



哈爾濱工業大學 (深圳)  
HARBIN INSTITUTE OF TECHNOLOGY

## 实验报告

开课学期: 2021 夏季

课程名称: 计算机设计与实践

实验名称: CPU 设计

实验性质: 综合设计型

实验学时: 52 地点:           

学生班级: 19 级 10 班

学生学号: 190111026

学生姓名: 郭毅安

评阅教师:                                 

报告成绩:                                 

实验与创新实践教育中心制

2021 年 5 月

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计的功能描述（含所有实现的指令描述，以及单周期/流水线 CPU 频率）

设计项目：单周期 CPU 和流水线 CPU

包含指令：

add: 把寄存器 rs1 加上寄存器 rs2, 结果写入寄存器 rd。忽略算术溢出。

sub: 从寄存器 rs1 减去寄存器 rs2, 结果写入寄存器 rd。忽略算术溢出。

add: 将寄存器 rs1 和寄存器 rs2 进行按位与, 结果写入寄存器 rd。

or: 将寄存器 rs1 和寄存器 rs2 进行按位或, 结果写入寄存器 rd。

xor: 将寄存器 rs1 和寄存器 rs2 进行按位异或, 结果写入寄存器 rd。

sll: 把寄存器 rs1 左移 rs2 位, 低位补 0, 结果写入寄存器 rd。rs2[4:0]表示左移位数, 高位忽略。

srl: 把寄存器 rs1 右移 rs2 位, 高位补 0, 结果写入寄存器 rd。rs2[4:0]表示右移位数, 高位忽略。

sra: 功能描述: 把寄存器 rs1 右移 rs2 位, 高位补 rs1[31], 结果写入寄存器 rd。rs2[4:0]表示右移位数, 高位忽略。addi: 把符号扩展的立即数加到寄存器 rs1 上, 结果写入寄存器 rd。忽略算术溢出。

