计算机组成原理第5次实验报告

实验名称：存储器和单周期CPU实现

学号：2313546 姓名：蒋枘言 班次：1078

**一、实验目的**

1. 了解只读存储器 ROM 和随机存取存储器 RAM 的原理。

2. 理解 ROM 读取数据及 RAM 读取、写入数据的过程。

3. 理解计算机中存储器地址编址和数据索引方法。

4. 理解同步 RAM 和异步 RAM 的区别。

5. 掌握调用 xilinx 库 IP 实例化 RAM 的设计方法。

6. 熟悉并运用 verilog 语言进行电路设计。

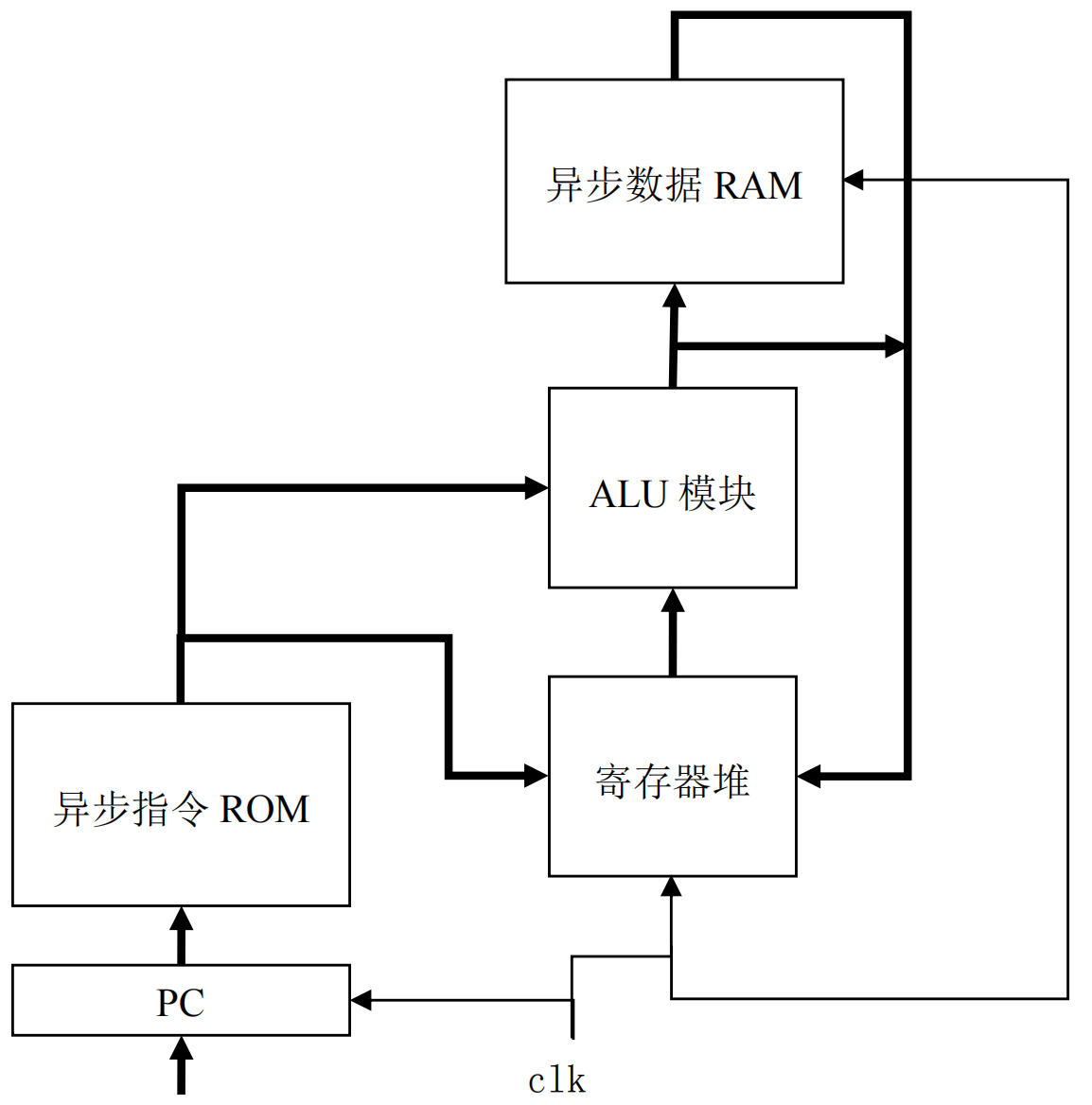
7. 为后续设计 CPU 的实验打下基础。

**二、实验内容说明**

请根据实验指导手册完成ROM存储器实验和单周期CPU实验，并撰写实验总结，要求：

1. ROM存储器实验请完成验证，总结收获，并对比跟之前的同步异步RAM实验有何不同，分析总结原因。
2. 单周期CPU实验，原理图应基于实验指导手册中的图7.1，在分析表7.4中指令执行过程时，可以在图7.1基础上辅助画线表示执行过程。
3. R型指令和I型指令挑两条分析总结执行过程，J型指令就1条，请直接分析总结执行过程。注意，这些指令已经在inst\_rom.v里面写好，所以请找到对应的指令，逐个分析。从指令的二进制编码开始，分析介绍代码是如何一步一步完成运算并执行的。
4. （提高要求，不强求做出来）把ALU实验中添加的三个指令，自行添加到这个单周期CPU中，注意指令码只要跟现有的不冲突就行，不必限制在标准的MIPS指令格式。

