计算机组成原理实验

Lab 3 实验手册

汇编程序设计

Made by TA

2023 年 4 月 10 日

目录

在阅读实验手册之前，你需要了解的内容包括：

1. 实验手册为实验 PPT 讲解的额外补充，用于明确实验细节。
2. 实验手册的每一部分内容都有着对应的作用。当你遇到困难无法继续时，请确保你已经认真查阅了实验手册中的全部内容！如果你依然对实验内容有所疑问，欢迎你在群聊或私聊中提出你的问题，我们会在许可的范围内进行解答。
3. 请保证实验内容为自己独立完成。我们将对重复率过高的实验结果进行严肃处理。
4. 为了保证区分度，实验的部分内容难度较大。请量力而行，不要在超出自身能力范围外的部分投入过多的精力。

祝大家实验顺利！

1. 前言

在先前的课程中，大家或许已经接触到了基于 LC-3 架构的汇编语言。事实上，不同的架构都会有对应的汇编语言，RISC-V 也不例外。使用汇编语言编程可以更加自由地操纵硬件结构，获得更高的运行效率。相比高级语言，汇编语言可以帮助我们感知 CPU 的运行过程和原理，从而能够对 CPU 和应用程序之间的联系和交互形成一个清晰的认识。

一个功能完整的 CPU 含有众多的控制信号，这些信号用于控制 CPU 的各个部件，如寄存器、ALU、PC、内存等。不同的控制信号保证了 CPU 能够按照我们的预期工作。在仿真阶段，我们可以通过 Vivado 的波形图查看 CPU 内部的控制信号，但是在实际的硬件环境中，我们并没有办法直接查看这些信息。因此，我们需要将这些信息通过某种方式输出，并支持对其的实时修改，进而实现在硬件层面上对 CPU 运行的控制（如单步运行、断点调试等）。这就是外设与调试单元（Peripherals and Debug Unit，PDU）在实验设计中的作用。

在 Lab3 中，我们将通过简单的设计帮助大家熟悉汇编程序与 PDU 的使用规则，为后续的 CPU 实验做好准备。

2. 主要内容

本文档主要介绍的内容如下：

1. 一些关于外设的前置知识，如果你对于一些外设的使用已经非常了解（如边沿检测、分时复用等）可以跳过一部分，但是串口的使用相信对于大多数同学还是陌生的，建议大家仔细阅读。
2. RARS 软件的使用和部分汇编代码的使用。

你需要完成的内容概括如下：

> 详细的实验内容见文档结尾，此处给出大致内容以方便同学们带着目的去阅读文档

- 编写给定要求的汇编程序，实现斐波那契-卢卡斯数列的计算；
- 阅读理解助教给出的 PDU 程序，并学习其使用方法（文档中对外设的介绍对你完成这部分至关重要）；
- 扩展必做中的汇编代码使其支持外设和一些特殊的要求。

3. FPGAOL 与外设原理

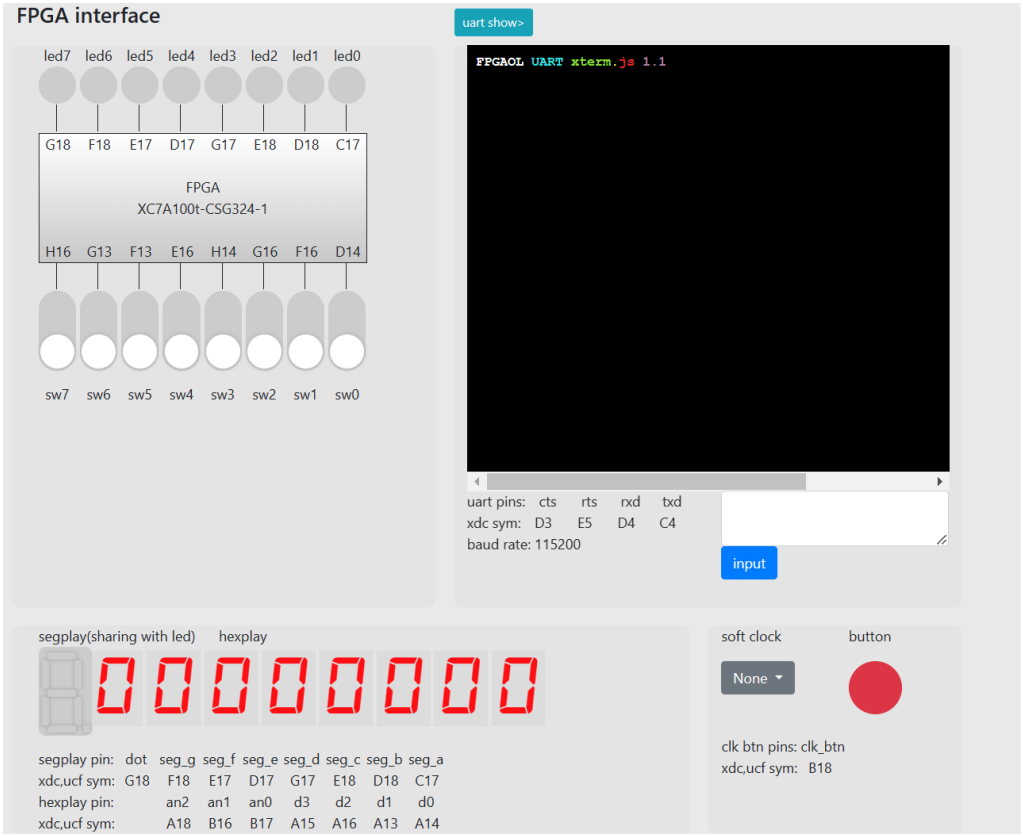
在这一章节中，我们将对于实验过程中可能用到的外设知识进行补充性讲解。你不必阅读所有的内容，根据自身需要进行查询即可。但我们推荐基础薄弱或没有学过数电的同学完整阅读下面的内容。

