# 流水线CPU设计文档

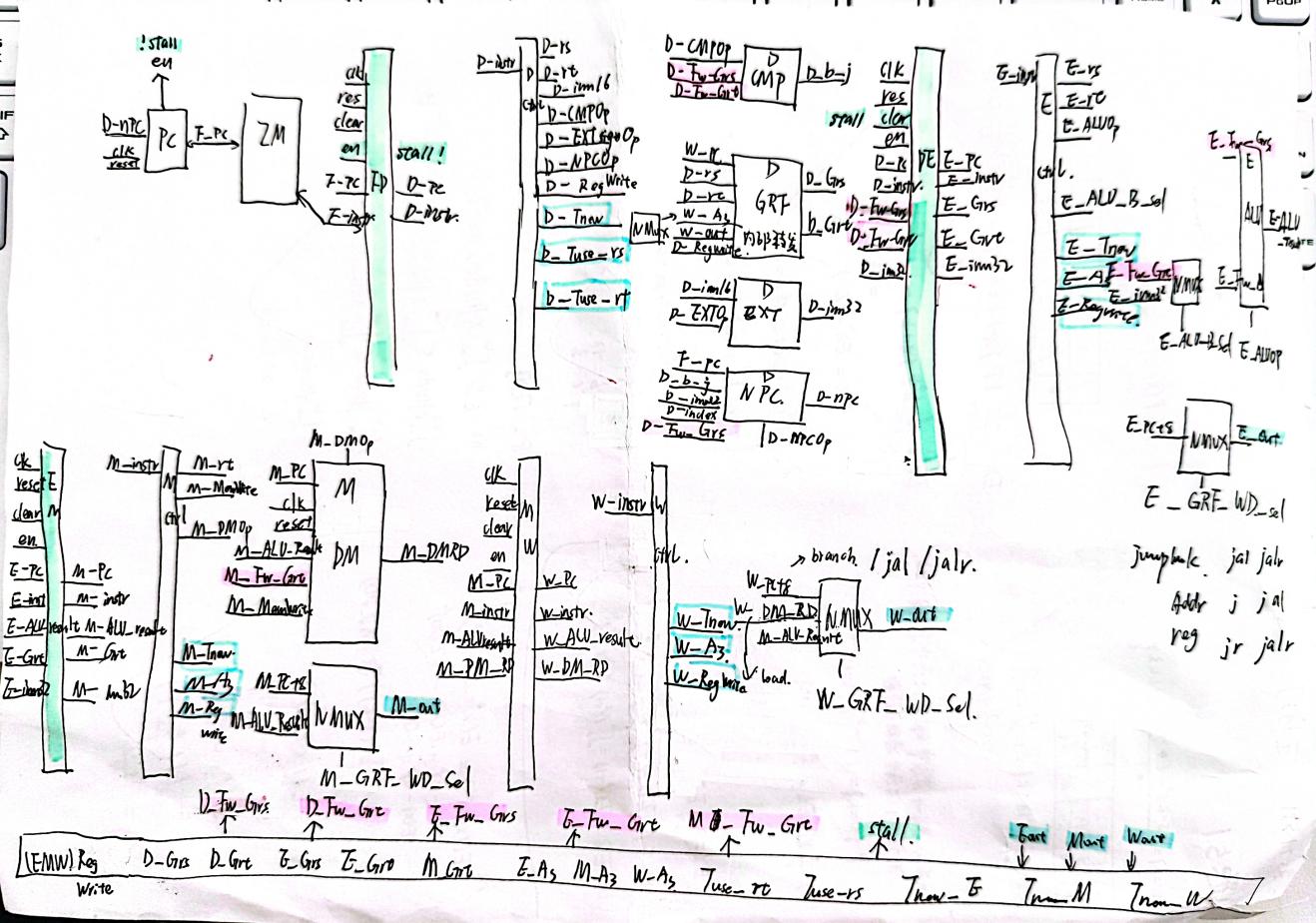
## CPU设计方案综述

支持指令：

addu、subu、ori、lui

sw、lw、sb、lb、sh、lh

beq、jal、j、jalr、jr



### 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 |

#### F\_IM

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | F\_pc[31:0] | I | 程序计数器信号 |
| 2 | F\_Instr[31:0] | O | 现在执行de机器码 |

#### 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\_Fw\_Grs | I | RD1  参与运算的第一个数 |
| 2 | B[31:0]  E\_Fw\_B | 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 |

