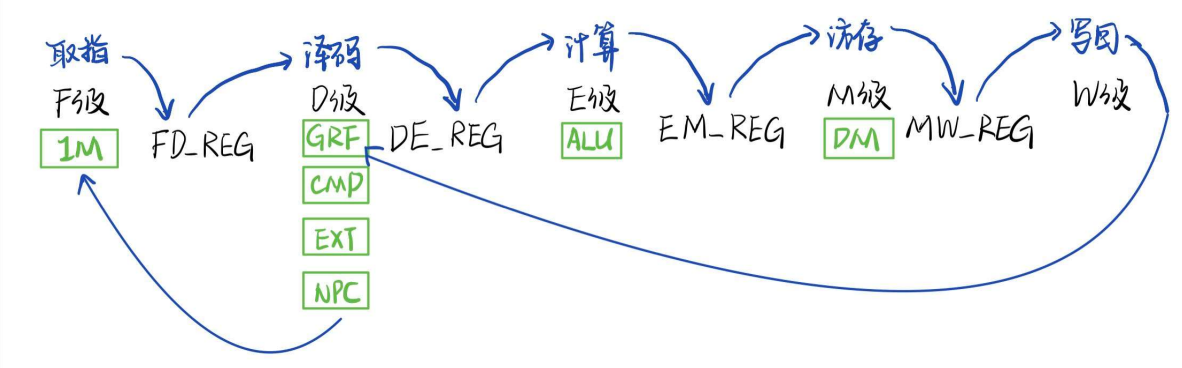
计算机组成原理实验报告

一、CPU设计方案综述

（一）总体设计概述

本CPU为Verilog实现的流水线MIPS - CPU，支持的指令集包含{addu,subu,ori,lw,sw,beq,lui,j,jal,jr,nop}。为了实现这些功能，CPU主要包含了IM,GRF,CMP,EXT,NPC,ALU,DM，各级流水线寄存器。



（二）关键模块定义

F级：

1、PC

- PC功能：输出当前位于F级的指令的地址
- 同步复位后，PC复位到0x0000_3000

端口名	端口输入\输出信号	端口描述
input clk	clk	时钟信号
input reset	reset	同步复位信号，将PC复位到0x0000_3000
input PC_en	~Stall	暂停信号，即PC的使能端。当PC_en为0时，PC在时钟上升沿不改变值
input [31:0] D_NPC	D_NPC	D级NPC计算出的下一条指令所在地址
output [31:0] F_PC	F_PC	当前F级指令所在地址

2、IM

- IM功能：保存当前程序所有指令，并输出当前F级指令

信号名	信号描述
input [31:0] F_PC	当前F级指令在IM中的地址
output [31:0] F_IM_instr	当前F级指令，即IM[PC[13:2]]

3、FD_REG

流水内容：F_instr，F_PC

使能信号：EN = ~STALL

D级：

1、GRF

功能：

a.读出数据（通过D_GRF_A1和D_GRF_A2读出D_GRF_RD1和D_GRF_RD2），支持内部转发，即将W级即将在下一个时钟周期写入的数据直接读出

b.【W级】写入数据（当W_RegWrite == 1且处于时钟上升沿时，将W级指令中W_GRF_WD写入W_GRF_A3（非0号寄存器！）

c.同步复位清零

信号名	信号描述
input clk	时钟信号
input reset	同步复位信号
input [4:0] D_GRF_A1	D级指令的第一个5位寄存器读数地址输入信号
input [4:0] D_GRF_A2	D级指令的第二个5位寄存器读数地址输入信号
input [4:0] W_GRF_A3	W级指令的5位寄存器写入地址输入信号
input [31:0] W_GRF_WD	W级指令的32位写入数据
input W_RegWrite	W级指令的grf写使能信号
input [31:0] W_PC	当前W级指令在IM中的地址
output [31:0] D_GRF_RD1	D级指令从a1号寄存器读出的数据
output [31:0] D_GRF_RD2	D级指令从a2号寄存器读出的数据

