计算机组成原理
课程设计报告

专业班级：

学生姓名：

指导老师：墙威

中国地质大学计算机学院

2022 年 6 月

目 录

设计一 单周期 MIPS CPU 的设计

1 实验目的

- 掌握硬布线控制器设计的基本原理

- 在 Logisim 平台中设计实现 MIPS 单周期 CPU

2 主要任务

- 绘制单周期 MIPS CPU 数据通路

- 实现单周期硬布线控制器

- 测试联调

3 实验过程

（1）构建单周期 MIPS 主机通路

- PC：32 位寄存器

- 数据存储器：地址 10 位（字地址），数据 32 位

- 寄存器文件：可从下面两种中选择其一

封装好的“MIPS Regifile”电路

CS3410 Components 中的“Register File”（推荐这个，可以直接看到寄存器的值）

- 运算器 ALU：封装好的“MIPS ALU”电路

- 指令存储器：

需自行添加一个 ROM

数据位宽为 32 bits（因为 MIPS 指令字是 32 位）

地址位宽合理设置，使其能够装载完整的排序程序

- 单周期硬布线控制器

将其输出控制信号与对应组件相连

需要设计其内部结构（见步骤 2）

（2）设计单周期 MIPS 控制器

- 输入信号：指令字中的 Opcode，Func 字段

- 输出信号：各种控制信号

- 设计组合逻辑电路，得到以下各种信号输出：

指令类型（详见 MIPS 指令集）：

利用比较器，通过比较 Opcode 的值，判断指令是否为某种 I 型指令：LW、SW、BEQ、 BNE 或 ADDI（指令译码逻辑）；

利用比较器和简单逻辑门电路，通过比较 Func 的值，判断指令是否为某种 R 型指令： ADD、SLT 或 SysCall；

利用简单逻辑门电路，判断指令是否为运算型 R 指令：R_TYPE

- ALU_OP 的值：

根据不同的指令（OP 和 FUNC 字段的值）确定执行该指令时 ALU 需要进行何种运算

- 控制信号：

利用简单逻辑门电路，得到 RegDst、RegWrite、MemToReg、MemWrite、AluSrc、Beq、 Bne、Halt（详见慕课：控制信号功能说明表）