andi: 把符号扩展的立即数与寄存器 rs1 的值进行按位与, 结果写入寄存器 rd。

ori: 把符号扩展的立即数与寄存器 rs1 的值进行按位或, 结果写入寄存器 rd。

xori: 把符号扩展的立即数与寄存器 rs1 的值进行按位异或, 结果写入寄存器 rd。

slli: 把寄存器 rs1 左移 shamt 位, 低位补 0, 结果写入寄存器 rd。

srl: 把寄存器 rs1 右移 shamt 位, 高位补 0, 结果写入寄存器 rd。

srai: 把寄存器 rs1 右移 shamt 位, 高位补 rs1[31], 结果写入寄存器 rd。

lw: 从地址(rs1)+sxt(offset)读取四个字节, 经符号扩展后写入寄存器 rd。

jalr: 将 PC 设置为(rs1)+sxt(offset), 把计算出的地址的最低位设为 0, 并将原 PC+4 的值写入寄存器 rd。rd 默认为 1。

sw: 将寄存器 rs2 存入内存地址(rs1)+sxt(offset)。

beq: 若寄存器 rs1 和寄存器 rs2 的值相等, 把 PC 的值设置为当前值加上符号扩展的偏移量。

bne: 若寄存器 rs1 和寄存器 rs2 的值不相等, 把 PC 的值设置为当前值加上符号扩展的偏移量。

blt: 若寄存器 rs1 的值小于寄存器 rs2 的值, 把 PC 的值设置为当前值加上符号扩展的偏移量。

bge: 若寄存器 rs1 的值大于等于寄存器 rs2 的值, 把 PC 的值设置为当前值加上符号扩展的偏移量。

lui: 将符号扩展的 20 位立即数逻辑左移 12 位, 结果写入寄存器 rd。

jal: 把下一条指令的地址保存到寄存器 rd, 然后把 PC 设置为当前值加上符号扩展的立即数。rd 默认为 1。

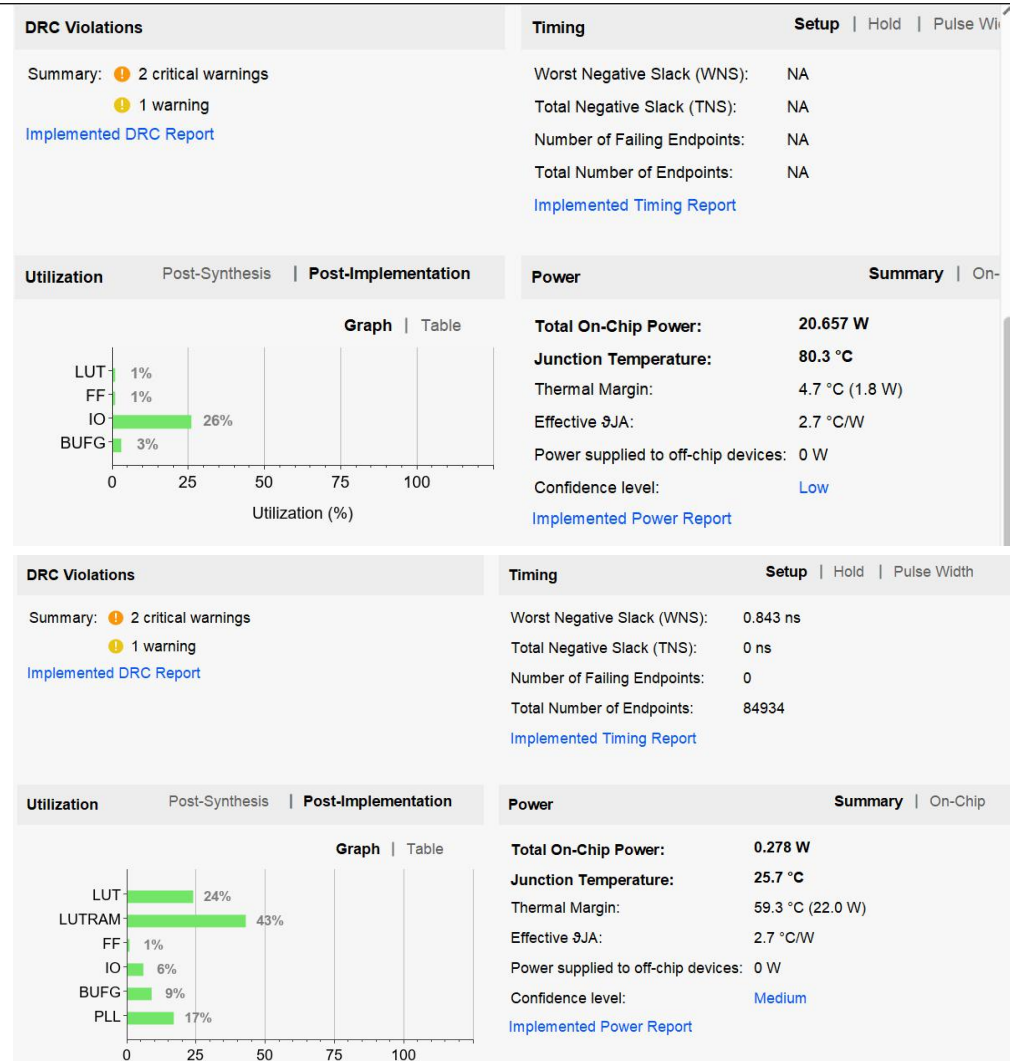
单周期频率: 60hz

流水线频率: 60hz

## 设计的主要特色（除基本要求以外的设计）

无

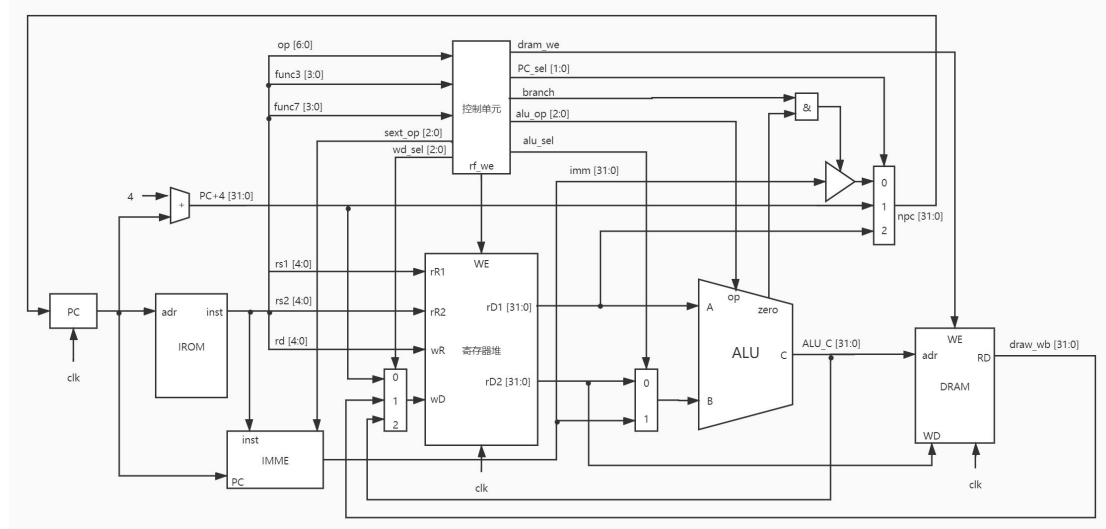
## 资源使用情况、功耗数据截图（实现后）



# 1 单周期 CPU 设计与实现

## 1.1 单周期 CPU 整体框图

(要求：无需画出模块内的逻辑，但要标出模块之间信号线的信号名和位宽，以及说明每个模块的功能含义)