2、EXT

功能：对16位立即数进行扩展

信号名	信号描述
input [15:0] D_EXT_imm16	D级指令的16位立即数输入
input [2:0] D_EXTop	扩展功能选择信号
output [31:0] D_EXT_imm32	D级指令扩展结果

3、CMP

功能：对于跳转指令，通过比较进行特定条件的判断决定是否跳转。

信号名	信号描述
input [31:0] D_CMP_A	
input [31:0] D_CMP_B	
input [4:0] D_CMPop	分支类型
output [31:0] D_CMP_sig	比较结果的输出，分支或不分支

4、NPC

功能：计算下一条指令的地址

信号名	信号描述
input [31:0] D_PC	确定跳转的指令：D_PC //
input D_cmp_sig	对branch类指令是否分支的标志
input [25:0] D_imm26	
input [31:0] reg_rs	跳转寄存器时，该寄存器内的数据
input [2:0] NextPCType	在CU中得到NPC的计算方式
output [31:0] D_NPC	计算得到的下一条指令地址，传回F级
input [31:0] F_PC	不跳转：F_PC

5、D_CU

信号名	信号描述
input [31:0] D_instr	
output [26:0] D_imm26/16	
output [4:0] D_rs_addr	
output [4:0] D_rt_addr	
output [4:0] shamt	
output [2:0] NextPCType	下一条指令地址计算类型（j/branch/normal...）
output [1:0] D_EXTop	立即数扩展类型
output [2:0] D_CMPop	比较功能选择，即是哪一种branch类指令

output D_cal_r(i)/D_load/D_store/D_branch/D_j_to_reg : 将指令划分为R型计算指令、I型计算指令、内存访问指令、Branch类指令和Jump类指令分析。为下文转发、阻塞提供便利。

6、DE_REG

流水内容：D_instr[31:0] , D_PC[31:0] , D_FW_rs[31:0] (转发更新过的rs值) ， D_FW_rt[31:0] (转发更新过的rt值), D_EXT_imm32[31:0]

控制信号*reset = reset | STALL （当暂停时要将DE_REG流水寄存器清空flush）

E级：

1、 ALU

信号名	信号描述
E_FW_rs	E_ALU_A
E_ALU_B	
ALUControl	运算方式
E_ALU_res	计算结果

2、E_CU

信号名	信号描述
input [31:0] E_instr	
output [4:0] E_rs_addr	E_rs_addr用于与后两级可能写入的地址进行对比，用于转发。
output [4:0] E_rt_addr	
output [2:0] ALUSrc_B	选择ALUB端口运算数，1：E_FW_rt 2：E_EXT_imm32
output ALUControl	运算选择
output E_RegDst	写回寄存器堆的地址来源
output E_MemtoReg	写入寄存器数据选择信号 0：DM_out 1: PC+8 2: EXT 3: ALU

3、EM_REG

流水内容：E_instr[31:0]， E_PC[31:0]， E_ALU_res[31:0]， E_FW_rt[31:0]， E_EXT_imm32[31:0]

M级：

1、DM

信号名	信号描述
input M_MemWrite	写使能信号
input [31:0] M_DM_A	M_ALU_res
input [31:0] M_DM_WD	M_FW_rt
input [31:0] M_PC	
output [31:0] M_DM_out	
input [1:0] load_sel	lw,lh,lb选择信号
input [1:0] store_sel	sw,sh,sb选择信号

2、M_CU

信号名	信号描述
input [31:0] M_instr	
output [4:0] M_rt_addr	
output M_MemWrite	
output [2:0] M_MemtoReg	写回寄存器堆数据的选择信号
output [31:0] M_RegDst	写回寄存器堆的地址来源
output M_load	判断M级指令是否为load型指令
output [1:0] load_sel	lw,lh,lb选择信号
output [1:0] store_sel	sw,sh,sb选择信号

