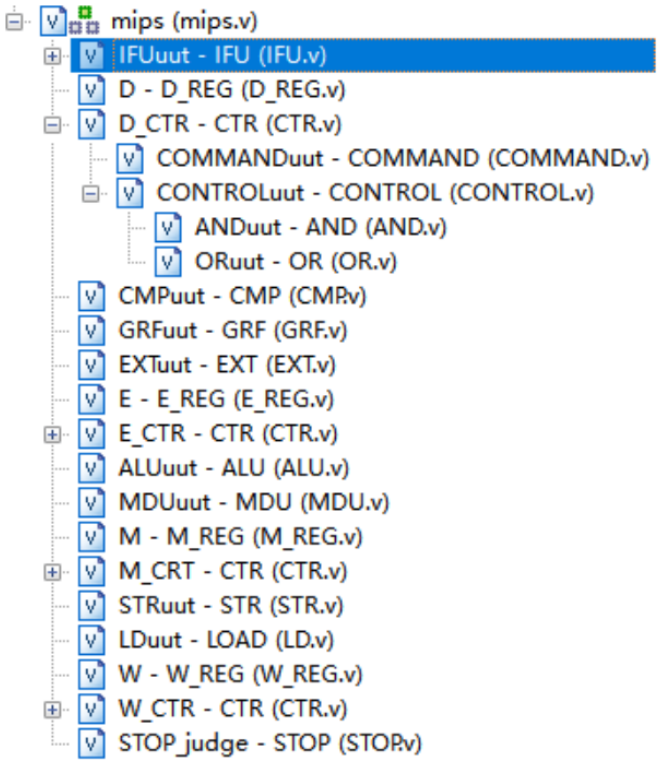
设计文档

设计草稿

ISA包括 add、addi、sub、and、andi、or、ori、slt、sltu、lw、lh、lb、sw、sh、sb、mult、multu、div、divu、mthi、mtlo、mfhi、mflo、beq、bne、lui、jal、jr、nop

- Verilog文件的结构如下：



冒险的解决

因为加入了乘除法，对阻塞部分进行更新

阻塞

T_use

T_use	rs	rt
alu_R	1	1
alu_I	1	inf
load	1	inf
md	1	1
mt	1	inf
mf	inf	inf
save	1	2
branch	0	0
lui	inf	inf
jal	inf	inf
jr	0	inf
nop	inf	inf

- inf置为3

T_new

T_new	E	M	W
alu	1	0	0
load	2	1	0
save	/	/	/
md	/	/	/
mt	/	/	/
mf	1	0	0
branch	/	/	/
lui	1	0	0
jal	0	0	0
jr	/	/	/
nop	/	/	/

- /处置为0

阻塞的判断

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

主要模块

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

新增模块说明

STR

端口说明

端口名称	方向	大小	说明
STR_op	input	[1:0]	区分sw、sh、sb的控制信号
addr	input	[15:0]	ALU得到的address
data	input	[31:0]	GRF读出、未处理的data
STR_WE	output	[3:0]	字节使能
STR_out	output	[31:0]	STR处理后的data

控制信号

STR_op	操作
1	sw
2	sh
3	sb

LOAD

端口说明

端口名称	方向	大小	说明
LOAD_op	input	[1:0]	区分lw、lh、lb的控制信号
addr	input	[15:0]	ALU得到的address
data	input	[31:0]	DM读出、未处理的data
LOAD_out	output	[31:0]	准备写入寄存器的data

控制信号

LOAD_op	操作
1	lw
2	lh
3	lb

MDU

端口说明

端口名称	方向	大小	说明
clk	input	[0:0]	时钟信号
res	input	[0:0]	同步复位
mt	input	[0:0]	mthi/mtlo
start	input	[0:0]	开始乘除法的控制信号
MDU_op	input	[2:0]	MDU的控制信号
A	input	[31:0]	操作数1
B	input	[31:0]	操作数2
HI	output	[31:0]	regHI
LO	output	[31:0]	regLO
busy	output	[0:0]	MDU在执行乘除法中

控制信号

MDU_op	操作
0	mtlo
1	mthi
2	mult
3	multu
4	div
5	divu

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	HI
4	LO

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
3	and
4	slt
5	slu

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	branch
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. 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？
   - 因为实际CPU中的乘除模块速度显著慢于加减与或运算，整合进ALU会降低CPU的运行速度
   - 因为乘除法需要两个寄存器来保存结果，为MDU模块提供HI、LO两个寄存器可以让MDU模块在不影响其他模块运行的情况下单独执行指令
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。
   - 在真实的流水线CPU中，乘法模块由多个乘法器组成，每个周期计算4位乘法，计算结果在几个周期后累加得到最终结果
   - 除法使用试商法，每个周期计算 4 位的商，经过 8 个周期计算结束。
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？

```
E_stop_MDU = (((busy == 1)|(start == 1))&(E_md|E_mf|E_mt));
```

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）

- 将两控制种信号合二为一（DM_op、DM_EN），省去了使用DM_op判断SW、SH、SB的过程，且更清晰地体现了待储存的字节
- 便于和DM模块统一控制信号

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？
   - 不是1byte，因为两个端口都是[31:0]大小，在写入前与读出都都需要对数据进行一些处理再使用
   - 按字读写仅使用按字使能，需要额外的读写控制信号进行数据处理，会使数据通路延长，降低读写效率
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？
   - 译码时采取了宏定义、指令分类（alu、alu_imm、alu_R...）手段
   - mips.v中将F、D、E、M、W级用注释分隔，分级书写
   - 减少了代码量且使代码可读性提高，降低了添加指令的难度
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？
   - 遇到了与乘除指令相关的冲突
     1. md、mf、mt之间

        若MDU的busy高电平，则直接将需要使用MDU的指令阻塞在D级

     2. mf与其他指令之间

        在CTR中添加mf的T_new，在转发模块中加入HI、LO的转发
   - 测试样例见结尾
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。
   - 由于alu指令、load/store指令的正确性已经在p5检验，因此只需要测试由MDU的加入引发的冲突与新添加指令的正确性

- 手动构造样例：

  ①测试所有新加入指令的正确性：逐条检验即可

  ②测试MDU冲突：mf类+branch/store类、alu/load类 + mt/md类、md/mt/mf类

  ③load、store单独进行测试

- 借用了自动评测机进行复查无误

测试样例

```
addi $t0,$t0,1023
add $t1,$t0,$t0
sub $t2,$0,$t1
and $t3,$t0,$t0
andi $t4,$t3,511
or $t5,$t3,$t4
ori $t6,$t5,1024
lui $s0,0xffff
lui $s1,0xffff
ori $s0,$s0,0xffff
ori $s1,$s1,0xfffe
slt $s2,$s1,$s0
sltu $s3,$s1,$s0
ori $s1,$0,1
slt $s2,$s1,$s0
sltu $s3,$s1,$s0
mult $t0,$t1
mfhi $s4
mflo $s5
div $t1,$t0
mfhi $t6
mflo $t7
mthi $t2
mtlo $t3
mfhi $s4
mflo $s5
divu $s4,$s4
mthi $0
mtlo $0
mfhi $t1
mflo $t2
beq $t1,$t2,lable_0
nop
addi $t1,$t1,1
lable_0:
addi $t2,$t2,1
bne $t1,$t2,lable_1
nop
addi $t1,$t1,1
lable_1:
addi $t1,$t1,1
```