取指模块：将指令从存储器中读取出来的过程。

译码模块：将存储器中取出的指令进行翻译的过程。

执行模块：对指令进行真正运算的过程。

访存模块：存储器访问指令将数据从存储器中读出，或者写入存储器的过程。

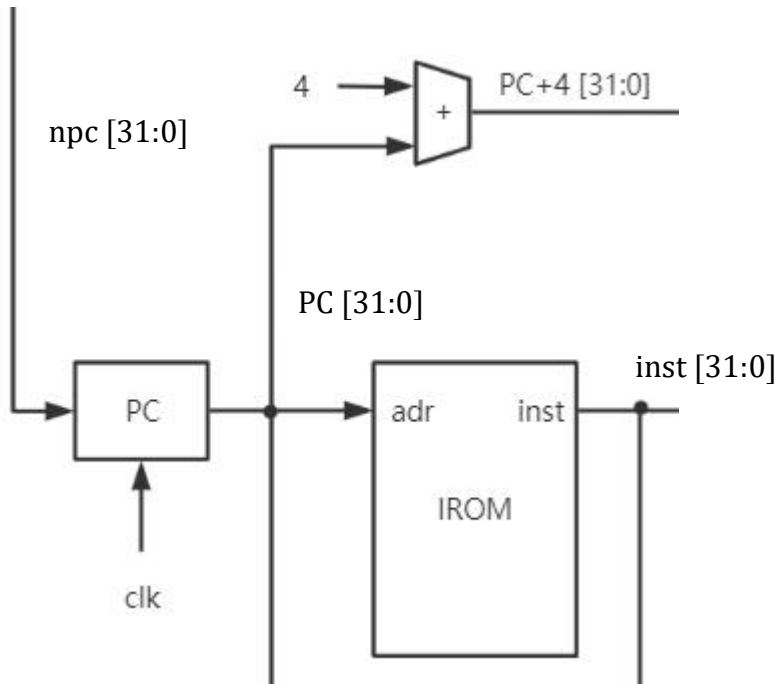
写回模块：将指令执行的结果写回通用寄存器组的过程。

控制模块：根据不同指令发出不同控制信号

## 1.2 单周期 CPU 模块详细设计

（要求：各个模块的详细设计图，要包含内部的子模块，以及关键性逻辑，标出信号名和位宽，并有详细说明）

取指模块：

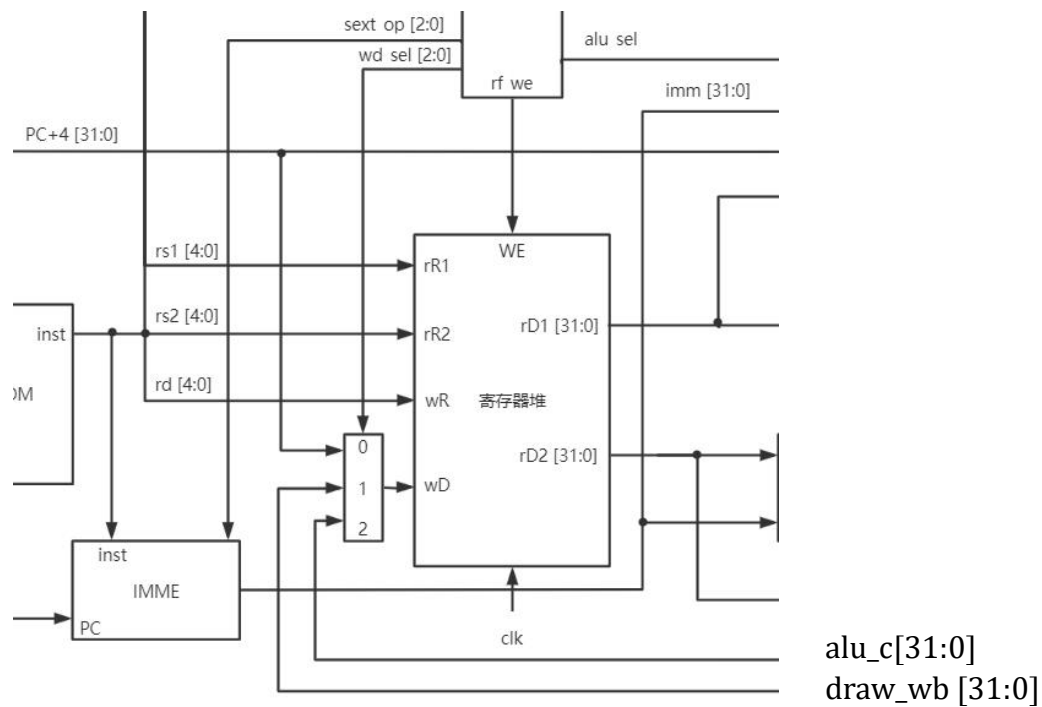


系统以当前 pc 的值为地址，从 IROM 中取出指令，然后传至下一模块。

PC: 32bit 寄存器，存储着当前指令的地址

IROM 是一个单端口的只读存储器

译码模块：

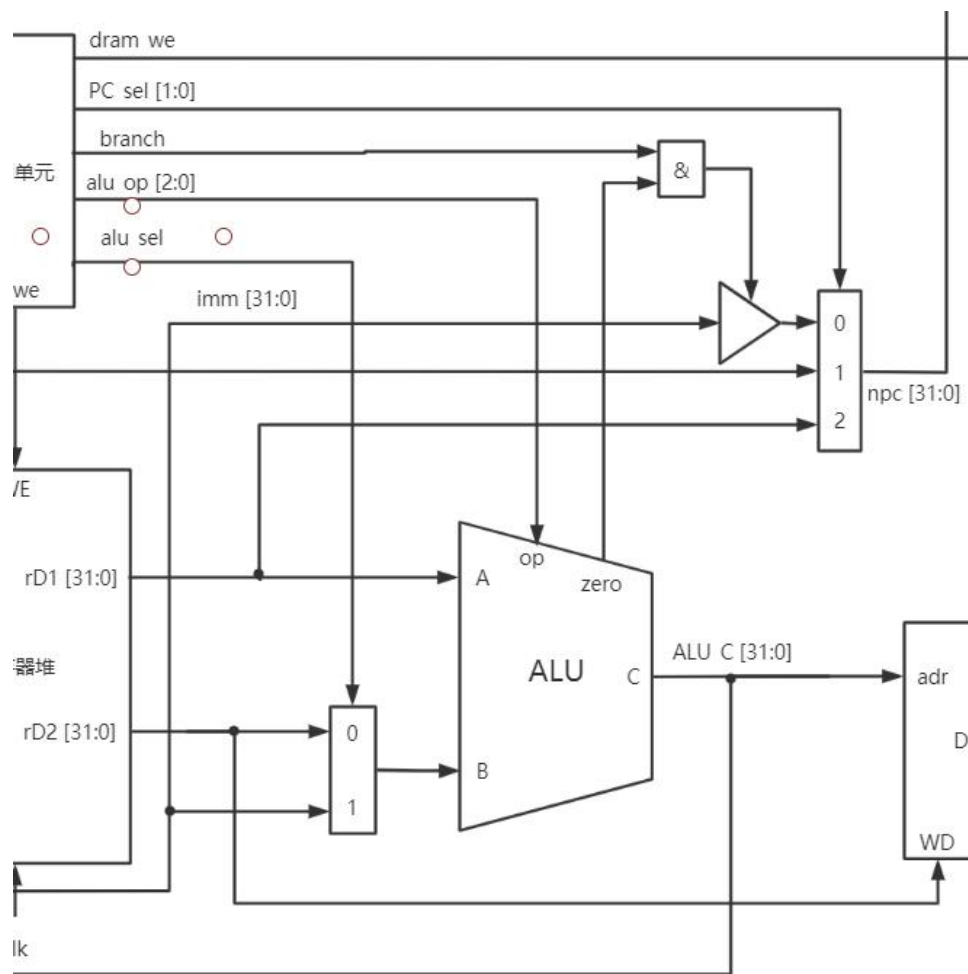


将存储器中取出的指令进行翻译的。经过译码之后得到指令需要的操作数寄存器索引，可以使用此索引从通用寄存器组（RF）中将操作数读出。同时在 IMME 中进行立即数的拓展，将其拓展为 32 位，供后续单元使用。同时寄存器内的值取出后也传递给后续单元使用。

RF 包含 32 个 32bit 寄存器

IMME 是符号拓展单元

执行模块：



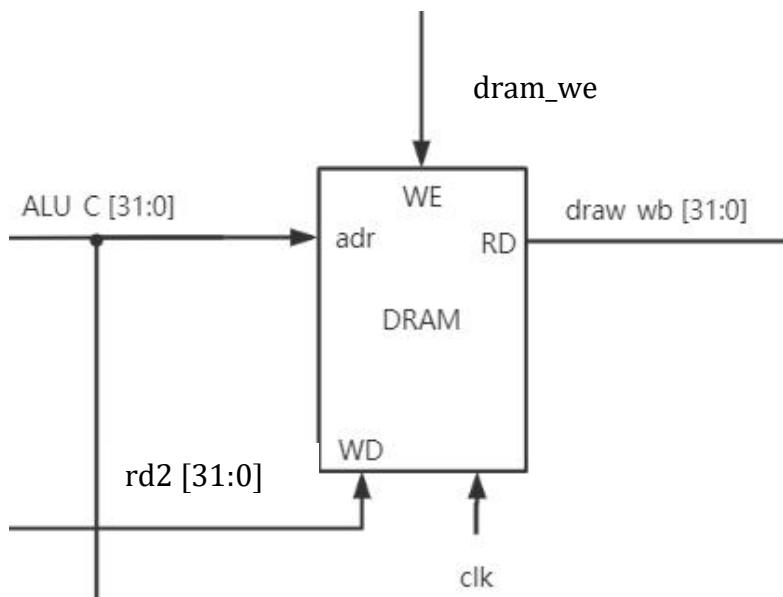
寄存器数据、立即数等作为运算原数据传入 ALU，在 ALU 中进行运算，运算结果

c 写回或传入 DRAM 中。

在这一单元中还有 pc 的更新逻辑，pc 的更新来源有三个：`+4`、立即数偏移量和 `rd1`，由多路选择器选择后进行输出从而更新 pc。

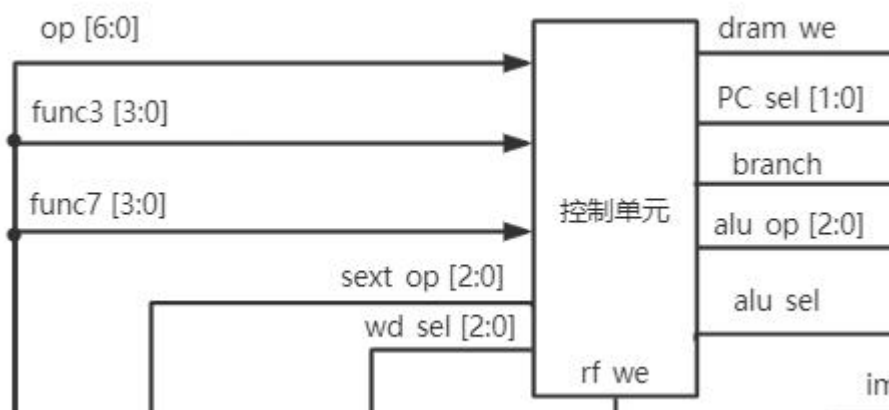
ALU 内部：多路选择器选择 ALU 输出，译码器选择具体运算

访存、写回模块：



存储器访问指令将数据从存储器中读出，或者写入存储器。  
**DRAM** 是一个按地址访问的存储器，可读可写

控制模块：



控制器的功能是根据指令的操作码 (opcode) 和功能码 (funct7 和 funct3)，产生指令执行所需的控制信号。控制信号一般包括 3 类：功能选择信号、多路选择信号和分支控制信号



### 1.3 单周期 CPU 仿真及结果分析

(要求：包含逻辑运算指令、访存指令、跳转指令的仿真截图，以及结果分析)