> 注：本部分内容整理自 FPGAOL 教程、数电实验讲义以及部分网络资料。

3.1 FPGAOL 平台简介

FPGAOL (https://fpgaol.ustc.edu.cn)是由中国科大计算机教学实验中心组织开发的、基于 Web 端的线上硬件实验平台。用户可以远程访问平台部署好的 FPGA（Nexys 4 DDR）集群，

上传本地生成好的比特流文件，并交互地控制 FPGA，实时获得 FPGA 的输出。需要注意的是，该结果是基于实际运行而非仿真产生的，所以可以确保其与线下操作 FPGA 开发板的结果相同。同时，由于线上设备具备出色的采样性能，平台能够精确发现人眼难以观察到的信号变化，从而为用户快速调试程序提供便利。



上图是 FPGAOL 平台所提供的交互平台示意图。FPGAOL 上拥有的输入输出设备包括：串口、七段数码管、LED、开关、按键等。通过这些 I/O 设备，我们可能涉及到的交互操作包括：

(1) 将数据输出到七段数码管上进行显示；

(2) 将数据输出到 LED 进行显示；

(3) 将开关的信息作为输入传输给 CPU；

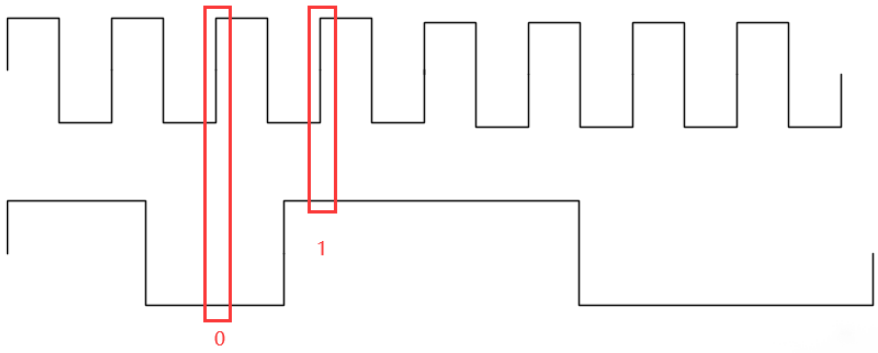
(4) 通过串口在用户与 FPGA 之间传递信息。（单独介绍）

3.2 边沿检测

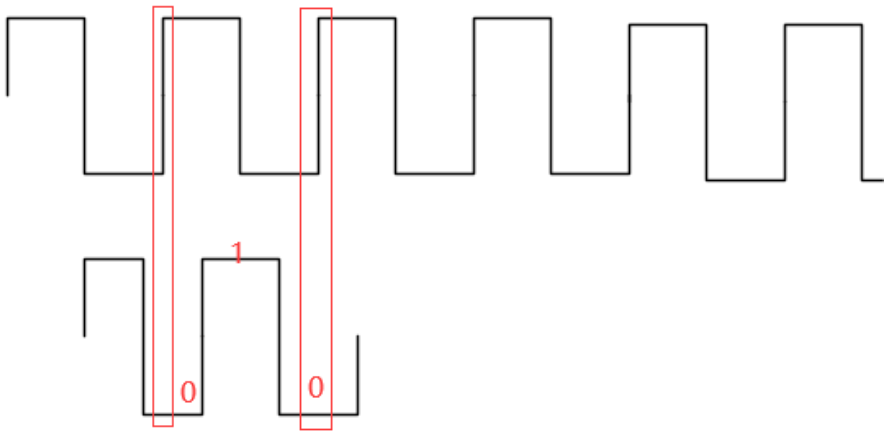
在 Verilog OJ 47 中，我们已经实现了基本的双边沿检测程序。下面，我们对边沿检测的原理进行简要的回顾。

边沿检测是对于输入信号的基本操作。它将输入信号的高低电平变化转化为一定宽度的脉冲信号，进而实现对后续电路的控制。根据信号变化的不同，边沿检测可以分为上升沿检测、下降沿检测以及双边沿检测。根据输出信号的特征，边沿检测可以分为同步检测与异步检测。

对于同步检测，我们通过比较输入信号在相邻两个时钟周期内的数值从而判断是否发生了电平变化。以下图为例，上方的是 clk 波形，下方的是输入信号波形。在某一个时钟上升沿时，输入信号为低电平（0）；在下一个时钟上升沿时，输入信号为高电平（1）。此时我们就检测到了输入信号的上升沿变化，进而输出一个时钟周期的脉冲信号。



