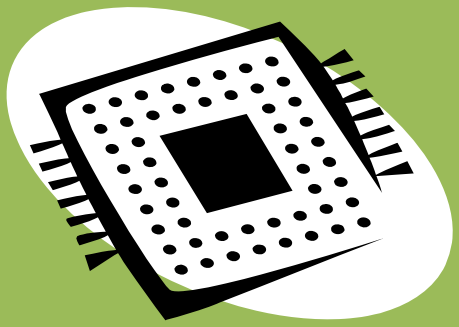
华中科技大学

2018

计算机组成原理 ·实验报告·

专　　业：	计算机科学与技术
班　　级：	CS1601
学　　号：	U201614532
姓　　名：	吕鹏泽
电　　话：	18703816020
邮　　件：	103349015@qq.com
完成日期：	2018-12-23



计算机科学与技术学院

目 录

1 单周期 CPU 设计实验

1.1 设计要求

利用运算器组成实验设计的 32 位 ALU，存储系统实验中构建的寄存器文件以及 Logisim 中其他功能部件构建一个 32 位 MIPS CPU 单周期处理器，该处理器应支持如图 1-1 所示的 8 条核心指令集，包括 add, slt, addi, lw, sw, beq, bne 和 syscall。最终设计完成的 CPU 应能运行教师提供的标准测试程序，程序存储在 Logisim 的指令存储器中，运行结果存储在数据存储器中。

#	MIPS指令	RTL功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt]
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm)
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

图 1-1 核心指令集 8 条

1.2 方案设计

根据 CPU 的组成结构，CPU 应当划分为存储器（包括指令存储器和数据存储器）、寄存器、ALU 以及控制器几部分。其中存储器可以直接使用 Logism 的功能部件，寄存器和 ALU 在之前的实验中已经设计完毕，而控制器的设计依托于数据通路的形式，因此首先要设计出数据通路，然后根据数据通路设计控制器从而给出具体的控制信号，最后进行调试，完成 CPU 的设计。

1.2.1 设计取指令数据通路

对于单周期 CPU 来说，每当时钟周期到来时，PC 计数器的值要”+1”，然后以

PC 为地址访问指令存储器，获得将要执行的指令。因此取指令操作涉及 PC、加法器和指令存储器。其中 PC 可以使用 32 位寄存器来实现，指令存储器使用 RAM 组件时间，”+1”操作使用加法器组件实现，由于在 Logism 中 32 位的存储器是按照字进行编址的，因此 PC 每次加 1（若是按照字节编址，则 PC 每次加 4）。据此可以得到取指令的数据通路如图 1-2 所示：

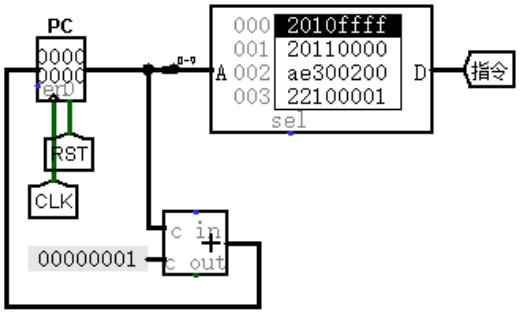


图 1-2 取指令数据通路设计

此外，为避免单周期 CPU 的资源冲突，指令存储器和数据存储器不能共用，”+1”操作也不能共用 ALU，均要为独立器件。

1.2.2 设计 R 型指令数据通路

对于 R 型指令，执行 R[$rd]←R[$rs] OP R[$rt]的操作，每一条指令均包含 3 个寄存器编号$rs、$rt 和$rd，因此需要将 3 个寄存器编号分离出来连接到寄存器文件中，然后由$rs 和$rt 指定的寄存器的数据从寄存器文件的输出端口输出，接到 ALU 上参与运算，在经过一小段延时后 ALU 的运算结果送到寄存器文件的数据输入端口，在下一个时钟到来时更新由$rd 指定的寄存器的值。据此可以得到 R 型指令的数据通路如图 1-3 所示：

