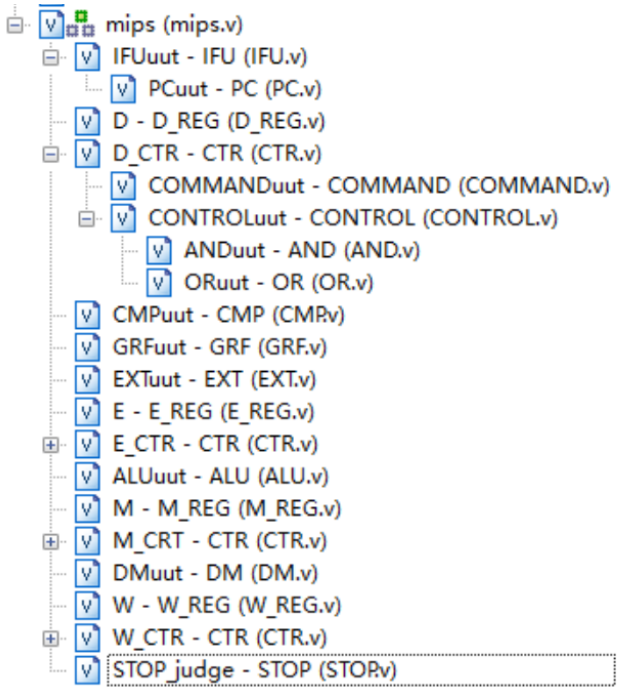
设计文档

设计草稿

ISA包括 add、sub、ori、lw、sw、beq、lui、nop、jal、jr

- Verilog文件的结构如下：



冒险的解决

阻塞

阻塞的目的是解决不能用转发解决的问题，如lw下一条指令是beq

- T_use、T_new在CONTROL的OR逻辑模块中用组合逻辑直接输出值

T_use

T_use	rs	rt
add	1	1
sub	1	1
ori	1	inf
lw	1	inf
sw	1	2
beq	0	0
lui	inf	1
jal	inf	inf
jr	0	inf
nop	inf	inf

- inf置为3

T_new

T_new	E	M	W
add	1	0	0
sub	1	0	0
ori	1	0	0
lw	2	1	0
sw	/	/	/
beq	/	/	/
lui	1	0	0
jal	0	0	0
jr	/	/	/
nop	/	/	/

- /处置为0

阻塞的判断

```
assign E_stop_rs = (E_A3 == D_rs) && (D_rs != 0) && (E_T_new > T_use_rs);
assign E_stop_rt = (E_A3 == D_rt) && (D_rt != 0) && (E_T_new > T_use_rt);
assign M_stop_rs = (M_A3 == D_rs) && (D_rs != 0) && (M_T_new > T_use_rs);
assign M_stop_rt = (M_A3 == D_rt) && (D_rt != 0) && (M_T_new > T_use_rt);

assign stop = E_stop_rs | E_stop_rt | M_stop_rs | M_stop_rt;
```

注：此处的A3都指A3_target（判断的逻辑与是否产生数据无关）

阻塞的操作

- 清空E_REG（等同于插入nop）
- 冻结D_REG
- 冻结IFU

转发

转发的具体思路就是将未写入GRF内寄存器reg[A3]的、但已经可以稳定输出（已经在REG中）的值提供给需要reg[A3]最新值的指令。需要注意的是转发目标可以是D、E、M级

- 转发部分实现如下

```
//E/M-->D
assign D_Forward_RD1 = (D_rs == 0)? 0:
                       (D_rs == E_A3)? E_WD:
                       (D_rs == M_A3)? M_WD:D_RD1;
assign D_Forward_RD2 = (D_rt == 0)? 0:
                       (D_rt == E_A3)? E_WD:
                       (D_rt == M_A3)? M_WD:D_RD2;
//M/W-->E
assign E_Forward_RD1 = (E_rs == 0)?0:
                       (E_rs == M_A3)? M_WD:
                       (E_rs == W_A3)? W_WD:E_RD1;
assign E_Forward_RD2 = (E_rt == 0)?0:
                       (E_rt == M_A3)? M_WD:
                       (E_rt == W_A3)? W_WD:E_RD2;
//W-->M
assign M_Forward_RD2 = (M_rt == 0)?0:
                       (M_rt == W_A3)? W_WD:M_RD2;
```

