# 流水线CPU设计文档

## CPU设计方案综述

支持指令：

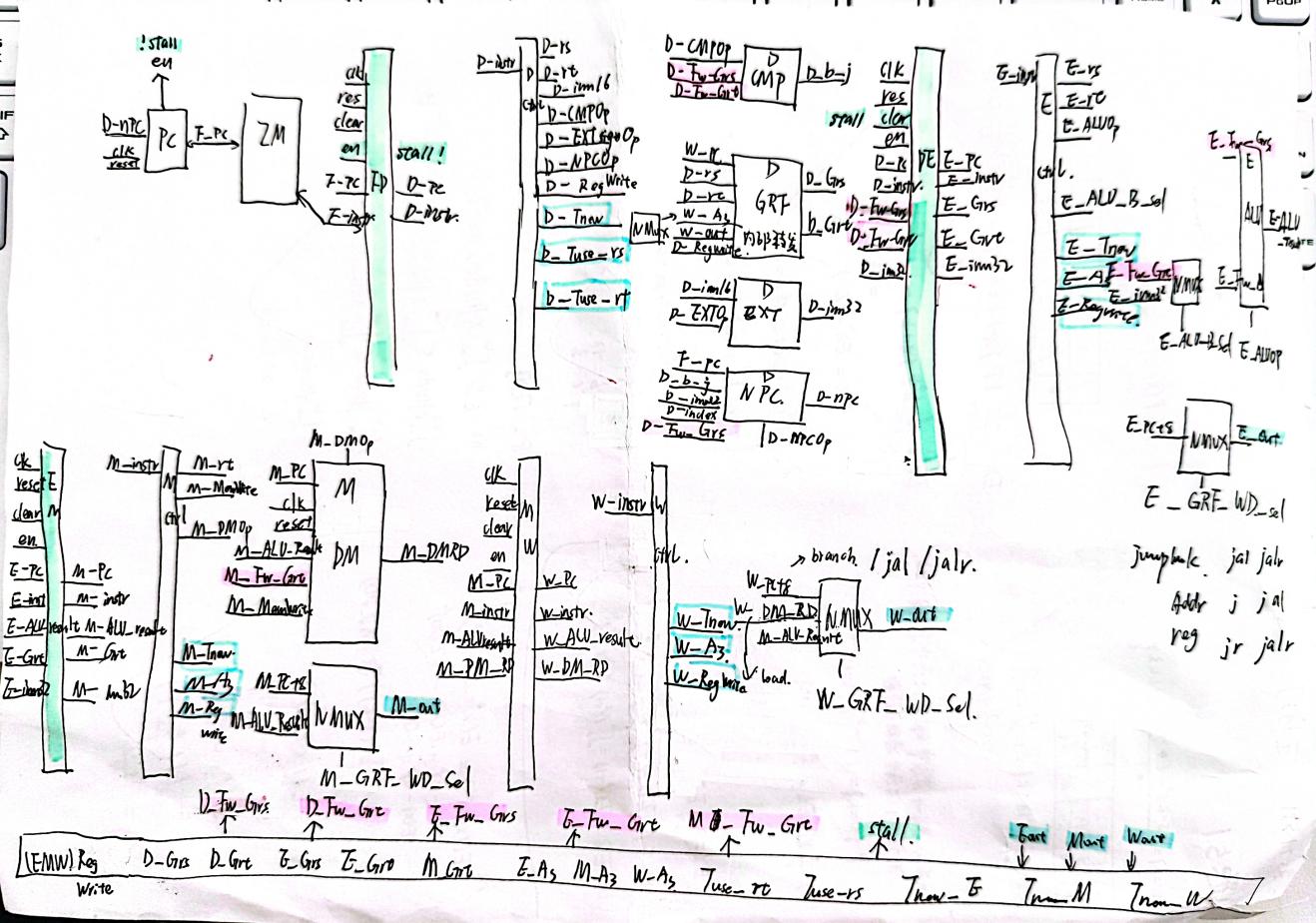
add, sub, and, or, slt, sltu, lui

addi, andi, ori

lb, lh, lw, sb, sh, sw

mult, multu, div, divu, mfhi, mflo, mthi, mtlo

beq, bne, jal, jr



### 官方Tb的造成的改动

#### Mips.v

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号，将PC值置为**0x00003000**  高电平有效 |
| i\_inst\_rdata | I | F\_instr |
| M\_data\_rdata | I | DE的din 未经处理的M\_DM\_out |
| i\_inst\_addr | O | F\_pc |
| m\_data\_addr | O | M\_ALU\_result |
| m\_data\_wdata | O | M\_BE 直接输出 |
| m\_data\_byteen | O | M\_BE 直接输出 |
| m\_inst\_addr | O | M\_pc |
| w\_grf\_we | O | W\_RegWrite |
| w\_grf\_addr | O | W\_A3 |
| w\_grf\_wdata | O | W\_out |
| w\_inst\_addr | O | W\_pc |

assign i\_inst\_addr = F\_pc;

assign F\_instr = i\_inst\_rdata;

assign m\_inst\_addr = M\_pc;

assign m\_data\_addr = M\_ALU\_result;

assign w\_grf\_we = W\_RegWrite;

assign w\_grf\_addr = W\_A3;

assign w\_grf\_wdata = W\_out;

assign w\_inst\_addr = W\_pc;

### 功能模块

#### F\_PC

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号，将PC值置为**0x00003000**  高电平有效 |
| 3 | D\_npc[31:0] | I | 要更新的PC值 |
| 4 | F\_pc[31:0](reg) | O | 现在的PC值 |

#### D\_NPC

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | F\_pc[31:0] | I | 32位地址信号 |
| 2 | D\_b\_j | I | A1(rs)、A2(rt)对应GRF寄存器中的值是否相等  1：相等  0：不相等 |
