计算机组成原理

ThinPAD 指令计算机系统设计和实现

实验报告

第 24 组

谢云桐[1] 吕松霖[2] 李祥泽[3]

[1] 2018011334 计 83

[2] 2017010073 计 82

[3] 2018011331 计 83

1.目标概述

实现一个基于简化的 RISC-V 指令系统的 32 位多周期 CPU。

该 CPU 具有以下功能：

1. 执行来自 RV32I 的 19 条基本指令，运行基础版本的监控程序；
2. 执行来自 RV32B 的 3 条扩展指令；
3. 检测并处理异常，执行 CSRRW、ECALL、MRET 等 6 条指令，运行中断和异常版本的监控程序；
4. 对用户态的内存空间做分页映射，运行页表版本的监控程序。

2.主要模块

2.1数据通路

数据通路由寄存器组（**RegFile.v**）、ALU（**ALU.v**）、立即数生成器（**ImmGen.v**）等 3 个模块，以及若干中间寄存器和多路选择器组成。

寄存器组实现了 RV32I 要求的 32 个通用寄存器。可以同时输入 2 个读取地址和 1 个写入地址，对应 2 个数据输出和 1 个数据输入。读出为组合逻辑，写入需要使能信号和时钟上升沿。

ALU 实现了 19+3 条指令所需的计算功能。得益于 RISC-V 的指令编码设计，对于 RV32I 的指令，可以直接使用指令字的 funct3 和 funct7 字段来编码 ALU 的功能。这样一来，译码器无需做任何额外操作，而只需将对应字段传给 ALU 即可。

但是，由于有来自于 RV32B 的扩展指令，且这些指令与 RV32I 的 R 型和 I 型计算指令使用了相同的 OpCode 和 funct3，造成编码混乱。只能增加 1 位控制信号以简化判断逻辑。

立即数生成器会将指令字中的不同字段按不同的立即数格式做拼接和扩展，同时生成 5 种格式（I、J、B、U、L）的立即数，并按控制信号输出所需的。

2.2控制器

控制器由译码器（**Decoder.v**）模块、顶层模块中的阶段状态机，以及若干中间寄存器和多路选择器组成。

阶段状态机共有 8 个状态，其中 1 个是按下 RST 时进入的空状态，另 1 个是死机状态；剩余 6 个，分别对应指令执行的取指、译码、执行、访存、写回 5 个阶段以及异常处理的 1 个阶段。

状态机的次态由译码器生成。在时钟边沿到来，且内存访问没有阻塞的情况下，状态机将自动转移到次态。

译码器以指令字、状态机的阶段、ALU 的“零标志位”、CPU 的运行模式以及 3 个异常标志位为输入，生成每个阶段所需的控制信号和阶段状态机的次态，同时在发生异常时根据指令类型和执行阶段生成相应的错误码（`mcause`）。

2.3内存和串口

两片内存和串口统一由 `RamController` 模块控制。

该模块接管内存和串口的一切输入输出信号，并将其包装成抽象的、更接近组合逻辑的控制。以这种方式，译码器将不需要为内存相关阶段设立多个状态。

另外，为了解决内存访问和其他操作用时差别较大的问题，在该模块中会输出“读写完成”信号，用于在读写过程中阻塞阶段状态机。

2.4异常处理器

`ExcepHandler` 模块实现 CSR 的读写、当前特权状态的维护等与异常和中断相关的功能。

2.5页表和 MMU

`MMU` 模块位于顶层模块与 RamController 之间，实现虚实地址翻译等功能。

该模块接管 RamController 的一切信号，并向顶层提供相同的控制约定。在其中用一个状态机实现了 Sv32 标准的地址翻译逻辑。同时，该模块也实现了页表相关的异常检查逻辑。