3、MW_REG

流水内容：M_instr[31:0] , M_PC[31:0] , M_ALU_res[31:0] , M_DM_out[31:0] , M_EXT_imm32[31:0]

W级：

1、W_CU

信号名	信号描述
input [31:0] W_instr	
output [2:0] W_MemtoReg	写回寄存器堆数据的选择信号
output W_RegWrite	写回寄存器堆的写使能信号
output [31:0] W_RegDst	写回寄存器堆的地址来源

####

（三）重要机制实现方法

1. 阻塞

产生原因：解决转发没有办法解决的问题——新数据根本没有产生**【Tuse < Tnew 且满足转发条件】**，只能阻塞后面的指令（将指令暂停在D级）减缓流水直到需要的数据产生再恢复正常。

实现：对PC_en, FD_REG_en,DE_REG_reset进行操作。Tuse：**指令进入 D 级后，其后的某个功能部件再经过多少时钟周期就必须要使用寄存器值。**Tnew：**位于 E 级及其后各级的指令，再经过多少周期就能够产生要写入寄存器的结果。**

```
assign  Tuse_rs = (D_branch | D_j_to_reg) ? 3'd0 :
                  (D_cal_r | D_cal_i | D_load | D_store) ? 3'd1 : 3'd6;
assign  Tuse_rt = (D_branch) ? 3'd0 :
                  (D_cal_r) ? 3'd1 :
                  (D_store) ? 3'd2 : 3'd6;

wire [2:0] Tnew_E = (E_load) ? 3'd2 :
                    (E_cal_r | E_cal_i) ? 3'd1 : 3'd0;
wire [2:0] Tnew_M = (M_load) ? 3'd1 : 3'd0;
```

```
wire stall_rs_1 = ((Tuse_rs < Tnew_E) & (D_rs_addr == E_RegDst) & (E_RegDst != 5'd0));
wire stall_rs_2 = ((Tuse_rs < Tnew_M) & (D_rs_addr == M_RegDst) & (M_RegDst != 5'd0));
wire stall_rt_1 = ((Tuse_rt < Tnew_E) & (D_rt_addr == E_RegDst) & (E_RegDst != 5'd0));
wire stall_rt_2 = ((Tuse_rt < Tnew_M) & (D_rt_addr == M_RegDst) & (M_RegDst != 5'd0));
assign STALL = (stall_rs_1 | stall_rs_2 | stall_rt_1 | stall_rt_2);
```

2. 转发

产生原因：部分先写再读的情况出现时，需要的（读）GRF[rs]值或者GRF[rt]值在后边已经被算出来，但没来得及在通用寄存器堆里更新，用转发可以立即得到最新的GRF[rs]值或者GRF[rt]值从而得到正确计算结果。

实现：将寄存器堆的读出值GPR[rs]和 GPR[rt]不断地在各个流水级寄存器之间流水并且存储，然后通过后续指令生成的控制信号X_RegDst与当前指令的rs_addr、rt_addr来判断是不是要发生转发。

需要转发的端口：D_CMP 模块的两个输入、E_ALU 模块的两个输入和 M_DM 模块的M_DM_WD

GRF的内部转发：

```
//read
assign D_GRF_RD1 = (D_GRF_A1 == 5'd0) ? 32'd0 :
                   (D_GRF_A1 == W_GRF_A3) ? W_GRF_WD : REG[D_GRF_A1];

assign D_GRF_RD2 = (D_GRF_A2 == 5'd0) ? 32'd0 :
                   (D_GRF_A2 == W_GRF_A3) ? W_GRF_WD : REG[D_GRF_A2];
```