然而，当输入信号的变化周期小于时钟周期时，我们就不能使用同步检测了。如下图所示，在相邻两个时钟周期的上升沿时，输入信号均为低电平，因此我们认为输入信号没有发生变化，但实际上输入信号在这个时钟周期内发生了两次电平翻转。此时我们就会使用到异步检测。



异步检测的原理是直接将输入信号作为敏感变量加入到 always 基本块的头部。**这种做法在大部分情况下都是极为不推荐的！**在 FPGA 综合时，我们需要避免非时钟信号、复位信号等出现在敏感变量列表中。所幸的是，在对外设信号进行边沿检测时，小于一个时钟周期的电平变化是极为罕见的，因此同步检测就足以支撑所有的应用场景了。

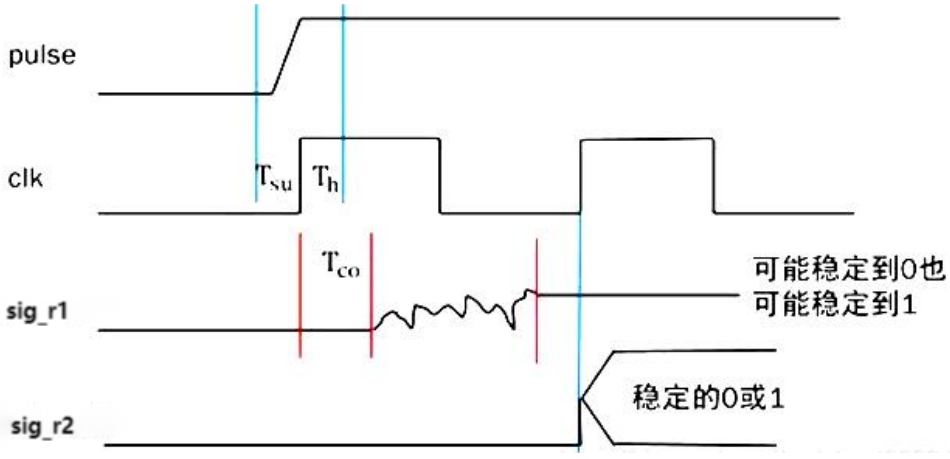
要实现同步的边沿检测，最直接的想法是两级寄存器法：用第二级寄存器锁存住某个时钟上升沿到来时的输入电平，第一级寄存器锁存住下一个时钟沿到来时的输入电平。如果这两个寄存器锁存住的电平信号不同，就说明检测到了边沿，具体是上升沿还是下降沿可以通过组合逻辑来实现。一个典型的 Verilog 程序如下所示：

```
1   module edge_capture(
2       input clk,
3       input rstn,
4
5       input sig_in,  // Signal input
6       output pos_edge,
7       output neg_edge
8   );
9
10  reg sig_r1, sig_r2;
11
12  always @(posedge clk or negedge rstn) begin
13      if (!rstn) begin
14          sig_r1 <= 0;
15          sig_r2 <= 0;
16      end
17      else begin
```

```
18              sig_r1 <= sig_in;
19              sig_r2 <= sig_r1;
20          end
21      end
22
23      assign pos_edge = (sig_r1 && ~sig_r2) ? 1 : 0;
24      assign neg_edge = (~sig_r1 && sig_r2) ? 1 : 0;
25      endmodule
26
```

上面讨论过程都是建立在理想情况下的。在实际的电路情况中（例如 FPGA 开发板上），时序电路的信号均存在建立与保持时间。如果输入信号的变化恰好出现在 clk 的变化边沿中，则第一级寄存器采集到的信号可能并不是明确的高低电平，而是一种中间的结果。此时，第一级寄存器的输出会进入到亚稳态，进而传递给后续的寄存器，影响整个电路的工作情况。

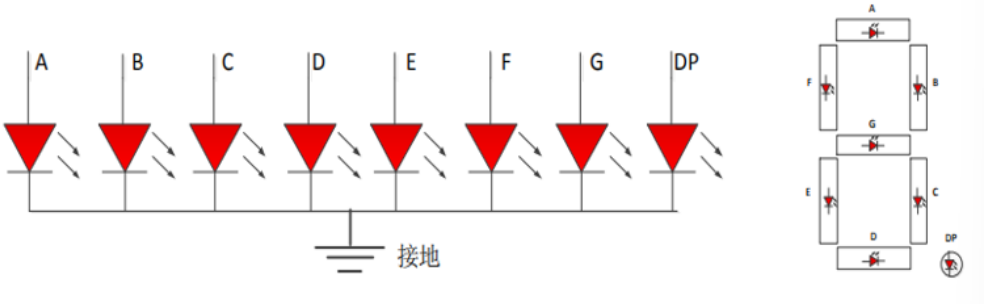
如下图所示，图中 Tco 为第一级寄存器 sig_r1的状态建立时间（即 clock to output），一般情况下，亚稳态的决断时间（即从进入亚稳态到稳定下来的时间）不会超过一个时钟周期，因此在下一个 clk 上升沿到来之前，sig_r1 已经稳定下来（可能稳定到 0 也可能稳定到 1），这样第二级寄存器就会采集到一个稳定的状态，从而把亚稳态限制在第二级寄存器之前，保证了整个电路输出的稳定性。



