P6_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
busy	乘除槽正在进行计算
start	乘除槽开始进行计算

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。

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

P6_new	and	or	slt	sltu	addi	andi	lb	lh	sb	sh
$T_{use}(rs)$	1	1	1	1	1	1	1	1	1	1
$T_{use}(rt)$	1	1	1	1					2	2

P6_new	mult	multu	div	divu	mfhi	mflo	mthi	mtlo	bne
$T_{use}(rs)$	1	1	1	1			1	1	0
$T_{use}(rt)$	1	1	1	1					0

```
    wire [1:0] Tuse_rs = (add_ID | sub_ID | ori_ID | lw_ID | sw_ID | lui_ID | and_ID |
or_ID | slt_ID | sltu_ID | addi_ID | andi_ID | lb_ID | lh_ID | sb_ID | sh_ID | mthi_ID |
mtlo_ID | mult_ID | multu_ID | div_ID | divu_ID)?2'b01:
                              (beq_ID | jr_ID | bne_ID)? 2'b00:
                              2'b11;

    wire [1:0] Tuse_rt = (add_ID | sub_ID | and_ID | or_ID | slt_ID |sltu_ID | mult_ID |
multu_ID | div_ID | divu_ID)? 2'b01:
                              (sw_ID | sb_ID | sh_ID)? 2'b10:
                              (beq_ID | bne_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	add	sub	ori	lw	sw	beq	lui	jal	jr	nop
功能部件(结果来源)	ALU	ALU	ALU	DM			ALU	PC		
EX	1	1	1	2			1	0		
MA				1				0		
WB				0				0		

P6_new_Intsr	and	or	slt	sltu	addi	andi	lb	lh	sb	sh
功能部件(结果来源)	ALU	ALU	ALU	ALU	ALU	ALU	DM	DM		
EX	1	1	1	1	1	1	2	2		
MA							1	1		
WB							0	0		

P6_new_Intsr	mult	multu	div	divu	mfhi	mflo	mthi	mtlo	bne
功能部件(结果来源)					mdb	mdb			
EX					1	1			
MA									
WB									

```
    wire [1:0] Tnew_EX = (add_EX | sub_EX | ori_EX | lui_EX | and_EX | or_EX | slt_EX |
sltu_EX | addi_EX | andi_EX | mfhi_EX | mflo_EX)? 2'b01:
                              (lw_EX | lb_EX | lh_EX)? 2'b10: 2'b00;
    wire [1:0] Tnew_MA = (lw_MA | lb_MA | lh_MA)?2'b01:2'b00;
```

rs策略矩阵

	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

$T_{use}\backslash T_{new}$	EX	EX	EX	MA	MA	MA	WB	WB	WB
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);
    wire Stall_md = ((busy | start) & (mult_ID | multu_ID | div_ID | divu_ID | mfhi_ID |
mflo_ID | mthi_ID | mtlo_ID));

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

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
clr	null	false	Stall	true	false

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

RegDst	type
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;
```

ALUOp (4-bit)

ALUOp	type
4'b0000	add
4'b0001	sub
4'b0010	or
4'b0011	lui
4'b0100	and
4'b0101	slt
4'b0110	sltu
4'b0111	addi

```
    wire [32:0] A_ext = {A[31],A};
    wire [32:0] B_ext = {B[31],B};
    wire [32:0] temp = A_ext + B_ext;

  always @(*) begin
      case(F)
          `ADD:    C = A + B;
              `ADDI:  C = (temp[32] == temp[31])?temp[31:0]:32'd0;
          `SUB:    C = A - B;
          `OR:     C = A | B;
          `SL16:   C = B << 16;
              `AND:       C = A & B;
              `SLT:   C = ($signed(A) < $signed(B))? 32'b1: 32'b0;
              `SLTU:  C = (A < B)? 32'b1: 32'b0;
          default:C = 0;
```

```
        endcase
```

mdOp (4-bit)