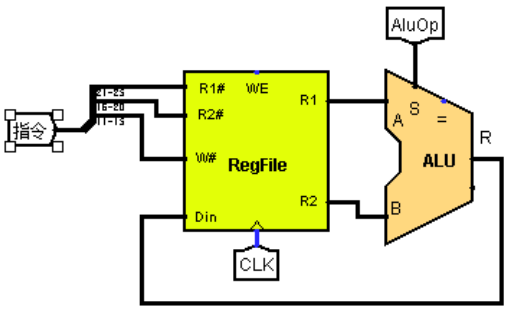


图 1-3 R 型指令数据通路

MIPS 访存指令属于 I 型指令，访存地址等于$rs 的值加上 16 位立即数，地址运算通过 ALU 完成，所以需要将$rs 的值（由寄存器文件的 R1 输出）和立即数（从指令中分离出来）送入 ALU，此外由于 ALU 是 32 位的，从指令中分离出来的 16 位立即数要符号拓展成 32 位。对于 Load 指令，访问得到的数据要写回寄存器$rt 中，因此需要将数据存储器的输出端口接到寄存器文件的 Din 上。对于 Store 指令，需要将寄存器$rt 的值写道数据存储器中，因此需要将寄存器文件的 R2 接到数据存储器的 Din 端。初步得到访存指令的数据通路如图 1-4 所示：

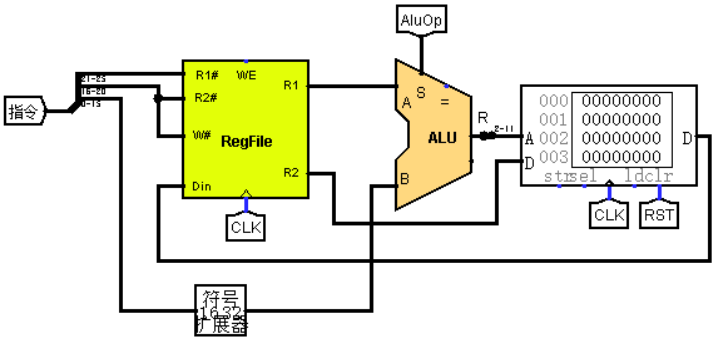


图 1-4 访存指令的部分数据通路

同 R 型指令数据通路比较可知，寄存器文件的 W#会接入$rd 和$rt，D_{in} 会接入数据存储器的 D 和 ALU 的 R，ALU 的 B 端口会接入 R2 和立即数，为了将两个数据通路统一到一个电路中，在 W#、B、D 端口增加多路选择器来选取正确的信号，多路选择器的控制信号可以有后面设计的控制器给出。得到访存指令和 R 型指令的混合数据通路如图 1-5 所示：

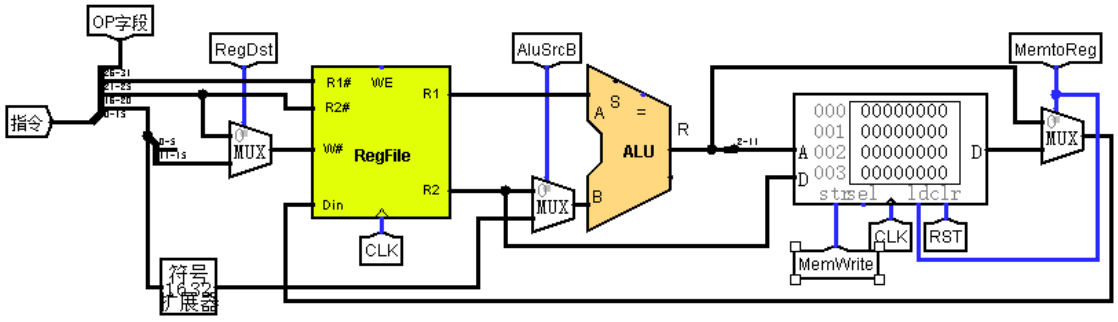


图 1-5 访存指令和 R 型指令的混合数据通路

1.2.3 设计条件转移指令数据通路

bne 和 beq 指令要实现 if(R[$rs] =(≠) R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})，

其中条件语句可以在控制器中实现，给出控制信号，PC ← PC + $SignExt_{18b}$({imm, 00})的操作则要在图 1-5 所示的数据通路上增加由 PC + $SignExt_{18b}$({imm, 00})送到 PC 的数据通路，增加该数据通路后会有两条线路接入 PC，则使用多路选择器进行选择，控制信号由控制器根据 if 条件给出。由于在 Logism 中指令存储器是按照字编址的，因此立即数无需左移两位。此外，为了实现停止指令，将控制器生成的 Halt 信号接到 PC 的使能端即可。修改后的数据通路如图 1-6 所示：