| 3 | D\_imm32[31:0] | I | 16位immediate的32位**符号扩展** （beq） |
| 4 | D\_index[25:0] | I | 26位instr\_index（j、jal） |
| 5 | D\_Fw\_Grs[31:0] | I | rs寄存器保存值（jalr、jr）PC  GPR[rs] |
| 6 | D\_NPCOp[2:0] | I | NPC将会是  000：PC+4  001：PCbranch（beq）  010：PC31..28instr\_index02 PCJ(al)(jal、j)  011：GPR[rs] PCJ(al)r(jalr、jr) |
| 7 | D\_npc[31:0] | O | 下一个更新的PC值 |

#### D\_GRF

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | W\_pc[31:0] | I | pc 32位地址 |
| 2 | clk | I | 时钟信号 |
| 3 | reset | I | 同步复位信号，将寄存器置为**0**；高电平有效 |
| 4 | W\_RegWrite | I | 写使能信号  1:可向 GRF 中写入数据  0:不能向GRF 中写入数据 |
| 5 | A1[4:0] | I | rs对应寄存器 |
| 6 | A2[4:0] | I | rt对应寄存器 |
| 7 | A3[4:0]  (RegAddr)  W\_A3 | I | RegDst决定:  000：rt  001：rd（R型add、sub、jalr）  010：$ra（jal） |
| 8 | WD[31:0]  (RegData)  W\_out | O | MemtoReg决定:  000：MemAddr / ALUresult  001：RD(DmData)  010:PC+4(jal、jalr)  32位数据输入信号 |
| 9 | RD1[31:0]  D\_Grs | O | 输出A1指定的寄存器中的32位数据 |
| 10 | RD2[31:0]  D\_Grt | O | （$rt）  输出A2指定的寄存器中的32位数据 |

#### D\_CMP

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | D\_CMPOp[2:0] | I |  |
| 2 | D\_Fw\_Grs[31:0] | I |  |
| 3 | D\_Fw\_Grt[31:0] | I |  |
| 4 | D\_b\_j | O |  |

#### D\_EXT

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | imm[15:0] | I | 原16位的立即数信号 |
| 2 | EXTsign | I | 无符号或符号扩展的选择信号  0：无符号扩展  1：符号扩展 |
| 3 | imm32[31:0] | O | 扩展后的32位的立即数信号 |

