中国科学技术大学计算机学院

《计算机组成原理实验》报告

实验题目：流水线 CPU 设计

学生姓名：黄瑞轩

学生学号：PB20111686

完成日期：2022.4.24

计算机实验教学中心制

2020 年 09 月

实验题目

流水线 CPU 设计

实验目的

- 理解流水线 CPU 的结构和工作原理
- 掌握流水线 CPU 的设计和调试方法，特别是流水线中的数据相关和控制相关的处理
- 熟练掌握数据通路和控制器的设计和描述方法

实验环境

- Nexys4-DDR
- Vivado 2019.1

实验 1: 无相关设计的流水线 CPU

● 先导器件设计

由于仍是 RISC-V 架构的 CPU，单周期 CPU 中大部分器件可复用。需要再进行先导设计的器件主要是流水段寄存器，这些寄存器见附件 IF_ID.v、ID_EX.v、EX_MEM.v 和 MEM_WB.v。

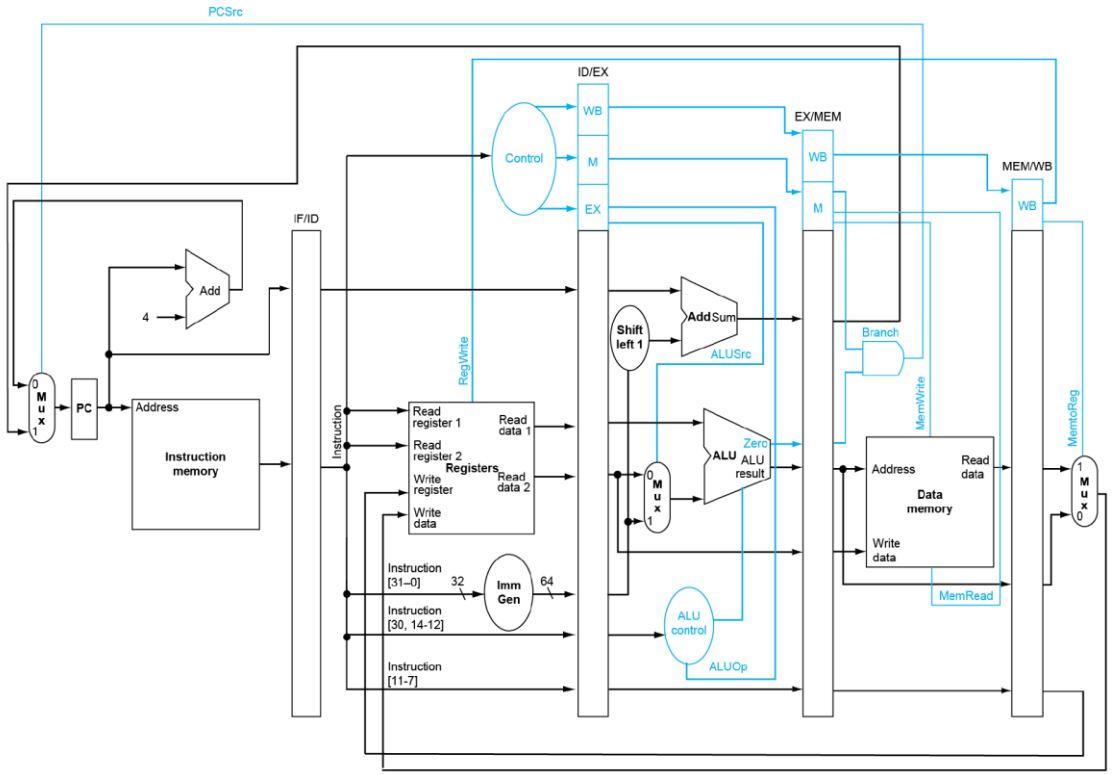
● 无相关处理的流水线 CPU 设计

➔ *要求实现的功能*

可执行 add, addi, sub, auipc, lw, sw, beq, blt, jal, jalr 这 10 条指令。并且能配合外设和调试单元 PDU，实现对 CPU 的下载测试。

