# Lab5 测试程序流水线虚拟仿真实验报告

黄奔皓*

上海交通大学

# 目录



*感谢陈老师的悉心指导

# 1 实验目的

1、理解流水线 CPU 指令执行过程。2、理解流水线冒险处理的概念。3、理解流水线硬件结构及对冒险处理方法的多样性。

# 2 实验任务

## 2.1 测试代码 1 的仿真

### 2.1.1 Execution Table 的分析

我们分四种模式分析五级流水线 CPU 执行过程：模式 1：with forward with flush，模式 2：no forward with flush，模式 3：with forward no flush，模式 4：no forward no flush。以下是 execution table：

EXECUTION TABLE — FULL LOOPS — CPU Cycles

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| addi t1, t1, 2 | F | D | X | M | W | | | | | | | | | | | | | | |
| beq t1, x0, 32 | | F | – | D | X | M | W | | | | | | | | | | | | |
| addi t1, t1, –1 | | | | F | D | X | M | W | | | | | | | | | | | |
| addi t0, t0, 3 | | | | | F | D | X | M | W | | | | | | | | | | |
| jal x0, –24 | | | | | | F | D | X | M | W | | | | | | | | | |
| add tp, tp, t0 | | | | | | | F | | | | | | | | | | | | |
| beq t1, x0, 32 | | | | | | | | F | D | X | M | W | | | | | | | |
| addi t1, t1, –1 | | | | | | | | | F | D | X | M | W | | | | | | |
| addi t0, t0, 3 | | | | | | | | | | F | D | X | M | W | | | | | |
| jal x0, –24 | | | | | | | | | | | F | D | X | M | W | | | | |
| add tp, tp, t0 | | | | | | | | | | | | F | | | | | | | |
| beq t1, x0, 32 | | | | | | | | | | | | | F | D | X | M | W | | |
| addi t1, t1, –1 | | | | | | | | | | | | | | F | | | | | |
| add tp, tp, t0 | | | | | | | | | | | | | | | F | D | X | M | W |

(a) with forward with flush

EXECUTION TABLE — FULL LOOPS — CPU Cycles

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| addi t1, t1, 2 | F | D | X | M | W | | | | | | | | | | | | | | | |
| beq t1, x0, 32 | | F | – | – | D | X | M | W | | | | | | | | | | | | |
| addi t1, t1, –1 | | | | | F | D | X | M | W | | | | | | | | | | | |
| addi t0, t0, 3 | | | | | | F | D | X | M | W | | | | | | | | | | |
| jal x0, –24 | | | | | | | F | D | X | M | W | | | | | | | | | |
| add tp, tp, t0 | | | | | | | | F | | | | | | | | | | | | |
| beq t1, x0, 32 | | | | | | | | | F | D | X | M | W | | | | | | | |
| addi t1, t1, –1 | | | | | | | | | | F | D | X | M | W | | | | | | |
| addi t0, t0, 3 | | | | | | | | | | | F | D | X | M | W | | | | | |
| jal x0, –24 | | | | | | | | | | | | F | D | X | M | W | | | | |
| add tp, tp, t0 | | | | | | | | | | | | | F | | | | | | | |
| beq t1, x0, 32 | | | | | | | | | | | | | | F | D | X | M | W | | |
| addi t1, t1, –1 | | | | | | | | | | | | | | | F | | | | | |
| add tp, tp, t0 | | | | | | | | | | | | | | | | F | D | X | M | W |

(b) no forward with flush

EXECUTION TABLE — FULL LOOPS — CPU Cycles

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| addi t1, t1, 2 | F | D | X | M | W | | | | | | | | | | | | | | |
| beq t1, x0, 32 | | F | – | D | X | M | W | | | | | | | | | | | | |
| addi t1, t1, –1 | | | | F | D | X | M | W | | | | | | | | | | | |
| addi t0, t0, 3 | | | | | F | D | X | M | W | | | | | | | | | | |
| jal x0, –24 | | | | | | F | D | X | M | W | | | | | | | | | |
| add tp, tp, t0 | | | | | | | F | D | X | M | W | | | | | | | | |
| beq t1, x0, 32 | | | | | | | | F | D | X | M | W | | | | | | | |
| addi t1, t1, –1 | | | | | | | | | F | D | X | M | W | | | | | | |
| addi t0, t0, 3 | | | | | | | | | | F | D | X | M | W | | | | | |
| jal x0, –24 | | | | | | | | | | | F | D | X | M | W | | | | |
| add tp, tp, t0 | | | | | | | | | | | | F | D | X | M | W | | | |
| beq t1, x0, 32 | | | | | | | | | | | | | F | D | X | M | W | | |
| addi t1, t1, –1 | | | | | | | | | | | | | | F | D | X | M | W | |
| add tp, tp, t0 | | | | | | | | | | | | | | | F | D | X | M | W |