#### E\_ALU

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | A[31:0]  E\_ALU\_B | I | RD1  参与运算的第一个数 |
| 2 | B[31:0]  E\_ALU\_A | I | ALUSrc：  0：E\_Fw\_Grt  1：E\_imm32  参与运算的第二个数 |
| 3 | E\_ALUOp[2:0] | I | 000：无符号加  001：无符号减  010：或立即数  011：lui |
| 5 | E\_ALUresult[31:0] | O | MemAddr  A与B做运算后的结果 |

#### E\_MD

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | clk | I |  |
| 2 | reset | I |  |
| 3 | start | I | E\_is\_md |
| 4 | A[31:0]  E\_Fw\_Grs | I |  |
| 5 | B[31:0]  E\_Fw\_Grt | I |  |
| 6 | MDOp[3:0] | I |  |
| 7 | HI | O |  |
| 8 | LO | O |  |
| 9 | out | O | E\_MD\_out |
| 10 | busy | O |  |

#### M\_BE

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | MemAddr | I | M\_ALU\_result |
| 2 | Din | I | M\_Fw\_Grt |
| 3 | BEOp | I | M\_BEOp |
| 4 | m\_data\_wdata | O | 对写入数据处理 |
| 5 | m\_data\_byteen | O | 四位字节使能 |

#### M\_DE

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | MemAddr | I | M\_ALU\_result |
| 2 | Din | I | m\_data\_rdata 读出的RD |
| 3 | DEOp | I | M\_DEOp |
| 4 | dout | O | M\_DM\_RD |

### 流水线寄存器

#### FD\_Reg

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I |  |
| reset | I |  |
| clear | I | 0 |
| en | I | Stall! |
| F\_pc | I |  |
| F\_instr | I |  |
| D\_pc | O |  |
| D\_instr | O |  |

#### DE\_Reg

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I |  |
| reset | I |  |
| clear | I | stall |
| en | I | 1 |
| D\_pc | I |  |
| D\_instr | I |  |
| D\_Fw\_Grs | I |  |
| D\_Fw\_Grt | I |  |
| D\_imm32 | I |  |
| E\_pc | O |  |
| E\_instr | O |  |
| E\_Grs | O |  |
| E\_Grt | O |  |
| E\_imm32 | O |  |

#### EM\_Reg

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I |  |
| reset | I |  |
| clear | I | 0 |
| en | I | 1 |
| E\_pc | I |  |
| E\_instr | I |  |
| E\_ALUresult | I |  |
| E\_Grt | I |  |
| E\_imm32 | I |  |
| M\_pc | O |  |
| M\_instr | O |  |
| M\_ALUresult | O |  |
| M\_Grt | O |  |
| M\_imm32 | O |  |

#### MW\_Reg

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I |  |
| reset | I |  |
| clear | I | 0 |
| en | I | 1 |
| M\_pc | I |  |
| M\_instr | I |  |
| M\_ALUresult | I |  |
| M\_DM\_RD | I |  |
| M\_imm32 | I |  |
| W\_pc | O |  |
| W\_instr | O |  |
| W\_ALUresult | O |  |
| W\_imm32 | O |  |

### 分布式ctrl

#### D\_ctrl

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| D\_instrl[31:0] | I | D\_instr |
| D\_rs | O | rs地址 |
| D\_rt | O | rs地址 |
| D\_imm16 | O | Imm16 |
| D\_CMPOp | O | CMP选择 |
| D\_EXTOp | O | EXT选择 |
| D\_NPCOp | O | NPC选择 |
| D\_Tnew | O | Tnew |
| D\_Tuse\_rs | O | Tuse\_rs |
| D\_Tuse\_rt | O | Tuse\_rt |