所以，为了保证系统的稳定性，我们建议大家在实现边沿检测时，均使用三级寄存器的方法。在对系统稳定性要求较高的数字系统中，可以采用更多级的寄存器来减小亚稳态发生概率，提高系统稳定性。

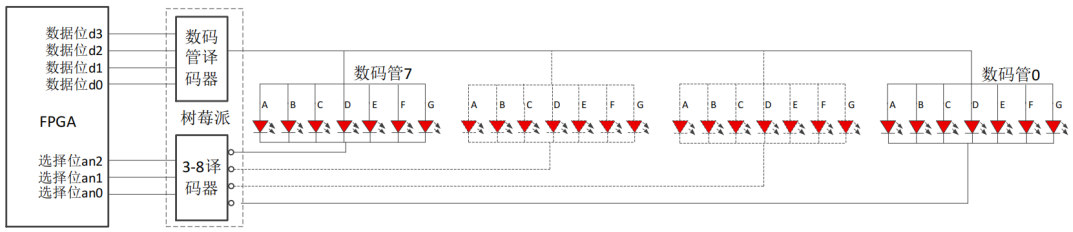
3.3 七段数码管、分时复用与时钟分频

七段数码管本质上是由 8 个 LED（发光二极管）构成，其中 7 个 LED 组成数字本身，1 个 LED 组成小数点。所有 LED 的阴极共同连接到一端并接地，而阳极分别由 FPGA 的 8 个输出管脚控制。当输出管脚为高电平时，对应的 LED 亮起。如下图所示，通过控制 8 个 LED 的亮灭情况，七段数码管便能显示出不同的字符，例如，当 A~F 的 6 个 LED 亮起，而 G、DP 两个 LED 熄灭时，数码管显示的便是字符“0”。



在有多个数码管的情况下，我们通常采用分时复用的方式轮流点亮每个数码管，并保证在同一时间只会有一个数码管被点亮。对于当前点亮的数码管，我们会只传输其应当显示的内容。分时复用的扫描显示利用了人眼的视觉暂留特性，如果公共端的控制信号刷新速度足够快，人眼就不会区分出 LED 的闪烁，从而认为这些数码管是同时点亮的。一般而言，我们建议数码管的扫描频率为 50Hz，也就是说，如果要驱动 8 个数码管，需要一个 400Hz 的时钟。

由于实验平台上的管脚数量有限，我们对数码管的显示方式进行了一定的简化：在使能方面，仅使能由 AN[2 : 0] 所表示的二进制数对应的数码管；在显示的数字方面，直接显示 D[3 : 0] 对应的 16 进制数。例如，若 AN = 3’b010，D = 4’b1010，则数码管在下标为 2 的数码管上显示字符“A”。



FPGA 开发板的 E3 管脚连接了一个 100MHz 频率的时钟晶振，可用作时序逻辑电路的时钟信号。如果我们需要一个其它频率的时钟信号应该怎么办呢？一般的做法是通过设置

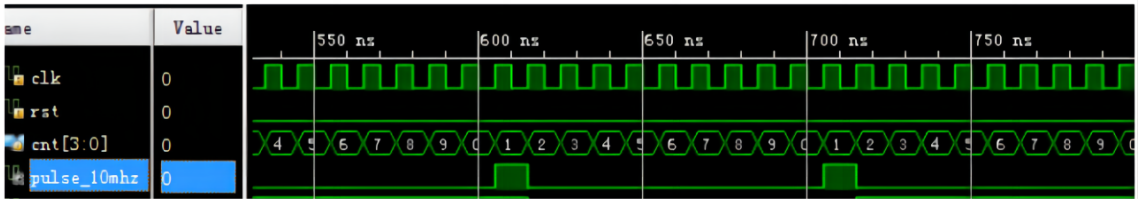
计数器产生一个低频的脉冲信号，然后再利用该脉冲信号控制其他逻辑的控制信号。

下面的代码通过 pulse_10mhz 信号实现了一个 10MHz 的时钟。

```
module pulse_10mhz(
    input clk, rst,
    output reg led
);
    reg [3:0] cnt;
    wire pulse_10mhz;
    always (@posedge clk) begin
        if(rst)
            cnt <= 4'b0;
        else if(cnt >= 9)
            cnt <= 4'b0;
        else
            cnt <= cnt + 4'b1;
    end
    assign pulse_10mhz = (cnt == 4'h1)
endmodule