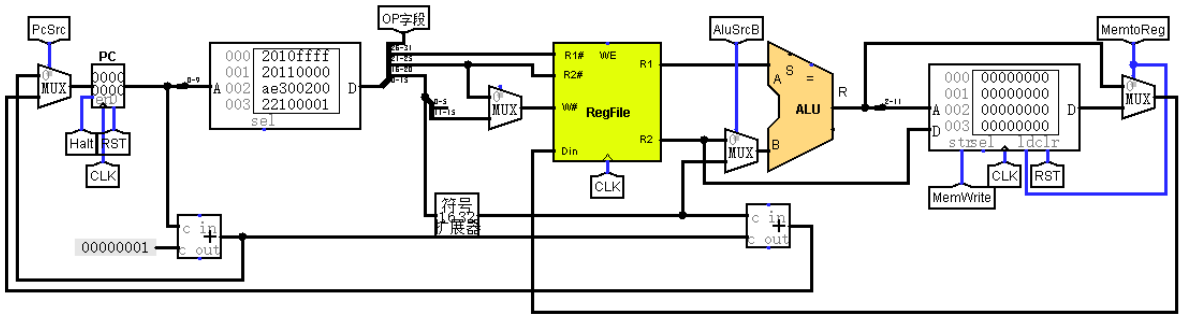


图 1-6 增加跳转指令后的数据通路

1.2.4 设计控制器

设计完数据通路后，即可按照指令特性与数据通路走向，确定对于每一个指令的控制信号，得到各控制信号的产生条件如表 1-1 所示：

表 1-1 控制信号功能说明

#	控制信号	信号说明	产生条件
1	MemToReg	写入寄存器的数据来自于存储器	lw 指令
2	MemWrite	写内存控制信号	sw 指令
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	运算器控制操作符	加法，比较运算
6	AluSrcB	运算器第二输入选择	lw、sw、addi 指令
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，取反后接入 PC 使能端	syscall 指令

控制信号的产生条件与当前执行的指令有关，可以使用比较器判断当前执行的指令，通过查阅 MIPS 指令集得到各指令的操作码如表 1-2 所示，对于 R 型指令来说，由 FUNC 来确定操作类型。

表 1-2 各指令对应的操作码

MIPS 指令	操作码(6 位)	FUNC 字段(5 位)
add	00H	20H
slt	00H	2aH
addi	08H	/
lw	23H	/
sw	2bH	/
beq	04H	/
bne	05H	/
syscall	00H	0cH

根据表 1-2 的信息和比较器得到指令译码逻辑的电路图如图 1-7 指令译码逻辑电路所示：

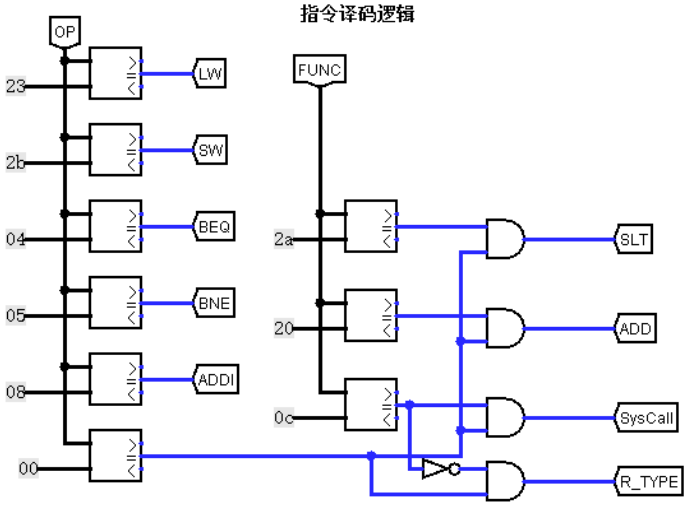


图 1-7 指令译码逻辑电路

本实验实现的 8 条指令中 ALU 的运算只有加法操作和比较操作，且当指令为 slt 时执行比较操作，则可使用多路选择器并将 slt 作为选择器的控制信号得到 ALU_OP，电路图如图 1-8 所示。

