执行

```
// EX
功能单元状态数组[`算术逻辑单元功能单元][`完成标志] <= ... // 在此处填充内容
...                                     // 在此处填充内容
```

写回

```
// WB
始终 â (*) 开始
    写选择 = 0;
    寄存器写 = 0;
    读控制 = 0;

    如果 (FUS[`FU_JUMP][`FU_DONE] 与 JUMP_WAR) 开始
        写选择信号 = 3位二进制数4;
        寄存器写使能 = 1位二进制数1;
        读控制信号 = 功能单元状态数组[`功能单元跳转][`目的地址高位:`目的地址低位];
    结束
    否则，如果 (功能单元状态数组[`功能单元算术逻辑单元][`功能单元完成] 且 算术逻辑单元写后读冲突) 开始
        写选择信号 = 3位二进制数0;
        寄存器写入 = 1'b1;
        读取控制 = 功能单元状态（FUS）[`功能单元_算术逻辑单元（FU_ALU）][`目的寄存器高位（DST_H）:`目的寄存器低位（DST_L）];
    结束
    ...
结束
```

```
// WB
如果（功能单元状态（FUS）[`功能单元_跳转（FU_JUMP）][`功能单元完成（FU_DONE）] 且 跳转写后读冒险（JUMP_WAR））开始功能单元状态（FUS）[`功能单元_跳转（FU_JUMP）] <= 32'b0;寄存器重命名状态（RRS）[功能单元状态（FUS）[`功能单元_跳转（FU_JUMP）][`目的寄存器高位（DST_H）:`目的寄存器低位（DST_L）]] <= 3'b0;// 确保写后读（RAW）...结束// 算术逻辑单元（ALU）
    ...;                                // 在此处填充内容
// MEM
    ...;                                //在此处填写内容
// MUL
    ...;                                //在此处填写内容
// DIV
    ...;                                //在此处填写内容
```

ROM

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		将内存地址为x0 + 4处的数据加载到寄存器x2	
2	00802203	8		将内存地址为x0 + 8处的数据加载到寄存器x4	结构冒险
3	004100b3	C		将寄存器x2和x4中的值相加，结果存入寄存器x1	
4	fff08093	10		将寄存器x1的值减1，结果存入寄存器x1	WAW
5	00c02283	14		将地址为12(x0)处的内容加载到寄存器x5	
6	01002303	18		将地址为16(x0)处的内容加载到寄存器x6	
7	01402383	1C		将地址为20(x0)处的内容加载到寄存器x7	
8	402200b3	20		用寄存器x4的值减去寄存器x2的值，结果存于寄存器x1	
9	ffd50093	24		将寄存器x10的值减3，结果存于寄存器x1	
10	00520c63	28		若x4等于x5，则跳转到标签label0	
11	00420a63	2C		若x4等于x4，则跳转到标签label0	
12	00000013	30		将x0的值加0后存回x0	
13	00000013	34		将x0的值加0后存回x0	
14	00000013	38		将x0的值加0后存回x0	

ROM

NO.	指令	地址	标签	ASM	注释
15	00000013	3C		将立即数0加到x0寄存器，结果存于x0	
16	000040b7	40	标签0:	将立即数4左移12位后加载到x1寄存器	
17	00c000ef	44		跳转到偏移量为12的地址并将返回地址存于x1	
18	00000013	48		将立即数0加到x0寄存器，结果存于x0	
19	00000013	4C		将立即数0加到x0寄存器，结果存于x0	
20	ffff0097	50		将0xffff0左移12位后加到PC寄存器，结果存于x1	
21	0223c433	54		将x7寄存器的值除以x2寄存器的值，结果存于x8	
22	025204b3	58		将x4寄存器的值乘以x5寄存器的值，结果存于x9	圣哈/原始数据/写后写
23	022404b3	5C		将x8和x2相乘，结果存于x9	WAR
24	00400113	60		将立即数4与x0相加，结果存于x2	
25	000000e7	64		跳转到x0地址加0处，并将返回地址存于x1	
26	00000013	68		将立即数0与x0相加，结果存于x0	
27	00000013	6C		将立即数0与x0相加，结果存于x0	

