简单多周期 CPU 实验：实验报告

刘泓尊　2018011446　计 84

一、实验目的

1. 加深对 SRAM 时序的理解
2. 掌握指令执行阶段的含义，熟悉执行阶段所需要的信号
3. 理解一个简单程序的执行过程，深入理解多周期 CPU 的特点

二、实验细节

整个项目分为 IF 模块，ID 模块，EX 模块，REG_FILE 模块，IO(MEM)模块和 CPU 控制模块。因为指令存储和数据存储共用一个总线，所以取指和访存不能同时进行，当然目前也没有引入流水的实现。多周期处理器相对于单周期 CPU 的优点是它可以根据指令类型来省略一部分阶段，从而提高运行效率。

时钟使用 50MHz 时钟。

• CPU 控制就是一个大的状态机：

IF 阶段：从 SRAM 读数据，地址是 `pc[21:2]`。可以使用多个周期完成。

ID 阶段：译码，判断指令类型，从指令中解析出寄存器地址、立即数等信息。同时在这一阶段判断 `beq` 是否进行跳转。

EX 阶段：按指令类型在 ALU 中执行。ALU 是组合逻辑。

MEM 阶段：将结果写入 SRAM，或者从 SRAM 取数据。（只有 lw, sw 指令涉及到这个阶段，其他指令可以跳过），可以使用多个周期完成。

WB 阶段：将运算结果写回寄存器。我选择在这一阶段更新 pc.

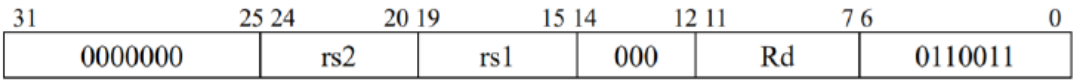
• 其他模块：

REG_FILE: 读寄存器是组合逻辑，写寄存器是时序逻辑。注意寄存器 0 始终要保持为 0。

MEM：注意 SRAM 的地址以 32 位为单位，需要忽略 CPU 中地址的后两位。

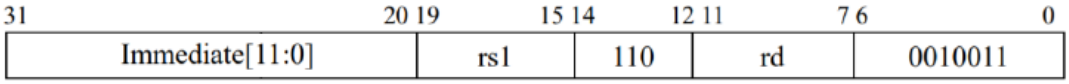
• 对于通过测试的 5 种指令分析如下：

• ADD：`add rd, rs1, rs2`. R 类型，忽略溢出



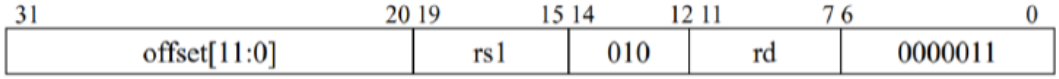
• ORI：`ori rd, rs1, immediate` I 类型

将 12 位立即数 `imm` 符号拓展之后和寄存器 `rs1` 按位或，将结果写入 `rd`



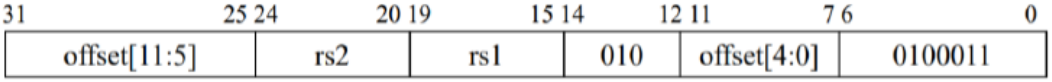
• LW: `lw rd, offset(rs1)` I 类型

将 12 位立即数 `offset` 进行符号扩展后和寄存器 `rs1` 相加，并将结果对应地址开始的 32 位字写入寄存器 `rd`



• SW `sw rs2, offset(rs1)` S 类型

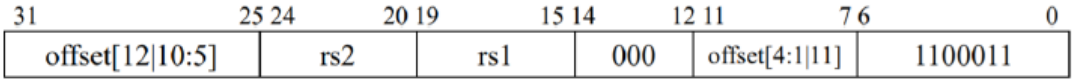
将 12 位立即数 `offset` 符号扩展后和寄存器 `rs1` 相加，并将寄存器 `rs2` 中的内容写入对应地址。



