P6 Document

设计草稿

Hazard

以E和D冒险为例:

1. rs_D/rt_D != 0
2. RegWrite_E == 1
3. rs_D = WriteReg_E

Tuse

指令	rs	rt
add	1	1
sub	1	1
and	1	1
or	1	1
slt & sltu	1	1
addi	1	/
andi	1	/
ori	1	/
lui	1	/
lw & lh & lb	1	/
sw & sh & sb	1	2
beq & bne	0	0
jr	0	/
jal	/	/
mult & multu	1	1
div & divu	1	1
mthi & mtlo	1	/
mfhi & mflo	/	/

Tnew

部件	E	M	W
PC	0	0	0
ALU	1	0	0
MDU	1	0	0
DM	2	1	0

> ALU: add, sub, and, or, slt, sltu, addi, andi, ori, lui
> MDU: mfhi, mflo
> DM: lw, lh, lb
> PC: jal
> 无: sw, sh, sb, beq, bne, jr, mthi, mtlo
> 单独判断：mult, multu, div, divu

CTRL

```
input wire [5:0] op;
input wire [5:0] func;
reg [21:0] tmp;
```

指令	RegWrite	RegDst	ALUSrc	MemOp	WDSrc1	WDSrc2	NPCOp	ALUOp	EXTOp	CMPOp
add(000000+100000)	1	01	0	000	00	0	000	000	00	00
sub(000000+100010)	1	01	0	000	00	0	000	001	00	00
and(000000+100100)	1	01	0	000	00	0	000	011	00	00
or(000000+100101)	1	01	0	000	00	0	000	010	00	00
slt(000000+101010)	1	01	0	000	00	0	000	100	00	00
sltu(000000+101011)	1	01	0	000	00	0	000	101	00	00
addi(001000)	1	00	1	000	00	0	000	000	01	00
andi(001100)	1	00	1	000	00	0	000	011	00	00
ori(001101)	1	00	1	000	00	0	000	010	00	00
lui(001111)	1	00	1	000	00	0	000	000	10	00
lw(100011)	1	00	1	100	00	1	000	000	01	00
lh(100001)	1	00	1	101	00	1	000	000	01	00
lb(100000)	1	00	1	110	00	1	000	000	01	00
sw(101011)	0	00	1	001	00	0	000	000	01	00
sh(101001)	0	00	1	010	00	0	000	000	01	00
sb(101000)	0	00	1	011	00	0	000	000	01	00
beq(000100)	0	00	0	000	00	0	001	001	00	00
bne(000101)	0	00	0	000	00	0	001	001	00	01
jal(000011)	1	10	0	000	01	0	010	000	00	00
jr(000000+001000)	0	00	0	000	00	0	011	000	00	00
md & mt	0	00	0	000	00	0	000	000	00	00
mfhi & mflo	1	01	0	000	10	0	000	000	00	00

指令	MDUOp
mult	0000
multu	0001
div	0010
divu	0011
mfhi	0100
mflo	0101
mthi	0110
mtlo	0111
无关	1111

注：`add`与`sub`实际上为`addu`与`subu`，不考虑溢出情况。

md: mult, multu, div, divu
mt: mthi, mtlo

RegDst_E

[4:0] WriteReg_E

> 00: rt_E
> 01: rd_E
> 10: 31

ALUSrc_E

[31:0] ALUb

> 0: WriteData_E
> 1: EXTOut_E

WDSrc1_E

[31:0] ALUOut_E

> 00: ALUResult
> 01: PC_E + 8
> 10: MDUOut

WDSrc2_W

[31:0] WD

> 0: ALUOut_W
> 1: DMOut_W

NPC

NPCOp	NPC
00	PC_F + 4
01	PC_D + 4 + sign_extend(offset||00)
10	PC_D31..28||instr_index||00
11	GPR[rs_D]

ALU

ALUOp	运算
000	加
001	减
010	或
011	与
100	小于置1（有符号）
101	小于置1（无符号）

MDU

MDUOp	运算	延迟
0000	有符号乘	5
0001	无符号乘	5
0010	有符号除	10
0011	无符号除	10
0100	读HI	0
0101	读LO	0
0110	写HI	0
0111	写LO	0

EXT

EXTOp	扩展
00	零扩展
01	符号扩展
10	加载至高位

BE & DE

MemOp	运算
000	DM无关
001	写字
010	写半字
011	写字节
100	读字
101	读半字
110	读字节

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？

因为乘除法所需时间较普通运算长，整合进ALU会使ALU所在流水级延时过长，使得运行效率降低；使用独立的寄存器，可让cpu在进行乘除法计算的同时进行其他与乘除法无关的指令。

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。

通过二进制的位运算，循环移位以及加减法实现的。

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？

当E级检测到乘除法指令时即将Start信号置1，经过一个周期后在MDU模块中将Busy信号置1，经过相应周期后Busy信号置0。

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）

统一性：将所有load类指令的实际行为均由字节使能信号控制，使得所有的写指令均能通过一种方法实现，行为更加统一，符合其功能统一的特点。

清晰性：四位字节使能信号每一位对应一个字的一个字节。0代表对位字节不进行写操作，1代表对位字节进行写操作。从字节使能信号就能直接看出需要对一个字的哪几个字节进行写入操作，较为清晰。

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？

实际从DM获得的数据和向DM写入的数据都是由一个字，在字的数据上进行加工达到按字节读和写的效果。当我们只需要对字节或半字访问时，按字节访问内存性能更有优势。如果此时还采用按字访问，则需要首先将整个字从内存中拿出来，然后再从字中寻找，效率会降低。

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？

抽象手段：在译码器译码时将指令根据执行模块，调用寄存器和具体行为等进行归类，例如将mult,div等归为md类，mfhi,mflo归为mf类。同一类指令的控制信号类似，只有在个别控制信号上有区别。在确定控制信号执行或逻辑时更加清楚也更加方便进行迭代。

规范手段：在数据通路实例化各个组成部分并进行连接时，按流水线等级顺次实例化，使得流水线的层级更加清楚。同时在命名信号时也都在后缀标注流水级，如PC_M。