#### M\_DM

| 序号 | 信号名 | 方向 | 描述 |
| --- | --- | --- | --- |
| 1 | M\_pc[31:0] | I | pc 32位地址信号 |
| 2 | clk | I | 时钟信号 |
| 3 | reset | I | 同步复位到0  高电平有效 |
| 4 | M\_MemWrite | I | DM的写入使能信号  0：禁止  1：允许（sh、b、w） |
| 5 | M\_ALU\_result[31:0] | I | ALUresult |
| 6 | M\_DMOp[2:0] | I | 向内存中读写方式  000：lw、sw  001：lh、sh  010：lb、sb |
| 7 | din[31:0]  M\_Fw\_Grt | I | DM写入32位数据 |
| 8 | dout[32:0]  M\_DM\_RD | O | DM读出32位数据 |

### 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 |

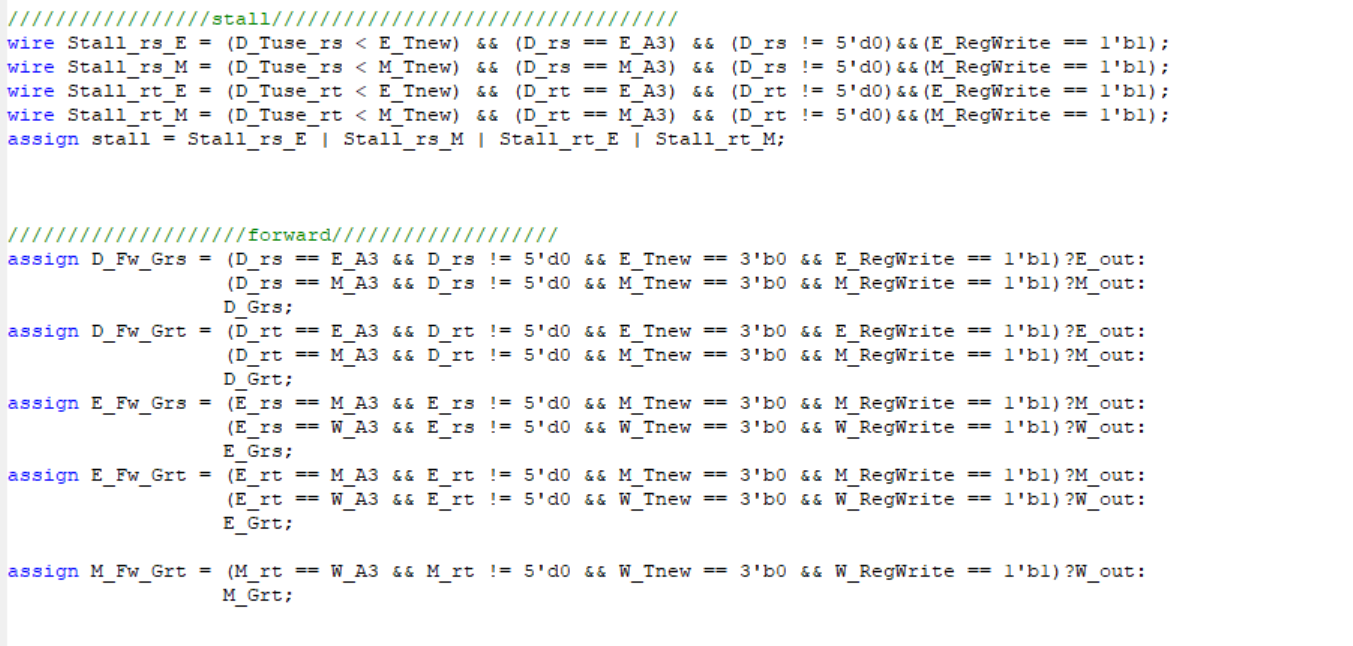
#### M\_ctrl

| 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 | X | 2 | 1 | 0 | 0 |
| shift | X | 1 | 2 | 1 | 0 | 0 |
| shiftv | 1 | 1 | 2 | 1 | 0 | 0 |
| load | 1 | X | 3 | 2 | 1 | 0 |
| store | 1 | 2 | X | X | X | X |
| branch | 0 | 0 | X | X | X | X |
| jal | X | X | 0 | 0 | 0 | 0 |
| jr | 0 | X | X | X | X | X |
| lui | X | X | 1 | 0 | 0 | 0 |
| jalr | 0 | X | 0 | 0 | 0 | 0 |

## 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. **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。**

**ori $1,$0,1**

**sw $1, 0($0)**

**lw $2, 0($0)  
beq $2, $0, label  
nop**

会有数据冒险，T\_new > T\_use还得阻塞

1. **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的下一条指令是延迟槽中的指令。如果写回PC+4的话，当出现jr $ra时，将会重复执行延迟槽中的指令。所以需要写回PC+8，跳回第一条没有被执行的指令。