（3）CPU 测试实验资料中的“8 条指令单独测试用例 2019-12-7.zip”给出了测试程序用到的 8 条指令的单 独测试用例；

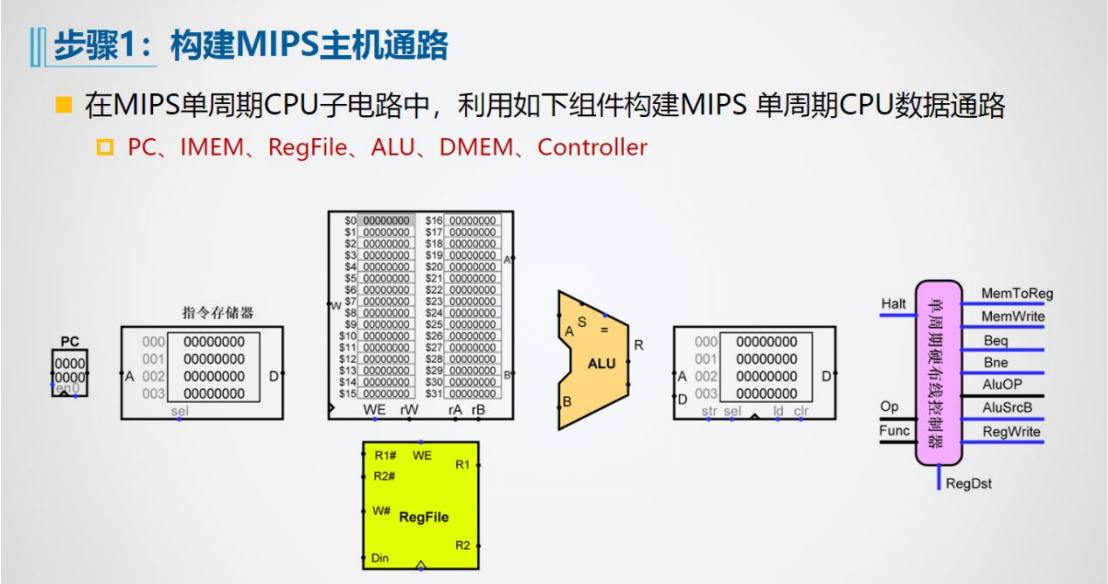
.asm：测试用例程序的说明及对应的汇编语句；

.hex：测试用例程序的 MIPS 指令序列，16 进制表示，需要装载入指令存储器；

最后一个 sort 是完整的排序程序；

4 电路图及相关说明

（1）构建 MIPS 主机通路

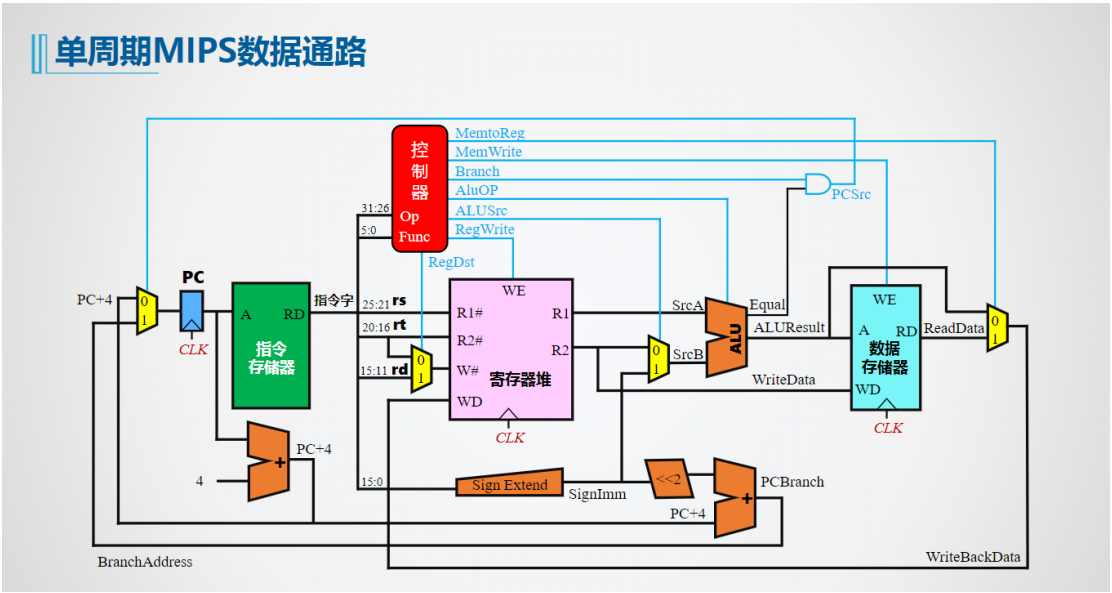


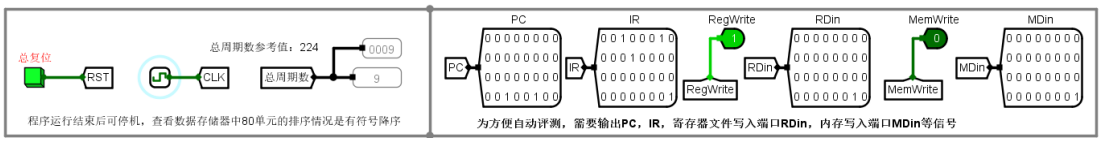
（2）MIPS CPU 指令格式

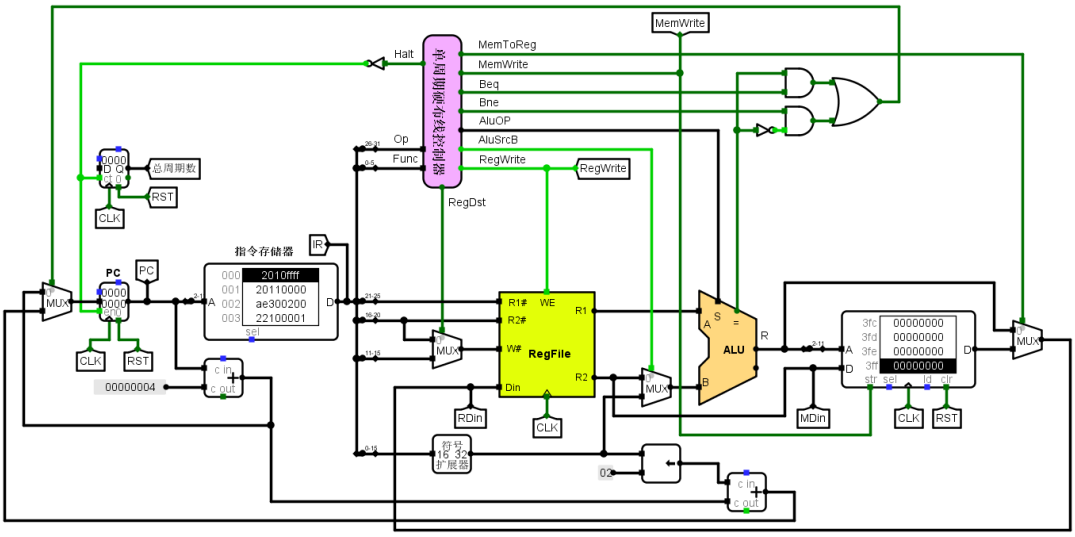
核心指令集 （可实现内存区域冒泡排序）

#	MIPS指令	RTL功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt] 溢出时产生异常，且不修改R[$rd]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt] 小于置1，有符号比较
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm) 溢出产生异常
4	lw $rt,imm($rs)	R[$rt]←$Mem_{4B}$(R[$rs]+$SignExt_{16b}$(imm))
5	sw $rt,imm($rs)	Mem_{4B}(R[$rs]+$SignExt_{16b}$(imm))←R[$rt]
6	beq $rs,$rt,imm	if(R[$rs] = R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
8	syscall	系统调用，这里用于停机