**三、实验原理图（单周期CPU简单原理图）**

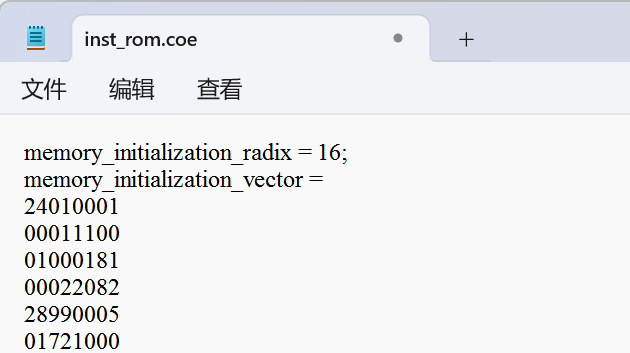


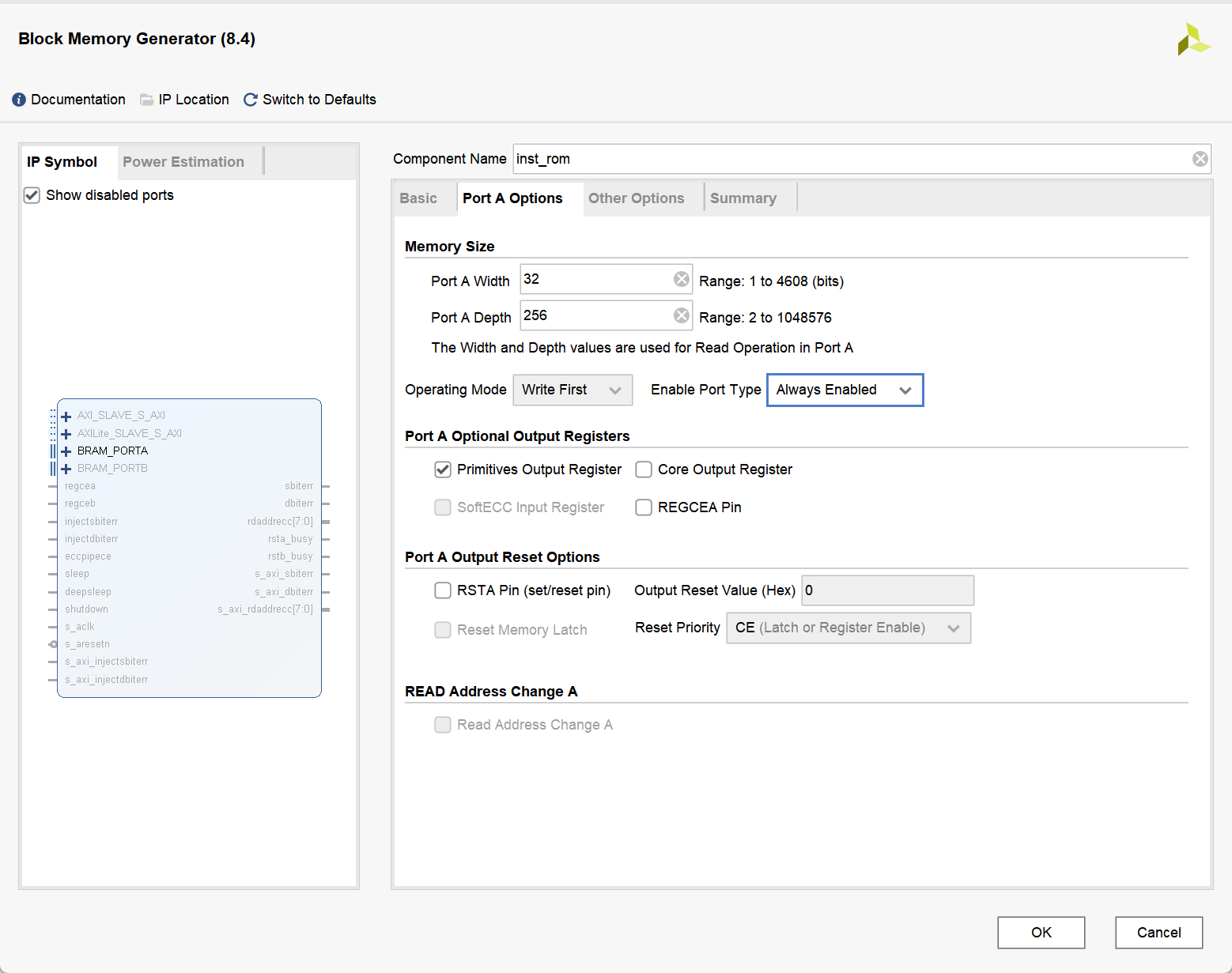
**四、实验步骤**

**（一）ROM存储器实现**

打开Vivado，对同步ROM、异步ROM分别新建一个工程。