逻辑运算指令：

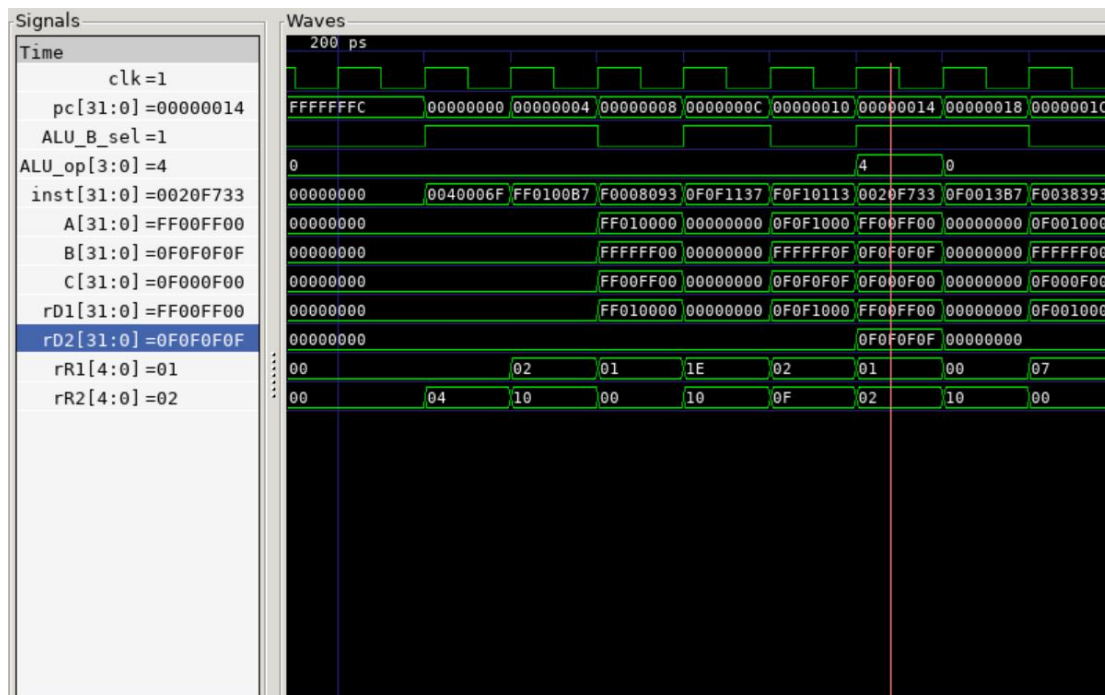
and 反汇编：

```

and.dump
1
2 and:      file format elf32-littleriscv
3
4
5 Disassembly of section .text.init:
6
7 00000000 <_start>:
8   0: 0040006f          jal x0,4 <reset_vector>
9
10 00000004 <reset_vector>:
11   4: ff0100b7          lui x1,0xff010
12   8: f0008093          addi x1,x1,-256 # ff00ff00 <_end+0xff00df00>
13  c: 0f0f1137          lui x2,0xf0f1
14  10: f0f10113          addi x2,x2,-241 # f0f0f0f <_end+0xf0eef0f>
15  14: 0020f733          and x14,x1,x2
16  18: 0f0013b7          lui x7,0xf001
17  1c: f0038393          addi x7,x7,-256 # f000f00 <_end+0xefff00>
18  20: 00200193          addi x3,x0,2
19  24: 48771c63          bne x14,x7,4bc <fail>
20
21 00000028 <test_3>:
22  28: 0ff010b7          lui x1,0xff01
23  2c: ff008093          addi x1,x1,-16 # ff00ff0 <_end+0xfefeff0>
24  30: f0f0f137          lui x2,0xf0f0f
25  34: 0f0f0113          addi x2,x2,240 # f0f0f0f0 <_end+0xf0f0d0f0>
26  38: 0020f733          and x14,x1,x2
27  3c: 00f003b7          lui x7,0xf00
28  40: 0f038393          addi x7,x7,240 # f000f0 <_end+0xefe0f0>
29  44: 00300193          addi x3,x0,3
30  48: 46771a63          bne x14,x7,4bc <fail>

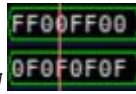
```

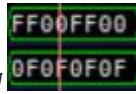
波形图：



分析：

地址为 0014 时执行 and 指令，将 x1 和 x2 两个寄存器内的数据进行与运算，结果存入寄存器 x14 内。控制信号 alu\_op 值为 4，表示此时 ALU 进行与运算。



A、B 的值分别为 ，运算结果 C 为 0F000F00，结果正确。

访存指令：

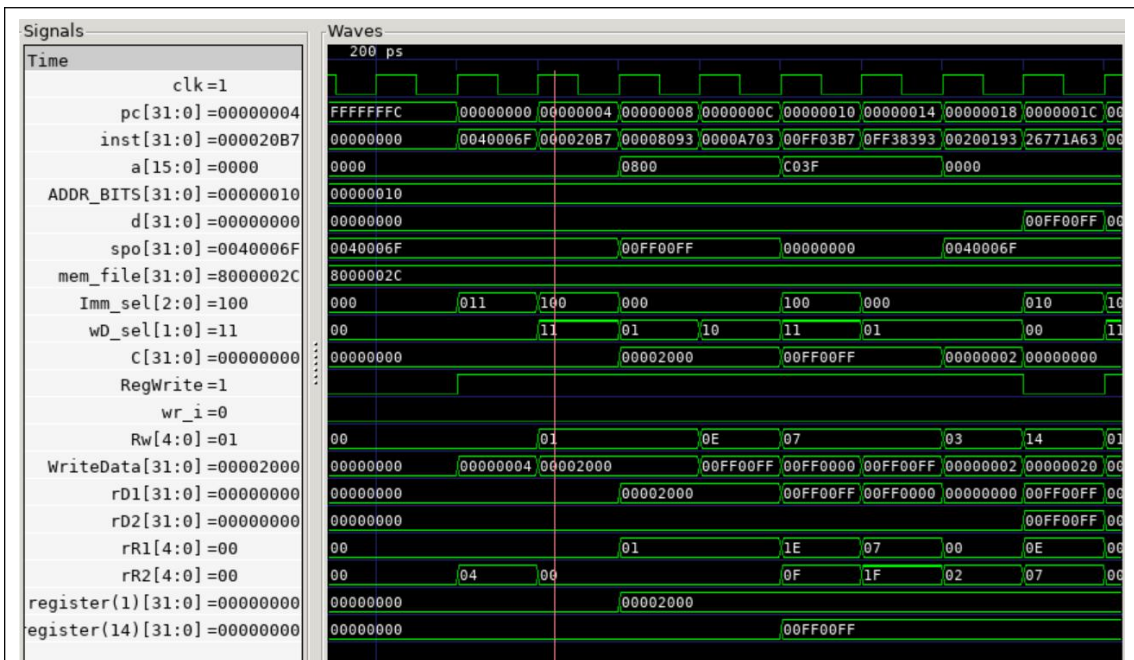
lw 指令反汇编：

```

lw.dump
1
2 lw:      file format elf32-littleriscv
3
4
5 Disassembly of section .text.init:
6
7 00000000 <_start>:
8   0: 0040006f          jal x0,4 <reset_vector>
9
10 00000004 <reset_vector>:
11   4: 000020b7          lui x1,0x2
12   8: 00008093          addi x1,x1,0 # 2000 <begin_signature>
13  c: 0000a703          lw  x14,0(x1)
14  10: 00ff03b7          lui x7,0xff0
15  14: 0ff38393          addi x7,x7,255 # ff00ff <_end+0xfee0ef>
16  18: 00200193          addi x3,x0,2
17  1c: 26771a63          bne x14,x7,290 <fail>
18
19 00000020 <test_3>:
20  20: 000020b7          lui x1,0x2
21  24: 00008093          addi x1,x1,0 # 2000 <begin_signature>
22  28: 0040a703          lw  x14,4(x1)
23  2c: ff0103b7          lui x7,0xff010
24  30: f0038393          addi x7,x7,-256 # ff00ff00 <_end+0xff00def0>
25  34: 00300193          addi x3,x0,3
26  38: 24771c63          bne x14,x7,290 <fail>
27

```

波形图：



分析：

由反汇编可知，地址为 000c 时进行访存指令 lw，其将 MEM[reg[x1]+0]中的数据传入寄存器 x14。0c 时，x1 的值为 0200，经过运算加上偏移量 0 后为 0200，目标寄存器为 0E (x14)，写回数据为 00FF00FF。写回后，x14 内的数据为 00FF00FF。