（3）单周期 MIPS 数据通路





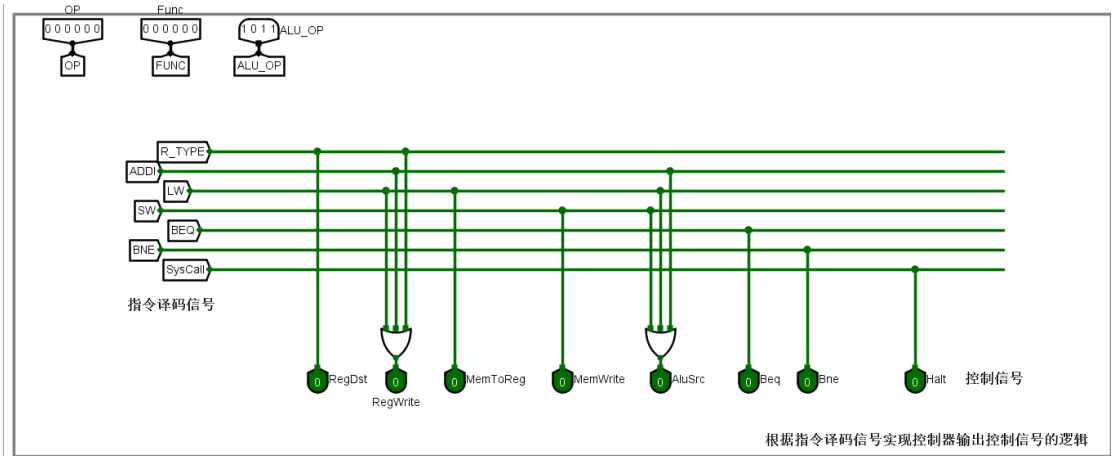


（4）完善硬布线控制器内部逻辑

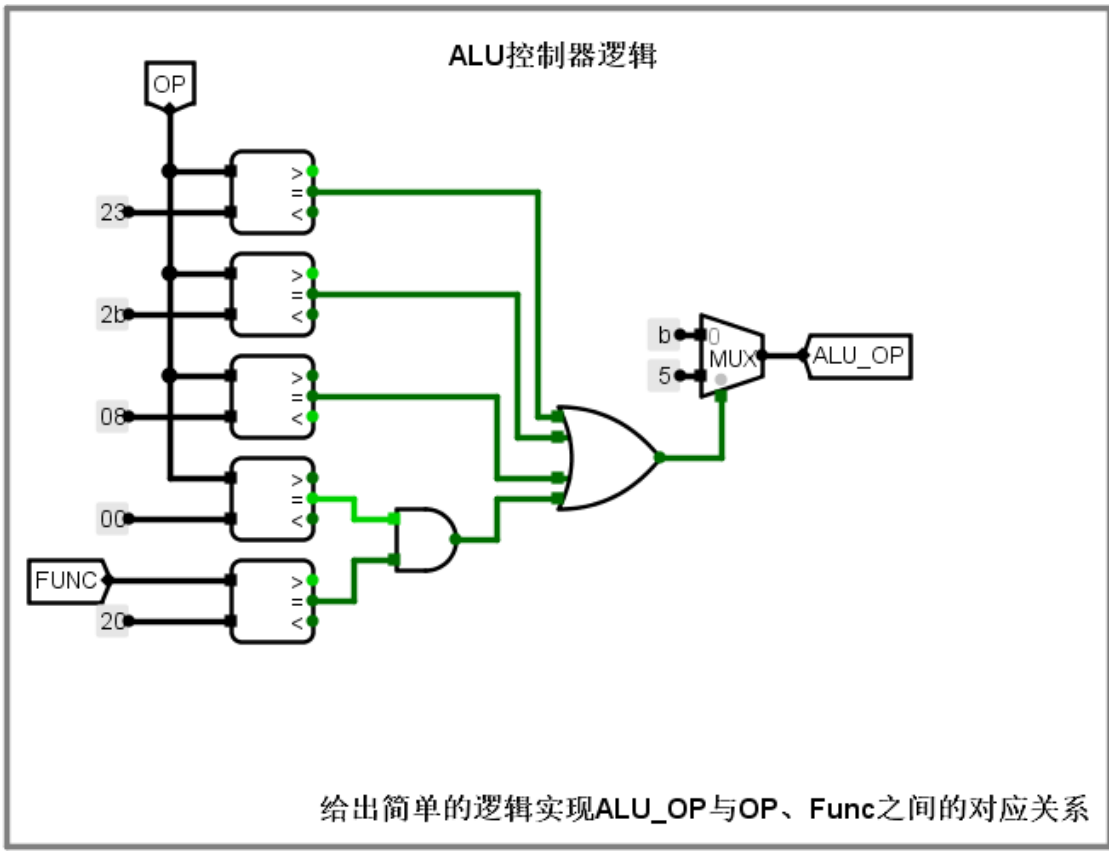
对于控制器输出信号的设计，要根据硬布线控制器中所包含的 9 中控制信号进行分析，如下图。

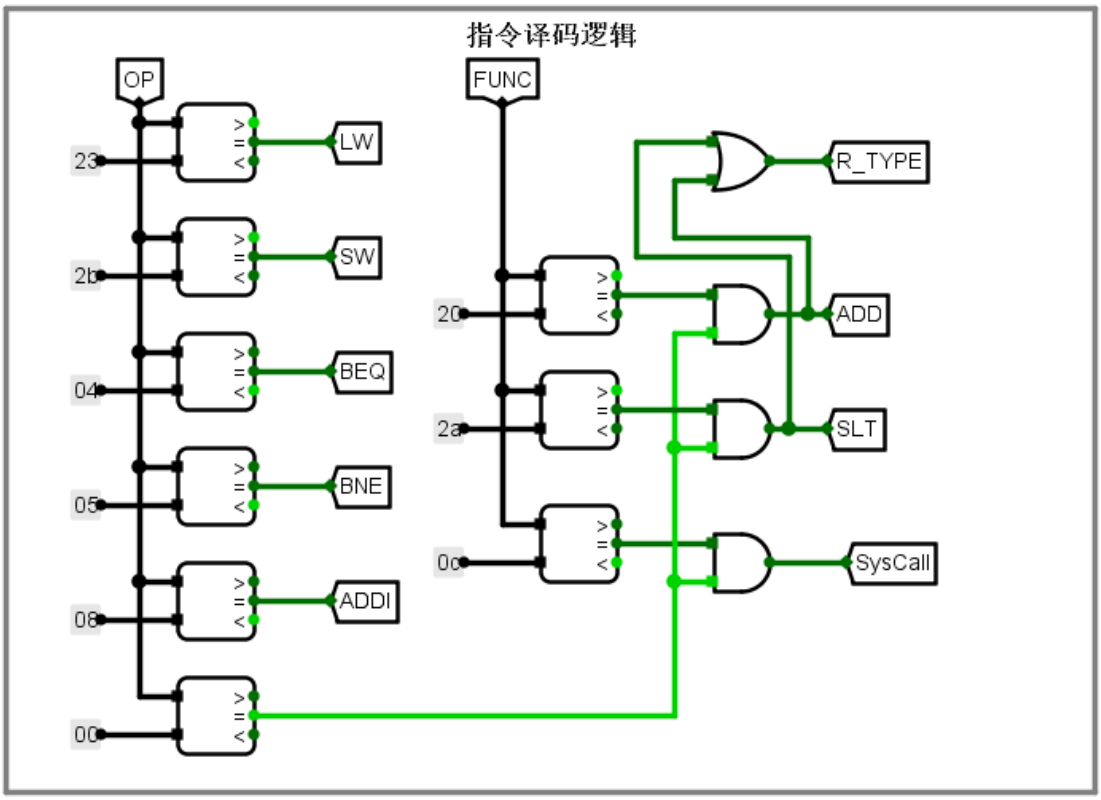
控制信号功能说明 （8条核心指令集）

#	控制信号	信号说明	产生条件
1	MemToReg	写入寄存器的数据来自存储器	lw指令
2	MemWrite	写内存控制信号	sw指令　未单独设置MemRead信号
3	Beq	Beq指令译码信号	Beq指令
4	Bne	Bne指令译码信号	Bne指令
5	AluOP	运算器操作控制符	加法，比较两种运算
6	AluSrcB	运算器第二输入选择	Lw指令，sw指令，addi
7	RegWrite	寄存器写使能控制信号	寄存器写回信号
8	RegDst	写入寄存器选择控制信号	R型指令
9	Halt	停机信号，取反后控制PC使能端	syscall指令