3.成果展示

在上交材料中含有我们编写的若干汇编测试程序，一部分是正确功能的展示，另一部分会引发异常。在每个文件中都用注释写明了操作方法和期望输出。可用于展示我们实现的功能。

以上测试程序的运行截图，出于排版整洁的原因，附于本文最后。

需要说明的是，在实现了页表之后，3 个 Access Fault 异常就不再能够触发了。因为触发这些异常需要访问非法的地址，而非法的地址在本实验中一定是未映射的地址；所以一定会优先触发相应的 Page Fault。

4.分工

我们的设计和调试工作大多共同完成的，具体代码的编写按模块分工如下：

谢云桐：顶层模块、寄存器组、MMU；

吕松霖：译码器、立即数生成器；

李祥泽：ALU、内存控制器、异常处理器。

5.心得和体会

通过本次实验，我们加深了对于 CPU 的结构、原理和运行方式的理解。特别是对于以前不甚了解的 CPU 内部的一些对外封装的控制细节，有了第一手的经验。

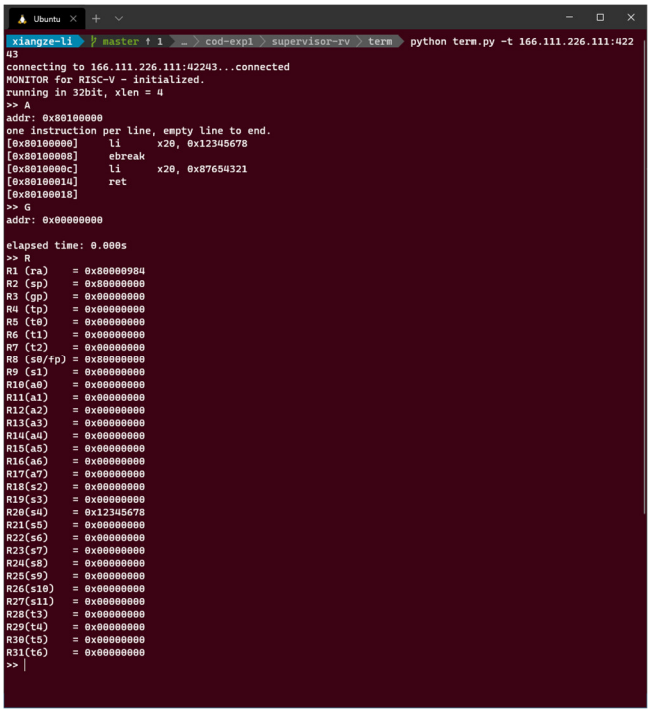
我们对于 RISC-V 指令系统、特别是指令字编码的巧妙设计也感到非常敬佩；体会到要设计一个既方便硬件设计、又方便软件使用的指令系统的困难。

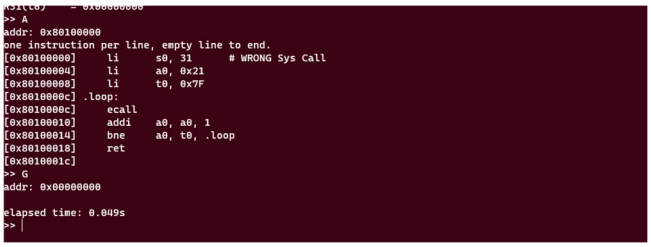
另外，我们还更熟练地掌握了使用 EDA 软件进行硬件设计和调试的方法。我们的许多 Bug 都是通过 EDA 的前仿真和后仿真发现的。甚至在前仿真环境中运行了 kernel 和仅一行的用户程序。不过，我们最后还是没有使用 ILA 直接截取硬件中信号的技术。主要是因为没能够在编译后的电路中找到和代码中对应的信号。