跳转指令：

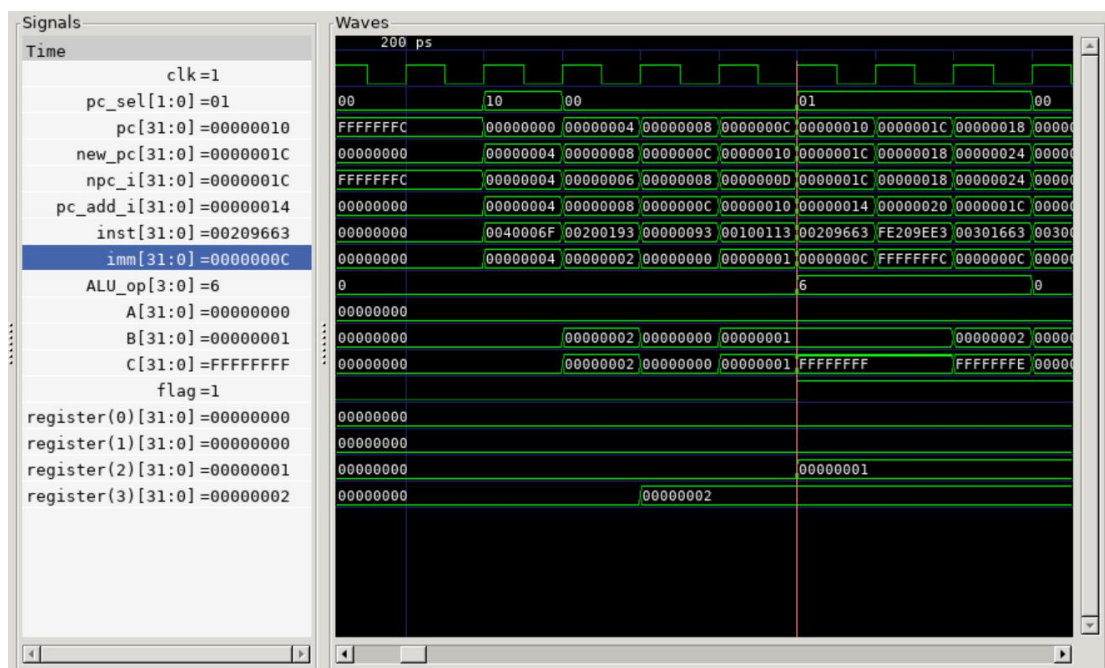
bne 反汇编：

```

bne.dump
1
2 bne:      file format elf32-littleriscv
3
4
5 Disassembly of section .text.init:
6
7 00000000 <_start>:
8   0: 0040006f          jal x0,4 <reset_vector>
9
10 00000004 <reset_vector>:
11   4: 00200193          addi  x3,x0,2
12   8: 00000093          addi  x1,x0,0
13   c: 00100113          addi  x2,x0,1
14  10: 00209663          bne  x1,x2,1c <reset_vector+0x18>
15  14: 2a301a63          bne  x0,x3,2c8 <fail>
16  18: 00301663          bne  x0,x3,24 <test_3>
17  1c: fe209ee3          bne  x1,x2,18 <reset_vector+0x14>
18  20: 2a301463          bne  x0,x3,2c8 <fail>
19
20 00000024 <test_3>:
21  24: 00300193          addi  x3,x0,3
22  28: 00100093          addi  x1,x0,1
23  2c: 00000113          addi  x2,x0,0
24  30: 00209663          bne  x1,x2,3c <test_3+0x18>
25  34: 28301a63          bne  x0,x3,2c8 <fail>
26  38: 00301663          bne  x0,x3,44 <test_4>
27  3c: fe209ee3          bne  x1,x2,38 <test_3+0x14>
28  40: 28301463          bne  x0,x3,2c8 <fail>
29

```

波形图:



分析:

由反汇编可以得到，地址为 0010 时会运行 bne 指令，跳转进入 001c。地址为 0010 时，ALU\_op 为 6，表示此时 ALU 进行相等判断的运算，同时得到结果 flag 为 1，表示不相等。跳转条件成立，pc 直接跳入 001c。

## 2 流水线 CPU 设计与实现

### 2.1 流水线的划分

(要求：画出流水线的划分，并标明每个阶段 CPU 完成的功能)



取指 (IF)：将指令从存储器中读取出来的过程。

译码 (ID)：将存储器中取出的指令进行翻译

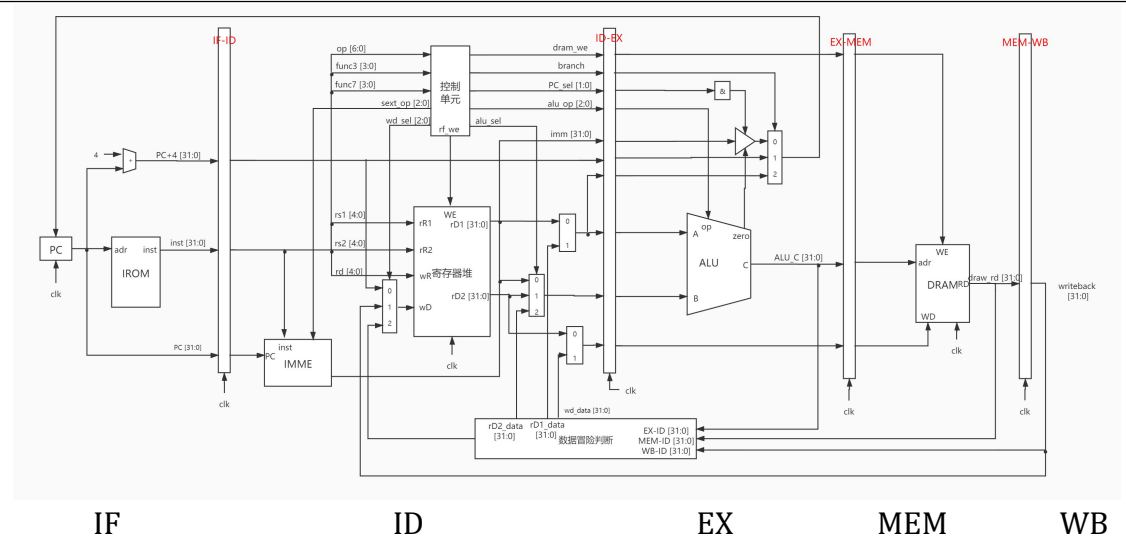
执行 (EX)：对指令进行真正运算的过程。

访存 (MEM)：存储器访问指令将数据从存储器中读出，或者写入存储器的过程。

写回 (WB)：将指令执行的结果写回通用寄存器组的过程。

## 2.2 流水线 CPU 整体框图

(要求: 无需画出模块内的逻辑, 但要标出模块之间信号线的信号名和位宽, 以及说明每个模块的功能含义)



取指模块将指令从存储器中读取出来的过程。

译码模块将存储器中取出的指令进行翻译的过程。经过译码之后得到指令需要的操作数寄存器索引, 可以使用此索引从通用寄存器组 (Register File, Regfile) 中将操作数读出。

执行模块对指令进行真正运算的过程。譬如, 如果指令是一条加法运算指令, 则对操作数进行加法操作; 如果是减法运算指令, 则进行减法操作。在“执行”阶段的最常见部件为算术逻辑部件运算器 (Arithmetic Logical Unit, ALU), 作为实施具体运算的硬件功能单元。

访存模块中存储器访问指令将数据从存储器中读出, 或者写入存储器的过程。

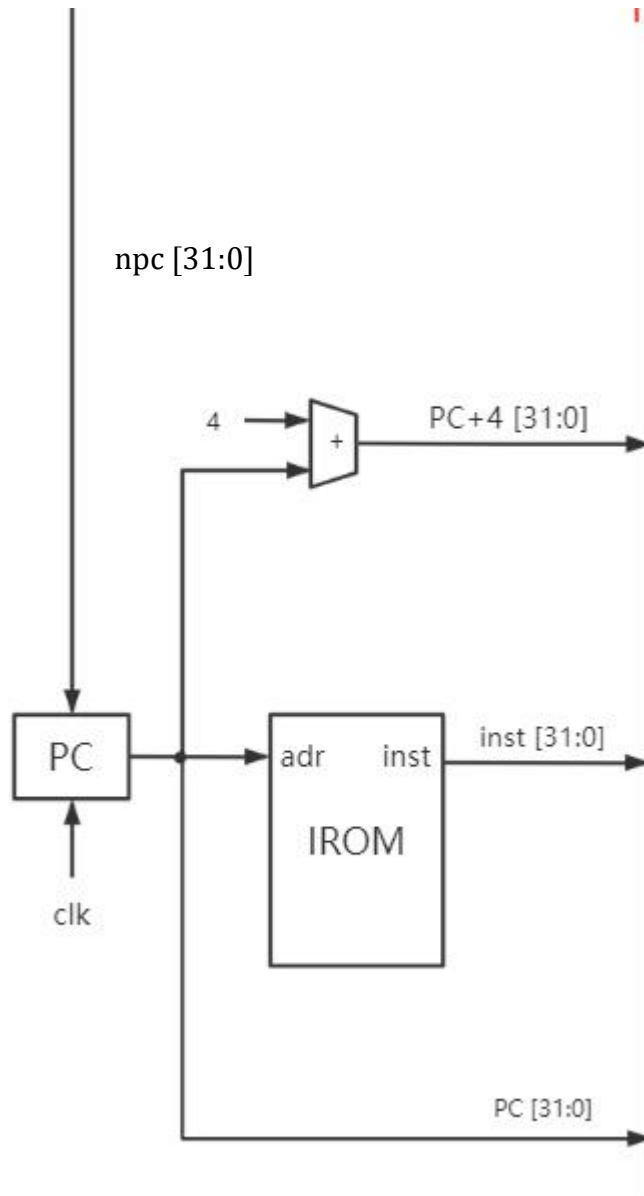
写回模块将指令执行的结果写回通用寄存器组的过程。如果是普通运算指令, 该结果值来自于“执行”阶段计算的结果; 如果是存储器读指令, 该结果来自于“访存”阶段从存储器中读取出来的数据。