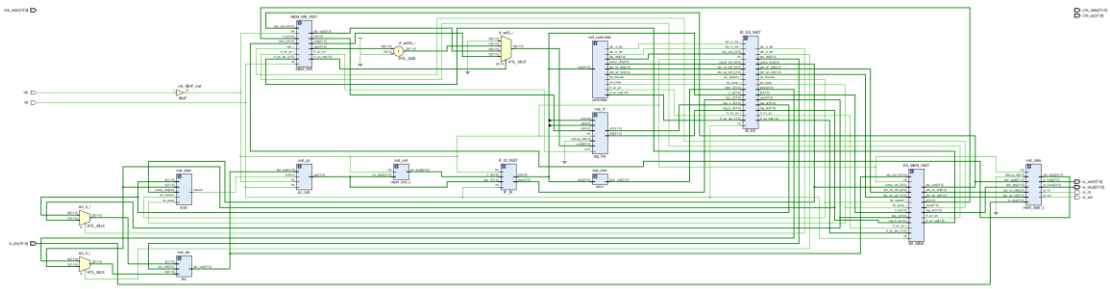
➔ *CPU 设计和核心数据通路*

设计采用 COD5 教材方案，数据通路图如下。



➔ *CPU RTL 电路图*

美观起见，这里展示的 RTL 电路图是去除 DBG_BUS 和 IO_BUS 后的。



➔ ***CPU 核心代码***

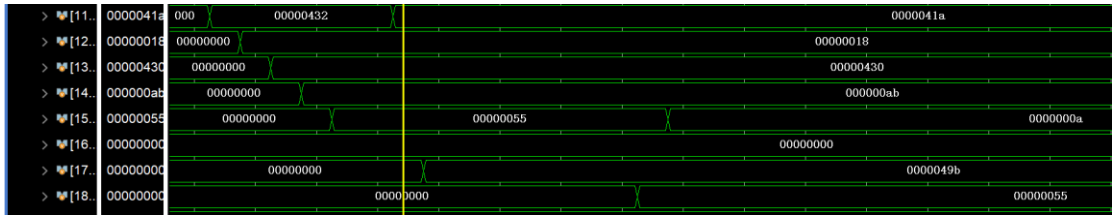
见附件 core_step1.v。

➔ ***CPU 仿真测试***

因为这里做的流水线 CPU 还没有加入相关处理，所以编写的测试汇编程序也不能出现相关问题。这里使用的仿真代码见附件 step1.asm，COE 文件见 step1.coe，仿真文件见附件 step1_tb.v。

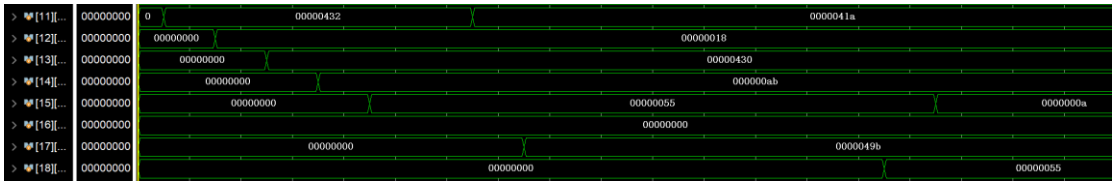
a0	10	0x00000000
a1	11	0x0000041a
a2	12	0x00000018
a3	13	0x00000430
a4	14	0x000000ab
a5	15	0x0000000a
a6	16	0x00000000
a7	17	0x0000049b
s2	18	0x00000055