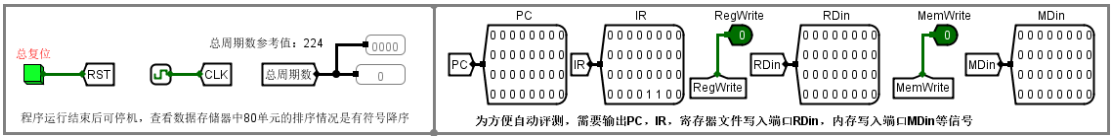
ALU 控制逻辑的设计，由于该 MIPS CPU 设计中有关的 8 条核心 MIPS 指令中，对于 ALU 运算逻辑单元中只涉及到加法和比较，因此这一部分可以大大简化。只有运行 *STL* 指令时，需要选择比较运算，其余都是加法运算。



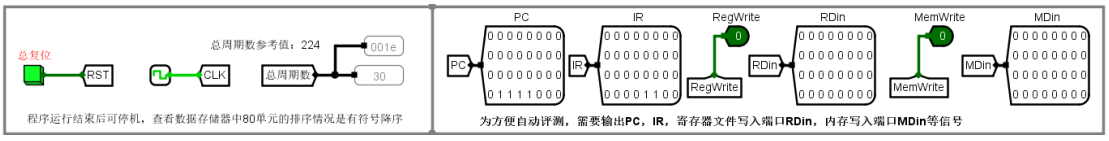


5 实验结果与测试数据

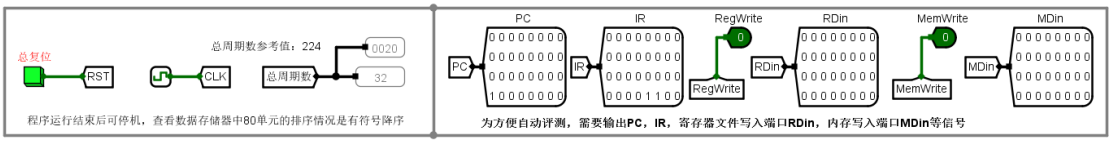
指令存储器载入 syscall_halt.hex



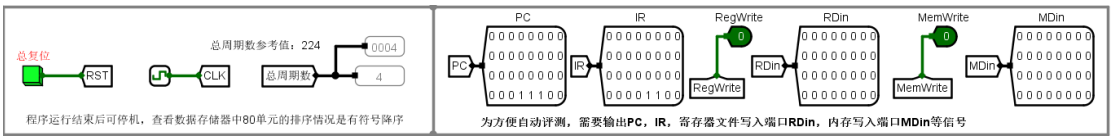
电路复位后，指令存储器载入 addi.hex



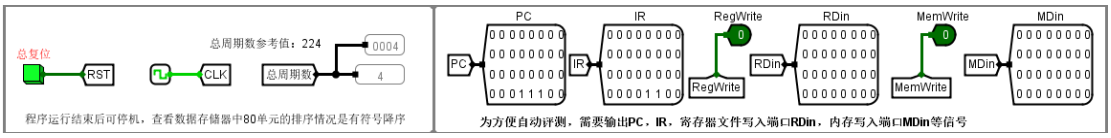
电路复位后，指令存储器载入 add.hex



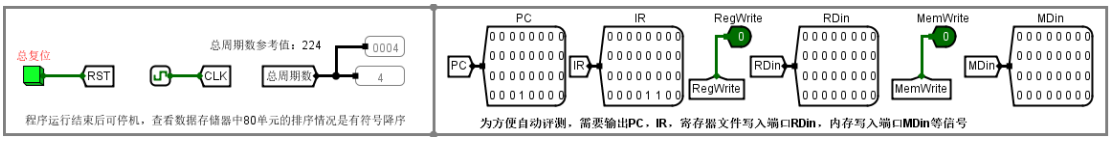
电路复位后，指令存储器载入 beq.hex



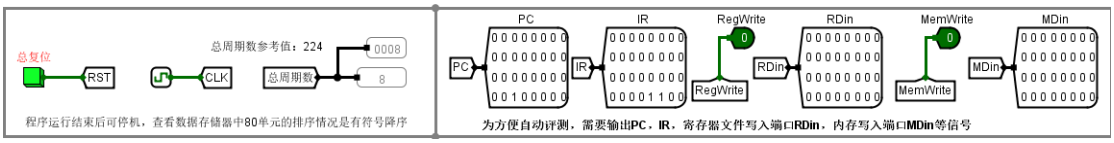
电路复位后，指令存储器载入 bne.hex



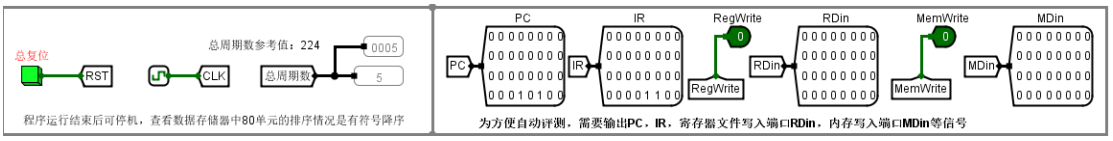
电路复位后，指令存储器载入 slt.hex



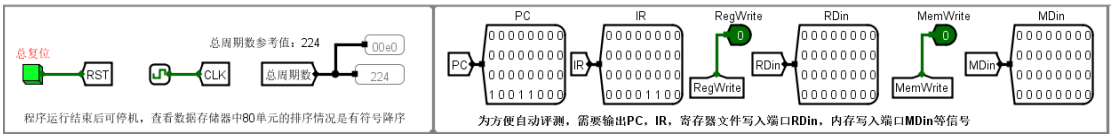
电路复位后，指令存储器载入 sw.hex



电路复位后，指令存储器载入 lw.hex



电路复位后，指令存储器中载入排序程序 sort.hex




```
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
010 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
030 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
050 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
070 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
080 00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff
```