按要求添加文件，并且调用Xilinx库IP生成同步ROM，对它们进行上箱验证。

对于同步ROM，需要创建一个IP盒，在此之前还需要创建一个.coe文件，如下图所示：



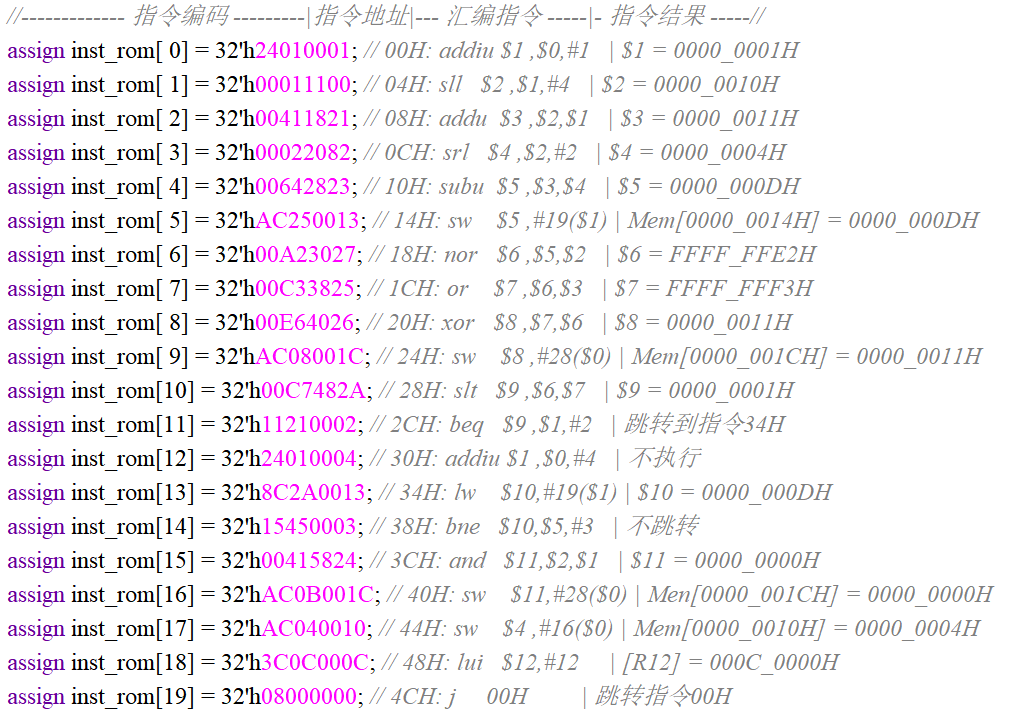