```

对应的波形图如下：



3.4 串口与通信编码

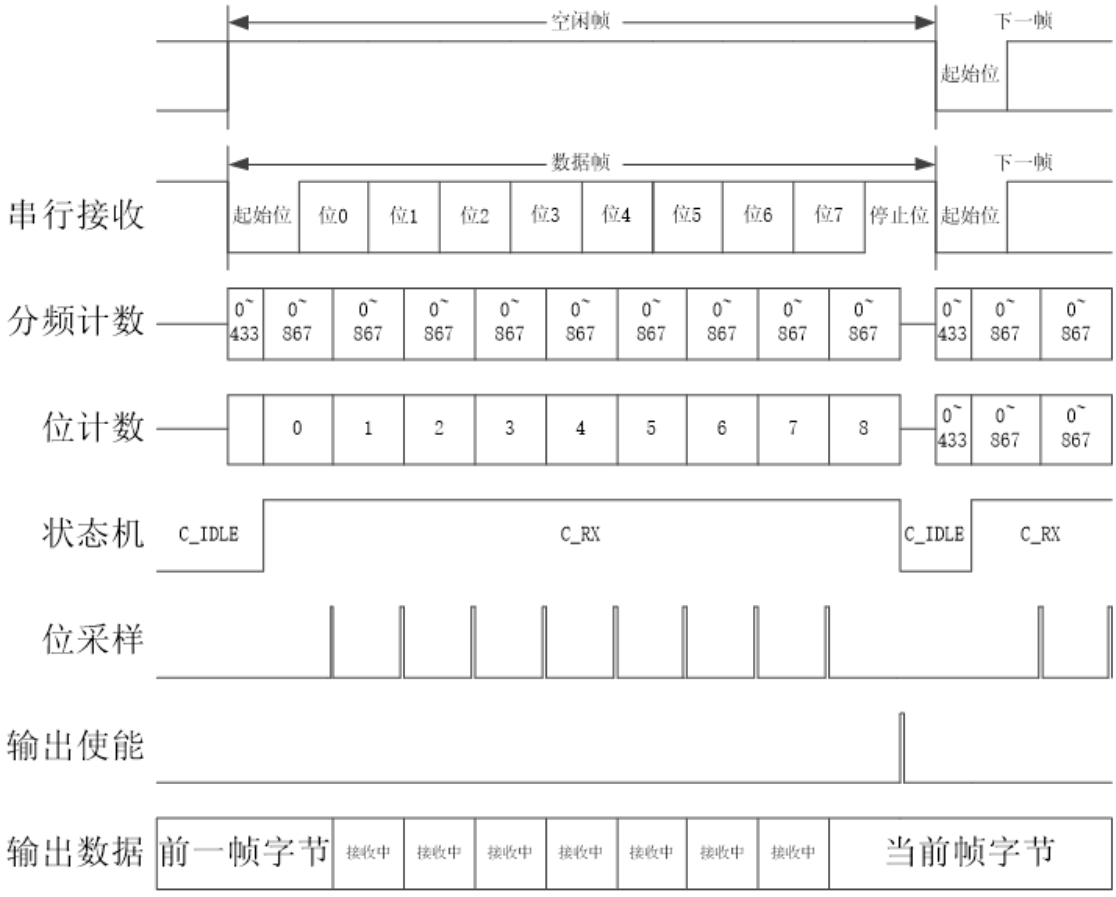
从广义上来说，采用串行接口进行数据通信的接口都可以称为串口，如 SPI 接口、IIC 接口等，但我们所说的串口一般是指通用异步收发器（Universal Asynchronous Receiver/-Transmitter），简称 UART，主要包含 RX、TX、GND 三个接口信号。其中 GND 为共地信号， TX、RX 负责数据的发送和接收。在嵌入式系统开发中，串口是一种必备的通信接口，在系统开发测试阶段和实际工作阶段都起着非常重要的作用。在我们使用的 Nexys4DDR 开发板中， UART 通信与 USB 烧写功能集成在了一个 microUSB 接口中。

说到这里，你可能对串口的工作原理依然十分困惑。下面是一个简单的串口回显程序，

你可以将其烧写成 bit 流并在 FPGAOL 平台上实际运行。在串口窗口中输入的内容会被原样输出到串口窗口中。

```
1    module uart_test (
2        input uart_tx,       // Transmitter: User >>>> FPGA
3        output uart_rx       // Receiver: FPGA >>>> User
4    );
5        assign uart_rx = uart_tx;
6    endmodule
7
```

在实际的串口通信中，我们会分别设计接收模块与发送模块（针对 FPGA 而言）。当用户需要向 FPGA 发送数据时，只需要将其交付给接收模块的输入；当用户需要 FPGA 向自己传输数据时，只需要将数据交付至发送模块的输入。



上面是串口通信的接收模块工作原理图。通俗来说，接收模块需要将串行发送的单 bit 数据电平信号转换为原始数据。什么意思呢？例如串行数据序列为“00011010”，即输入信号为低电平-低电平-低电平-高电平-高电平-低电平-高电平-低电平，接收模块需要根据输入