- W-->D在GRF内部

- 转发部分参考了学长的[博客](P5 课下学习 — 流水线 CPU 设计 (1) - 北航计算机组成原理 | Test Blog = FlyingLandlord's Blog)

主要模块

主要模块包含CTR、GRF、ALU、IFU、DM、EXT、STOP、REG（D、E、M、W）、CMP

新增模块说明

CTR

- 由于采用了分布式译码，要在不同级实例化CONTROL与COMMAND，因此将两者合并为一个模块多次进行实例化

控制信号列表

	add	sub	ori	lw	sw	beq	lui	jal	jr
EXT_op	x	x	0	1	1	x	2	x	x
ALU_op	0	1	2	0	0	1	0	x	x
PC_op	0	0	0	0	0	1	0	2	3
DM_WE	0	0	0	0	1	0	0	0	0
GRF_WE	1	1	1	1	0	0	1	1	0
GRF_addr	1	1	0	0	x	x	0	2	x
GRF_data	0	0	0	1	x	x	0	2	x
ALU_src	0	0	1	1	1	0	1	x	x

- 控制信号整体保留了P4的设计

端口说明

端口名称	方向	大小	说明
command	input	[31:0]	这一级执行的指令
stage	input	[1:0]	调用CTR的阶段
rs	output	[4:0]	command[25:21]
rt	output	[4:0]	command[20:16]
rd	output	[4:0]	command[15:11]
imm15	output	[15:0]	command[15:0]
imm25	output	[25:0]	command[25:0]
控制信号	output	[n:0]	如上表
T_use_rs	output	[1:0]	此指令rs的T_use
T_use_rt	output	[1:0]	此指令rt的T_use
T_new	output	[1:0]	此指令在本阶段的T_new
A3	output	[4:0]	此指令将要写入的寄存器编号
A3_target	output	[4:0]	此指令将要写入的寄存器编号

- A3与A3_target的区别：A3确保数据已经产生且指令为写寄存器指令

```
assign A3 = (valid) ? A3_target : 0;
```

STOP

- STOP模块负责判断何时阻塞

端口说明

端口名称	方向	大小	说明
T_use_rs	input	[1:0]	rs的T_use
T_use_rt	input	[1:0]	rt的T_use
E_T_new	input	[1:0]	E级T_new
M_T_new	input	[1:0]	M级T_new
W_T_new	input	[1:0]	W级T_new（暂未使用）
D_rs	input	[4:0]	/
D_rt	input	[4:0]	/
E_A3	input	[4:0]	/
M_A3	input	[4:0]	/
stop	output	[0:0]	阻塞信号

CMP

- 分支指令从E级转移到了D级，因此添加了CMP模块判断是否跳转

端口说明

端口名称	方向	大小	说明
RD1	input	[31:0]	D_RD1
RD2	input	[31:0]	D_RD2
PC_op	input	[1:0]	PC_op = 1即为beq
zero	output	[0:0]	是否跳转到PC+4+imm15

- RD1与RD2都是转发过后的值

层级结构

F级（Fetch）

- 包含IFU、CTR_F

IFU

- NPC计算方法：

```
assign F_NPC = (D_zero === 1)?(D_PC + 4 + {{14{D_imm15[15]}},D_imm15,2'b00})://beq
               (D_PC_op === 2)?({D_PC[31:28],D_imm25,2'b00})://jal
               (D_PC_op === 3)?(D_Forward_RD1)://jr
               (F_PC + 4);//others
```