RAM

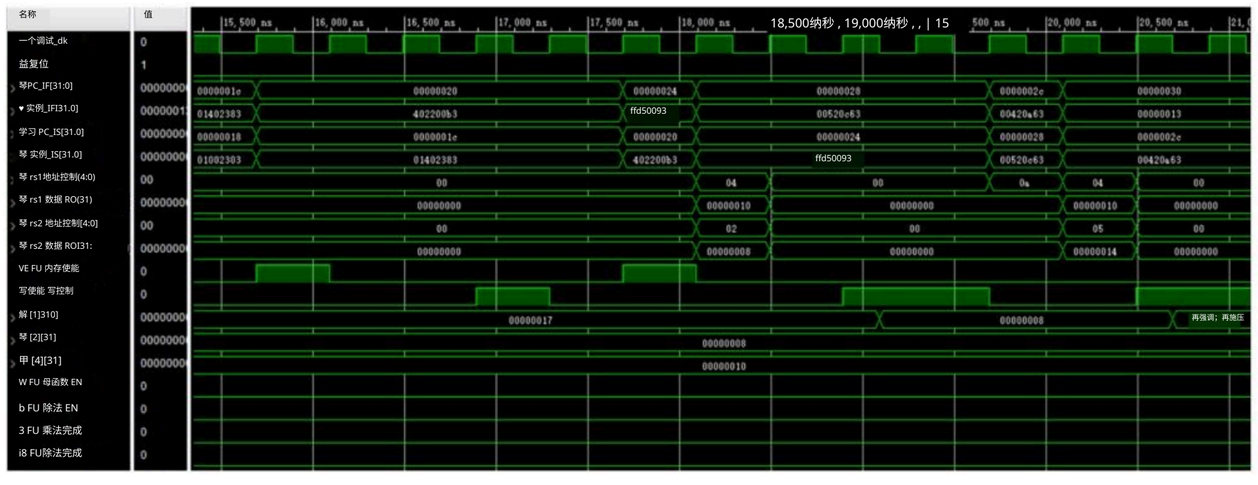
NO.	数据	地址
0	000080BF	0
1	00000008	4
2	00000010	8
3	00000014	C
4	FFFF0000	10
5	0FFF0000	14
6	FF000F0F	18
7	F0F0F0F0	1C
8	00000000	20
9	00000000	24
10	00000000	28
11	00000000	2C
12	00000000	30
13	00000000	34
14	00000000	38
15	00000000	3C

NO.	说明	地址
16	00000000	40
17	00000000	44
18	00000000	48
19	00000000	4C
20	A3000000	50
21	27000000	54
22	79000000	58
23	15100000	5C
24	00000000	60
25	00000000	64
26	00000000	68
27	00000000	6C
28	00000000	70
29	00000000	74
30	00000000	78
31	00000000	7C