控制模块根据不同指令发出不同控制信号, 使得各个模块能根据当前指令类型因地制宜地进行相应而又不同的操作。

## 2.3 流水线 CPU 模块详细设计

(要求: 各个模块的详细设计图, 要包含内部的子模块, 以及关键性逻辑, 标出信号名和位宽, 并有详细说明; 数据冒险与控制冒险的解决方法必须要详细说明)

取指模块:



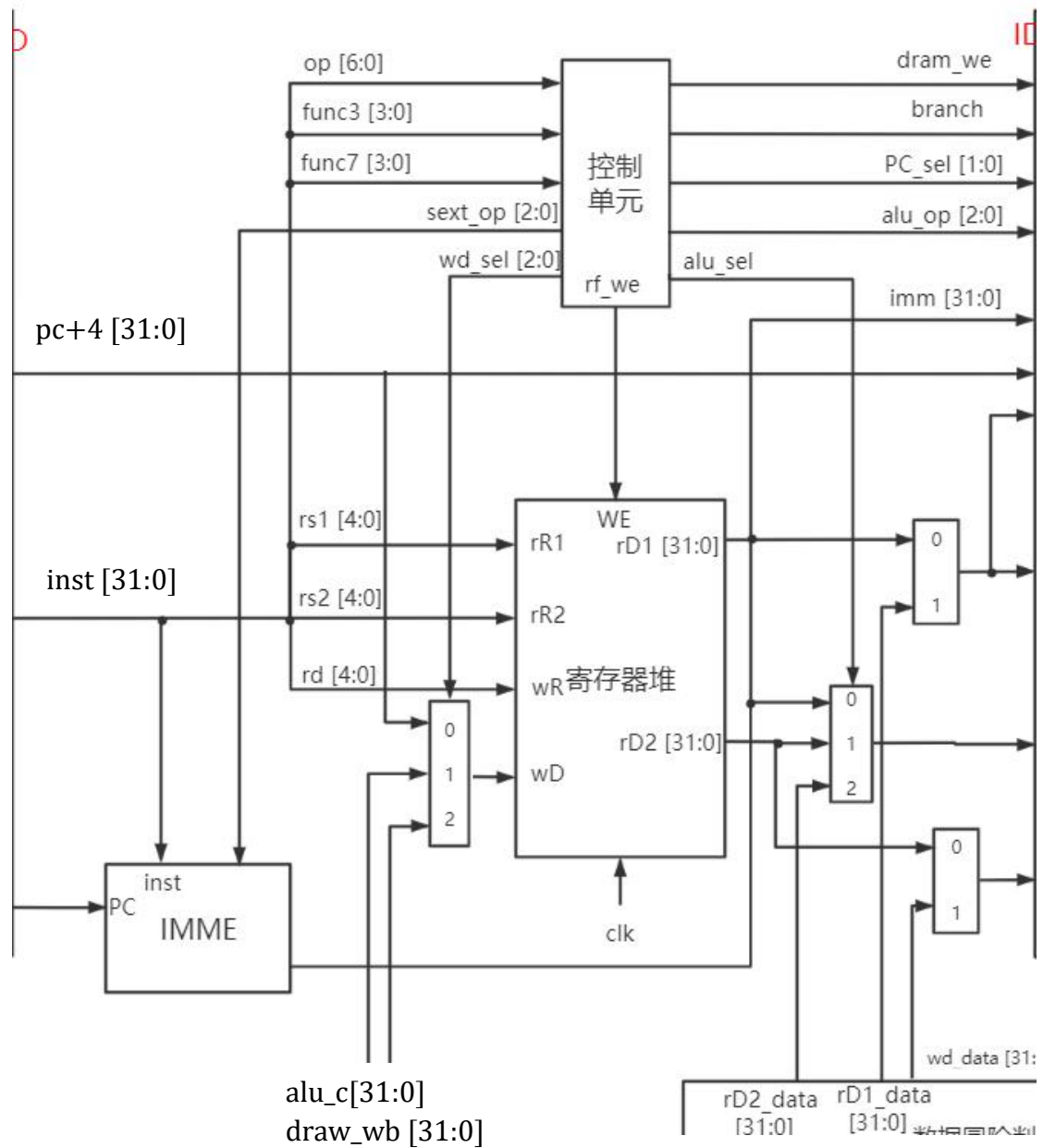
系统以当前 pc 的值为地址, 从 IROM 中取出指令, 然后传至下一模块。

PC: 32bit 寄存器, 存储着当前指令的地址

IROM 是一个单端口的只读存储器



译码模块：



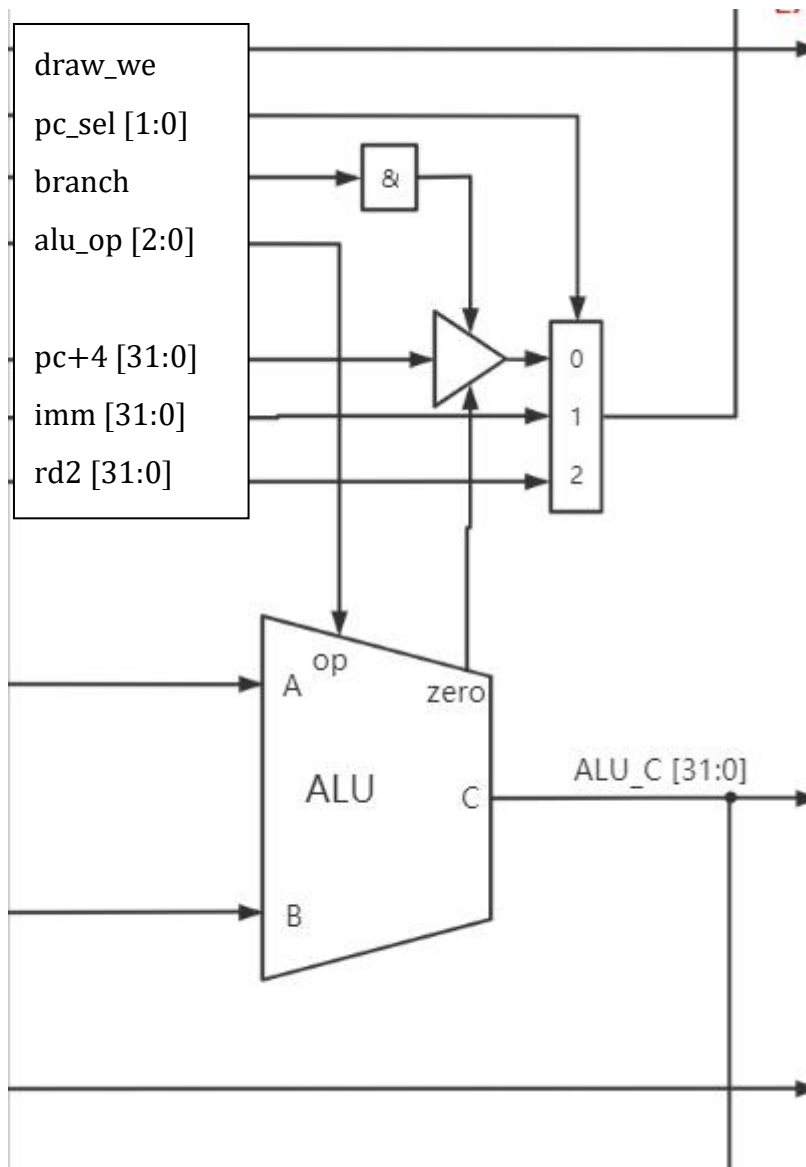
将存储器中取出的指令进行翻译的。经过译码之后得到指令需要的操作数寄存器索引，可以使用此索引从通用寄存器组（RF）中将操作数读出。同时在 IMME 中进行立即数的拓展，将其拓展为 32 位，供后续单元使用。同时寄存器内的值取出后也传递给后续单元使用。