D级（Decode）

- 包含GRF _R、CMP、CTR_D、REG_D

E级（Execute）

- 包含ALU、EXT、CTR_E、REG_E

M级（Memory）

- 包含DM、CTR_M、REG_M

W级（WriteBack）

- 包含GRF_W、CTR_W、REG_W

注：GRF_R与GRF_W只是为了区分GRF功能，实际上实例化的GRF只能有一个

一些信号的说明

GRF

控制信号

1. GRF_addr

   选择对寄存器A3进行输入

GRF_addr	A3
0	rt
1	rd
2	31

2. GRF_data

选择输入寄存器的值WD

GRF_data	WD
0	ALU_result
1	DM_out
2	PC+4

3. GRF_WE

   选择是否写入GRF

GRF_WE	Write?
0	√
1	×

ALU

控制信号

1. ALU_src

   选择ALU的运算数B

ALU_src	B
0	ALU_RD2
1	EXT_out

2. ALU_op

   选择ALU的运算R

ALU_op	R
0	add
1	sub
2	or

IFU

控制信号

1. PC_op

   选择下一条指令的PC计算方式（根据执行的指令C）

PC_op	C
1	beq
2	jal
3	jr

DM

控制信号

1. DM_WE

   选择是否写入DM

DM_WE	Write?
0	√
1	×

EXT

控制信号

1. EXT_op

   选择EXT的方式

EXT_op	WD
0	zero_ext
1	sign_ext
2	<<16

CONTROL

控制信号列表

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。

    - 因为b类指令的T_use从1变成了0，很容易出现T_new>T_use的情况，流水线被阻塞，此时其效率等同于不提前判断分支的流水线

    ```
    ori $t0,$0,1023
    ori $t1,$0,1023
    beq $t0,$t1,lable
    ...
    lable:
    ```

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？

    - jal的下一条指令（PC+4）是延迟槽中的指令，在jal执行时会进入流水线。如果回写PC+8，则在执行 jr $ra 时会重复执行延迟槽中的内容。

3. 我们要求大家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？

    - 只有流水寄存器的输入是即时的、稳定的、可靠的输入。功能部件的输出有延迟、不稳定、不可靠，若作为转发来源数据，则可能会引发写异常

4. 我们为什么要使用 GRF 内部转发？该如何实现？

    - GRF内部转发实现了W-->D的转发

    - 实现如下：

    ```
    assign RD1 = (A1 == 0) ? 32'b0 :
                 (A1 == A3)? WD : REG[A1];
    assign RD2 = (A2 == 0) ? 32'b0 :
                 (A2 == A3)? WD : REG[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？

- 见上文转发部分

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。

   要更改以下模块：

   - CTR模块内的CONTROL模块，调整控制信号与T_use、T_new的逻辑
   - STOP模块，调整阻断的逻辑
   - 根据指令种类调整
     - 计算指令：在E级修改ALU模块
     - 跳转指令：在D级修改CMP模块
     - 储存指令：在M级修改DM模块输入

7. 确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。

   本人采用了分布式译码。

   - 优势：较为灵活，“现译现用”且有效降低了流水级间传递的信号量，便于书写与修改
   - 缺点：在DEMW级都需要译码，会增加代码量，可能会产生一些不易发现的bug

测试样例

1. ORI

```
ori $t0, $t0, 1
ori $t1, $t1, 2333
ori $t2, $t2, 0x7fff
ori $t3, $t2, 0xffff
```

2. ADD

```
ori $t0, $t0, 2
ori $t1, $t1, 0xffff
add $t2, $t0, $t0
add $t3, $t1, $t1
```

3. SUB

```
ori $t0, $t0, 2
ori $t1, $t1, 0xffff
sub $t2, $t0, $t0
sub $t3, $t1, $t1
sub $t4, $t1, $t0
sub $t4, $t0, $t1
```

4. LUI

```
lui $t0, 0xffff
ori $t0, $t0, 0xffff
lui $t1, 1023
lui $t2, 2578
```

5. JAL&JR

```
jal label
nop
ori $t0,$0,1023
ori $t2,$0,20
add $ra,$t2,$ra
label:
lui $t1,0xffff
jr $ra
ori $t1,0xffff
```