(c) with forward no flush

EXECUTION TABLE — FULL LOOPS — CPU Cycles

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| addi t1, t1, 2 | F | D | X | M | W | | | | | | | | | | | | | | | | | |
| beq t1, x0, 32 | | F | – | – | D | X | M | W | | | | | | | | | | | | | | |
| addi t1, t1, –1 | | | | | F | D | X | M | W | | | | | | | | | | | | | |
| addi t0, t0, 3 | | | | | | F | D | X | M | W | | | | | | | | | | | | |
| jal x0, –24 | | | | | | | F | D | X | M | W | | | | | | | | | | | |
| add tp, tp, t0 | | | | | | | | F | – | D | X | M | W | | | | | | | | | |
| beq t1, x0, 32 | | | | | | | | | F | D | X | M | W | | | | | | | | | |
| addi t1, t1, –1 | | | | | | | | | | F | D | X | M | W | | | | | | | | |
| addi t0, t0, 3 | | | | | | | | | | | F | D | X | M | W | | | | | | | |
| jal x0, –24 | | | | | | | | | | | | F | D | X | M | W | | | | | | |
| add tp, tp, t0 | | | | | | | | | | | | | F | – | D | X | M | W | | | | |
| beq t1, x0, 32 | | | | | | | | | | | | | | F | D | X | M | W | | | | |
| addi t1, t1, –1 | | | | | | | | | | | | | | | F | D | X | M | W | | | |
| add tp, tp, t0 | | | | | | | | | | | | | | | | F | D | X | M | W | | |

(d) no forward no flush

Figure 1: Execution Tables

| 代码段1 | 模式1： with forward with flush | 模式2： no forward with flush | 模式3： with forward no flush | 模式4： no forward no flush |
|---|---|---|---|---|
| 执行周期数 | 19 | 20 | 19 | 22 |
| 被执行forward的指令，执行forward次数和原因 | addi t1, t1, 2, 原因为控制冒险，下一条指令beq在ID阶段就要用到t1 | / | addi t1, t1, 2, 原因为控制冒险，下一条指令beq在ID阶段就要用到t1 | / |
| | 执行的次数为1次 | | 执行的次数为1次 | |
| 被执行flush操作的指令和原因 | add tp, tp, t0，因为beq判断为真，程序回到 loop: beq x6,x0,fi | add tp, tp, t0，因为beq判断为真，程序回到 loop: beq x6,x0,fi | / | / |
| | addi t1, t1, -1，**分支预测错误** | addi t1, t1, -1，**分支预测错误** | | |
| 被执行stall的指令、原因，被执行stall的时刻的时钟周期序号 | CLC=3: beq t1, x0, 32, 此时t1的数据还没能获取 | CLC=3,4 : beq t1, x0, 32, 此时t1的数据还没能获取, 由于没有forwarding过程，t1的值需要在上一条指令的WB阶段才能被读取 | CLC=3: beq t1, x0, 32, 此时t1的数据还没能获取 | CLC=3,4 : beq t1, x0, 32, 此时t1的数据还没能获取, 由于没有forwarding过程，t1的值需要在上一条指令的WB阶段才能被读取 |
| | | | | CLC=9, add tp, tp, t0，此时t0还未WB，所以ID无法进行 |
| | | | | CLC=15, add tp, tp, t0，此时t0还未WB，所以ID无法进行 |

Figure 2: 对代码段 1 四种模式 execution table 的分析

可以看到，在不同的模式下，执行的周期数不同，有 forward 功能的执行模式下，因为无需等待需要使用的值写回寄存器后再读取，所以需要的时钟数更少。而在均没有 forward 功能的情况下，有 flush 功能的模式 (b) 不会受到 addi t0 t0 3 和 add tp, tp t0 指令间数据冒险的影响，从而比无 flush 的模式 (d) 少使用两个时钟周期。