RARS 执行完毕后寄存器状态



流水线 CPU 仿真结果

为了再验证正确性，使用 LabH4 设计的单周期 CPU 加载同样的 COE 文件。



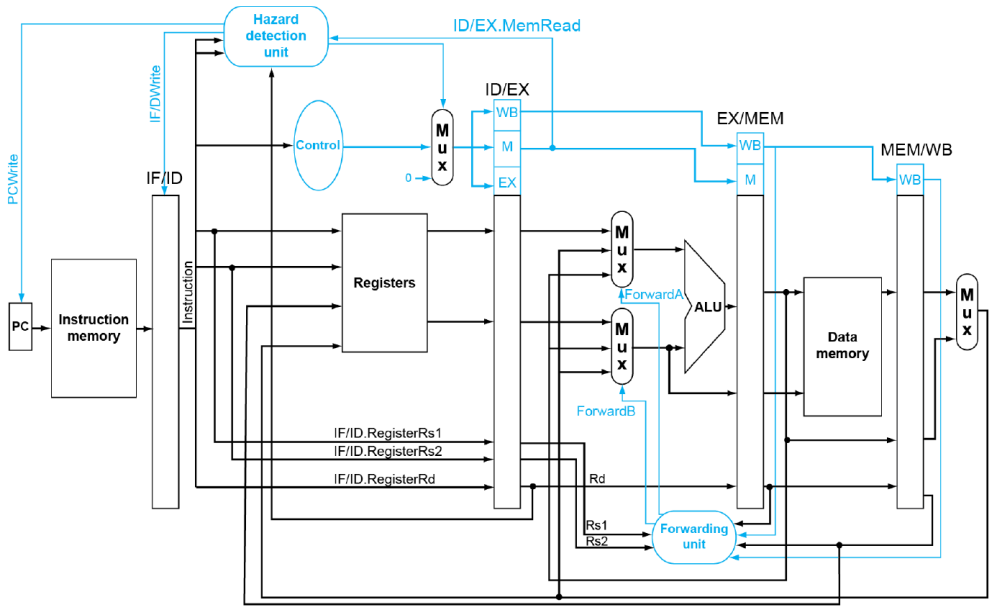
单周期 CPU 仿真结果

与流水线 CPU 仿真结果相比，内容和时序都完全正确。

实验 2: 仅有数据相关处理的流水线 CPU

➔ ***CPU 设计和核心数据通路***

带有前递和互锁功能的流水线 CPU 数据通路简图如下，前递元件的代码见附件 forward.v，互锁元件的代码见附件 detection.v。



前递条件设置：

```
if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10
if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10


if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
        and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs1))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
        and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs2))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

多选器控制	源	解释
ForwardA = 00	ID/EX	ALU 的第一个操作数来自寄存器堆
ForwardA = 10	EX/MEM	ALU 的第一个操作数来自上一个 ALU 计算结果的前递
ForwardA = 01	MEM/WB	ALU 的第一个操作数来自数据存储器或者更早的 ALU 计算结果的前递
ForwardB = 00	ID/EX	ALU 的第一个操作数来自寄存器堆
ForwardB = 10	EX/MEM	ALU 的第一个操作数来自上一个 ALU 计算结果的前递
ForwardB = 01	MEM/WB	ALU 的第一个操作数来自数据存储器或者更早的 ALU 计算结果的前递

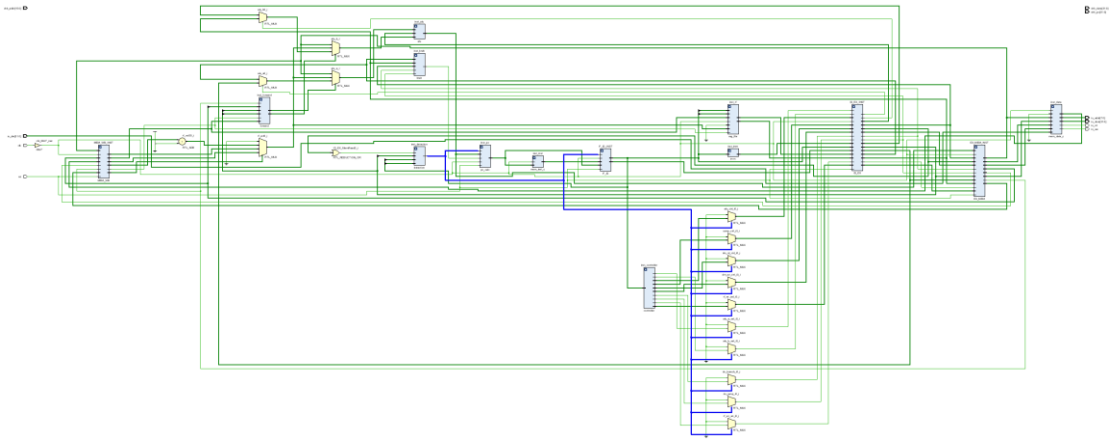
互锁条件设置：