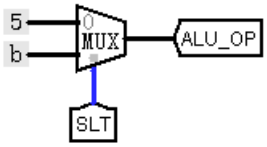


图 1-8 ALU 控制逻辑

经过指令译码逻辑电路获得指令译码信号后，结合表 1-1 各控制信号的产生条件，可以使用简单的组合逻辑的设计得到控制信号，控制信号的设计如图 1-9 所示：

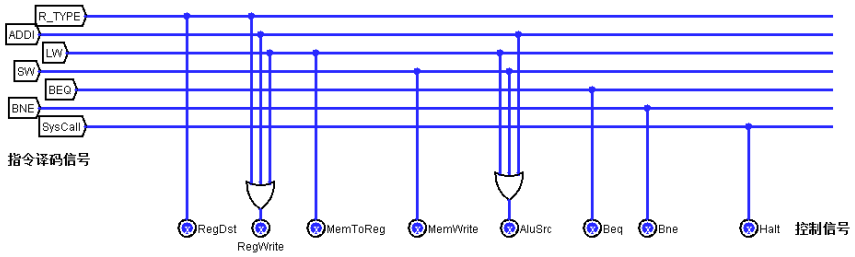


图 1-9 控制器控制信号

加上 ALU_OP，所有的控制信号均已得到，将其封装成一个模块以方便连接到数据通路中，封装图如图 1-10 所示：

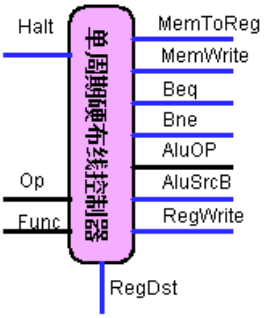


图 1-10 控制器模块

1.2.5 设计单周期 MIPS CPU

将上一步实现的控制器接入到数据通路，将控制信号和对应的控制端口连接，即可构建单周期 CPU，其中和 PC 连接的多路选择器的控制信号 PcSrc 由 Beq 和 Bne 经过组合逻辑实现，具体的关系为 PcSrc=Beq&Equal+Bne&~Equal。综上，可以得到单周期 CPU 的设计图如图 1-11 所示：

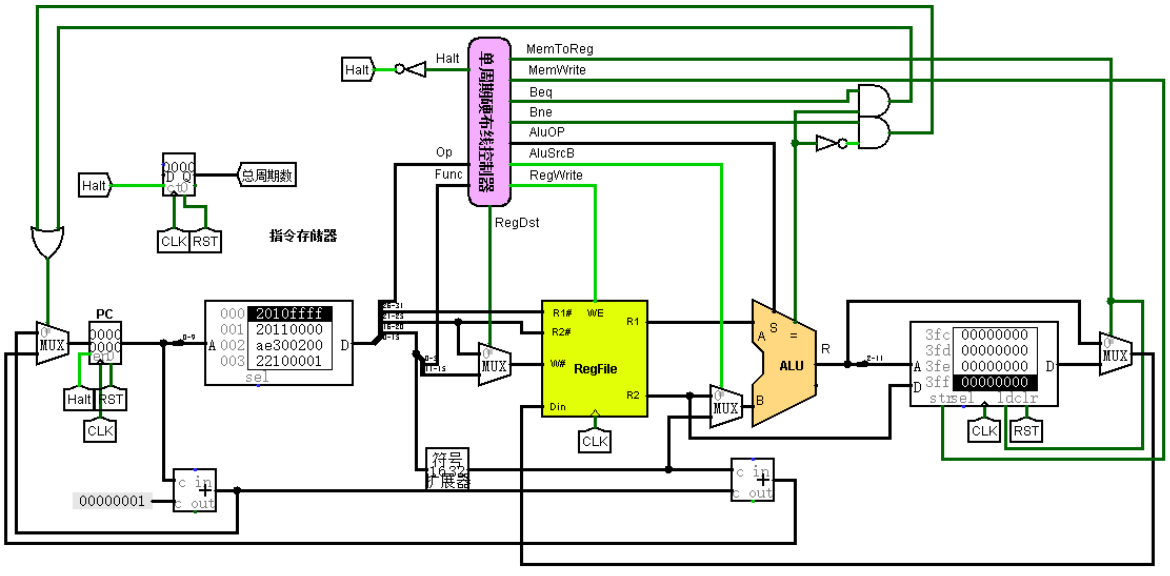


图 1-11 单周期 MIPS CPU 电路图

1.3 实验步骤

（1） 根据方案设计，实现如图 1-6 所示的数据通路

（2） 根据表 1-2 各指令的操作码利用比较器实现指令译码逻辑电路

（3） 分析 ALU 在各指令下参与的运算设计出 ALU_OP 信号

（4） 根据表 1-1 各控制信号的产生条件设计出其余控制信号

（5） 将封装好的控制器连接到数据通路中，将控制信号接入到对应端口

（6） 加载测试程序，观察运行结果，对执行错误的地方单步调试查找错误原因，然后修改错误直到程序得到正确运行结果。

1.4 故障与调试

1.4.1 PC 计数器的值问题

故障现象：指令存储器的值每次移动 4 条指令。

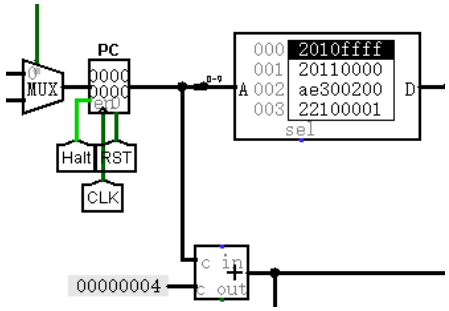


图 1-12 PC 计数器值问题

原因分析：如图 1-12，PC 寄存器设置为上升沿刷新，每当时钟到来时，指令存储器就会移动 4 条指令，即只定位到第 0、4、8…条指令。这是因为在 Logism 中 32 位的存储器是按照字进行编址的，因此将 PC+4 改为 PC+1 即可。

