# Verilog HDL 完成流水线处理器开发

# 目录

# 一、 设计说明

1. 指令集：add, sub, and, or, slt, lw, sw, beq, j, addi
2. 处理器为流水线设计，支持阻塞和转发

# 二、 测试代码

代码主要用来测试各种数据相关、阻塞以及 10 条指令执行的正确性。

```
        addi $1, $0, 0x10
        addi $3, $0, 0x2
        addi $5, $0, 0x3
        addi $6, $0, 0x4
        sub  $2, $1, $3
        and  $12, $2, $5
        or   $13, $6, $2
        add  $14, $2, $2
        add  $1, $1, $2
        add  $1, $1, $3
        add  $1, $1, $5
        sw   $1, 0($0)
        lw   $2, 0($0)
        and  $4, $2, $5
        or   $8, $2, $6
        add  $9, $4, $2
        beq  $1, $2, label
        and  $4, $2, $5
        or   $8, $2, $6
        add  $9, $4, $2
        add  $1, $1, $2
        add  $1, $1, $3
        add  $1, $1, $5
label:  lw   $10, 0($0)
loop:   addi $7, $7, 1
        slt  $16, $7, $14
        beq  $16, $0, exit
        j    loop
exit:
```

# 三、 结果对比

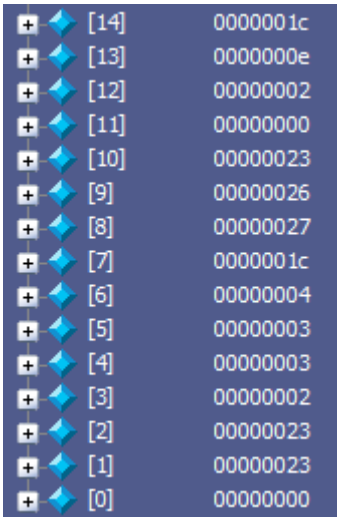
## 1. Mars 运行结果

| | | |
|---|---|---|
| $zero | 0 | 0x00000000 |
| $at | 1 | 0x00000023 |
| $v0 | 2 | 0x00000023 |
| $v1 | 3 | 0x00000002 |
| $a0 | 4 | 0x00000003 |
| $a1 | 5 | 0x00000003 |
| $a2 | 6 | 0x00000004 |
| $a3 | 7 | 0x0000001c |
| $t0 | 8 | 0x00000027 |
| $t1 | 9 | 0x00000026 |
| $t2 | 10 | 0x00000023 |
| $t3 | 11 | 0x00000000 |
| $t4 | 12 | 0x00000002 |
| $t5 | 13 | 0x0000000e |
| $t6 | 14 | 0x0000001c |

Data Segment

| Address | Value (+0) | Value (+4) |
|---|---|---|
| 0x00000000 | 0x00000023 | 0x00000000 |
| 0x00000020 | 0x00000000 | 0x00000000 |
| 0x00000040 | 0x00000000 | 0x00000000 |

## 2. ModelSim 仿真结果




Memory Data - /Test_TB/MIPS/DM/RAM - Default

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000023 | 00000000 | 00000000 |
| 00000008 | 00000000 | 00000000 | 00000000 |

# 四、 心得体会

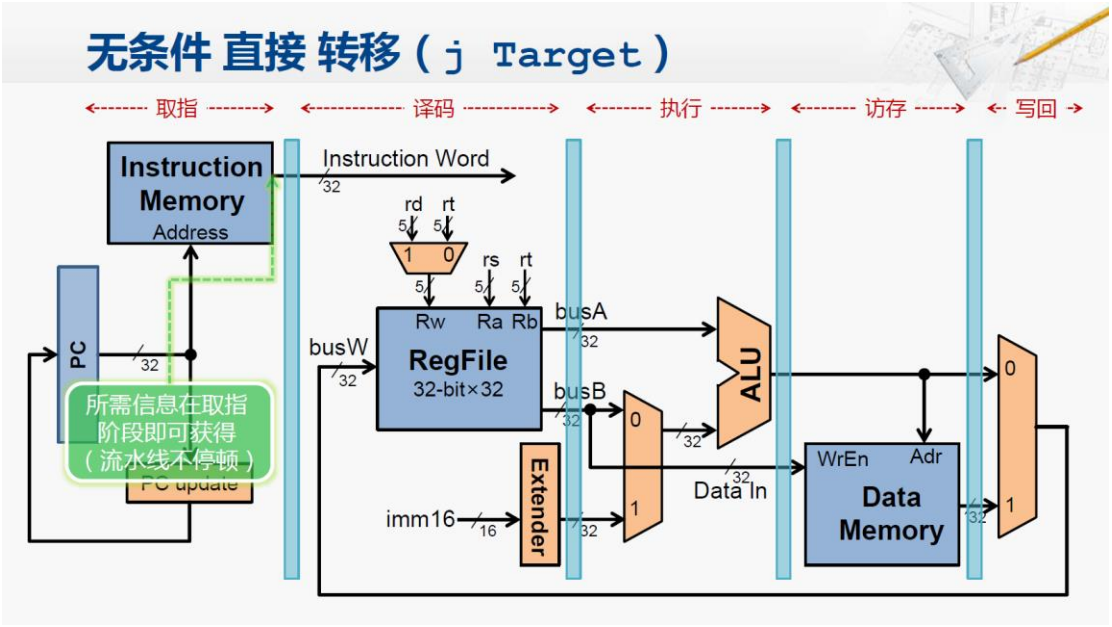
刚开始做这个实验的时候，我是想在没有转发和阻塞的程序上直接修改，结果发现越改越乱，这时又有别的考试，就把这件事暂时放一边了。然而等我忙完再回来看这些代码基本上已经看不懂了，于是我索性从头开始，认真阅读了课本上关于冒险的内容，并在 Coursera 上看了一下北京大学《计算机组成》的 MOOC，确保自己理解了各种冒险产生的原因。然后我就一一对照课本中提到的转发和阻塞条件，在课本给出的数据通路的基础上对数据通路进行了完善，然后按照完善后的草图来编写代码。大约用了 16 个小时左右完成了 10 条指令支持所有课本中考虑到的转发和阻塞的 CPU。

我认为完成这个实验的关键在于：

1. 理解阻塞和转发产生的原因，知道这 10 条指令何时需要或产生数据；
2. 完整的数据通路，一张完整的数据通路图能让思路保持清晰；
3. 代码的命名规范，变量的名称应能表达它的作用以及它处于哪一个流水级。我选择了在变量名称后加一个字母来表示改变量是哪一级产生的信号，如 PCJumpF, PCBranchD, PCPlus4F, PCF, PCPlus4D, …这样的代码更容易维护，在调试时会方便许多。

# 五、 创新之处

北大的 MOOC 中提到 j 指令所需信息均可以在取指阶段获得，因此可以将 j 指令放在取指阶段处理，从而避免流水线的停顿（如图）。




我尝试采用了这种方法，并成功避免了 j 指令导致的流水线停顿。在实现的过程中我认为需要注意的一点是在确定 NPC 时，一定要优先选择译码阶段由 beq 产生的跳转地址，否则当 beq 指令后是一条 j 指令时，beq 就无法起作用。