**（二）单周期CPU实现**

打开Vivado，新建名称为single\_cycle\_cpu的工程，导入以下文件，进行上箱验证：



包含的指令如下：



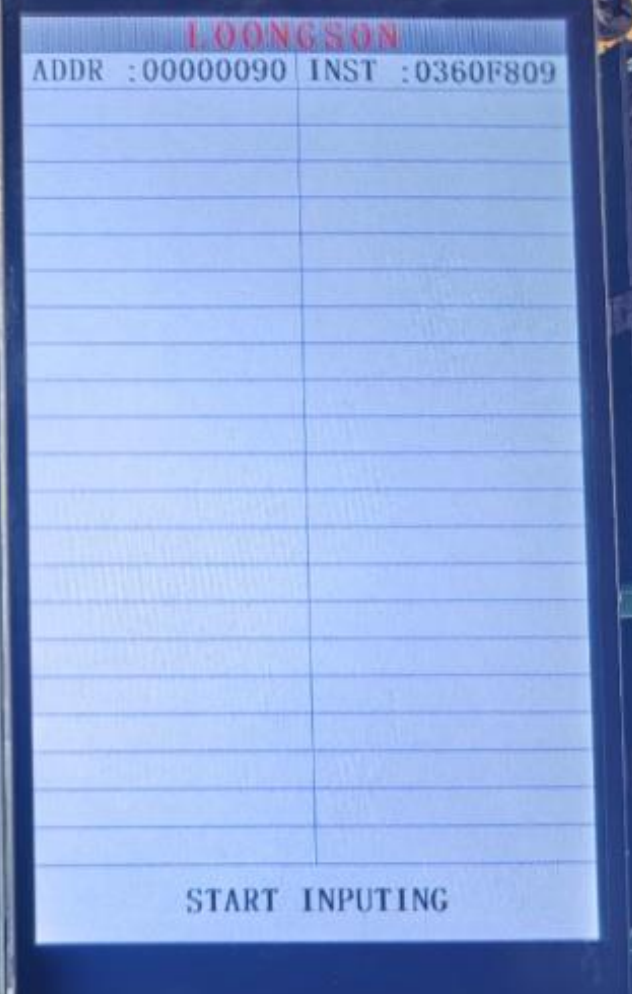
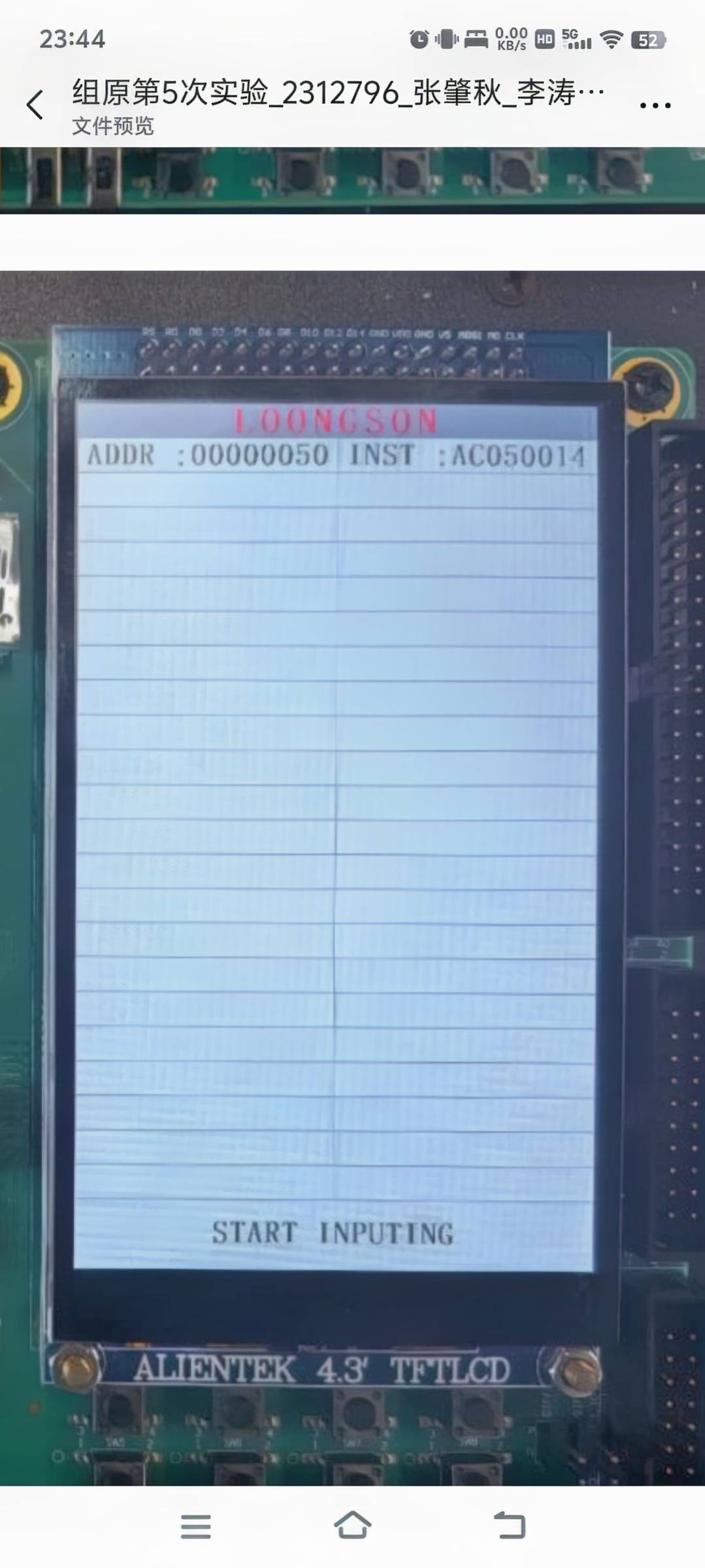
**五、实验结果分析**