3、控制信号总表

控制信号	说明
input [31:0] instr	当前流水的指令
[25:0] imm26	立即数字段
[15:0] imm16	立即数字段
[4:0] rs_addr	指令对应rs寄存器编号
[4:0] rt_addr	指令对应rt寄存器编号
[4:0] rd_addr	指令对应rd寄存器编号
[4:0] shamt	对应偏移量字段
cal_r	R型计算指令：addu,subu
cal_i	I型计算指令：ori
load	从DM中读取数据：lw,lb,lh
store	向DM存数据：sw,sb,sh
branch	进行特定条件判断再决定是否跳转的指令：beq
j_to_reg	跳转到特定寄存器的指令，一般是将GPR[rs]作为PC跳转地址： jr
j_to_addr	跳转到指定地址的指令： j
j_and_link	跳转到指定地址并链接(GPR[31] <- PC + 8) : jal
Lui	lui指令，在EXT模块即可得到结果
ALUSrc_A	logical shift:1(GPR[rt]) 0:GPR[rs]
[2:0] ALUSrc_B	logical shift:0(shamt); cal_r:1(GPR[rt]); cal_i/lui/store/load:2(EXT); 否则，3:(GPR[rt])
MemWrite	DM写使能信号，store型指令时为1
[3:0] ALUControl	
[2:0] MemtoReg	写入GRF数据的选择：load:0(DM_OUT) j-link:1(PC+8) lui:2(EXT_OUT) 其他（alu运算结果）:3(ALU_res)
RegWrite	寄存器堆的写使能信号，RegDst不为0的时候就为1
[4:0] RegDst	写入GRF地址来源 cal_r: rd_addr; cal_i/lui/load: rt_addr; j-link:31; 否则为0
[2:0] NextPCType	决定下一个PC值的类型：branch:0 j/j-link:1 jr:2 否则：3(normal)
[1:0] Ext	

控制信号	说明
[2:0] D_CMPop	beq:0 其他：7
[1:0] load_sel	
[1:0] store_sel	

二、测试方案

（一）典型测试样例

1、单条指令测试：

ori:

```
55@00003000: $ 8 <= 00000001
65@00003004: $ 9 <= 00000000
75@00003008: $11 <= 00007fff
85@0000300c: $12 <= 0000ffff
```

addu:

```
ori $0, $0, 1
ori $1, $1, 0
ori $3, $3, 0x7fff
ori $4, $3, 0xffff
addu    $5, $4, $3
addu    $6, $4, $4
```

subu:

```
ori $3, $3, 0x7fff
ori $4, $3, 0xffff
subu $5, $4, $3
subu $6, $0, $4
```

lui:

```
lui $4, 0xffff
lui $2, 0x4678
lui $0, 0xffff
lui $3, 0xffff
```

```
55@00003000: $ 4 <= ffff0000
65@00003004: $ 2 <= 46780000
85@0000300c: $ 3 <= ffff0000
```

跳转:

```
jal A
ori $s1, $s1, 0x1111
addu $ra, $ra, $t2
ori $s1, $s1, 0x2222
A:
lui $t1, 200

jal t
ori $t0, $t0, 0x1111
subu $t1, $t1, $t0
ori $t2, $t2, 0x1111
beq $t2, $t2, B
nop
t:
```

sw/lw:

```
ori $t0, $t0, 6
addu $t1, $t0, $t0
subu $t2, $t0, $t1
addu $t3, $t2, $t2
subu $t4, $t0, $t3
sw $t4, 16($t4)
sw $t3, -4($t1)
lw $t5, 16($t4)
lw $t6, -4($t1)
```

2、冲突测试：依据讨论区测试样例testpoint1~10分别测试与std_ans对拍。

```
@00003000: $ 8 <= 0000007b
@00003004: $ 4 <= 0000007b
@00003008: $ 4 <= 00000000
@0000300c: $11 <= 007b0000
@00003010: $11 <= 00000000
@00003014: $12 <= 00000000
@00003018: $ 8 <= 00003028
@0000301c: $ 4 <= 00003028
@00003024: $ 8 <= 00003038
@00003028: *00000000 <= 00003038
@0000302c: $ 4 <= 00003038
@00003038: $31 <= 00003040
@00003050: $31 <= 00003058
@00003058: $ 8 <= 00003038
@0000305c: $16 <= 00003038
@00003060: $17 <= 00003038
@00003064: $18 <= 00003038
@00003068: *00000000 <= 00003038
```