### 2.1.2 寄存器值的分析

在不同的执行模式下，执行同一程序的差别主要来自有无 forward 带来的 stall 的数量不同，以及有无 flush 带来的对寄存器的操作不同。

| 代码段1 | 分析：单周期CPU架构下(RIPES)，执行需（ 11 ）个时钟周期， 执行后，x6=（0），x5=（6），x4=（6） | | | |
|---|---|---|---|---|
| 实际执行仿真情况 | 模式1： with forward with flush | 模式2： no forward with flush | 模式3： with forward no flush | 模式4： no forward no flush |
| Reg X6= | 0 | 0 | -1 | -1 |
| Reg X5= | 6 | 6 | 6 | 6 |
| Reg X4= | 6 | 6 | 15 | 15 |
| 执行结果正确与否 | 正确 | 正确 | 错误 | 错误 |
| 若不正确 如何修改 | / | / | 错误的原因是没有对指令进行正常的flush操作, 修改方式是:<br>1. 在addi x6,x6,-1前加addi x6,x6,0<br>2.在add x4,x4,x5前加addi x4, x4, 0 | 错误的原因是没有对指令进行正常的flush操作, 修改方式是:<br>1. 在addi x6,x6,-1前加addi x6,x6,0<br>2.在add x4,x4,x5前加addi x4, x4, 0 |

Figure 3: 对四种模式下代码段 1 执行后 Reg 值的分析

## 2.2 自行选择代码观察 CPU 的执行过程 (选做)

我们选择一段会发生数据冒险的代码。

```
1 lui x10, 0
2 ori x1, x10, 1024
3 lui x5, 1
4 addi x25, x0, 1
5 sw x25,0(x1)
6 lw x2, 0(x1)
7 and x4,x2,x5
8 addi x4, x4, 1
```

Figure 4: 一段会发生数据冒险的代码

观察代码可知，在第 6 行完成读取数据进 x2 寄存器之前（MEM），第 7 行就用到了 x2 的值（EX），从而产生了数据冒险，且该数据冒险无法由前递修复，必须产生一个 stall。