设计二 多周期 MIPS CPU 的设计

1 实验目的

- 掌握多周期 MIPS CPU 设计原理
- 掌握微程序控制器设计的基本原理
- 利用微程序控制器的设计实现多周期 MIPS 处理器

2 主要任务

- 绘制多周期 MIPS CPU 数据通路
- 实现微程序控制器
- 测试联调

3 实验过程

（1）构建多周期 MIPS 主机通路

- 存储器：地址 10 位（字地址），数据 32 位，同时存储指令和数据
- 寄存器文件：同单周期 MIPS CPU
- 运算器 ALU：同单周期 MIPS CPU
- 添加几个专用寄存器（均为 32 位）：PC，IR，DR，A，B，C
- 微程序控制器：将其输出控制信号与对应组件相连，需要设计其内部结构（见步骤 2）

（2）设计微程序控制器

- 输入信号：指令字中的 Opcode，Func 字段，时钟信号，复位信号

- 输出信号：多路选择器的选择信号，存储器的访问控制信号，寄存器的写使能信号，微指

令中的微命令- 运算器的控制信号：设计 ALU 控制器逻辑

微指令中的微命令 ALU_Control：其值决定运算器的运算选择控制信号 ALU_OP 的值

- 指令译码信号：设计指令译码逻辑（与单周期 MIPS CPU 相同）

- 微程序地址转移逻辑设计：

利用 Excel 文件得到四位的微程序入口地址的逻辑表达式

“微程序地址转移逻辑自动生成 ext(2020-9-1).xlsx”