信号的电平变化，结合串口通信协议将其恢复成数据 00011010。

我们为大家提供了已经编写完成并经过测试的 Receive.v 文件，用于实现数据的传输。具体的模块接口以及使用说明请参见相应文件的注释。

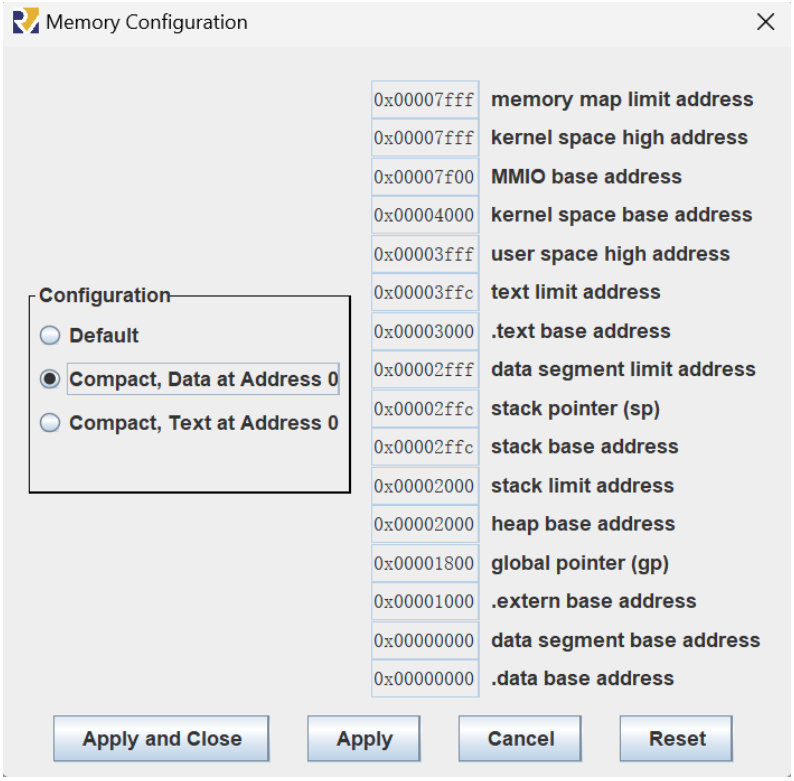
4. RARS 软件与汇编程序(选做相关)

RARS (github 链接：https://github.com/TheThirdOne/rars) 是基于 RISC-V 架构的汇编综合实验平台。RARS 程序提供了汇编器、仿真器（参考 ICS LabA&S）等功能，同时也包含了方便的外设接口与信息查询接口，被广泛用于 RISC-V 汇编程序的编写与测试之中。

RISC-V 汇编程序涉及到的相关知识点请参考课本与指令集手册。如果你以前未接触过，或不熟悉汇编语言编程的话，可以参考这里：https://riscv-programming.org/book/riscv-book.html。本次实验中，你可能会关注下面的内容：4-Assembly language、6-The RV32I ISA 以及 9.4-Busy waiting。

下面，我们将对本次实验中可能用到的相关内容进行介绍。

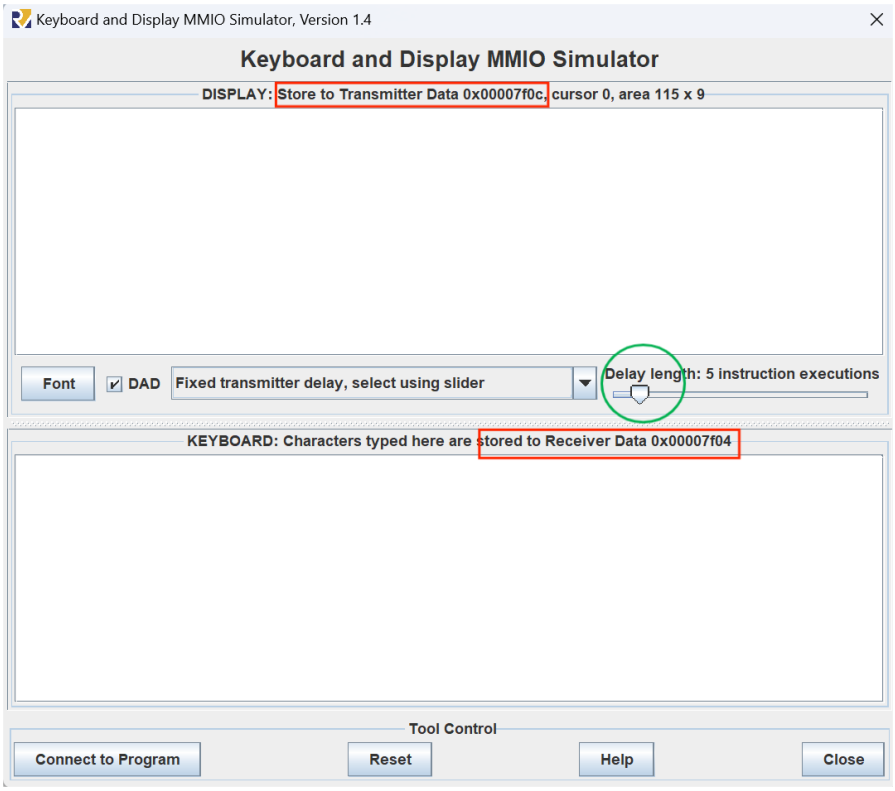
4.1 RARS 内存设置



在主界面上点击 Settings » Memory Configuration 即可打开上图所示的子界面。在 RARS 中，内存空间被划分为许多部分。按照功能可以概括为：程序段、数据段、栈空间与堆空间、内核空间以及 MMIO 保留地址。根据设置的不同，各个部分的相对位置与大小也有所不同。**本实验以及后续实验所采用的均为紧凑型（Compact, Data at Address 0）内存分布，**保证数据段的范围为 0x0000~0x2ffc，程序段的范围为 0x3000~0x3ffc。