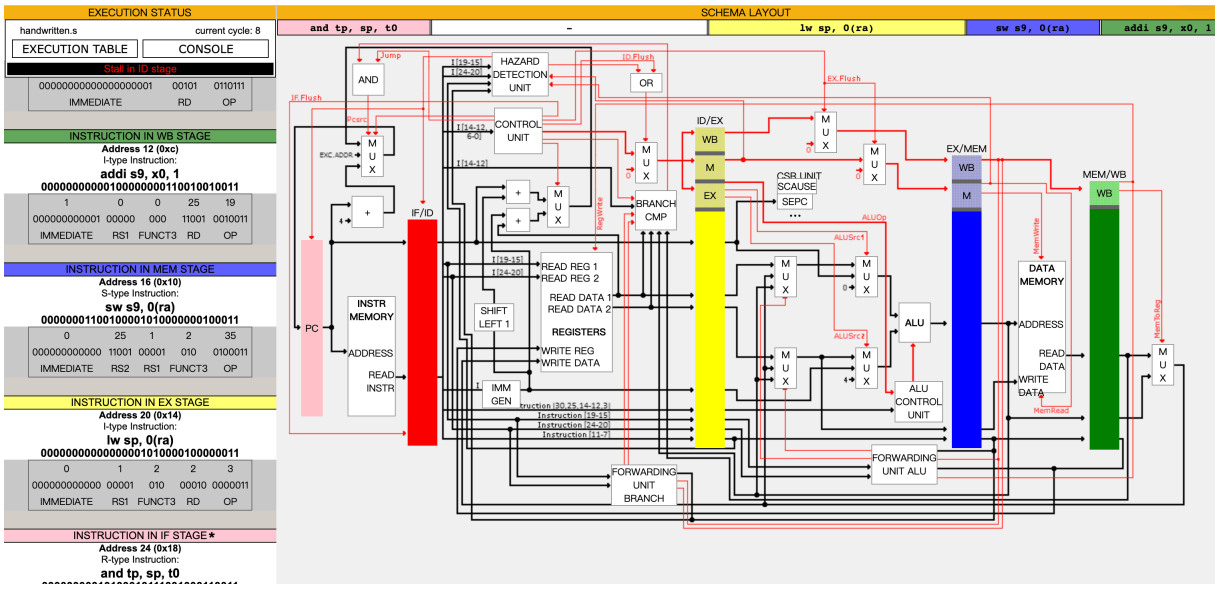


Figure 5: 发生数据冒险，进行 stall 的时间节点截图

结合单步仿真图和 execution table 知，CPU 确实通过 stall 来处理这样的数据冒险。

EXECUTION TABLE

FULL LOOPS

| | CPU Cycles | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| lui a0, 0 | F | D | X | M | W | | | | | | | | |
| ori ra, a0, 1024 | | F | D | X | M | W | | | | | | | |
| lui t0, 1 | | | F | D | X | M | W | | | | | | |
| addi s9, x0, 1 | | | | F | D | X | M | W | | | | | |
| sw s9, 0(ra) | | | | | F | D | X | M | W | | | | |
| lw sp, 0(ra) | | | | | | F | D | X | M | W | | | |
| and tp, sp, t0 | | | | | | | F | – | D | X | M | W | |
| addi tp, tp, 1 | | | | | | | | | F | D | X | M | W |

Figure 6: 该代码的 execution table

## 2.3 测试代码 2 的仿真

我们通过表格的方式指出 PC 为哪些的指令会被做数据冒险处理或控制冒险处理，以及相应的操作。

| PC值 | 十进制 | 指令 | 冒险的种类 | 执行的操作 |
|---|---|---|---|---|
| 0x0 \| 0x4 | 0 \| 4 | lui x10, 0 \| ori x4, x10, 1024 | 数据冒险 | Forwarding |
| 0x2c \| 0x30 | 44 \| 48 | jal sum \| sw x12 0(x4) | 控制冒险 | Flush |
| 0x34 \| 0x38 | 52 \| 56 | lw x19, 0(x4) \| sub x18, x19, x12 | 数据冒险 | Stall, Forwarding |
| 0x40 ~ 0x5c | 64 ~ 92 | addi x5, x5, -1 \| ori x18, x5, -1 \| xori x18, x18, 1365 \| addi x19, x0, -1 \| andi x20, x19, -1 \| or x16, x20, x19 \| xor x18, x20, x19 \| and x17, x20, x16 | 数据冒险 | Forwarding |
| 0x64 \| 0x68 | 100 \| 104 | j loop2 \| addi x5, x0, -1 | 控制冒险 | Flush |
| 0x68 ~ 0x78 | 104 ~ 120 | addi x5, x0, -1 \| slli x18, x5, 15 \| slli x18, x18, 16 \| srai x18, x18, 16 \| srli x18, x18, 15 | 数据冒险 | Forwarding |
| 0x7c \| 0x80 | 124 \| 128 | fi: j fi \| add x18, x0, x0 | 控制冒险 | Flush |
| 0x84 \| 0x8c | 132 \| 140 | loop:lw x19,0(x4) \| add x18, x18, x19 | 数据冒险 | Forwarding |
| 0x90 \| 0x94 | 144 \| 148 | addi x5, x5, -1 \| bne x5, x0, loop | 数据冒险 | Stall, Forwarding |
| 0x94 \| 0x98 | 148 \| 152 | beq x5, x0, loop \| slli x12, x18, 0 | 控制冒险 | Flush |
| 0x9c | 156 | jr ra | 控制冒险 | Flush |