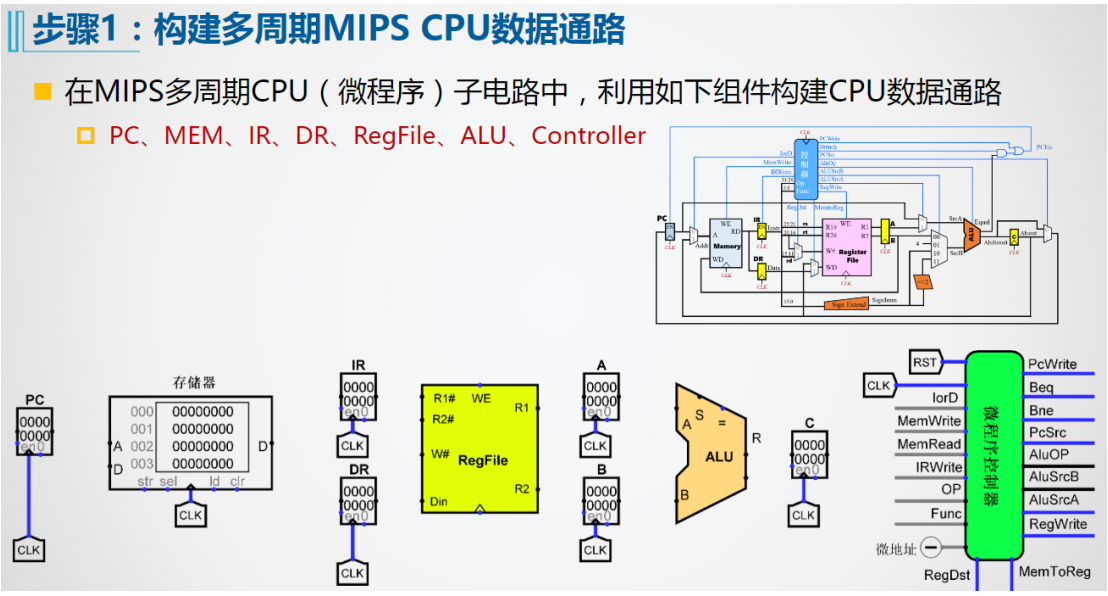
- 根据状态图构建微程序

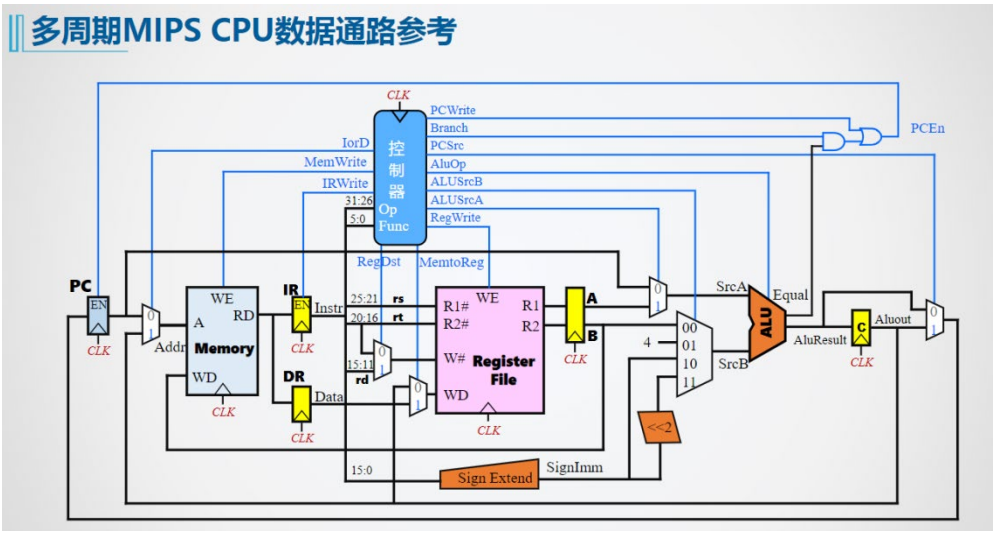
利用 Excel 文件自动得到控制存储器中存储的微指令：

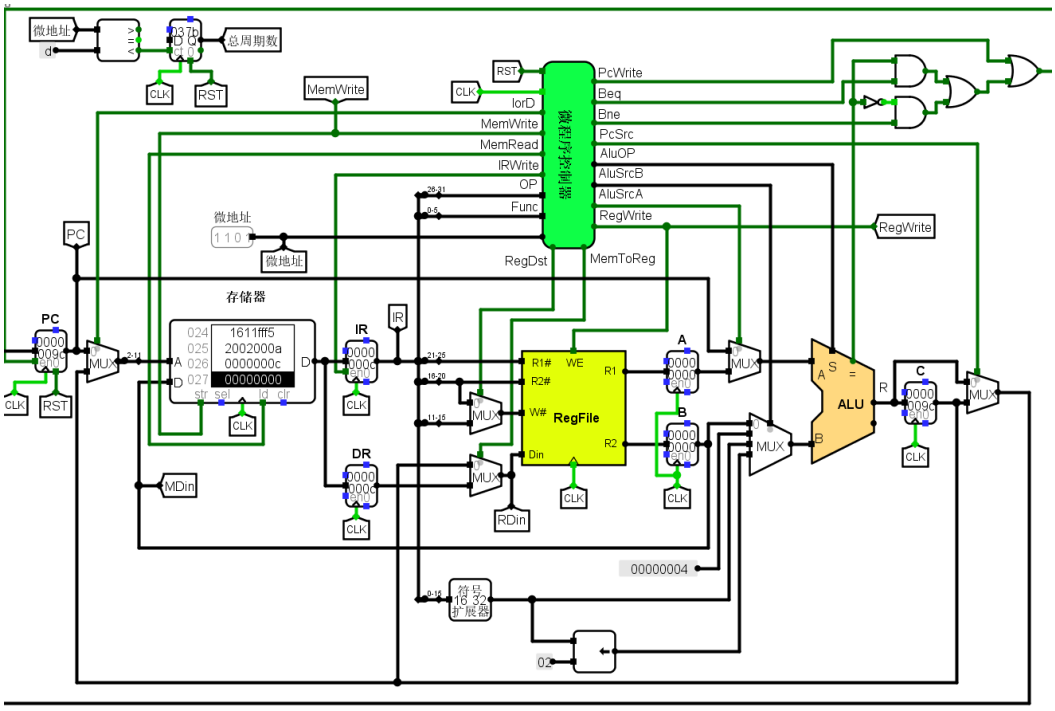
“微指令自动生成(2019-4-22).xlsx”

4 电路图及相关说明

(1) 构建多周期 MIPS CPU 数据通路



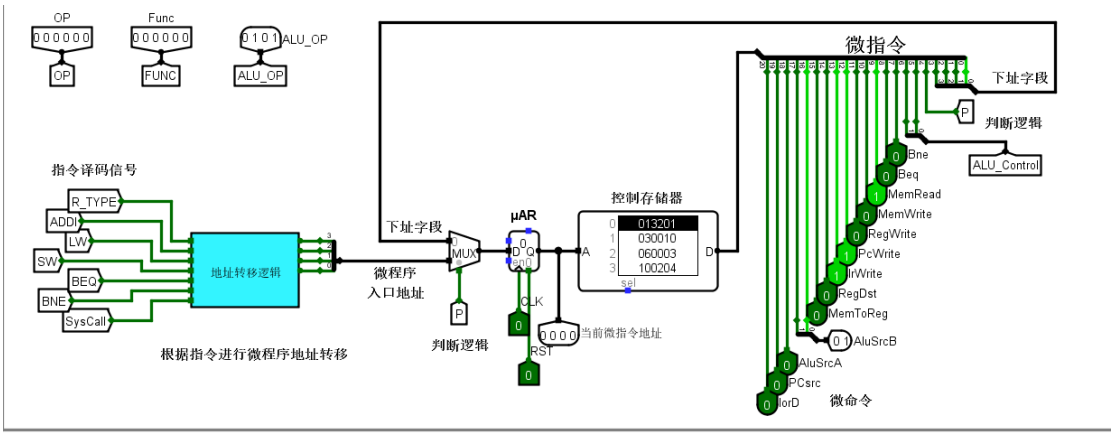

多周期MIPS CPU数据通路参考
CLK
控制器
Op
Func
PCWrite
Branch
PCSrc
AluOp
ALUSrcB
ALUSrcA
RegWrite
PCEn
IorD
MemWrite
IRWrite
RegDst
MemtoReg
PC
IR
DR
Memory
Register File
ALU
SrcA
SrcB
Equal
AluResult
Aluout
Instr
Addr
Data
Sign Extend
SignImm
<<2

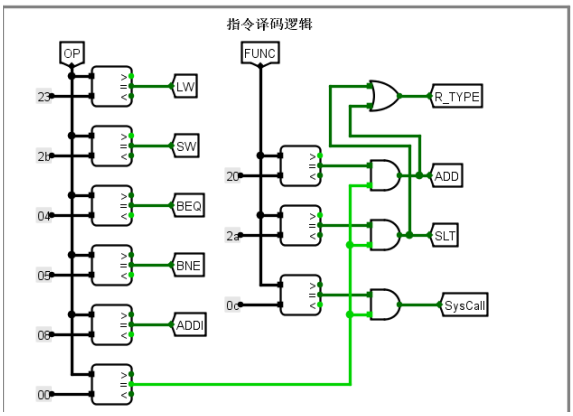

微地址
总周期数
MemWrite
微程序控制器
PcWrite
Beq
Bne
PcSrc
AluOP
AluSrcB
AluSrcA
RegWrite
IorD
MemRead
IRWrite
OP
Func
RegDst
MemToReg
存储器
PC
IR
DR
RegFile
ALU
MDin
RDin
00000004

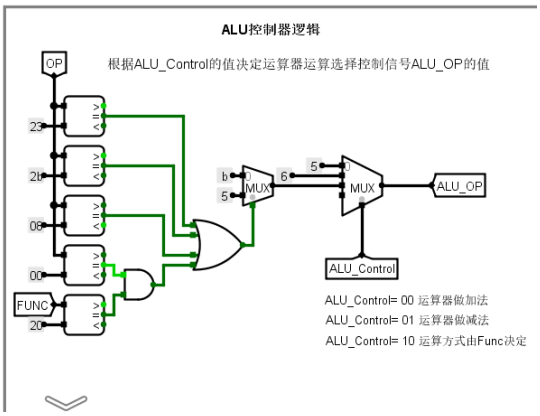
（2）设计微程序控制器

控制信号功能说明 （8条核心指令集）

#	控制信号	信号说明	产生条件
1	PCWrite	PC写使能控制	取指令周期，分支指令执行
2	IorD	指令还是数据	0表示指令，1表示数据
3	IRwrite	指令寄存器写使能	高电平有效
4	MemWrite	写内存控制信号	sw指令
5	MemRead	读内存控制信号	lw指令 取指令
6	Beq	Beq指令译码信号	Beq指令
7	Bne	Bne指令译码信号	Bne指令
8	PcSrc	PC输入来源	顺序寻址还是跳跃寻址
9	AluOP	运算器操作控制符 4位	ALU_Control控制，00加，01减，10由Funct定
10	AluSrcA	运算器第一输入选择	
11	AluSrcB	运算器第二输入选择	Lw指令，sw指令，addi
12	RegWrite	寄存器写使能控制信号	寄存器写回信号
13	RegDst	写入寄存器选择控制信号	R型指令
14	MemToReg	写入寄存器的数据来自存储器	lw指令