RF 包含 32 个 32bit 寄存器

IMME 是符号拓展单元



执行模块：



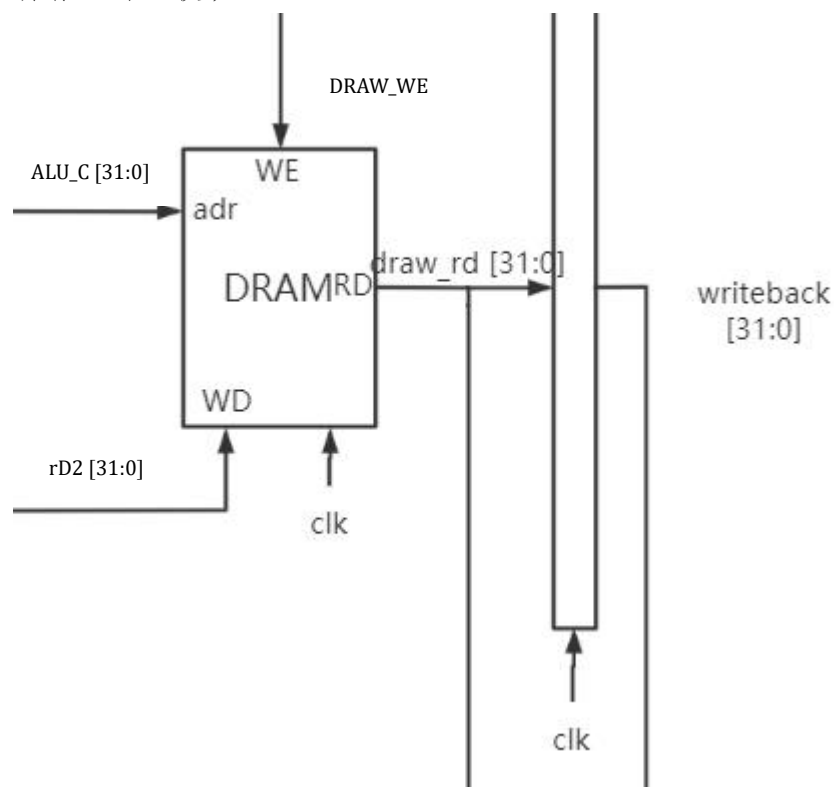
寄存器数据、立即数等作为运算原数据传入 ALU，在 ALU 中进行运算，运算结果

c 写回或传入 DRAM 中。

在这一单元中还有 pc 的更新逻辑，pc 的更新来源有三个：**+4**、立即数偏移量和 `rd1`，由多路选择器选择后进行输出从而更新 pc。

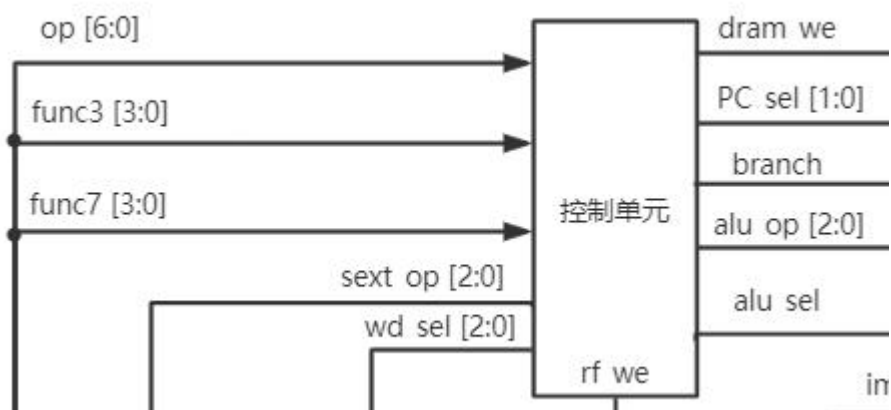
ALU 内部：多路选择器选择 ALU 输出，译码器选择具体运算

访存、写回模块：



存储器访问指令将数据从存储器中读出，或者写入存储器。  
**DRAM** 是一个按地址访问的存储器，可读可写

控制模块：



控制器的功能是根据指令的操作码 (opcode) 和功能码 (funct7 和 funct3)，产生指令执行所需的控制信号。控制信号一般包括 3 类：功能选择信号、多路选择信号和分支控制信号

数据冒险判断与数据前递模块：



该模块能根据当前数据判断是否存在冒险并且为何种冒险，同时承担这数据前递中转站的作用。根据数据冒险的类型，选择前递的数据。

数据冒险解决办法：

使用数据前递来解决数据冒险。首先根据当前数据判断出当前冒险是数据冒险三种情况中的哪一种，再选择相应的前递数据。

控制冒险解决办法：

使用流水线暂停来解决控制冒险。当当前指令为跳转指令并满足跳转条件后，CPU 首先判断暂停多长时间（多少周期），再根据判断结果进入暂停，直到进入跳转目标指令。

## 2.4 流水线 CPU 仿真及结果分析

(要求: 包含数据冒险、控制冒险的仿真截图, 以及结果分析)

数据冒险:

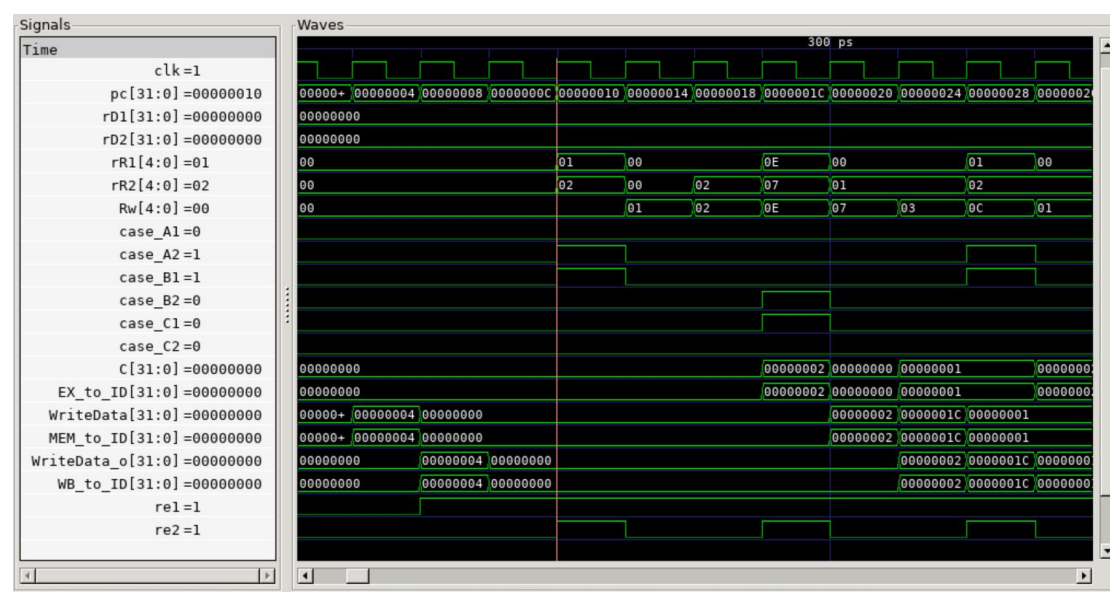
add 指令反汇编:

```

add.dump
1
2 add:      file format elf32-littleriscv
3
4
5 Disassembly of section .text.init:
6
7 00000000 <_start>:
8   0: 0040006f          jal x0,4 <reset_vector>
9
10 00000004 <reset_vector>:
11   4: 00000093          addi  x1,x0,0
12   8: 00000113          addi  x2,x0,0
13  c: 00208733          add  x14,x1,x2
14  10: 00000393          addi  x7,x0,0
15  14: 00200193          addi  x3,x0,2
16  18: 4c771663          bne  x14,x7,4e4 <fail>
17
18 0000001c <test_3>:
19  1c: 00100093          addi  x1,x0,1
20  20: 00100113          addi  x2,x0,1
21  24: 00208733          add  x14,x1,x2
22  28: 00200393          addi  x7,x0,2
23  2c: 00300193          addi  x3,x0,3
24  30: 4a771a63          bne  x14,x7,4e4 <fail>
25

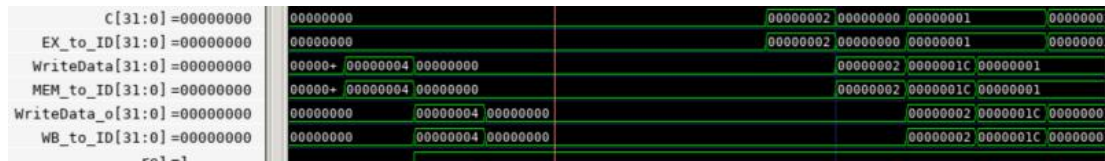
```