```
if (ID/EX.MemRead
    and ((ID/EX.RegisterRd = IF/ID.RegisterRs1)
        or (ID/EX.RegisterRd = IF/ID.RegisterRs2))) Hazard = 1
```

当 Hazard 有效时，IF/ID 以及 PC 都不更新，ID/EX 接收到的控制信号为全 0。

➔ ***CPU RTL 电路图***

美观起见，这里展示的 RTL 电路图是去除 DBG_BUS 和 IO_BUS 后的，蓝色线路部分指示 Hazard Detection 模块。



➔ ***CPU 仿真测试***

因为这里做的流水线 CPU 还没有加入控制相关处理，所以编写的测试汇编程序也不能出现控制相关问题。这里使用的仿真代码见附件 step2.asm，COE 文件见 step2.coe，仿真文件见附件 step1_tb.v。

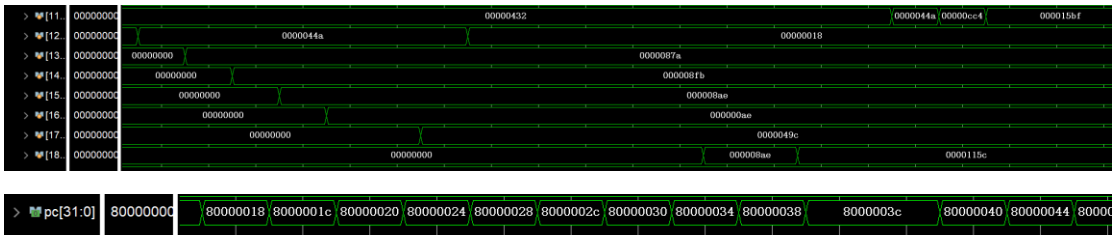
step2.asm