**（一）ROM存储器实现**

同步ROM的结果如下，发现均能正确读取到数据。



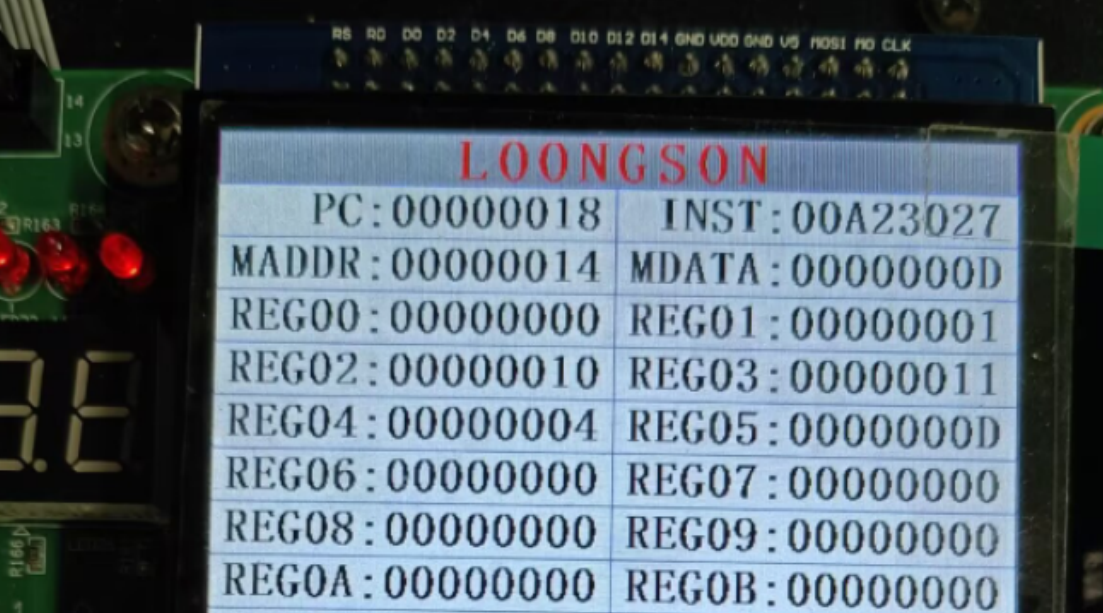
异步ROM的结果如下，发现同样能正确读取到数据。



**（二）单周期CPU实现**

每次执行 sw 指令后记录一次当前各个寄存器的值。

sw $5,#19($1)指令结束后，寄存器显示如下：



在这条指令结束后，其前面的代码段完成了对寄存器 $1~$5 的赋值，并将 $5 寄存器的值存在了内存：Mem[0000\_0014H]中。