解决方案：将连接在加法器的常量 4 改为常量 1 即可。

1.4.2 数据存储器运算结果存放地址问题

故障现象：如图 1-13，执行完测试程序后数据没有出现在期望的 80 位置处而出现在 200 地址处。

```
200  00000006 00000000 00000000 00000000  00000005 00000000 00000000 00000000
208  00000004 00000000 00000000 00000000  00000003 00000000 00000000 00000000
210  00000002 00000000 00000000 00000000  00000001 00000000 00000000 00000000
218  00000000 00000000 00000000 00000000  ffffffff 00000000 00000000 00000000
```

图 1-13 数据存储器地址问题

原因分析：经过 ALU 运算后得到的是按字节编址的地址，而数据存储器是按字编址的，因此需要将地址的低两位忽略。

解决方案：如图 1-14，使用分线器将地址线 0-9 位改为 2-11 位接到数据存储器的数据存储器的 A 端。

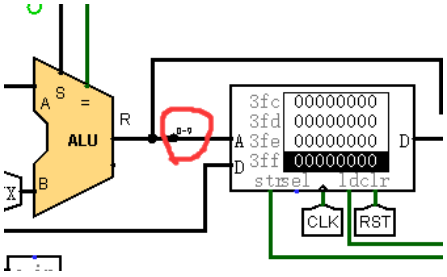


图 1-14 修改数据存储器的地址

1.4.3 程序运行周期数统计错误

故障现象：程序运行的总周期数一直增加。

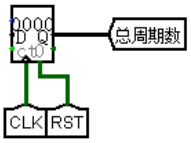


图 1-15 程序运行总周期数

原因分析：如图 1-15，每到时钟到来，计数器的值就会+1，当执行到 Syscall 指令时，接在 PC 寄存器使能端的 Halt 信号为 0，值不再改变，而周期计数器使能端仍有效，其值仍会改变，因此需要将 Halt 的信号接到计数器的使能端。

解决方案：将 Halt 信号接到周期计数器的使能端。

1.5 测试与分析

运行老师提供的测试程序，观察运行结果，测试程序的汇编代码见 1.5.1，测试程序的机器指令见 1.5.2。

1.5.1 测试程序源代码

```
.text
#############################################################################
#########
#本程序实现 0x80 开始的 8 个字单元的降序排序,此程序可在 mars mips 仿真器中运行,字地址 0x80
#############################################################################
#########
  .text
sort_init:
  addi $s0,$0,-1
  addi $s1,$0,0
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
```

```
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)
  addi $s0,$s0,1
  addi $s1,$s1,4
  sw $s0,128($s1)

  add $s0,$zero,$zero
  addi $s1,$zero,28     #排序区间
sort_loop:
  lw $s3,128($s0)
  lw $s4,128($s1)
  slt $t0,$s3,$s4
  beq $t0,$0,sort_next     #降序排序
  sw $s3, 128($s1)
  sw $s4, 128($s0)
sort_next:
  addi $s1, $s1, -4
  bne $s0, $s1, sort_loop

  addi $s0,$s0,4
  addi $s1,$zero,28
  bne $s0, $s1, sort_loop
```

```
addi    $v0,$zero,10            # system call for exit
syscall                         # we are out of here.
```

1.5.2 测试程序机器指令

v2.0 raw

2010ffff 20110000 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 22100001 22310004 ae300200 8020 2011001c 8e130200 8e340200 274402a 11000002 ae330200 ae140200 2231fffc 1611fff8 22100004 2011001c 1611fff5 2002000a c

1.5.3 测试结果分析

如图 1-16 所示，程序的最后一条指令为 Syscall，运行完毕后 PC 会停在 026H 的位置。理论的运行周期数为 224，观察运行结果总周期数与预期符合。