时间对比： std_ans:

```
34875@00006644: $ 4 <= 0000664c
34905@0000664c: $ 6 <= 33650c24
34915@0000664c: $ 6 <= 33650c24
34925@00006650: $ 6 <= 000015c2
34935@00006654: $13 <= 00008928
34945@00006658: $ 4 <= 0e820000
34955@0000665c: $ 1 <= 5fb20000
34965@00006660: $ 5 <= bf640000
34975@00006664: $22 <= bf640000
34985@00006668: $18 <= 1f160000
```

my_ans:

```
34785@00006644: $ 4 <= 0000664c
34815@0000664c: $ 6 <= 33650c24
34825@0000664c: $ 6 <= 33650c24
34835@00006650: $ 6 <= 000015c2
34845@00006654: $13 <= 00008928
34855@00006658: $ 4 <= 0e820000
34865@0000665c: $ 1 <= 5fb20000
34875@00006660: $ 5 <= bf640000
34885@00006664: $22 <= bf640000
34895@00006668: $18 <= 1f160000
```

三、思考题

1、 在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

PC的值由有组合逻辑模块NPC负责产生，NPC的输入是PC值(F_PC+D_PC)、立即数和寄存器数据。当非跳转分支指令时，NPC的来源是F_PC，当是跳转分支指令时，NPC的来源是D_PC。这是因为F_PC是D_PC的后一条地址，我们在普通情况，想要的是F_PC的下一条地址，所以是F_PC+4。D_PC+4一般就是F_PC，如果是D_PC+4就是把F_instr发送了两遍，是错误的。对于跳转分支指令，要是用F_PC，就相当于用了D_PC + 4，显然是错误的。

对于数据通路：

```
module D_NPC(
    input [31:0] D_PC,
    input [31:0] F_PC,
    input [25:0] D_imm26,
    input [31:0] reg_rs,
    input [2:0] NextPCType,
    input D_cmp_sig,
    output [31:0] D_NPC
     );
```

对于控制信号，需要产生一个NextPCType信号选择NPC计算方式，D_cmp_sig确定是否进行分支，这个信号在分支跳转指令的时候置1，其他时候置0。

2、对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

因为延迟槽的存在，所以跳转分支指令的下一条指令一定会被执行，为了避免这一条指令被重复执行，所以真正的jal指令的下一条指令是PC+8。

3、为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据?

电路具有延迟性，不论是ALU的计算结果还是取出的DM中的数据都是发生在时钟周期的后半段，如果直接从ALU或者DM部件取数据，取出来的大概率是错误数据。同时，例如add(D)-add(E),需要在E级完成计算才能传给D级开始D级的执行过程，会延长流水线的最长的执行阶段，即影响流水线效率。

4、“转发（旁路）机制的构造”中的 Thinking 1-4：

Thinking 1：为了实现转发机制，我们对这些输入前加上一个 MUX。这些 MUX 的默认输入来源是上一级中已经转发过的数据。如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。）

比如：

```
ori  $2,$2,1
addu $3,$2,$2
```

Thinking2：GPR 是一个特殊的部件，它既可以视为 D 级的一个部件，也可以视为 W 级之后的流水线寄存器。基于这一特性，我们将对 GPR采用内部转发机制。也就是说，当前 GPR 被写入的值会即时反馈到读取端上。我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢?

内部转发解决的是在一个周期内W存D读同一个寄存器的情况。如果不采用内部转发机制，可以用半写半读解决。

Thinking3：选择信号的生成规则是：只要当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0为。什么 0 号寄存器需要特殊处理?