#### E\_ctrl

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| E\_instrl[31:0] | I | E\_instr |
| E\_rs | O | rs地址 |
| E\_rt | O | rt地址 |
| E\_imm16 | O | Imm16 |
| E\_ALUOp | O | ALU选择 |
| E\_ALU\_B\_Sel | O | ALU\_B选择 |
| E\_Tnew | O | Tnew |
| E\_A3 | O | 写寄存器地址 |
| E\_RegWrite | O | 能不能写寄存器 |

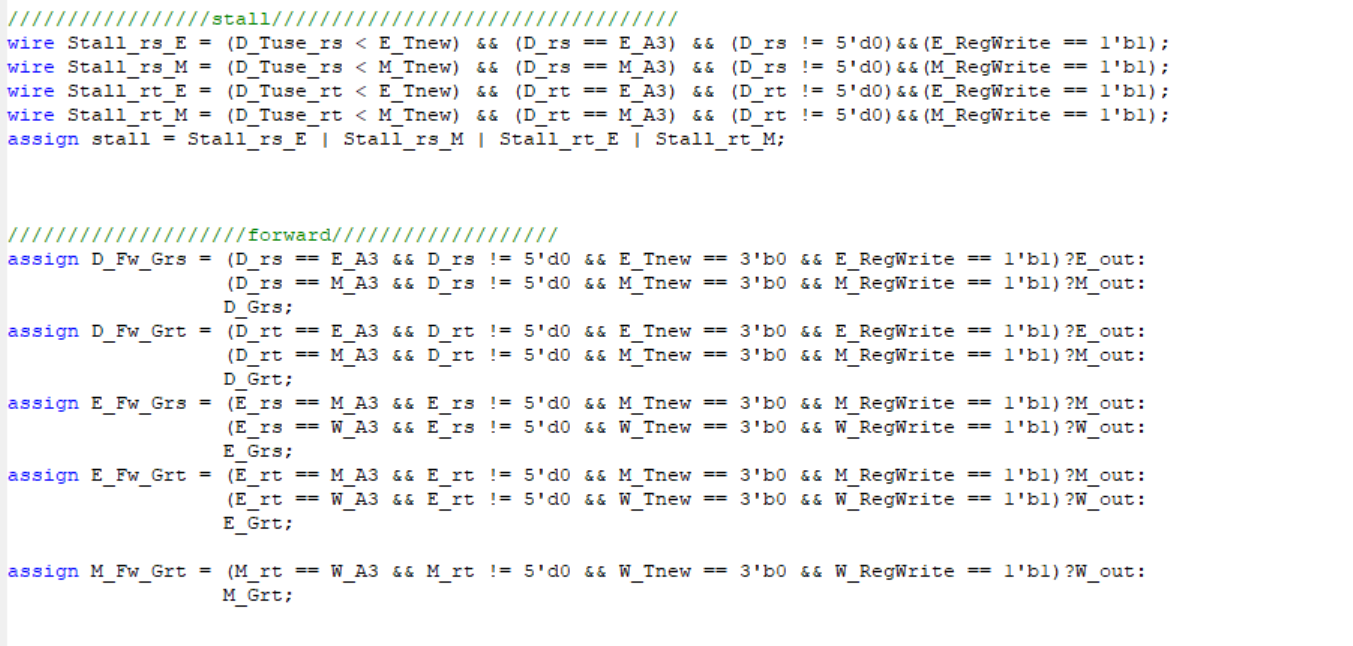
#### M\_ctrl

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| W\_instrl[31:0] | I | E\_instr |
| W\_rt | O | rt地址 |
| W\_ALUOp | O | ALU选择 |
| W\_ALU\_B\_Sel | O | ALU\_B选择 |
| W\_Tnew | O | Tnew |
| W\_A3 | O | 写寄存器地址 |
| W\_RegWrite | O | 能不能写寄存器 |

#### W\_ctrl

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| W\_instrl[31:0] | I | E\_instr |
| W\_rt | O | rt地址 |
| W\_Tnew | O | Tnew |
| W\_A3 | O | 写寄存器地址 |
| W\_RegWrite | O | 能不能写寄存器 |