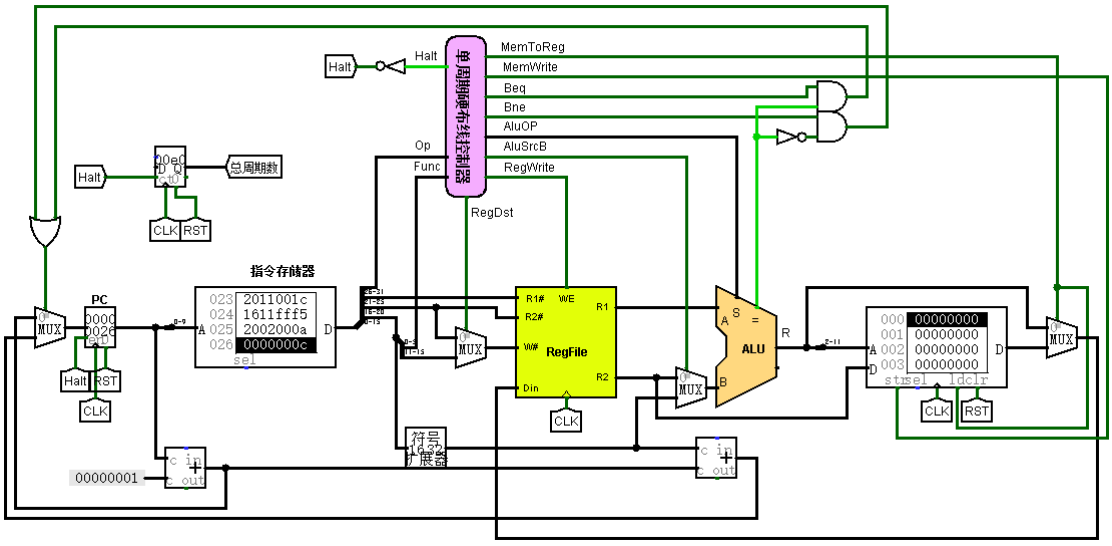


图 1-16 运行结果

如图 1-17，打开数据存储器，理论结果为运行后在 080H 的位置降序排列 6，5，4，3，2，1，0，-1，观察数据存储器可知测试结果与预期相符。

```
080  00000006 00000005 00000004 00000003  00000002 00000001 00000000 ffffffff
```

图 1-17 执行后数据存储器的内容

2 多周期 CPU 设计实验

2.1 设计要求

利用运算器组成实验设计的 32 位 ALU，存储系统实验中构建的寄存器文件以及 Logisim 中其他功能部件构建一个 32 位 MIPS CPU 多周期处理器，处理器的该处理器应支持如图 2-1 核心指令集 8 条所示的 8 条核心指令集，包括 add, slt, addi, lw, sw, beq, bne 和 syscall。最终设计完成的 CPU 应能运行教师提供的标准测试程序，程序和运行结果数据均存储在唯一的存储器中。

#	MIPS指令	RTL功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt]
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm)
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

图 2-1 核心指令集 8 条

2.2 方案设计

多周期 MIPS CPU 处理器可以在单周期 MIPS CPU 处理的基础上修改数据通路、重新设计控制器得到。多周期 MIPS CPU 和单周期 MIPS CPU 的不同体现在以下几点：

（1） 指令存储器和数据存储器共享

（2） 程序计数器的”+1”操作共享 ALU

（3） 主要功能部件的输出端增加寄存器，以共用部件时保存之前得到的数据

2.2.1 设计数据通路

根据多周期 CPU 的特点，在单周期 CPU 处理器架构（图 1-11）的基础上增加了以下部件：为了实现存储部件和运算部件 ALU 的共用，需要增加寄存器因此在设计数据通路时增加了数据寄存器 DR，用于存放从存储器读出的数据。增加指令寄存器 IR，用于存放从存储器读出的指令。在寄存器文件输出端增加寄存器 A 和 B，用于存放存寄存器文件读出的数据。在 ALU 的输出端增加寄存器 C，用于存放 ALU 的输出结果。

其中 A、B、C、DR 寄存器只用于保存上一时钟周期的数据，因此不需要增加专门的控制信号，只在时钟到来时更新数据即可。IR 存储器用于保存当前执行的指令，在当前指令未执行完时其值不能改变，因此需要专门的控制信号来控制 IR 数据的更新。

与单周期 CPU 不同，多周期 CPU 每一条指令执行的时钟周期数不相同，因此 PC 的值的更改需要由控制信号控制，以确保只有当执行完一条指令后 PC 的值才改变。