mdOp	type
4'b0001	mult
4'b0010	multu
4'b0011	div
4'b0100	divu
4'b0101	mthi
4'b0110	mtlo
4'b0111	mfhi
4'b1000	mflo

```
// in MulDivBlock
always @(*) begin
    case (op)
        4'b0001: {prod_HI, prod_LO} = ($signed(A) * $signed(B));//mult
        4'b0010: {prod_HI, prod_LO} = $unsigned(A) * $unsigned(B);//multu
        4'b0011: {prod_HI, prod_LO} = {$signed(A) % $signed(B), $signed(A) /
$signed(B)};//div
        4'b0100: {prod_HI, prod_LO} = {A % B, A / B};//divu
    endcase
end
```

```
     wire [31:0] mdb_output = (mdOp_EX == 4'b0111)? mdb_HI:
                                      (mdOp_EX == 4'b1000)? mdb_LO:
                                      32'd0;
```

load_type (3-bit)

load_type	type
3'b000	lw
3'b001	lbu
3'b010	lb

load_type	type
3'b011	lhu
3'b100	lh

```
assign data_out = (load_type == 3'b000)? data_input:
                  (load_type == 3'b001)? {24'b0, byte_sel}:
                  (load_type == 3'b010)? {{24{byte_sel[7]}}, byte_sel}:
                  (load_type == 3'b011)? {16'b0, hw_sel}:
                  (load_type == 3'b100)? {{16{hw_sel[15]}}, hw_sel}:
                  32'b0;
```

store_type (3-bit)

store_type	type
3'b001	sw
3'b010	sb
3'b100	sh

```
wire [3:0] byteen_MA = (store_type_MA == 3'b001)? 4'b1111: //word
                       ((store_type_MA == 3'b010))? byte_sel://byte
                       (store_type_MA == 3'b100)? hw_sel://half word
                       4'b0000;
```