因为0号寄存器的值始终是0，不能改变。

比如：

```
ori  $0,$0,1
addu $3,$0,$0
```

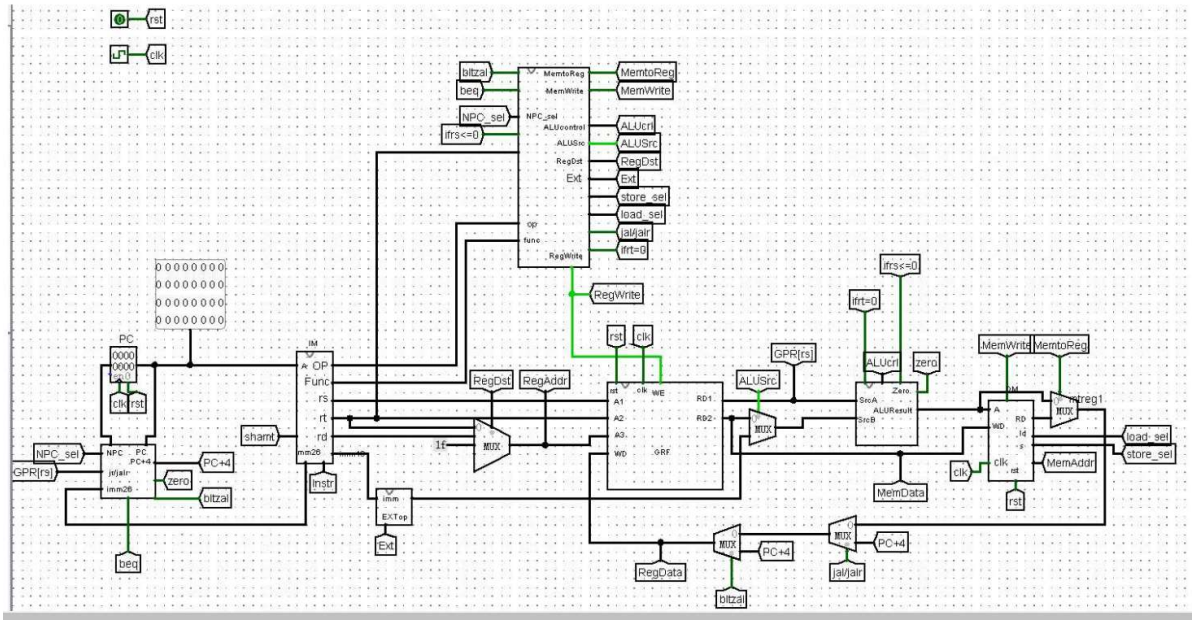
此时不能将ALU_res转发!

Thinking4：在有多个转发输入来源都满足条件时，最新产生的数据优先级最高。什么是“最新产生的数据”?

离当前指令所在级最近的下一级流水线寄存器所产生的值。

5、在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢?

如果翻译出的we（W_RegWrite）信号为0，则代表着GRF_A3为0，此时无论该指令是对0号寄存器操作，还是该指令不涉及写寄存器操作，都不对寄存器做任何改变。


rst
clk
bltzal
beg
NPC_sel
ifrs<=0
MemtoReg
MemWrite
ALUctrl
ALUSrc
RegDst
Ext
store_sel
load_sel
jal/jalr
ifrt=0
RegWrite
PC
IM
OP
Func
rs
rt
rd
imm26
shamt
Instr
RegDst
RegAddr
1f
MUX
GRF
RD1
RD2
WE
A1
A2
A3
WD
GPR[rs]
ALUSrc
ALUctrl
zero
SrcA
SrcB
ALUResult
MemWrite
MemtoReg
DM
RD
WD
rt/reg1
load_sel
store_sel
clk
MemAddr
MemData
rst
NPC_sel
GPR[rs]
NPC
PC
PC+4
jal/jalr
imm26
zero
bltzal
beg
imm
EXTop
Ext
RegData
PC+4
jal/jalr
bltzal