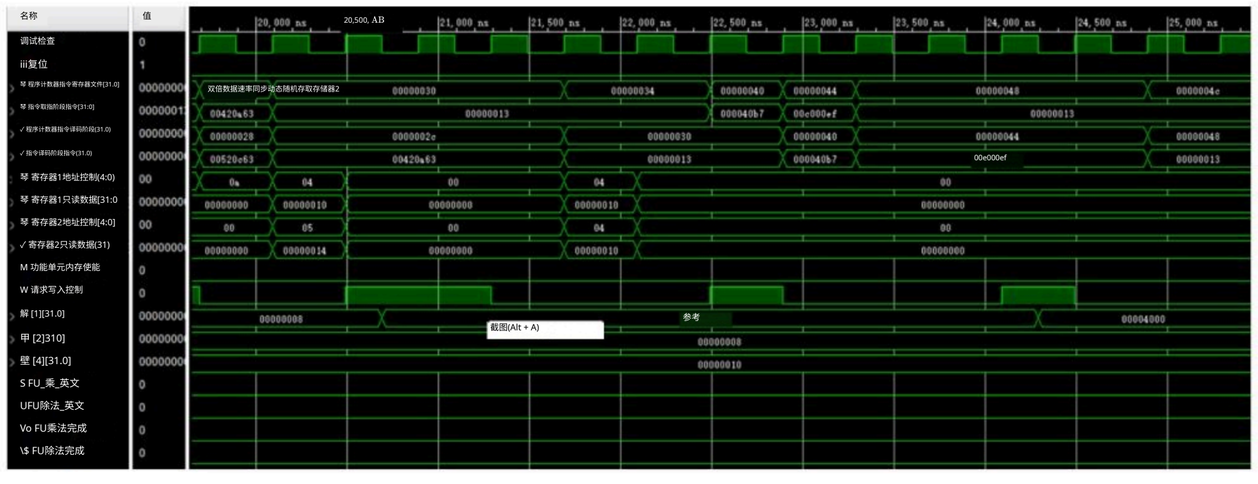
Simulation

Simulation

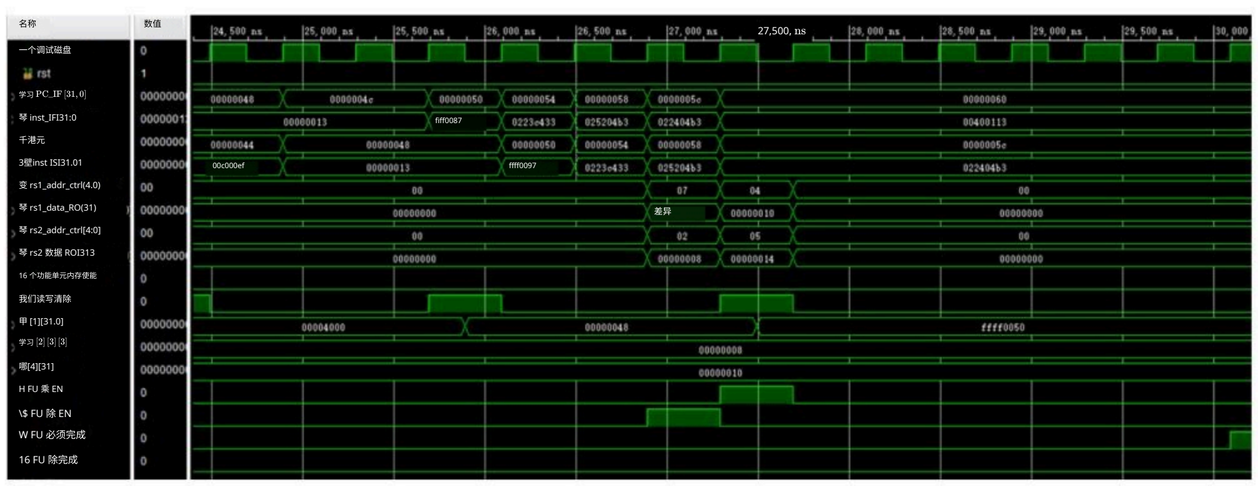
仿真



仿真

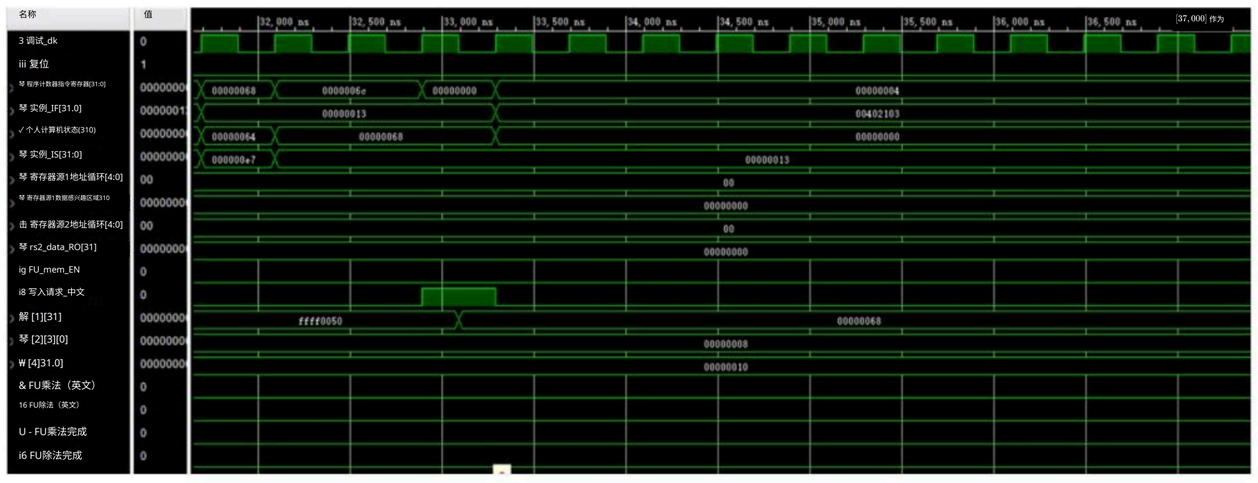


仿真



仿真

仿真



仿真

参考文献

- https://dl.acm.org/doi/pdf/10.1145/3369383
- https://zhuanlan.zhihu.com/p/496078836
- https://jasonren0403.github.io/scoreboard/