```
1  .text
2      addi a1, a1, 0x432
3      addi a2, a1, 0x18
4      addi a3, a2, 0x430
5      ori a4, a3, 0xab      一般数据相关
6      xori a5, a4, 0x55
7      andi a6, a5, 0x0ff
8      nop
9      xor a7, a1, a6
10     li a2, 0x18
11     sw a5, 0(a2)
12     nop
13     nop
14     nop
15     lw s2, 0(a2)          Load_Use 数据相关
16     add s2, s2, s2
17     nop
18     add a1, a1, a2        双重数据相关
19     add a1, a1, a3
20     add a1, a1, a4
```

step2.asm 内容

a1	11	0x000015bf
a2	12	0x00000018
a3	13	0x0000087a
a4	14	0x000008fb
a5	15	0x000008ae
a6	16	0x000000ae
a7	17	0x0000049c
s2	18	0x0000115c

RARS 执行完毕后寄存器状态



流水线 CPU 仿真结果

与 RARS 单步调试结果相比，内容和时序都完全正确。

➔ ***CPU 上板测试***

约束文件沿用 LabH4 的，这里不在附件中给出了。上板情况已经线下检查，这里仅以第 11 号寄存器的内容变化作为示例。

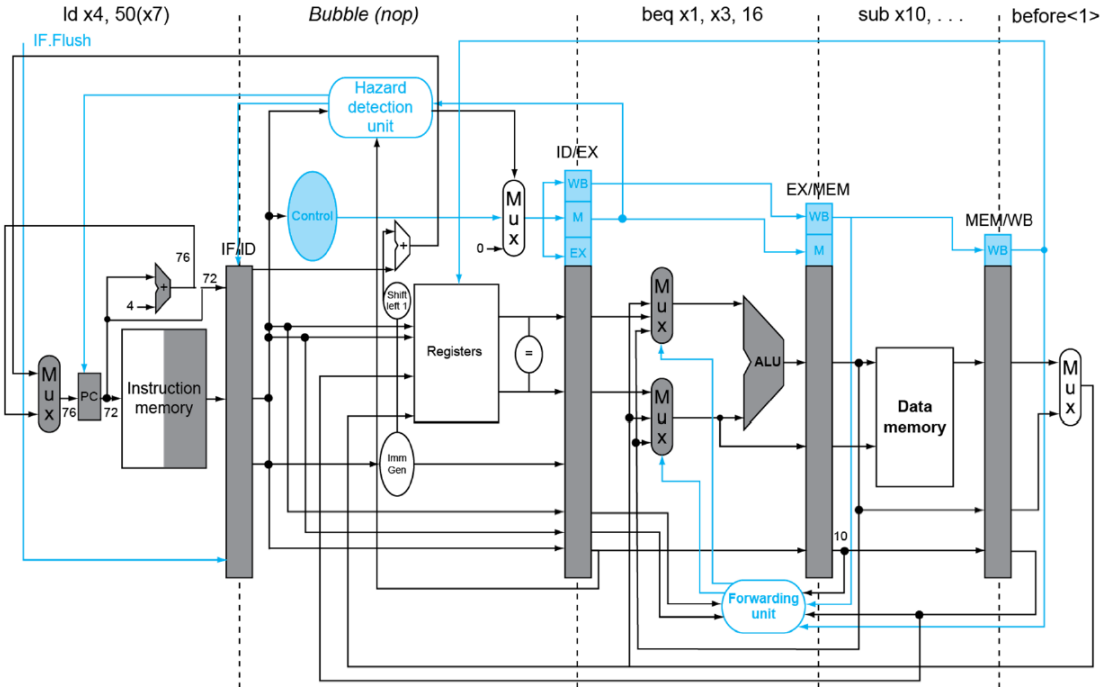
设置 chk_addr 为 100b（R11）→ 点击 chk，此时内容为 0 → 点击 step，此时内容变化

持续点击 step，可以看到内容按照波形图变化，验证成功。

实验 3: 有数据和控制相关处理的流水线 CPU

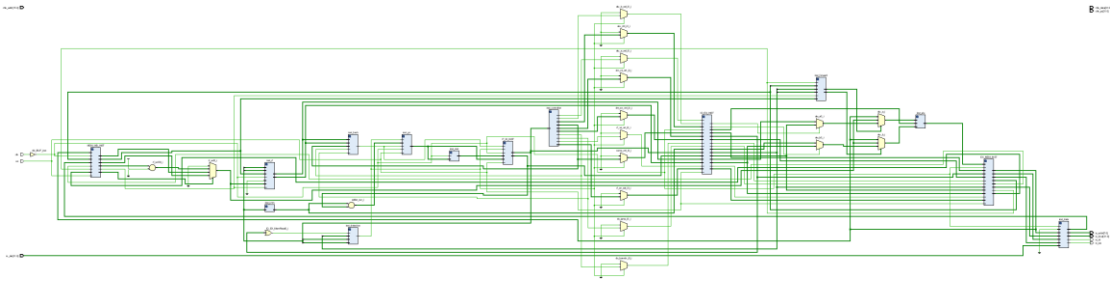
➔ ***CPU 设计和核心数据通路***

本设计处理控制相关的方式是，将 branch 的输出以及地址计算移动到 ID 段进行，如果分支成功，则将下一条指令设置为 nop。数据通路如下图所示。



➔ ***CPU RTL 电路图***

美观起见，这里展示的 RTL 电路图是去除 DBG_BUS 和 IO_BUS 后的。



➔ ***CPU 核心代码***

见附件 core_step3.v。

➔ ***CPU 仿真测试***

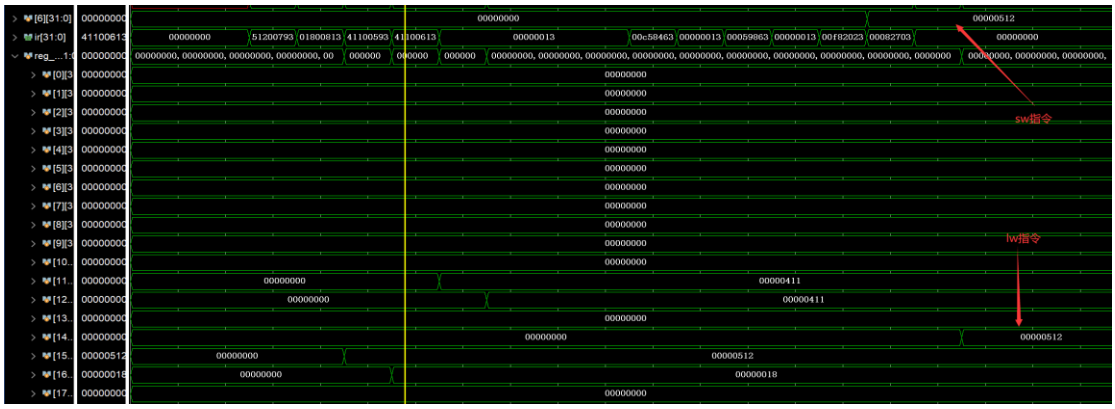
这里使用的仿真代码见附件 step3.asm，COE 文件见 step3.coe，仿真文件见附件 step1_tb.v。