Figure 7: 冒险类型和相应的操作

由于 forwarding 在仿真程序中无法直接看出，我们主要验证数据冒险中的 stall 和控制冒险中的 flush 操作。

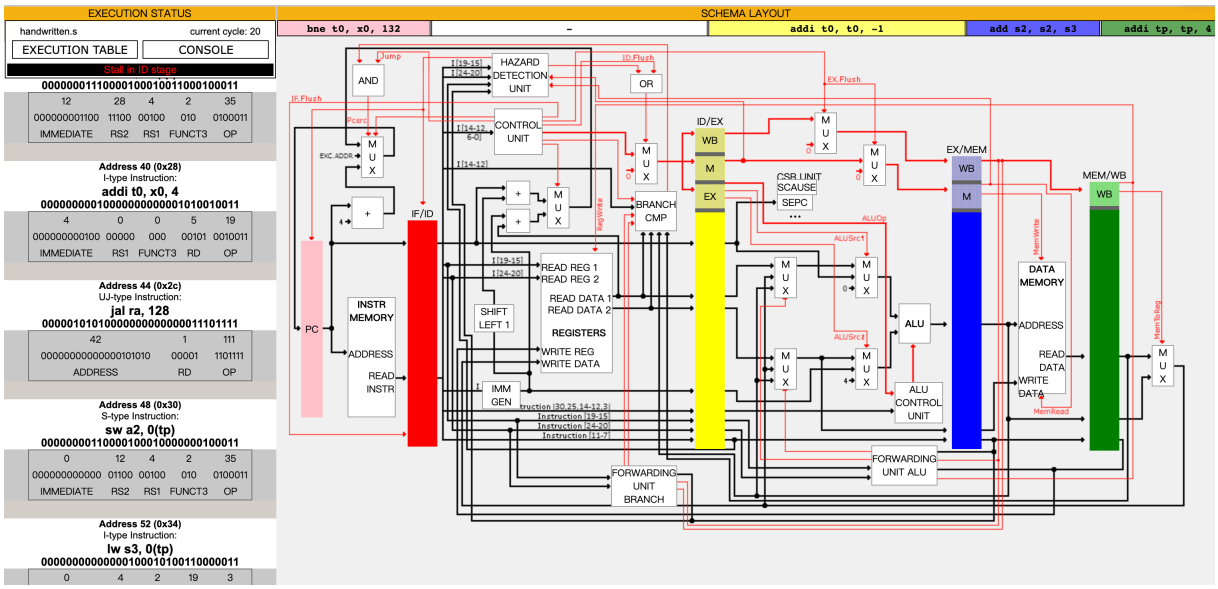


Figure 8: addi x5, x5, -1 | bne x5, x0, loop 数据冒险带来的 stall

图 8 中 bne 指令需要在 ID 阶段判断是否跳转，而此时 EX 阶段的 addi 还未完成对 x5 的计算，所以需要加入 stall。

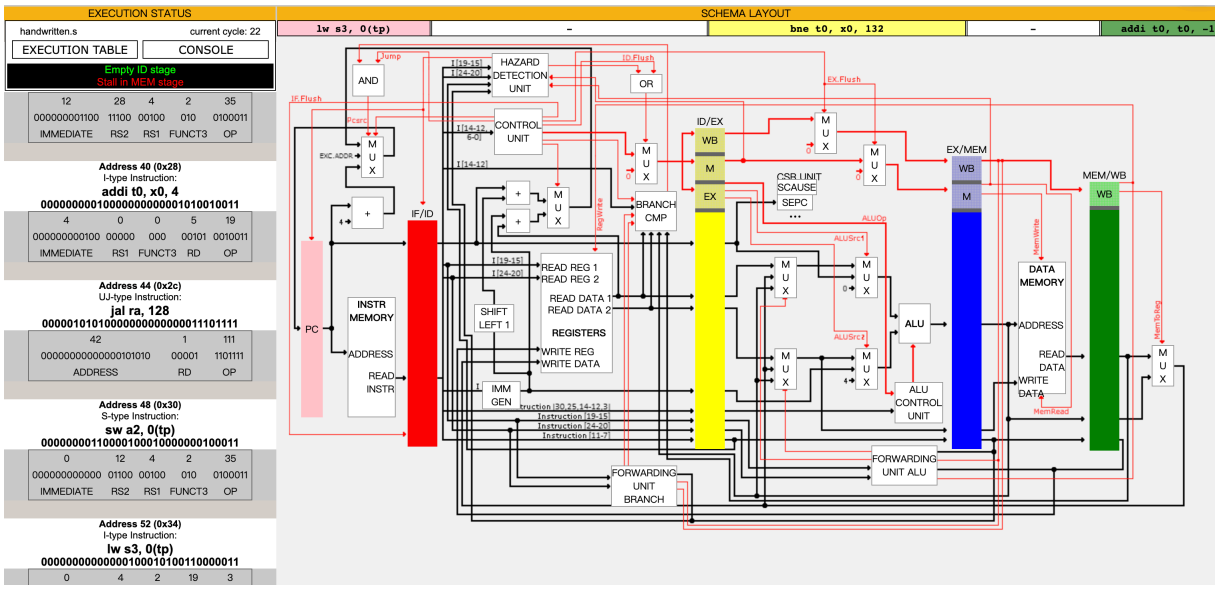


Figure 9: beq x5, x0, loop ｜ slli x12, x18, 0 控制冒险

图 9 中在 beq 指令完成是否跳转的判断之前，分支预测不跳转，但是分支预测错误，从而使 slli x12, x18, 0 被 flush。

# 3 拓展思考

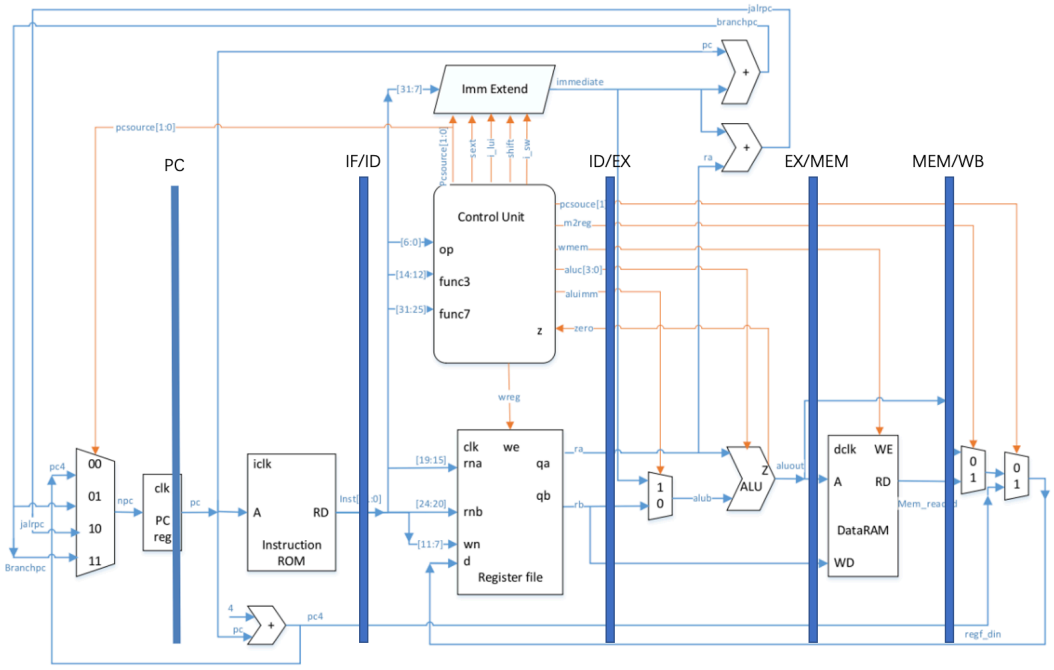


Figure 10: 五级流水线划分