测试方案

指令分类

R-type

- *ALU型指令（Grd <- Grs ? Grt）* —— (add、sub、or、and、slt、sltu)
- *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、addi、andi）
- ***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
   - 代码生成来自学长博客

```python
import random

list_R = ["addu", "subu", "and", "or", "nor", "xor", "sltu", "slt", "sllv", "srlv",
"srav"]
list_I = ["andi", "addiu", "ori", "xori", "lui", "slti", "sltiu"]
list_LS = ["lw", "sw", "lh", "lhu", "sh", "lb", "lbu", "sb"]
list_shift = ["sll", "srl", "sra"]
list_B = ["bne", "beq","bgtz", "blez", "bltz", "bgez"]
list_MD = ["mult", "multu", "div", "divu"]
list_MTMF = ["mfhi", "mflo", "mthi", "mtlo"]

#length是生成的指令所用到的寄存器个数
length = 8 #为了增大冒险概率，我们将寄存器的范围缩小到0~7

def R_test(file, n):
    for i in range(n):
        k = random.randint(0, 10000000) % len(list_R)
        rs = random.randint(0, 10000000) % length;
        rt = random.randint(0, 10000000) % length;
        rd = random.randint(0, 10000000) % length;
        s = "{} ${}, ${}, ${}\n".format(list_R[k], rd, rs, rt)
        file.write(s)

def I_test(file, n):
    for i in range(n):
        k = random.randint(0, 10000000) % len(list_I)
        rs = random.randint(0, 10000000) % length
        rt = random.randint(0, 10000000) % length
        imm = random.randint(-32768, 32768)
        abs_imm = random.randint(0, 65536)
        if list_I[k] == "lui":
            s = "{} ${},{}\n".format(list_I[k], rt, abs_imm)
        else:
            s = "{} ${}, ${}, {}\n".format(list_I[k], rt, rs, imm)
        file.write(s)

def LS_test(file, n):
    for i in range(n):
        k = random.randint(0,10000000) % len(list_LS)
        ins = list_LS[k]
        num = 0
        if(ins[1] == "w"):
            num = (random.randint(0,10000000) << 2) % 4096
        elif(ins[1] == "h"):
```

```
            num = (random.randint(0,10000000) << 1) % 4096
        else:
            num = (random.randint(0,10000000)) % 4096

        rt = random.randint(0, 10000000) % length
        s = "{} ${}, {}($0)\n".format(ins, rt, num)
        file.write(s)

def shift_test(file, n):
    for i in range(n):
        k = random.randint(0,10000000) % len(list_shift)
        shamt = random.randint(0, 10000000) % length
        rd = random.randint(0, 10000000) % length
        rt = random.randint(0, 10000000) % length
        s = "{} ${}, ${}, {}\n".format(list_shift[k], rd, rt, shamt)
        file.write(s)

def MD_test(file, n):
    for i in range(n):
        k = random.randint(0,10000000) % len(list_MD)
        rs = random.randint(0, 10000000) % length
        rt = random.randint(0, 10000000) % length
        if(list_MD[k] == "mult" or list_MD[k] == "mul"):
            s = "{} ${}, ${}\n".format(list_MD[k], rs, rt)
        else:
            s = "{} ${}, ${}\n".format(list_MD[k], rs, 8)
        file.write(s)

def MTMF_test(file, n):
    for i in range(n):
        k = random.randint(0,10000000) % len(list_MTMF)
        rs = random.randint(0, 10000000) % length
        s = "{} ${}\n".format(list_MTMF[k], rs)
        file.write(s)

def B_test(file, lable):
    k = random.randint(0,10000000) % len(list_B)
    if(k == 0 or k == 1):
        rs = random.randint(0, 10000000) % length
        rt = random.randint(0, 10000000) % length
        s = "{} ${}, ${}, {}\n".format(list_B[k], rs, rt, lable)
        file.write(s)
    else:
        rs = random.randint(0, 10000000) % length
```

```
            s = "{} ${}, {}\n".format(list_B[k], rs, str(lable))
            file.write(s)


    def b_begin(file, n):
        file.write("\nb_test_{}_one:\n".format(n))
        B_test(file, "b_test_{}_one_then".format(n))
        R_test(file,1)

        file.write("b_test_{}_two:\n".format(n))
        B_test(file, "b_test_{}_two_then".format(n))
        I_test(file,1)

        file.write("jal_test_{}:\n".format(n))
        file.write("jal jal_test_{}_then\n".format(n))
        I_test(file,1)

        file.write("end_{}:\n\n".format(n))

        R_test(file, 1)
        I_test(file, 1)
        LS_test(file, 1)
        shift_test(file, 1)

    def b_end(file, n):
        file.write("\nb_test_{}_one_then:\n".format(n))
        R_test(file, 1)
        I_test(file, 1)
        LS_test(file, 1)
        shift_test(file, 1)
        MD_test(file, 1)
        MTMF_test(file, 1)
        file.write("j b_test_{}_two\n".format(n))
        R_test(file, 1)

        file.write("\nb_test_{}_two_then:\n".format(n))
        R_test(file, 1)
        I_test(file, 1)
        LS_test(file, 1)
        shift_test(file, 1)
        MD_test(file, 1)
        MTMF_test(file, 1)
        file.write("jal jal_test_{}\n".format(n))
        file.write("addu $1, $ra, $0\n")
```

```
        file.write("\njal_test_{}_then:\n".format(n))
        R_test(file, 1)
        I_test(file, 1)
        LS_test(file, 1)
        shift_test(file, 1)
        MD_test(file, 1)
        MTMF_test(file, 1)
        file.write("addiu $ra,$ra, 8\n".format(n))
        B_test(file, "end_{}".format(n))
        R_test(file, 1)
        I_test(file, 1)
        LS_test(file, 1)
        shift_test(file, 1)
        MD_test(file, 1)
        MTMF_test(file, 1)
        file.write("jr $ra\n".format(n))


    with open("mips_code.asm", "w") as file:
        for i in range(length):
            temp = random.randint(-2147483648, 2147483648)
            s = "li ${} {}\n".format(i, temp)
            file.write(s)
        file.write("li $8, {}\n".format(random.randint(-2147483648, 2147483648) % 10000 + 1))

        for i in range(10):
            R_test(file, 2)
            MD_test(file,1)
            MTMF_test(file, 1)
            I_test(file, 2)
            LS_test(file, 2)
            MTMF_test(file, 1)
            I_test(file, 2)
            shift_test(file, 2)
            file.write("\n")
```

