P5_document

Hazard.v

input	explain
Instr_ID	ID阶段的指令码
Instr_EX	EX阶段的指令码
Instr_MA	MA阶段的指令码
A3_EX	EX阶段的寄存器写入地址
A3_MA	MA阶段的寄存器写入地址

T_use表格（需求者）

指令位于ID级时，还需要多少个周期才必须要使用数据。

	$T_{use}(rs)$	$T_{use}(rt)$
add	1	1
sub	1	1
ori	1	
lw	1	
sw	1	2
beq	0	0
lui	1	
jal		
jr	0	
nop		

```
wire [1:0] Tuse_rs = (add_ID | sub_ID | ori_ID | lw_ID | sw_ID | lui_ID)?2'b01:
                     (beq_ID | jr_ID)? 2'b00:
                     2'b11;

wire [1:0] Tuse_rt = (add_ID | sub_ID)? 2'b01:
                     (sw_ID)? 2'b10:
                     (beq_ID)? 2'b00:
                     2'b11;
```

T_new表格（供给者）

T_new: 供给指令位于某个流水级时还需要多少个周期才可以计算出结果并存储到流水寄存器中。

早于需求指令进入流水线的供给者还需要多少cycle能够产生需求者需要的寄存器值。

Intsr	功能部件(结果来源)	EX	MA	WB
add	ALU	1		
sub	ALU	1		
ori	ALU	1		
lw	DM	2	1	0
sw				
beq				
lui	ALU	1		
jal	PC	0	0	0
jr				
nop				

```
wire [1:0] Tnew_EX = (add_EX | sub_EX | ori_EX | lui_EX)? 2'b01:
                     (lw_EX)? 2'b10: 2'b00;
wire [1:0] Tnew_MA = lw_MA?2'b01:2'b00;
```

rs策略矩阵

$T_{use}\backslash T_{new}$	EX	EX	EX	MA	MA	MA	WB	WB	WB
	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

$T_{use}\backslash T_{new}$	EX	EX	EX	MA	MA	MA	WB	WB	WB
	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

如何理解策略矩阵

- 策略矩阵针对的是AT法中的Time部分，具体判断阻塞还是转发需要综合Adress部分
- 对于ALU型指令，如add，只需要关注 T_{new} 的ALU一列，其表示在当前流水级下，还有多少周期才能计算出结果并存入流水级寄存器。

```
    wire Stall_RS_EX = (Tuse_rs < Tnew_EX) & (A3_EX == RS_ID & RS_ID != 0);
    wire Stall_RS_MA = (Tuse_rs < Tnew_MA) & (A3_MA == RS_ID & RS_ID != 0);
    wire Stall_RT_EX = (Tuse_rt < Tnew_EX) & (A3_EX == RT_ID & RT_ID != 0);
    wire Stall_RT_MA = (Tuse_rt < Tnew_MA) & (A3_MA == RT_ID & RT_ID != 0);

    assign Stall = Stall_RS_EX | Stall_RS_MA | Stall_RT_EX | Stall_RT_MA;
```

Forward in mips.v

数据的供给者

- EX_MA级流水线寄存器

```
    wire [31:0] forward_EX_MA = (RegWrite_MA && (Mem2Reg_MA == 2'b00))? ALU_C_MA:
                                (RegWrite_MA && (Mem2Reg_MA == 2'b10))? pc_add4_MA +
32'd4:
                                32'h404;
```

- MA_WB级流水线寄存器

```
    assign grf_WD = (Mem2Reg_WB == 2'b00) ? ALU_C_WB :
                    (Mem2Reg_WB == 2'b01) ? dm_data_WB :
                    (Mem2Reg_WB == 2'b10) ? pc_add4_WB + 32'd4 : 32'd0;

    wire [31:0] forward_MA_WB = RegWrite_WB? grf_WD: 32'h405;
```

数据需求者

- ID级

```
    wire [31:0] RD1_ID_valid =  ((RS_ID == A3_MA) && (A3_MA != 5'b0))? forward_EX_MA:
                                ((RS_ID == A3_WB) && (A3_WB != 5'b0))? forward_MA_WB:
                                RD1_ID;

    wire [31:0] RD2_ID_valid =  ((RT_ID == A3_MA) && (A3_MA != 5'b0))? forward_EX_MA:
                                ((RT_ID == A3_WB) && (A3_WB != 5'b0))? forward_MA_WB:
                                RD2_ID;
```