最后，为了共用部件，需要在 ALU 和存储器的输入端口增加多路选择器和对应的控制信号来实现器件共用。最终设计得到的多周期数据通路如图 2-2 所示：

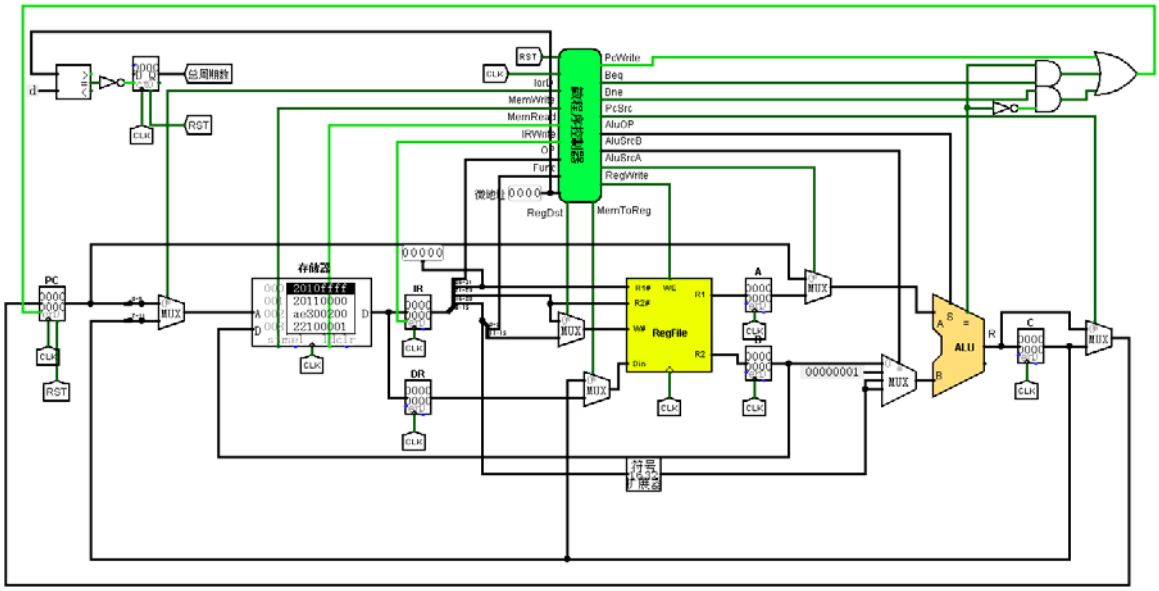


图 2-2 多周期 CPU 数据通路

各控制信号的说明见表 2-1.

表 2-1 控制信号功能说明

#	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

2.2.2 设计微程序控制器

（1） 微程序控制器结构

微程序控制器主要由控制存储器、地址转移逻辑、微地址寄存器三部分组成，其组成电路图如图 2-3 所示：

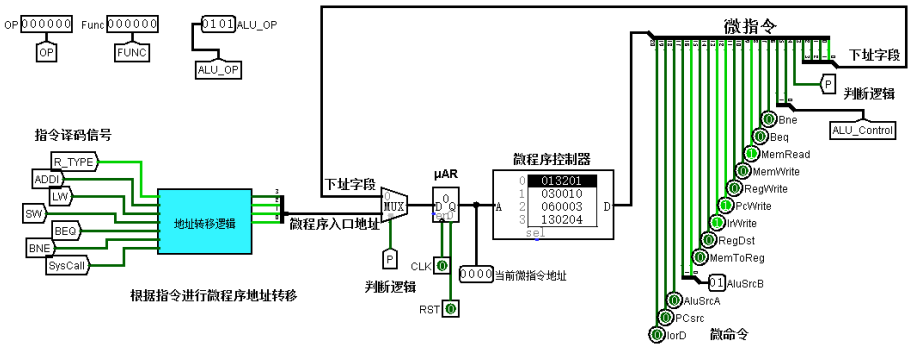


图 2-3 微程序控制器电路图

（2） 设计微指令

有了微程序控制器的数据通路，接下来要设计微指令。在多周期处理器中，指令的执行需要占用多个时钟周期，而且每个指令占用的时钟周期数也不尽相同，因此需要设计出每个指令状态变换图，根据该操作得到控制信号。设计的指令状态变换图如图 2-4 所示，每一条指令的前两个状态必定未取指令和译码操作，之后根据指令的不同进入不同的状态，在该指令执行完毕后又再次回到取指令状态。