addiu $1,$0,#1指令执行 $1=$0+1 ，因此寄存器 $1 的值应变为00000001。

sll $2,$1,#4指令将 $1 左移四位后再赋值给 $2 ，因此 $2 值应为00000010。

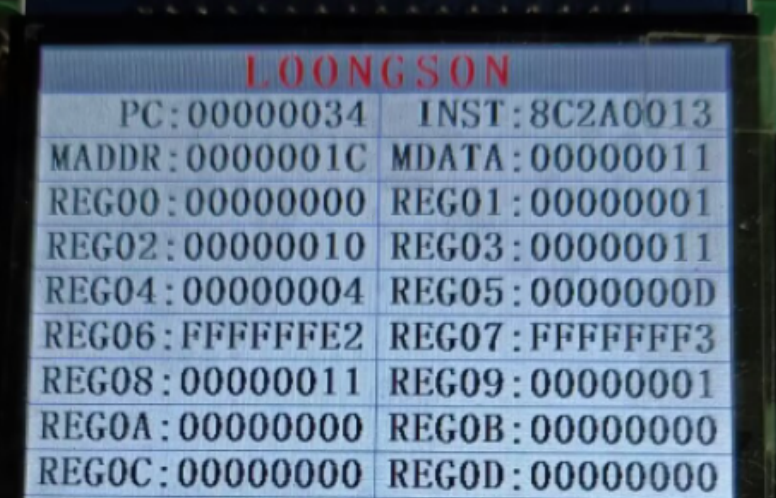
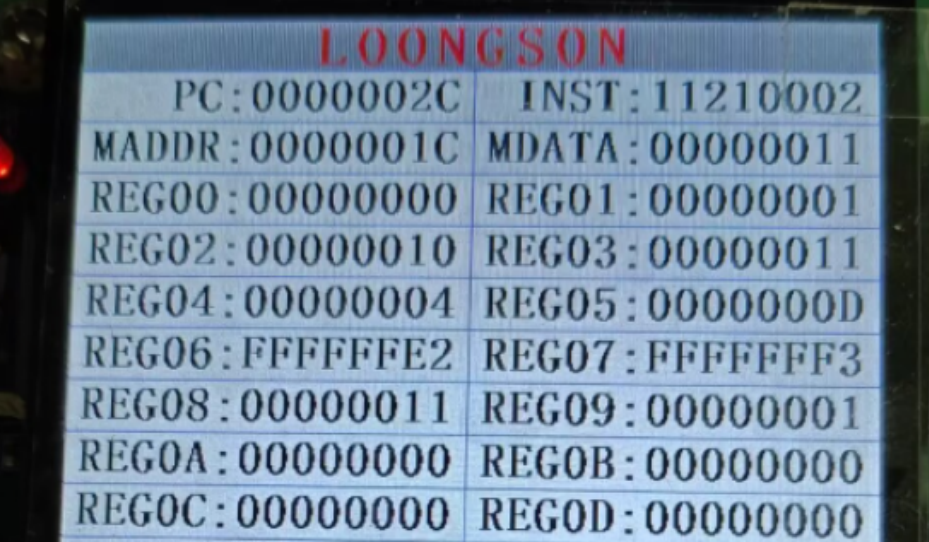
addu $3,$2,$1指令执行 $3=$2+$1 ,因此 $3 的值应为00000011。