- EX级

```
    //forward--select valid ALU_A and ALU_B
    wire [31:0] ALU_A_EX_valid = ((RS_EX == A3_MA) && (A3_MA != 5'b0))? forward_EX_MA:
                                 ((RS_EX == A3_WB) && (A3_WB != 5'b0))? forward_MA_WB:
                                 RD1_EX;

    wire [31:0] Grt_EX_valid =  ((RT_EX == A3_MA) && (A3_MA != 5'b0))? forward_EX_MA:
                                ((RT_EX == A3_WB) && (A3_WB != 5'b0))? forward_MA_WB:
                                RD2_EX;//将目前正确的Grt值，流水到MA级供sw指令使用，也可供R-
type-ALU指令使用

    wire [31:0] Grt_MA_valid;

    wire [31:0] ALU_B_EX_valid = (ALUSrc_EX == 1'b0) ? Grt_EX_valid:
                                 (ALUSrc_EX == 1'b1) ? ext32_EX : 32'b0;
```

- MA级

  再一次获取转发数据，确保DM中的Grt值是正确的

```
wire [31:0] Grt_MA_valid_valid = ((RT_MA == A3_WB) && (A3_WB != 5'd0))? forward_MA_WB:
Grt_MA_valid;
```

流水线寄存器

	PC	IF_ID	ID_EX	EX_MA	MA_WB
en	nStall	nStall	true	false	true

	PC	IF_ID	ID_EX	EX_MA	MA_WB
clr	null	false	Stall	true	false

	RegDst	ExtOp	jump	jumpSrc	branch	bOp	ALUSrc	ALUOP	MemWrite	ls_type	Mem2Reg	RegWrite
作用阶段	EX	ID	ID	ID	ID	ID	EX	EX	MA	MA	WB	WB
IF_ID												
ID_EX	1						1	1	1	1	1	1
EX_MA									1	1	1	1
MA_WB											1	1

Stall Explain

ID级指令静止，EX级插入一个nop

- PC不写入新数据
- IF_ID寄存器不写入新数据
- ID_EX寄存器清零，即插入nop(0x00000000)

Logic

```
module Stage_Reg(
        input clk,
        input reset,
        input en,
        input clr,
        input [31:0] data,
        output reg [31:0] data_out
    );

     initial begin
            data_out <= 0;
     end

     always @(posedge clk) begin
        if (reset || clr) begin
             data_out <= 0;
        end else begin
            if (en) begin
                data_out <= data;
            end
        end
     end
```

信号解释

jumpSrc (2-bit)

jumpSrc	type
2'b00	branch
2'b01	jump
2'b10	jr

```
assign next_pc_ID = (jumpSrc_ID == 2'b00) ? branch_offset_ID:
                    (jumpSrc_ID == 2'b01) ? jumpExt32_ID:
                    (jumpSrc_ID == 2'b10) ? RD1_ID_valid: 32'h404;
```

Mem2Reg (2-bit)

Mem2Reg	type
2'b00	ALU
2'b01	lw
2'b10	link

```
assign grf_WD = (Mem2Reg_WB == 2'b00) ? ALU_C_WB :
                (Mem2Reg_WB == 2'b01) ? dm_data_WB :
                (Mem2Reg_WB == 2'b10) ? pc_add4_WB + 32'd4 : 32'd0;
```

RegDst (2-bit)

RegDst	type
2'b00	无需写入寄存器的指令
2'b01	R-type，写入rd寄存器
2'b10	I-type，写入rt寄存器
2'b11	link，写入ra(31)

```
assign A3_EX =  (RegDst_EX == 2'b00)? 5'b0: //null
                (RegDst_EX == 2'b01)? RD_EX:
                (RegDst_EX == 2'b10)? RT_EX:
                (RegDst_EX == 2'b11)? 5'h1f:
                5'b00000;
```

ALU (4-bit)

ALU	type
4'b0000	add
4'b0001	sub
4'b0010	or
4'b0011	lui

```
always @(*) begin
        case(F)
            `ADD:    C = A + B;
            `SUB:    C = A - B;
            `OR:     C = A | B;
            `SL16:   C = B << 16;
            default:C = 0;
        endcase
    end
```

测试方案

指令分类

R-type

- *ALU型指令（Grd <- Grs ? Grt）* —— (add、sub、or)
- *ALU型指令（Grd <- Grt？ shamt）* —— (sll、srl、sra)
- *跳转型指令（PC <- Grs）* —— (jr)