波形图:



分析：

由反汇编可知，在地址为 000c 时，会出现数据冒险  
在波形图中，



是前递的数据及其来源。

在 0c 数据冒险来临时，判断逻辑判断为 A、B 两种冒险情况，同时 re2 拉高一个周期。通过数据前递，使得冒险得以消除。

结构冒险：

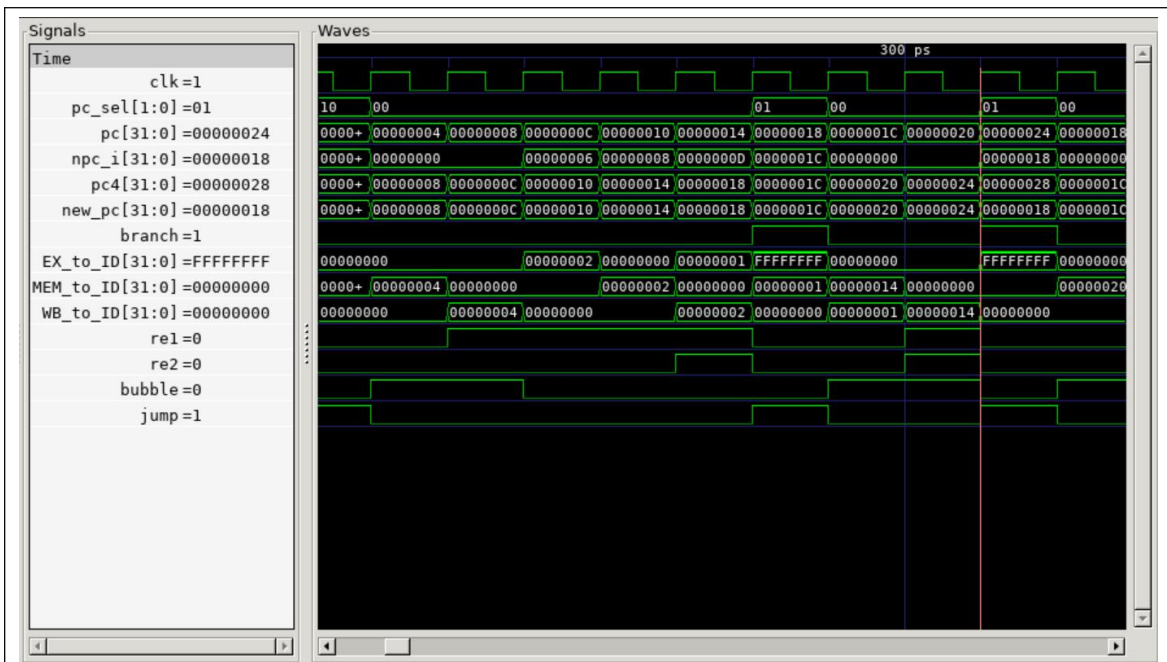
bne 指令反汇编：

```

bne.dump
1
2 bne:      file format elf32-littleriscv
3
4
5 Disassembly of section .text.init:
6
7 00000000 <_start>:
8   0: 0040006f          jal x0,4 <reset_vector>
9
10 00000004 <reset_vector>:
11   4: 00200193          addi x3,x0,2
12   8: 00000093          addi x1,x0,0
13  c: 00100113          addi x2,x0,1
14 10: 00209663          bne x1,x2,1c <reset_vector+0x18>
15 14: 2a301a63          bne x0,x3,2c8 <fail>
16 18: 00301663          bne x0,x3,24 <test_3>
17 1c: fe209ee3          bne x1,x2,18 <reset_vector+0x14>
18 20: 2a301463          bne x0,x3,2c8 <fail>
19
20 00000024 <test_3>:
21 24: 00300193          addi x3,x0,3
22 28: 00100093          addi x1,x0,1
23 2c: 00000113          addi x2,x0,0
24 30: 00209663          bne x1,x2,3c <test_3+0x18>
25 34: 28301a63          bne x0,x3,2c8 <fail>
26 38: 00301663          bne x0,x3,44 <test_4>
27 3c: fe209ee3          bne x1,x2,38 <test_3+0x14>
28 40: 28301463          bne x0,x3,2c8 <fail>
29

```

波形图：



分析：

**bubble** 为停顿表示变量。**bubble** 拉高表示流水线进入停顿。

由反汇编可以得到，地址为 0010 时跳转进入 001c。在波形途中，0010 时，检测到跳转判断条件成立，**jump** 拉高，随后 **bubble** 拉高，流水线进入暂停，并持续两个周期。两个周期后，**pc** 跳至 001c，**bubble** 回到低电平。

### 3 设计过程中遇到的问题及解决方法

(包括设计过程中的错误及测试过程中遇到的问题)

设计过程中犯过的错误:

- (1) 开始设计时对 CPU 结构及数据通路理解不够透彻
- (2) 设计之前没有先画数据通路图, 直接上手写代码, 导致走了一点弯路, 后来经过老师的提醒, 先完成了数据通路图。
- (3) 在设计单周期时对 pc 的更新机制开始时理解不透彻, 导致 bug 出现
- (4) 在流水线设计时对数据冒险的判断不够准确, 条件没有写全。

设计过程中遇到的问题:

- (1) 画数据通路时, 对多路选择的各个情况有疑惑, 后来经过思考得以解决。
- (2) 单周期设计时, 对 pc 变化的各个情况有疑惑, 后来通过和同学交流得以解决。
- (3) 流水线设计时, 对控制冒险不知如何解决, 后来同学建议使用停顿得以解决。

## 4 总结

(要求：个人收获以及对课程的建议)

通过这门《计算机设计与实践》的学习，我收获了很多、掌握了很多

- 1.对 verilog 这门语言的掌握进一步加深，已经能相当熟练地使用 vivado 来进行编程。
- 2.对 CPU 的构造、数据通路、实现方法有了更加深刻的理解；对 riscV 的指令集以及汇编有了更进一步的掌握。
- 3.对顶层开发更加熟练，代码能力进一步加强，调试能力更加成熟。
- 4.对 FPGA 以及硬件等领域有了更深的了解