srl $4,$2,#2指令把 $2 右移2位后传给 $4 ，因此 $4 的值应为00000004。

subu $5,$3,$4指令执行 $5=$3-$4 ，因此 $5的值应为0000000D。

sw $5,#19($1)指令把 $5 的值存入内存： Mem[0000\_0014H] ，为验证该结果，我们把 MADDR 输入14，读得 MDATA 为0000000D，写入内存成功。

之后的过程均正确，此处省略一条一条指令的讲解了。



**六、总结感想**

**（一）ROM与RAM对比**

**1. 数据持久性**

ROM：数据永久存储，无需电源维持。

RAM：数据仅在通电时存在。

**2. 写入方式**

ROM：出厂时写入，写入后不可修改，只可读。

RAM：运行时直接通过CPU或外设读写。

**3. 访问方式**

ROM：按地址顺序读取，随机访问较慢。

RAM：支持高速随机读写。

**（二）指令执行过程**

**1. sll（Shift Left Logical，逻辑左移） 以sll $2, $1, #4为例**

指令类型：R型指令

**二进制编码**

32位二进制值为：

000000 00000 00001 00010 00100 000000

分解为R型指令字段：

opcode: 000000（R型指令）

rs: 00000（未使用）

rt: 00001（源寄存器$1）

rd: 00010（目标寄存器$2）

shamt: 00100（十进制4，左移4位）

funct: 000000（对应sll）

**执行过程**

取指（IF）：从地址0x04（inst\_rom[1]）读取指令。

译码（ID）：识别为R型指令（opcode=0），功能码funct=0对应sll。

读取$1的值（假设$1 = 0x00000002）。

执行（EX）：将$1的值左移4位，0x00000002 << 4 = 0x00000020。

写回（WB）：结果0x00000020写入$2寄存器。

**2. lw（Load Word，加载字） 以lw $10, #19($1)为例**

指令类型：I型指令

**二进制编码**

32位二进制值为：

100011 00001 01010 0000000000010011

分解为I型指令字段：

opcode: 100011（对应lw）

rs: 00001（基址寄存器$1）

rt: 01010（目标寄存器$10）

immediate: 0000000000010011（十进制19，符号扩展为32位）

**执行过程**

取指（IF）：从地址0x34（inst\_rom[13]）读取指令。

译码（ID)：

识别为lw指令（opcode=0x23）。

读取基址寄存器$1的值（假设$1 = 0x00000000）。

地址计算（EX）：

address = $1 + 19 = 0x00000000 + 0x13 = 0x00000013。

访存（MEM）：从内存地址0x13读取32位数据（假设内存中0x13存储值为0x0000000D）。

写回（WB）：将0x0000000D写入$10寄存器。

**3. j（Jump，无条件跳转） 以j 00H为例**

指令类型：J型指令

**二进制编码**

32位二进制值为：

000010 00000000000000000000000000

分解为J型指令字段：

opcode: 000010（对应j）

target: 00000000000000000000000000（目标地址高26位）

**执行过程**

取指（IF）：从地址0x4C（inst\_rom[19]）读取指令。

译码（ID）：识别为j指令（opcode=0x02）。

目标地址计算（EX）：

当前指令地址为0x4C，PC+4 = 0x50（下一条指令地址）。

取PC+4的高4位（0x50的二进制高4位为0000）。

拼接目标地址：0000 || (target << 2) = 0x00000000。

更新PC：将PC设置为0x00，程序跳转到地址0x00执行。