4.2 RARS 键盘输入与输出

RARS 提供了用户与汇编程序之间进行 IO 的接口。在主界面上点击 Tools » Keyboard and Display MMIO Simulator 即可打开下图所示的子界面。这个界面中包含了键盘输入以及显示器输出两部分。



上方为显示模块，负责接收汇编程序传输的数据，并显示在上面的矩形区域之中。我们约定，内存单元 0x7f0c 代表输出数据寄存器，输出的数据以二进制 ASCII 编码的形式保存。例如，当汇编程序在 0x7f0c 地址写入数据 0x30 时，显示器会显示一个字符“0”。

下方为输入模块，负责接收用户输入的数据，并交付给汇编程序。我们约定，内存单元 0x7f00 为输入状态寄存器，0x7f04 为输入数据寄存器。当键盘输入数据时，输入状态寄存器会被置 1，输入数据寄存器会被设置为当前字符的二进制 ASCII 编码。当程序读取输入数

据寄存器的内容后，输入状态寄存器会被自动清零。

在用户输入数据时，每输入一个字符，输入数据寄存器就会被更新一次。由于数据寄存器只能存储一个字符的 ASCII 编码，因此在下一次输入到来前，程序需要读取数据寄存器中的内容，否则就会被下一次的用户输入覆盖。

注意图中绿圈标出的地方：这里代表着输出模块显示的延迟时间，即从写入数据输出寄存器到显示在屏幕上的时间间隔（指令条数）。当延迟较大时，程序写入输出数据寄存器的频率也应相应降低。**本实验中，请将该数值设定为 1，从而避免可能的时序问题。**

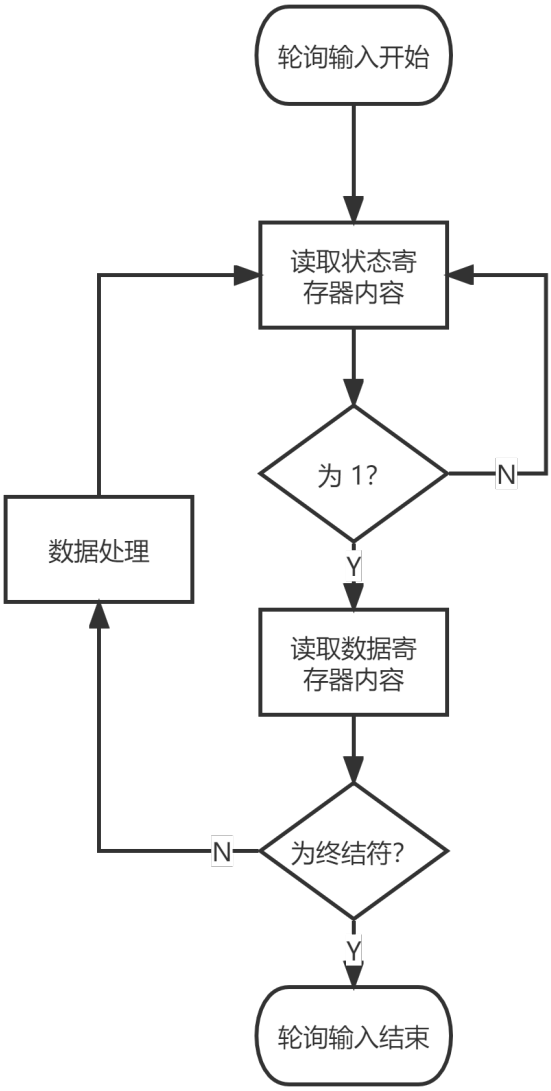
在程序运行之前，我们需要点击界面下方的 Connect to Program 按钮，建立汇编程序与 MMIO Simulator 之间的连接。每一次运行汇编程序时（单击 Run » Assemble），我们都需要按下 Reset 按钮，以重置三个寄存器中的状态、数据信息。

更多的使用说明可以点击界面下方 Help 按钮进行查询。

4.3 轮询与中断

程序如何知道用户此时正在输入呢？根据前面的介绍，我们需要读取输入状态寄存器中的数值，若为 0 则代表当前没有数据输入，若为 1 则代表当前有数据输入，进而需要读取输入数据寄存器中的内容。那在没有输入时，程序应该做些什么呢？我们可以让程序一直读取状态寄存器中的数值，直到读取到 1，代表用户输入了内容。这样的输入实现方式即为轮询输入。

下面的流程图展示了轮询输入的过程。



与轮询相对应的方式是中断。此时程序不会反复检查输入状态寄存器的内容，而是由硬件（CPU）代为监测。当监测到状态寄存器为 1 时，CPU 会终止当前程序的执行，转而执行特定的中断处理程序读取输入的数据，并存入其内部特定的寄存器。程序可以通过读取特定寄存器中的数据得到来自外界的输入。

