基于ZYNQ的FPGA部分实现LC-3架构：可行性分析，任务目标，以及architecture的规划

1. 可行性分析

材料：拥有一块别人友情赞助的ZYNQ开发板，其有一块Xilinx 7000 的arm处理器，一块Artix 7 的FPGA。

由于ARM处理器架构过于复杂，我们这里暂时不使用，只使用FPGA的部分。

我们的目标是实现LC-3架构，其拥有16种指令。

对于vivado的synthesis过程来说，中断处理开发难度较大不在计划之内，2^16位内存需要大量综合的时间。因此最终采取2^8\*2^16的内存，也就是2MB的存储空间，

在与电脑的通讯方面，我们将参考中科大的digital lab代码，使用coe文件动态烧写。

[2.3 LC-3 处理器 - Digital Lab 2024](https://soc.ustc.edu.cn/Digital/2024/lab8/LC_3/)

关于LC-3的编译器

感谢[laser/src/label.c at testing · PaperFanz/laser · GitHub](https://github.com/PaperFanz/laser/blob/testing/src/label.c)开发的laser库使得LC-3转机器码成为可能。具体实现细节我们就按黑箱操作就行，不过此类编译器的基本原理都是刚开始创造一个变量名和地址的mapping，称为symbol table。第二遍再进行assemble，遇到变量就用symbol table上对应的值替换。

1. 任务目标

通过FPGA实现LC-3架构

Input

一个按钮负责切换显示哪个寄存器

一个按钮复位所有寄存器

Output

10条线控制一组（4个）七段数码管，显示当前寄存器的值

四个LED显示当前查看的是哪个寄存器，9个GPR和PC还有MDR和MAR，一共12个寄存器

因此，经过长时间的研究，这个项目的难度已经被降到了最低，接下来的任务和最大难点就是如何简化并实现这个LC-3结构。

1. Architecture的规划

指令集分析：

Operate ADD/AND/NOT

Control BR/JMP/JSR

Data transfer LD/LDR/LEA/ST/STR

接下来应该研究在原本的LC-3架构上我们要保留哪些，删除哪些

开发流程将分为三个部分—我们先实现三种最简单的操作指令，然后再是数据搬运指令，最后是控制指令，符合先易后难的原则。

有一个概念很重要：微指令

微指令序列相当于一个虚拟的状态机，一个指令（如ADD）由多个微指令组成。

微指令储存在ROM中，读取速度非常快（每条都要单周期内完成），由COE文件进行memory的预设。

微指令具有易于扩展的优点，同时其逻辑清楚适合初学者编写。

在我们的设计中，微指令将以此类格式出现：

0-1: alu\_ctrl

2-11: reg\_ctrl

1. xx: mem\_ctrl

xx-xx+2: COND

xx+3-xx+6: next address（j-field）

微指令拥有自己的内存条和地址系统，有一个专门的微指令计数器，保证指令每周期更新一位。

1. field是满足COND后会跳转的位置。

但是对于有些需要等待的COND类型，就有可能把J-field改为其他的地址入口。内存等待微指令（R）就令地址不变，使得程序一直等待直到内存更新完毕。

条件跳转微指令（IR11）就通过NZP寄存器和当前IR的要求进行跳转。

这个微指令运行的架构参考的是

[LC3-CPU/ControlSignals/ControlStore.csv at main · MatthewKing2/LC3-CPU · GitHub](https://github.com/MatthewKing2/LC3-CPU/blob/main/ControlSignals/ControlStore.csv)

他的结构很清楚，还留下了极大的扩展空间。

写于2025/7/16，此时这个项目已经接近完成，还有一些微小的地方需要改动。接下来就是最后一步—把程序烧录进FPGA。

对于我的工作，有一些经验应该记录下来。

1.关于系统的调试

在这次开发中，一个大的通用中央处理器被分成Memory，regfile，control-logic等等内容来处理，这其实就是module test的基本思想。通过前期top-down的设计指导，程序的重心被移到了微指令部分，而其他部分的重点只在于连线。

这给我们以启示：对于大型项目，使用top-down设计，先搭好框架，再继续工作。

2.关于硬件描述语言的设计

在这次开发中，刚开始我想采取自己的设计思路—也就是把所有mux，处理逻辑和内存都在一起编写，后来看到matthexking的结构设计才恍然大悟。

在硬件描述语言中，有一个基本结构： topmodule =control module + memory module。

这样调试第一是对外界调用这个结构很友好—所有的数据传输都已经在内部完成了，第二是逻辑比较清晰—内存模块负责接收地址和enable，write-read-en，输入数据，然后输出内存内容就好。

而控制模块只需接受input，访问内存，读或写，然后把结果发送出去就行了。

3.

Vivado 综合工具看到 posedge btn 后，可能会认为 btn 是一个时钟信号（因为 posedge 通常用于时钟）。

FPGA 的最佳实践是：避免用异步信号（如按键）直接作为 always @(posedge) 的触发条件，否则可能导致时序问题

4.timing loop

1. vivado仿真过程中不报错且突然中断问题。

在Regfile的程序中，我写了一段比较经典的combinational loop的代码，一个MUX的输入在兜兜转转经过五条线之后又直接与输出相连，于是在运行仿真的时候，仿真器就停在那个点不动，current time一直保持同一个值，console不报错，本应呈现xxx的值也保持原样，导致我们根本没有办法去debug，所幸程序在中断时会把断点以一个绿色光标形式留在程序中，我们因此可以根据这个来挨个排除并修改。