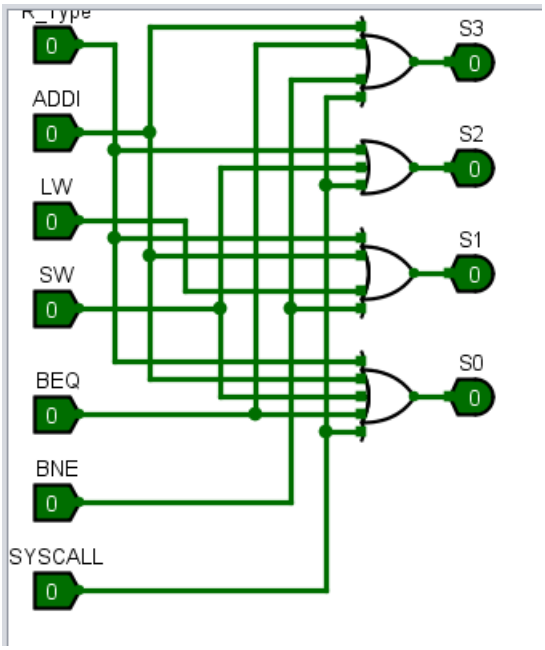




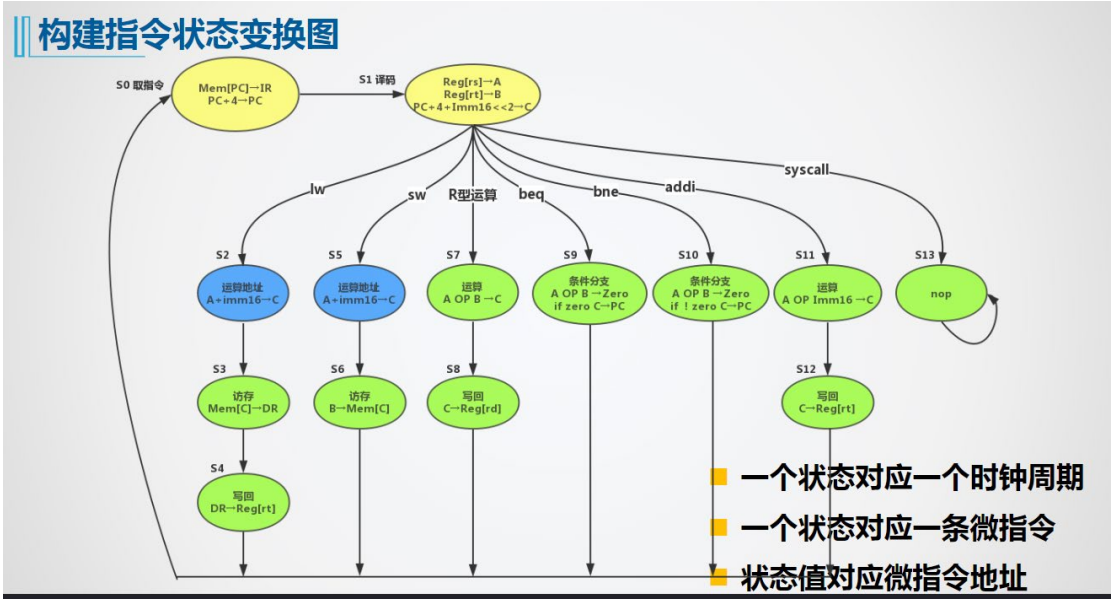
（3）实现微程序地址转移逻辑

填写微程序地址入口表，自动生成微程序地址转移逻辑子电路

机器指令译码信号							微程序入口地址				
R_Type	ADDI	LW	SW	BEQ	BNE	SYSCALL	入口地址 10进制	S3	S2	S1	S0
1						0	7	0	1	1	1
	1						11	1	0	1	1
		1					2	0	0	1	0
			1				5	0	1	0	1
				1			9	1	0	0	1
					1		10	1	0	1	0
1						1	13	1	1	0	1



构建状态指令变化图

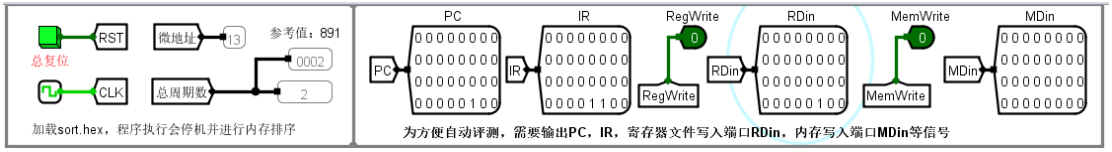


根据状态图构建微程序

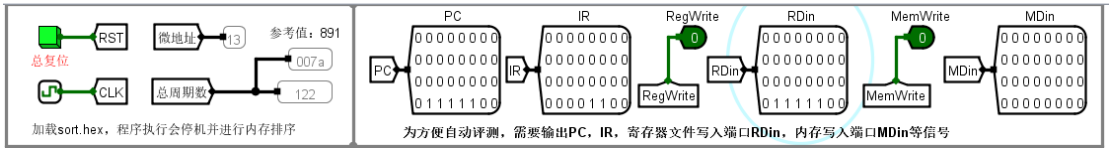
微指令功能	状态	微指令地址	IorD	PcSrc	AluSrcA	AluSrcB	MemToReg	RegDst	IrWrite	PcWrite	RegWrite	MemWrite	MemRead	BEQ	BNE	AluControl	P	下址字段
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00	0	0001
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000
LW1	2	0010	00	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00	0	0100
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00	0	0000
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00	0	0000
R1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10	0	1000
R2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	00	0	0000
BEQ	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	00	0	0000
BNE	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	00	0	0000
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	1100
ADDI2	12	1100	0	0	0	00	0	0	0	0	1	0	0	0	0	00	0	0000
SYSCALL	13	1101	1	0	0	00	0	0	0	0	0	0	0	0	0	11	0	1101