### Hazard

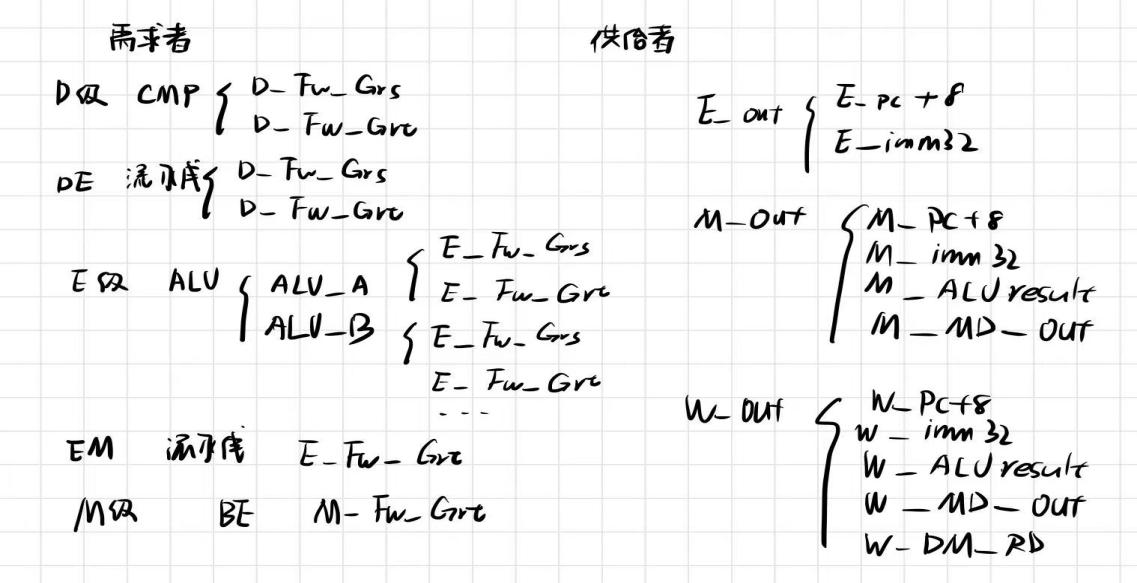


Tuse：指令进入 D 级后，其后的某个功能部件再经过多少时钟周期就必须要使用寄存器值

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

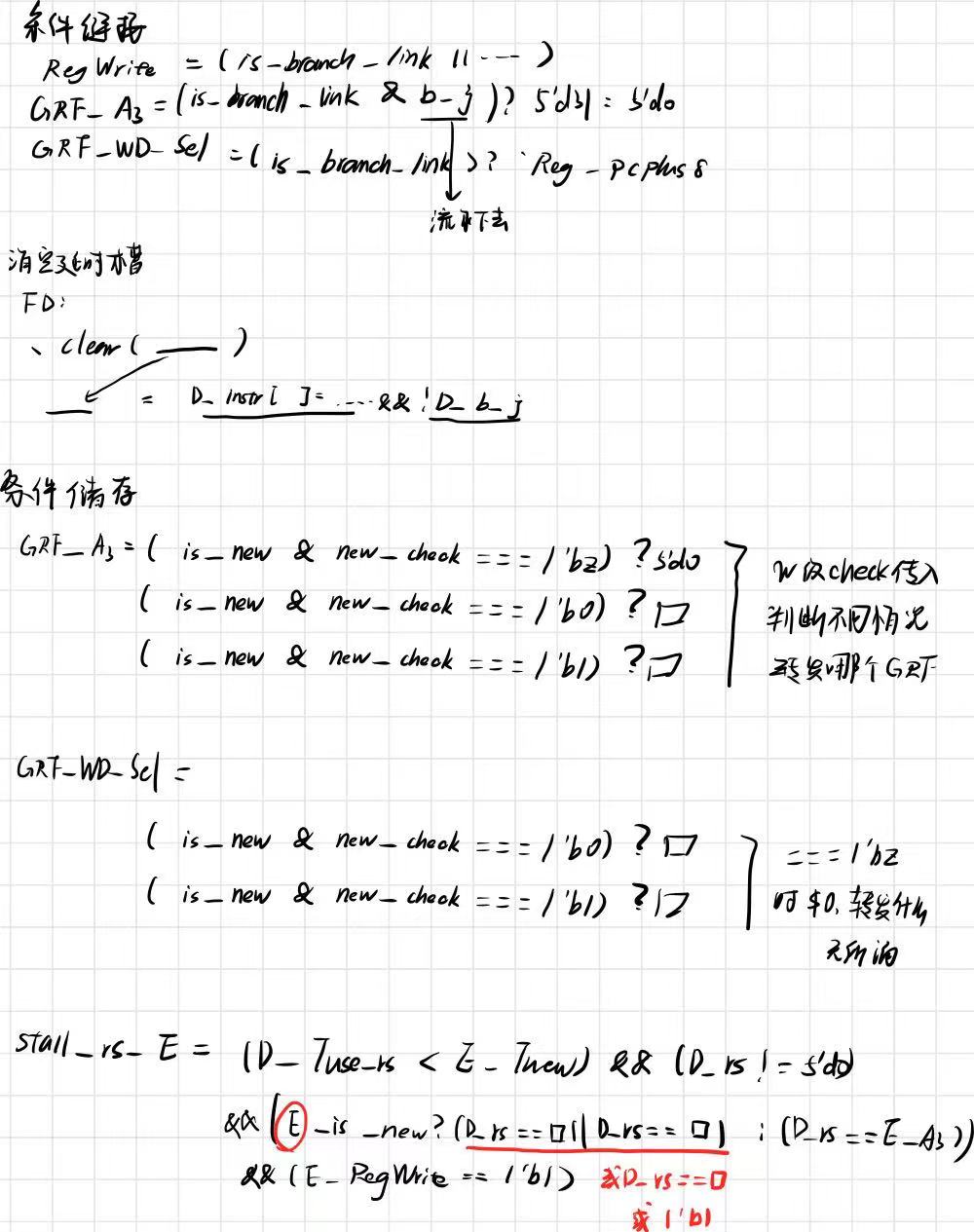
W 级的指令Tnew 恒为 0

Tnew@M = max(Tnew@E - 1, 0)



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 指令 | Tuse\_rs | Tuse\_rt | Tnew\_D | Tnew\_E | Tnew\_M | Tnew\_W |
| cacR | 1 | 1 | 2 | 1 | 0 | 0 |
| cacI | 1 | X5 | 2 | 1 | 0 | 0 |
| shift(R) | X5 | 1 | 2 | 1 | 0 | 0 |
| shiftv(R) | 1 | 1 | 2 | 1 | 0 | 0 |
| load | 1 | X5 | 3 | 2 | 1 | 0 |
| store | 1 | 2 | X0 | X0 | X0 | X0 |
| branch | 0 | 0 | X0 | X0 | X0 | X0 |
| jal | X5 | X5 | 0 | 0 | 0 | 0 |
| jr | 0 | X5 | X0 | X0 | X0 | X0 |
| jalr | 0 | X5 | 0 | 0 | 0 | 0 |
| j | X5 | X5 | X0 | X0 | X0 | X0 |
| Lui（I） | X5 | X5 | 1 | 0 | 0 | 0 |
| md | 1 | 1 | X0 | X0 | X0 | X0 |
| mf | X5 | X5 | 2 | 1 | 0 | 0 |
| mt | 1 | X5 | X0 | X0 | X0 | X0 |