- 其中左侧四选一 MUX 应该算作 ID 级，因为其控制信号在 ID 级就完整的产生了。
- 右上方两个加法器可以在 ID 级里面实现。
- 最右侧的二选一 MUX 可以在 EXE 阶段实现，但是需要做一些修改。我们在 EXE 阶段先做 aluout 和 pc4 的选择，然后在 WB 阶段再跟数据存储器 Data Memory 输出的数据做选择，从而得到写回寄存器堆的数据。
- forward 操作仅在 ID 级实现是不够的。它既需要在 ID 阶段实现，从而将后面级的寄存器值 forward 到 ID 阶段，用于分支控制指令是否跳转的判断，同时也要在 EXE 阶段实现，将上一指令 alu 输出值（存在 EX/MEM 寄存器中）的值 forward 到 EXE 阶段，用来为当前指令进行 ALU 的计算。
- stall 操作一方面控制 PC 寄存器和 IF/ID 寄存器使得其不更新，另一方面通过 MUX 控制 ID/EX，向 ID/EX 寄存器中输入空指令。
- flush 操作一方面控制 IF/ID 寄存器使 IF/ID 寄存器控制信号为 0，丢弃指令；另一方面通过 MUX 控制 ID/EX，向 ID/EX 寄存器中输入空指令。