I-type

- ***ALU-Imm型指令（Grt <- Grs？ imm）*** ——（ori、lui）
- ***Memory-Store型指令（Memory[Grs + offset] <- Grt）*** ——（sw、sb、sh）
- ***Memory-load型指令（Grt <- Memory[Grs + offset]）*** ——（lw、lb、lbu、lh）
- ***Branch型指令（Grs？Grt，PC <- PC + 4 + offset）*** ——（beq、bgez、bgtz、blez、bltz、bne）

J-type

- ***J-link型指令*** ——(jal)
- ***无条件Jump型指令***

	IF级	ID级	EX级	MA级	WB级
R-ALU		读取数据（Grs、Grt）	计算数据		写入数据
R-jump		读取数据并跳转			
I-imm		读取数据（Grt、imm）	计算数据		写入数据
I-store		读取数据（Grs+offset、Grt）	计算数据（地址）	写入数据	
I-load		读取数据（Grs+offset、Grt）	计算数据（地址）	读取数据	写入数据
I-branch		读取数据（Grs、Grt）比较数据并跳转			
J-link		扩展数据并跳转	传递跳转PC	传递跳转PC	写入pc+8

测试步骤

1. 编写测试代码
   - 包括所有阻塞情况
   - 测试所有转发通路
   - 可利用课程组提供的分析工具对代码测试强度进行评估
2. 将ISE输出与Mars输出比对
   - 可利用同学开发评测机进行快速比对
3. 大量随机数据强测
   - 代码生成思路参照P4数据生成思路

```python
import random

SUPPORTED_INSTRUCTIONS = [
    "add", "sub", "and", "ori", "lui",  # arithmetic
    "lw", "sw",  # storage
    "beq", "bne",  # B-type
    "j", "jal",  # jump
    "nop"
    # command jr will be generated according to the use of command jal
]
REGISTERS = [f"${i}" for i in range(32) if i != 1 and i < 26]


def weighted_choice():
    return random.choices(REGISTERS, weights=REGISTER_WEIGHTS, k=1)[0]


def generate_random_instruction(half):
    instruction = random.choice(SUPPORTED_INSTRUCTIONS)

    if instruction in ["add", "sub", "and"]:
        # add $t0, $t1, $t2 : op rd, rs, rt
        rd = weighted_choice()
        rs = weighted_choice()
        rt = weighted_choice()
        return f"{instruction} {rd}, {rs}, {rt}"

    # other instructions...


def main():
    # parser...

    high_weight_registers = random.sample(REGISTERS, 4)
    # print(f"High weight registers: {high_weight_registers}")
    global REGISTER_WEIGHTS
    REGISTER_WEIGHTS = [4 if reg in high_weight_registers else 1 for reg in REGISTERS]

    mips_code = generate_mips_code(args.num_instructions)
```

思考题

1. 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

> 提前分支判断会带增加阻塞的情况，从而降低效率
>
> ```
> lw $1, 0($0)
> ori $2, $0, 514
> beq $1, $2, label
> ```
>
> 对于上述代码，需要阻塞两次

2. 因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

> 因为，jr返回时应该执行延迟槽后面的指令

3. 我们要求所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），请思考为什么？

> 减少组合逻辑与时序逻辑的组合，减少毛刺，使流水线分级更清晰

4. 我们为什么要使用 GPR 内部转发？该如何实现？

> 解决结构冒险。由于流水线中grf的读写在不同级，所以当同时读和写grf时会出现结构冒险
>
> ```
> assign RD1 = (A3≠0 & A1\==A3 & WE) ? WD : grf[A1];
> assign RD2 = (A3≠0 & A2\==A3 & WE) ? WD : grf[A2];
> ```

5. 我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

> 见上文

6. 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

> 1. 添加控制信号
> 2. 可能需要修改ALU
> 3. 修改顶层模块，可能需要添加wire
> 4. 可能需要修改阻塞逻辑

7. 简要描述你的译码器架构，并思考该架构的优势以及不足。、

> 集中式译码。只需一次译码，然后流水各个信号。
> 优势：资源占用小，代码量小
> 不足：流水信号比较麻烦，添加控制信号时比较麻烦

8. [P5 选做] 请详细描述你的测试方案及测试数据构造策略。

> 见测试方案

9. [P5、P6 选做] 请评估我们给出的覆盖率分析模型的合理性，如有更好的方案，可一并提出。