1. **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？**

功能部件输出有延迟，如果在正确数据生成前转发写入了错误的数据，会导致数据的波动。流水寄存器中的输出数据是前一级输出的数据，在当前周期内时是稳定的。

1. **我们为什么要使用 GPR 内部转发？该如何实现？**

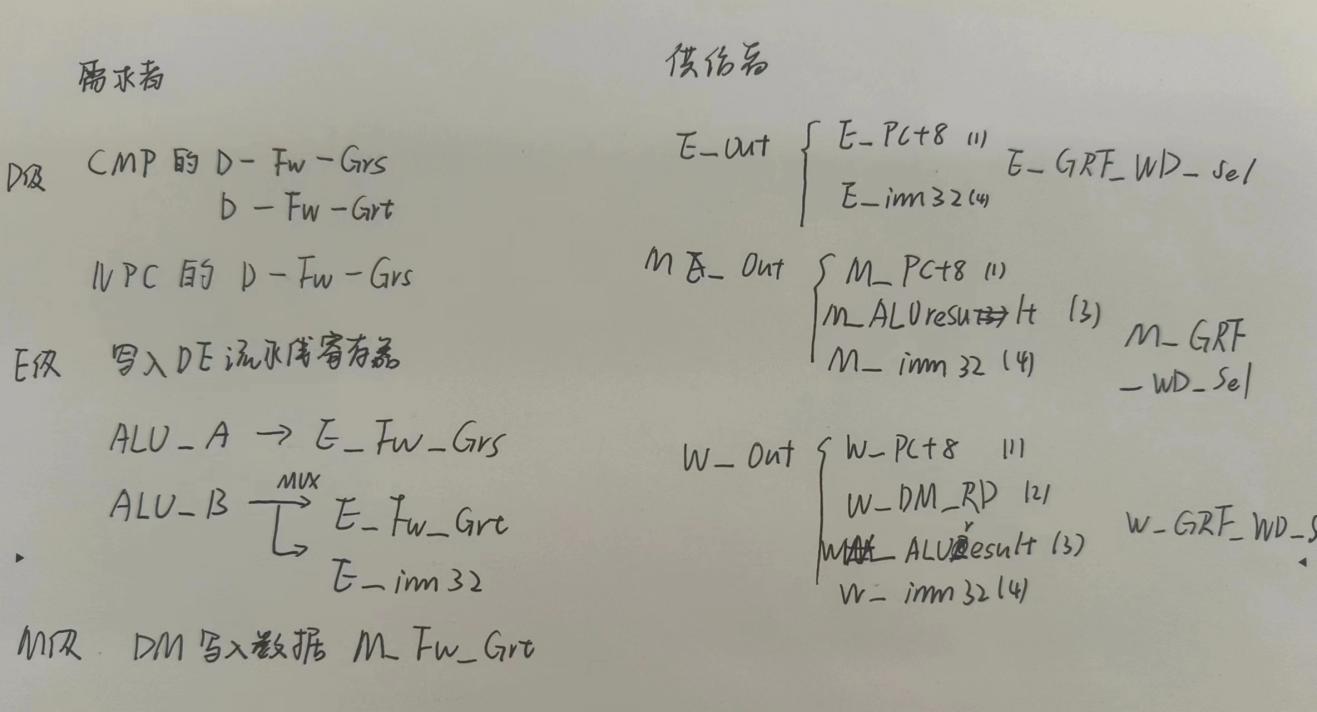
因为存在数据冒险，需要将W级写入寄存器操作转发到GRF

**实现：**

**assign RD1 = (RegWrite && A1!=0 && A1 == A3)?WD:grf[A1];**

**assign RD2 = (RegWrite && A2!=0 && A2 == A3)?WD:grf[A2];**

1. **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？**

****

1. **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。**

cacR/cacI指令：ctrl里Tnew、Tuse、ALUOp、RegWrite、A3、GRF\_WD\_Sel里具体写什么

访存指令：类比store、load复用

遇到条件写（DM种读出的数值决定了是否进行写入寄存器的操作）,把M级判断结果随instr一同流水到W，将W\_RegWrite与新增指令、是否写入信号一同控制。

Branch类指令：CMPOp

遇到条件写入 将D\_b\_j 一起流水到下一级，将E\_RegWrite与新增指令、是否写入信号一同控制，流到下一级。

遇到清空延时槽 reset>阻塞>清空

**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。**

分布式译码

每一级都部署一个控制器，负责译出当前级所需控制信号。见前文Ctrl端口定义。

优势：有效降低了流水级间传递的信号量，每一级需要用到的信号“现用现译”，AT法算Tnew很方便。

不足：需要实例化多个控制器，增加了后续流水级的逻辑复杂度。