5 实验结果与测试数据

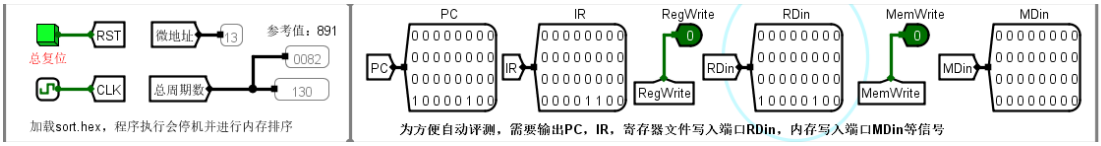
存储器载入 syscall_halt.hex



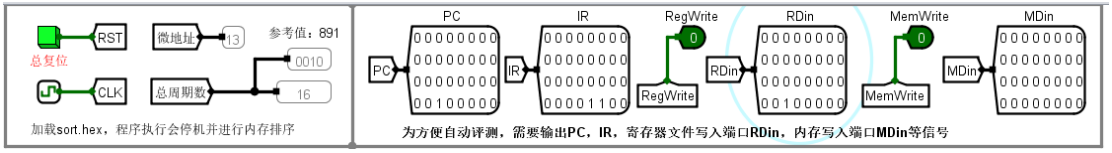
电路复位后，存储器载入 addi.hex



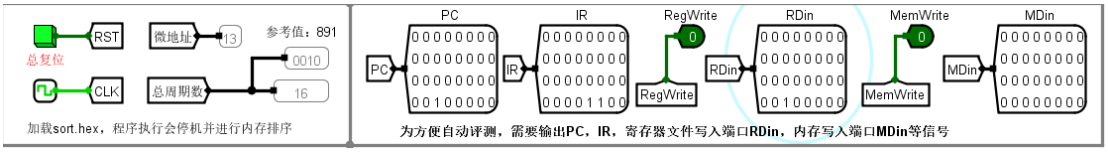
电路复位后，存储器载入 add.hex



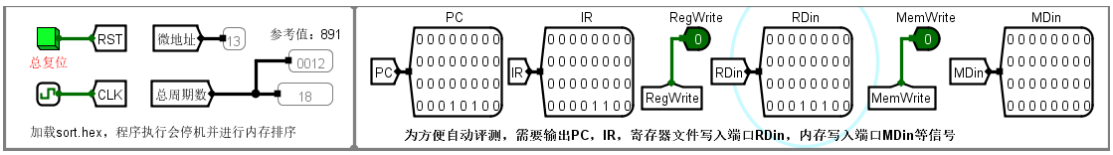
电路复位后，存储器载入 beq.hex



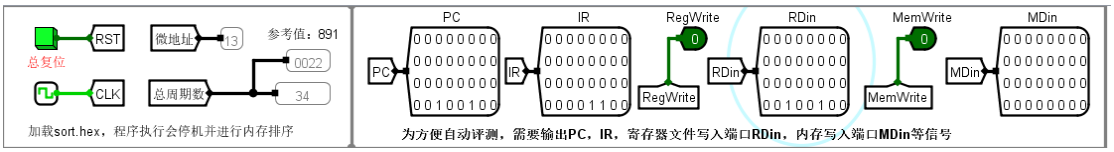
电路复位后，存储器载入 bne.hex



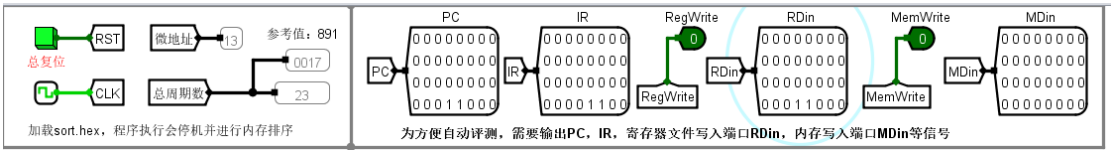
电路复位后，存储器载入 slt.hex



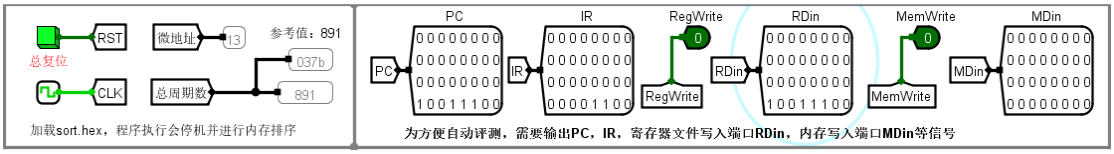
电路复位后，存储器载入 sw.hex



电路复位后，存储器载入 lw.hex



电路复位后，存储器载入 sort.hex




```
000  2010ffff 20110000 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001
010  22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 00008020 2011001c 8e130200 8e340200 0274402a 11000002 ae330200 ae140200
020  2231fffc 1611fff8 22100004 2011001c 1611fff5 2002000a 0000000c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
030  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
040  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
050  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
060  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
070  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
080  00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

心得体会及其他

在完成这个课程设计的过程中，我收获颇丰。一开始着手做 CPU 时，完全没有头绪，不知道如何下手，好在有慕课上的视频把 CPU 的架构和数据通路以及做的步骤讲解了，以及 CSDN 上的大佬的讲解，在老师和同学的帮助才最终完成了我的课程设计。在这个过程中，我了解了每一种指令时如何运行的、硬布线和微程序 CPU 的差别、如何做出硬布线和微程序控制器以及怎样利用 xlsx 文件自动生成表达式。虽然做的过程中有时候会非常烦躁，不过当我静下心来，解决问题后，当 CPU 跑起来的那一刻的成就感是无可替代的。