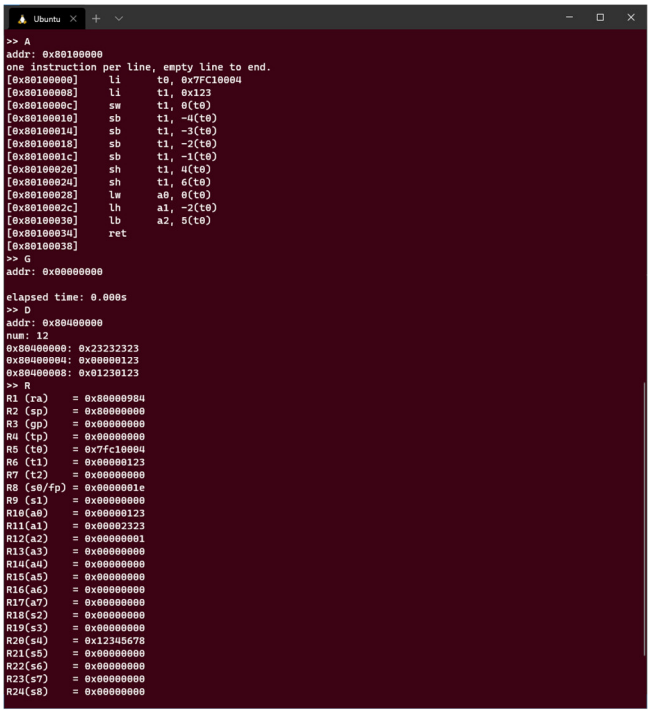
6.提交材料清单

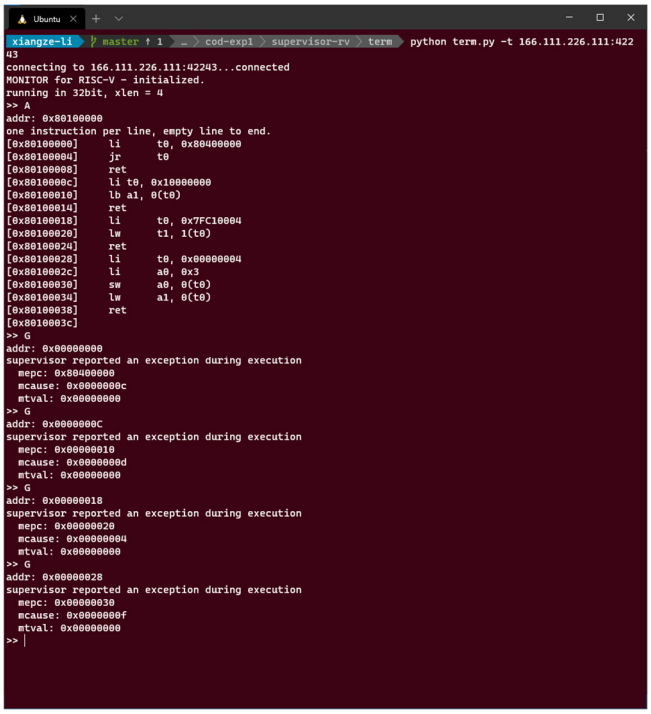
```
grp24.zip
|- Report.pdf (本文件)
|- thinpad_top.xpr
|- TestCases
|   |- TestCases
|   |   |- EBREAK.s
|   |   |- ECALL.s
|   |   \- Paging.s
|   \- FailCases
|       |- ECALL_Wrong.s
|       |- Instr_PageFault.s
|       |- Load_PageFault.s
|       |- Misalign_w_Paging.s
|       \- Store_PageFault.s
|- Kernels
|   \- (编译好的若干二进制文件)
\- thinpad_top.src
    |- sources_1/new
    |   |- ALU.v
    |   |- ClkGen.v
    |   |- Decoder.v
    |   |- ExcepHandler.v
    |   |- ImmGen.v
    |   |- MMU.v
    |   |- RamController.v
    |   |- RegFile.v
    |   \- thinpad_top.v
    \- (others)
```

7.测试程序运行截图









正例。左列上：EBREAK；下：带页表映射的各种长度访存；右列上：ECALL。

反例。右列中：用不正确的调用号 ECALL；下：3 种 Page Fault 和 1 种 Addr. Misalign。