4.4 I/O 的数据处理

注意到 RARS 中输入、输出时均以 ASCII 编码的形式进行，因此在进行 I/O 时，我们需要在汇编程序中进行数据的处理操作。

在数据输入时，假定我们在 Keyboard and Display MMIO Simulator 上输入了“25”，在

汇编程序看来，从输入数据寄存器中两次读取到内容分别为 0x32、0x35。我们需要将其解码成数据 2、5，再将其拼接成 25。注意：输入数据的长度是未知的，所以还需要额外约定一个终结符，用于指示数据输入的结束。你可以假定输入的内容只包含 16 进制数码与终结符。

在数据输出时，假定我们在输出数据寄存器中写入了 0x30，则 Keyboard and Display MIMO Simulator 会在屏幕上显示字符“0”。因此，我们需要将寄存器中的 32bit 数据逐一进行 ASCII 编码，再依次转移到输出数据寄存器上进行输出。

为了保证输出结果的可读性，我们会在适当的时候输出换行。在 Windows 环境下，换行符被视作 "$\backslash n\backslash r$"，其中 '$\backslash n$' 的 ASCII 编码为 0x0A，'$\backslash r$' 的 ASCII 编码为 0x0D。

5. 实验任务

本次实验所需完成的各项工作介绍如下：

> 请不要使用打表的方式完成斐波那契-卢卡斯数列相关的内容。

【必做部分】

- **设计汇编程序：计算斐波那契-卢卡斯数列的前 n 项，并生成 COE 文件。** 汇编程序的相关要求请参考 PPT。数列的前两项为 1、1，初始时存储在 0x0000 开始的连续地址处。请只使用 ppt 上给出的 10 条指令以及基于它们的伪指令实现。你可以在汇编程序中采用如下的方式设置相关的数值（仅供参考）：

```
    .data
        1   # a1 in 0x0000
        1   # a2 in 0x0004
    .text
        addi t0 x0 5    # Store n=5 in reg t0

```

  COE 文件是接下来 CPU 实验所需要使用的程序文件，请将其下载并保存好，以备后续实验的使用。

- **设计 32bits 移位寄存器，通过 PDU 实现对移位寄存器数据的实时操作。** PDU 模块会

完全给出，请根据演示视频与相关模块的接口定义，完成 Shift_reg 模块的逻辑设计，并将其下载到 FPGAOL 平台上进行功能测试。

【选做部分】

你可以任选若干项选做内容并完成。

- **增加外设交互，实现项数 n 的键盘输入。** 本项选做在必做内容的基础上进行。数据的前两项依然保持为 1、1，初始时存储在 0x0000 开始的连续地址处。你也可以将前两项设计为通过外设进行输入，但这样不会有额外的分数。 n 的输入建议采用十进制格式，这将有助于程序的可读性。请使用轮询输入的方式进行程序设计，并根据实验手册处理一系列可能遇到的问题。

- **增加外设交互，实现前 n 项的显示器输出。** 本项选做需要在完成键盘输入的基础上进行。输出的进制不做要求，但建议采用十六进制，这将有助于程序的可读性。请确保一行仅输出一个 32bits 整数，这需要你在适当的时刻输出换行符。建议大家采用 jal-ret 过程调用的方式进行程序设计，这将有效降低程序编写时的复杂度。

- **增加数据处理，消除输出数据的前导零。** 本项选做需要在完成显示器输出的基础上进行。你需要消除输出数据的前导零。例如：假定某一时刻运算结果为 0x00213d05，那么你需要输出的结果应当为 0x213d05。

- **设计汇编程序，支持对于大整数的运算与存储。** 本项选做在必做内容的基础上进行。当 $3 \leq n \leq 80$ 时，斐波那契-卢卡斯数列的结果有可能超出 32bits 整数的范围，为此我们需要重新设计汇编程序，支持 64bits 整数的运算与存储。请注意，你只能使用 RISC-V 32I 中的指令完成这一部分内容。对于项数 n 的输入、前 n 项数据的输出方式我们不做要求。正确性参考：$f_{80} = (533163ef0321e5)_H$

本次实验需要大家在实验平台上在线提交相关内容。你提交的文件结构应当满足下面的文件树格式：

```
/
└── lab3_姓名_学号_ver尝试编号
    ├── figs ..... 图片文件夹
    ├── Lab3_姓名_学号.pdf ..... 实验报告文件
    └── src ..... 需要提交的相关程序文件夹
        ├── Module_name.v ..... 非仿真 .v 文件
        └── Program_name.asm ..... 汇编源程序文件
```

```
├── ...
└── others.....................其他你打算提交的文件，如果没有可以无此文件夹
```

请将全部文件按照上面的格式压缩成一个文件，提交到实验平台上。

请确保你的实验报告至少包含以下的内容：

- 用于计算斐波那契-卢卡斯数列的核心代码介绍；
- 用于实现外设输入、输出的核心代码介绍（如果做了此选做）；
- 用于处理大整数运算与存储的核心代码介绍（如果做了此选做）。

我们也欢迎大家在实验报告中给出对于本次实验的反馈。

> 实验检查与报告提交的 DDL 按照各班各组的约定设置。超出 DDL 的检查与提交将按照规定扣除部分分数。请保证个人实验的独立完成！