思考题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

> [!ANS]
> 因为乘除操作较为复杂，需要多个周期才能完成，需要单独处理

2. 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

```
`define size 8
module mul_booth_signed(
    input wire [`size - 1 : 0] mul1,mul2,
    input clk,
    input wire [2:0] clk_cnt,//运算节拍，相当于状态机了，8位的话每次运算有4个拍
    output wire [2*`size - 1 : 0] res
    );

    //由于传值默认就是补码，所以只需要再计算“负补码”即可
    wire [`size - 1 : 0] bmul1,bmul2;
    assign bmul1 = (~mul1 + 1'b1) ;
    assign bmul2 = (~mul2 + 1'b1) ;//其实乘数2的负补码也没用到。
 //其实可以把状态机的开始和结束状态都写出来，我懒得写了，同学们可以尝试一下啊~
    parameter   zeroone       =   3'b00,
                twothree      =   3'b001,
                fourfive      =   3'b010,
                sixseven      =   3'b011;
    //y(i-1),y(i),y(i+1)三个数的判断寄存器，由于有多种情况，也可以看成状态机（也可以改写成状态机形式，
大家自己试试吧）
    reg [2:0] temp;

    //部分积
    reg [2*`size-1 : 0] A;
 //每个节拍下把相应位置的数据传给temp寄存器
    always @ (posedge clk) begin
        case(clk_cnt)
            zeroone  : temp <= {mul2[1:0],1'b0};
            twothree : temp <= mul2[3:1];
            fourfive : temp <= mul2[5:3];
            sixseven : temp <= mul2[7:5];
            default : temp <= 0;
        endcase
    end

    always @(posedge clk) begin
        if (clk_cnt == 3'b100) begin//如果节拍到4就让部分积归0，此时已经完成一次计算了
            A <= 0;
```

```
        end else case (temp)
            3'b000,3'b111 :   begin//这些是从高位到低位的判断，别看反了噢
                A <= A + 0;
            end
            3'b001,3'b010 : begin//加法操作使用补码即可，倍数利用左移解决
                A <= A + ({{8{mul1[`size-1]}},mul1} << 2*(clk_cnt-1));
            end
            3'b011 : begin
                A <= A + ({{8{mul1[`size-1]}},mul1} << 2*(clk_cnt-1) + 1);
            end
            3'b100: begin//减法操作利用“负补码”改成加法操作，倍数利用左移解决
                A <= A + ({{8{bmul1[`size-1]}},bmul1} << 2*(clk_cnt-1) + 1);
            end
            3'b101,3'b110 : begin
                A <= A + ({{8{bmul1[`size-1]}},bmul1} << 2*(clk_cnt-1));
            end
            default: A <= 0;
        endcase
    end
 //当节拍到4的时候写入结果寄存器。
    assign res = (clk_cnt == 3'b100) ? A : 0;
endmodule
```

3. 请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的?

> [!ANS]
> 修改P5中的Stall逻辑，当乘除槽处于start或busy状态，且ID级指令需要使用乘除槽时阻塞

```
    wire Stall_md = ((busy | start) & (mult_ID | multu_ID | div_ID | divu_ID | mfhi_ID |
mflo_ID | mthi_ID | mtlo_ID));
    assign Stall = Stall_RS_EX | Stall_RS_MA | Stall_RT_EX | Stall_RT_MA | Stall_md;
```

4. 请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

> [!ANS]
> 更符合实际CPU电路，在实际电路中这样做更加高效。对于外部存储单元，其只需要读取使能信号然后进行相应存储即可，无需增加其他存储逻辑，使外部设备有较为统一的存储逻辑，便于扩展。

5. 请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

> [!ANS]
> 不是一字节。当存储容量较小时按字节读和写会更加高效, 不需要添加多余的逻辑。

6. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

> [!ANS]
>
> 1. 在高层模块的排布中就屏蔽了实现的细节，只保留了此模块的端口，以及端口的功能。
> 2. 对于同一类型的指令增加多位宽type信号，功能模块根据不同type信号执行对应操作。例如：涉及乘除槽的指令，我设置一个mdOp信号来区分指令，从而减少了流水寄存器的接口数目，从而降低复杂度。
> 3. 是否阻塞全部放到冲突处理模块判断，尽可能遵循“高内聚低耦合”的原则。
> 4. 乘除槽与ALU分离，但是输出路径一致。不需要更改转发逻辑。

7. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

> [!ANS]
> 涉及到HI和LO寄存器的指令与其他读写寄存器指令冲突。通过设置合理的阻塞逻辑解决。

```
ori $2, $0, 128
lw $3, 0($0)
mult $3, $2
```

8. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略**，比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

> [!ANS]
> 我才用的方案是手动构造样例确保新指令的功能正确性，随机生成指令强测确保转发和阻塞的正确性。限制读写寄存器的范围，能够尽可能多的造成数据冒险，再加上大量随机数据，基本可以覆盖全部转发和阻塞情况。

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。