南京大学

《数字逻辑与计算机组成实验》
实验报告

实验十二：大实验——基于RISC-V架构的单周期CPU

姓名： 程子涵

学号： 201220015

班级：数据逻辑与计算机组成实验2班

院系： 计算机科学与技术系

邮箱： czh@smail.nju.edu.cn

实验时间： 2021/12/29

目录

7.6 LED控制指令

八、问题与解决方法

九、实验体会

一、完成效果

本实验完成了一个基于RISC-V架构的单周期CPU设计，可以使用键盘和显示器进行简单交互，完成命令的输入和解析，可以通过输入命令进行简单计算、屏幕颜色改变和开发板LED灯管的控制。

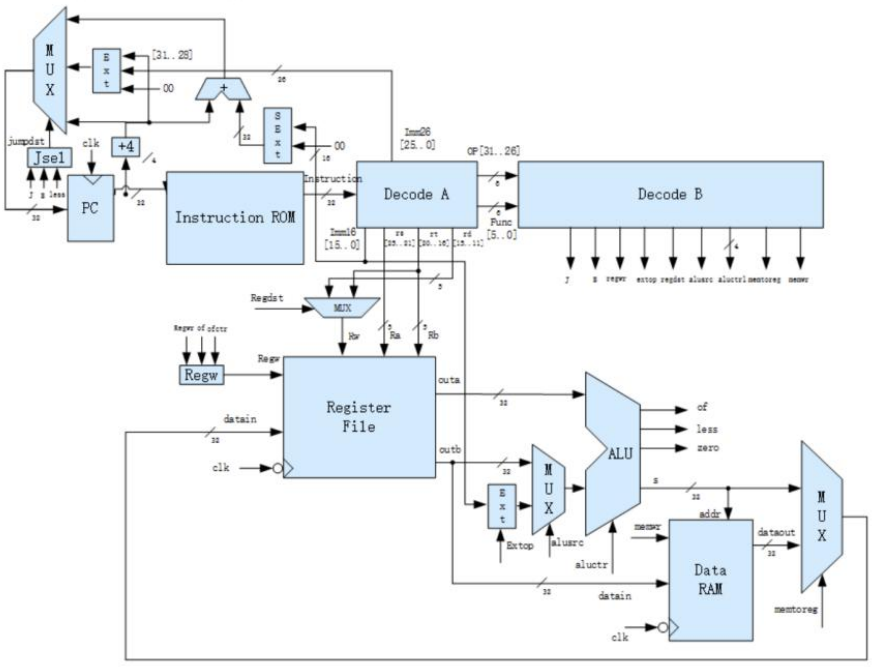
二、完成情况

独立完成CPU部分的设计、RISC-V架构常用指令的实现、外设部分的调试和系统软件的编写等。

三、主要结构介绍

3.1 架构介绍

该计算机系统架构参考实验手册上的架构框图：



主要组成部分包括指令计数器（PC）、二选一选择器、解码器、寄存器、ALU，另外有主存用于只读数据的存储和程序运行时栈的支持。

3.2 CPU部分

3.2.1 寄存器

CPU内部寄存器与RISC-V指令集要求的寄存器实现相同，一共有32个32位的寄存器，使用5bit作为寄存器地址。

3.2.2 指令

CPU指令部分实现了RISC-V架构中的算数和逻辑操作指令；部分Load、Store 指令和分支跳转指令。指令的实现主要参考实验十一。由于系统实现过程中没有更多需要，因此没有对指令集进行过多拓展。

指令使用256个单元的32位ROM进行存储，系统运行时在decoder模块进行指令解码之后转入相应流程执行。

3.3 主存部分

3.3.1 主存的实现

主存部分使用IP核自动生成的OnChipMemory来实现，一共有65536个32位存储单元，按照字节编址，存储地址从0x1000 0000开始。

3.3.2 主存的划分

主存一共划分为5个部分，各个部分的功能核大小如下：

1. VGA缓存部分：0x1000 0000 - 0x1000 20D0（共8400字节），CPU可以通过对这一区域进行读写来改变屏幕显示内容；
2. 键盘缓冲区：0x1000 20D0 - 0x1000 20D4（共4字节），键盘

输入有效时输入对应的ASCII码将会被送入内存该地址处；

3. 输入缓存：0x1000 2100 - 0x1000 2300（共512字节），CPU将输入的字符串保存到该区域，待以后进行字符串比较核命令解析时使用；

4. 只读数据段：0x1000 2400 - 0x1000 2A00（共1536字节），用于存放常量和字符串，例如开机时打印的欢迎语、帮助信息和“Hello World!”字符串等；