• BEQ: `beq rs1, rs2, offset`

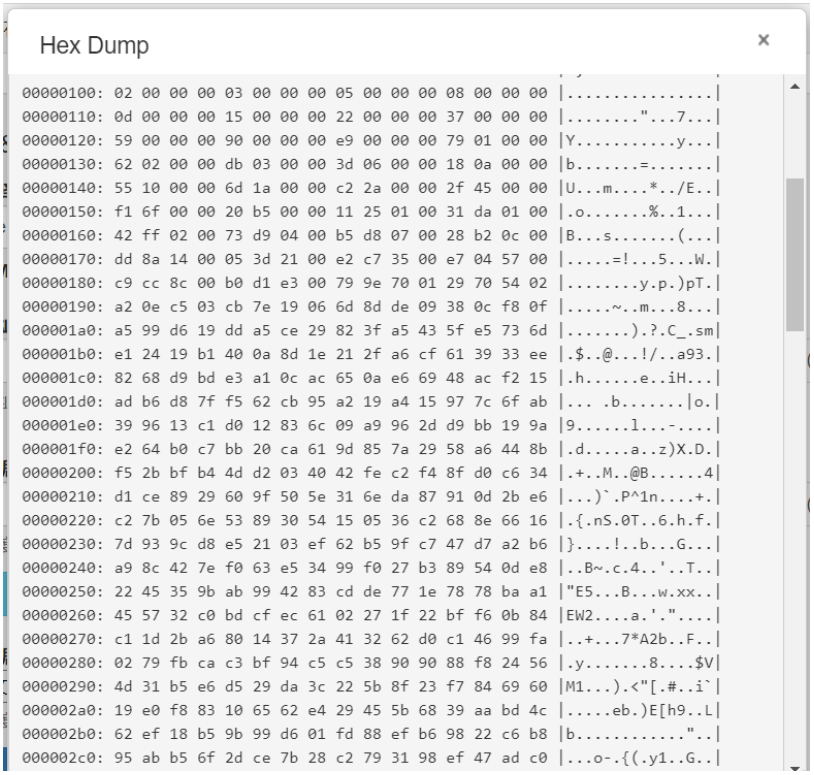
若 `rs1` 和 `rs2` 内容相等，则 `pc` 跳转到 `pc+offset`，否则 pc 跳转到 pc + 4

注意：`beq` 指令省略了 `offset` 末尾的 0，所以求 offset 时需要在末尾补个 0



三、实验数据

下图可以看到 `0x100` 开始的地址处已经正确写入 Fib 数列。



四、思考题

1. 指令同样被分为 5 个执行阶段，那么多周期相比于流水线的不同地方在哪里？（流水线多做了什么事情？）

流水线的 5 个模块是并行执行的，可以看做每个模块都是一个状态机，它们同步工作。同时每个阶段之后设置了流水线寄存器，保存上一个节拍的输出结果。

对于多周期来讲，因为 5 个阶段串行执行，所以不存在太多的数据冲突、结构冲突和控制冲突等，指令存储器和数据存储器也可以共用一个总线，5 个阶段称为一个大状态机即可。

但是对于流水线 CPU，要设计合理的流水线暂停和数据旁路等，来解决可能出现的冲突。并且因为取指和访存同步进行，需要有指令总线和数据总线来避免冲突。

2. 在实验的过程中遇到什么样的困难，是如何解决的。

a. 因为不太熟悉硬件设计，为了把每个模块独立出来，进行了很多次代码重构，最后勉强将每个阶段做成一个模块，最后在顶层文件中用 `wire` 连接起来

b. 一开始并不了解 latch, 导致一直无法调通代码。最后根据老师和助教为微信群中的提醒才得以解决。

c. 仿真尚不熟练，只会进行很基本的仿真操作，花费了一些时间。

五、实验小结

有了上次 SRAM 实验的基础，本次实验上手起来没有那么困难了，我主要的时间花在 CPU 的设计上，其实先把实现细节想好，实现起来并没有太大阻力。此外我复用了上一个实验的 SRAM 模块，也给本次实验减轻了不少压力。

本次实验初步实践了一个简单的 CPU，收获良多。老师和助教们在微信群里对 verilog 规范的提醒也让我收益很多，感谢老师和助教的悉心指导！