### 新增指令



## 测试方案

**code.txt 一定要放在dictionary下面！！！！！！！！！！**

**ALU与跳转冲突**

ori $2,$0,4

sw $2,0($0)

lw $1,0($0)

add $1,$1,$2

（add $3,$1,$0）

beq $1,$2,labe1

add $1,$1,$2

labe2: beq $0,$0,labe2

nop

labe1: beq $0,$0,labe1

nop

**ALU之间冲突(自动化生成)**

label0: ori $a1, $a3, 50774

label1: ori $0, $v0, 32154

label2: add $a3, $v0, $v1

label9: add $a0, $a1, $0

label10: add $v1, $a2, $a3

label11: sub $a0, $a0, $a1

label15: add $v0, $0, $a2

label16: add $0, $v1, $v1

label18: add $0, $a1, $a3

label21: ori $v0, $v1, 58765

label23: sub $a3, $a2, $v0

**ALU与访存指令冲突（自动化生成）**

label6: ori $v1, $a1, 5544

label8: add $a0, $a0, $v1

label9: ori $a2, $0, 4260

label10: sw $a1, 140($a2)

label13: add $v0, $a2, $a1

label15: ori $v1, $0, 8923

label16: sw $a2, 397($v1)

label17: add $a1, $a2, $a0

label18: ori $a2, $0, 4045

label19: lw $a2, 435($a2)

label21: ori $a0, $0, 4580

label22: lw $0, -472($a0)

label23: ori $a0, $0, 11576

label24: sw $a0, 292($a0)

label25: ori $0, $0, 6729

label26: lw $a2, 204($0)

label29: ori $a1, $0, 10200

label30: sw $a2, 8($a1)

label32: ori $a2, $0, 7954

**jr跳转冲突**

ori $2,0x0004

ori $1,0x3014

add $1,$1,$2

jr $1

nop

add $1,$1,$2

label1:beq $0,$0,label1

nop

label2:beq $0,$0,label2

nop

# 测不动了听天由命

## 

## **思考题汇总**

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？**

若整合进[ALU](https://so.csdn.net/so/search?q=ALU&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_62275188/article/details/_blank)，在乘除运算延迟的过程中，无法执行其他指令

单独出来，在乘除运算延迟的过程中，其他无关指令可以并行执行

独立的HI,LO寄存器存放结果，简化了CPU的数据通路的复杂度

1. **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。**

通过多次加法实现乘法，每次加法均在上一次加法的结果(部分积)的高位部分进行，且根据乘数的最低位决定在加被乘数或加零；每完成一次加法，结果（部分积）和乘数均右移1位，循环执行。

中间余数R初始化为0，被除数最高有效位为R最低位。中间余数重复减去除数B，判断差D，D为负数，则商位Q\_i为0，忽略差，D为正数，Q\_i为，R更新为D。中间余数左移一位，A的下一个有效位为R的最低有效位，循环执行。

1. **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？**

D级是md/mf/mt指令 且 E级start或者busy 阻塞

wire Stall\_MD\_E = (D\_isMDFT && (E\_MD\_start || E\_MD\_busy));

1. **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：对于sb、sh、sw 通用,不用挨个按地址条件进行位拼接

清晰性：独热码，需要哪个字节一目了然

1. **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？**

否，是一个字，然后经过字节使能信号（写）、不同地址条件（读）处理

​ sb，sh，lb，lh等对于字节操作的指令，按字节的读写效率会更高

按字读写需要读出后进行位拼接，写入前进行位拼接，复杂的组合逻辑使效率降低。

1. **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？**

指令分类

wire is\_cacR = (is\_add || is\_sub || is\_slt||is\_sltu || is\_and || is\_or || is\_sll || is\_sllv || is\_slt || is\_sltu );

wire is\_cacI = (is\_andi || is\_addi || is\_ori || is\_lui);

……

控制信号驱动的译码器，控制信号的赋值先从大类进行出发，一步步进行细分。

宏定义替代判断选择信号的常量，让代码可读性更高

1. **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？**

R类型写rd 后续使用rs/rt

M向DE转发

W向EM转发

前文已包含样例

load写rt 后续使用rs/rt

W向EM转发

前文已包含样例

md和mt、mf

如果E级的Busy或Start信号有效，D级是乘除法相关指令，将它们阻塞在D级

 mult $1,$2

 mflo $3

**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。**

非跳转指令我用随机生成，但即使我将寄存器编号局限在1-8，也无法保证相邻的指令构成冒险高

跳转指令单独构造，避免死循环和延时槽的非法行为