5. 栈区：0x7FFF F800 - 0x7FFF FFFF（共8192字节），用于为函数的过程调用提供支持。

3.4 系统程序

前面的CPU部分已经实现的RISC-V架构程序中需要使用到的大部分指令，因此只需要一套能够完成系统功能的汇编程序即可完成本次计算机系统的实现。

3.5 各类外设

外设部分主要是键盘和显示器。其实本实验中外设的问题总体来说更少，一方面因为前面的实验多次用到键盘和显示器，这两种外设的代码相对已经比较成熟，另一方面本实验中大部分内容都是通过软件来实现的，因此只需要编写合适的软件程序就可以完成各种字符显示、颜色更改等行为，这就简化了硬件的设计。

四、实验器材

（一）软件环境:

Quartus 17.1 Lite

（二）硬件环境:

（1）开发板：DE10 Standard

（2）FPGA：Intel Cyclone V SE 5CSXFC6D6F31C6N

五、CPU内部各部分设计思路

5.1 寄存器部分

CPU需要32个32位寄存器，显然寄存器部分的空间并不大，因此可以自行编写RAM实现。

但是由于某些指令需要同时读取两个寄存器中的内容，因此这一块RAM需要有两个读口，同时需要一个写口，并提供一个写使能。

这块RAM在时钟控制上只需要一个时钟信号，可以板上的50MHz时钟信号来作为时钟信号控制。

5.2 指令寄存器

指令寄存器其实就是一个ROM，读取RISC-V架构中需要用到的所有指令的OP code，提供一个读口。

5.3 译码器部分

译码器部分核心功能就是根据指令的前6位OP部分将指令不同部分进行划分，再根据划分出的这些部分继续进行分支处理。

六、存储部分实现思路

最初的思路是划分出多个存储单元，分别提供给各个不同的部件使用。但是这样的思路也遇到了问题，计算机系统的各个部分之间存在大量的数据交换的需要，划分成多个存储单元则数据交换会有巨大的困难。因此最后决定使用一个比较大的主存划分成不同的部分，这样数据交换的时候只需要在各个模块之内将自己分到的那一部分内存的内容进行修改，再将那一部分内容的内存地址传给其他部分。

计算机系统部分可以共享一个统一的主存，但是键盘的待处理队列、ASCII码查找表部分还是需要在模块内给出单独的存储空间，主存中只是给出两个ASCII码的缓冲区。

VGA显示部分，因为需要实现60Hz的刷新率，因此显存部分需要被以很高的频率访问，因此VGA部分需要和CPU主要部分独立开来，异步运行，因此这里也需要给VGA模块单独划分出一块内存。

七、功能介绍和实现思路

7.1 命令识别

如前所述，如果能用C语言进行系统软件编写，命令的识别就变得非常简单了,只需要将缓冲区内的字符串依次与预存的指令依次比较，如果识别出有效指令就转到指令的处理部分，如果无法识别指令返回错误信息即可。

7.2 斐波那契数列的计算

输入:

```
fib n
```

就可以计算出斐波拉契数列中第n项的值。该指令使用循环迭代的方式实现。

7.3 hello命令

输入:

```
hello
```

将会打印出“Hello, World！”消息。系统在识别出hello指令之后将需要打印的字符串地址（其实就是C语言中的指针）传送给print函数，后者将这些字符串“挪”到预留给显示器的位置，最后由显示器模块自动扫描这一块内存中的内容并修改自己显存中的内容，最终将消息打印在屏幕上。

7.4 颜色的修改

修改所有字符的颜色其实比较简单，设置一个全局变量用来表示系统中所有字符的颜色，进行颜色修改的指令的时候只需要修改该全局变量就能改变系统中所有字符的颜色。

7.5 清屏指令

清屏指令其实就是将预留给VGA部分的所有内容清空。

7.6 LED控制指令

可以设置10个LED控制全局变量表示LED的开关，LED控制指令其实就是将这些变量取反。输入：

```
LED n
```

就可以控制编号为n的LED的开关。

八、问题与解决方法

本次实验中遇到的最大问题其实是显示模块的设计方法问题。

之前的实验如显示器实验、字符交互实验中，显示模块的所有功能完全是通过硬件实现的，例如字符交互模块中命令提示符是通过每开辟一个新的行就用硬件向显存中写入固定内容来实现的。因此开始做这个实验的时候十分自然地就想用硬件的方式实现例如换行、命令提示符等功能。

但是随着实验的进行，用硬件进行编码的问题也逐渐暴露出来，一个是佢併伂比较多，另一个是与计算机系统的实现不统一。后来想到，既然已经实现了计算机系统的软件部分，不如各种问题都使用软件编程实现。使用软件实现比硬件实现顺畅了很多。

九、实验体会

本实验给我们一个机会自己动手从硬件到软件实现了一个比较简单的计算机系统，确实是对自己能力的一次很好的锻炼，也让自己对与计算机的组成和非x86指令集的运行原理有了更深的了解。

但是在进行实验时也有两点感受：实验一开始上手非常艰难，可以说是完全没有思路，课程网站上提供的资料还是有点简略。还有实现系统的时候发现CPU部分反而不是最难写的， 系统软件部分的编写才是最困难、最花时间的。