```
1  .text
2      li a5, 0x512
3      li a6, 0x18
4      addi a1, x0, 0x411
5      addi a2, x0, 0x411
6      nop
7      nop
8      nop
9      nop
10     beq a1, a2, not_exit
11     j exit
12 not_exit:
13     bne a1, x0, exit
14     and a1, a5, a6
15     or a5, a1, a2
16     add a7, a2, a2
17 exit:
18     sw a5, 0(a6)
19     lw a4, 0(a6)
```

step3.asm 内容

a1	11	0x00000411
a2	12	0x00000411
a3	13	0x00000000
a4	14	0x00000512
a5	15	0x00000512
a6	16	0x00000018
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000

RARS 执行完毕后寄存器状态



流水线 CPU 仿真结果

与 RARS 单步调试结果相比，内容和时序都完全正确。

➔ ***CPU 上板测试***

约束文件沿用 LabH4 的，这里不在附件中给出了。上板情况已经线下检查，这里仅以第 15、16 号寄存器的内容变化作为示例。

设置 chk_addr 为 100f

→点击 chk，查看内容

切换 chk_addr 为 1010，此时内容为 0 → 点击 step，内容变为 18

➔ ***CPU+PDU 排序程序上板测试***

由于此次排序程序上板测试现象与 LabH4 一样，以及线下也检查了相关的部分，因此仅给出 bit 文件（附件 cpu_sort.bit），不再给出上板照片。

总结篇

● 收获

这次实验中，我通过亲手设计一个流水线 CPU，并且处理数据相关和控制相关的有关问题，增进了对流水线 CPU 以及数据相关、控制相关的发生与解决方法的理解。除此之外，任务量和内容与 LabH4 基本一致。

● 建议

（1） 由于没有做多周期的 CPU 或者其他比较方法，看不出所做的流水线 CPU 性能与单周期 CPU 相比的实际差距在哪，可以改进。

（2） 本次实验出了制作流水线 CPU 结构外，没有什么新意，可以改进实验要求。