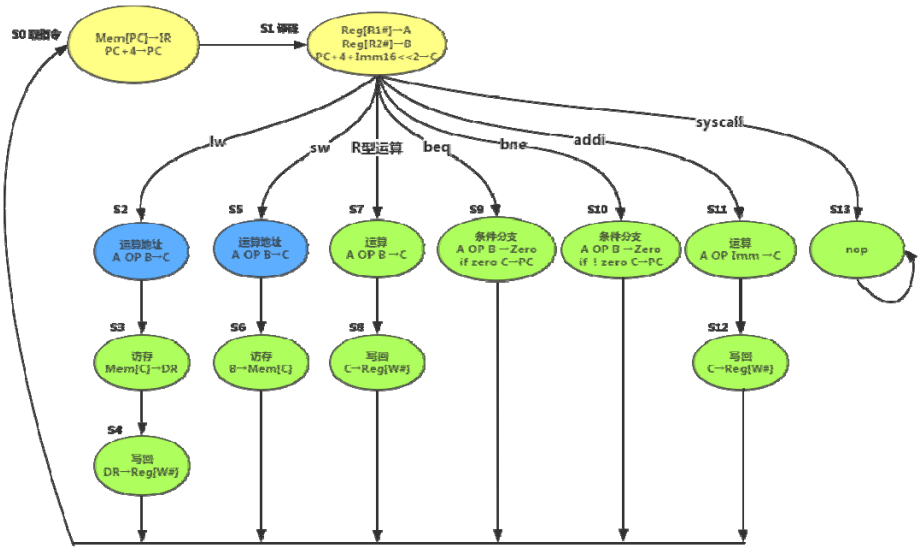


图 2-4 指令状态变换图

得到指令状态变换图后就要根据状态图构造微程序，每一个状态均可视为一条微指令。各指令的操作流程及控制信号如表 2-2——表 2-8 所示。

表 2-2 R 型指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=0, ALU_Control=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	AluSrcA=0, ALUSrcB=11, ALU_Control=00
运算	A OP B→C	ALUSrcA=1, ALUSrcB=0, ALU_Control=10
写回	C→Reg[W#]	RegDst=1, RegWrite=1, MemtoReg=0

表 2-3 addi 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=0, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
运算	A OP Imm→C	ALUSrcA=1, ALUSrcB=10, ALU_Control=00
写回	C→Reg[W#]	RegDst=0, RegWrite=1, MemtoReg=0

表 2-4 lw 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=0, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
运算地址	A OP B→C	ALUSrcA=1, ALUSrcB=10, ALU_Control=00
访存	Mem[C]→DR	MemRead=IorD=1
写回	DR→Reg[W#]	RegDst=0, RegWrite=1, MemtoReg=1

表 2-5 sw 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=0, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
运算地址	A OP B→C	ALUSrcA=1, ALUSrcB=10, ALU_Control=00
访存	B→Mem[C]	MemWrite=IorD=1

表 2-6 beq 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=1, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
条件分支	if(A==B) C→PC	PCSrc=1, ALUSrcA=1, ALUSrcB=00, BEQ=1, ALU_Control=10

表 2-7 bne 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=1, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
条件分支	if(A!=B) C→PC	PCSrc=1, ALUSrcA=1, ALUSrcB=00, BNE=1, ALU_Control=10

表 2-8 syscall 指令操作流程及控制信号

指令阶段	操作流程	控制信号
取指令	M[PC]→IR PC+4→PC	MemRead=IrWrite=PcWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, PcSrc=0, ALU_OP=00
译码及取操作数	Reg[R1#]→A Reg[R2#]→B PC+4+Imm16→C	ALUSrcA=0, ALUSrcB=11, ALU_OP=00
停机	nop	nop

根据各个指令的操作流程及控制信号表将控制信号填写到表 2-9 所示的微指令字段中，得到微指令。

表 2-9 微指令设计表

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	微指令	十六进制
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00	0	0001	0000100110010000000001	13201
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000	0001100000000000010000	30010
LW1	2	0010	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011	0011000000000000000011	60003
LW2	3	0011	1	0	0	11	0	0	0	0	0	0	1	0	0	00	0	0100	1001100000010000000100	130204
LW3	4	0100	0	0	0	11	1	0	0	0	1	0	0	0	0	00	0	0000	0001110001000000000000	38800
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110	0011000000000000000110	60006
SW2	6	0110	1	0	0	11	0	0	0	0	0	1	0	0	0	00	0	0000	1001100000100000000000	130400
R型运算1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10	0	1000	0010000000000001001000	40048
R型运算2	8	1000	0	0	0	11	0	1	0	0	1	0	0	0	0	10	0	0000	0001101001000010000000	34840
Beq	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	10	0	0000	0110000000001010000000	C0140
Bne	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	10	0	0000	0110000000001100000000	C00C0
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	1100	0011000000000000001100	6000C
ADDI2	12	1100	0	0	0	11	0	0	0	0	1	0	0	0	0	00	0	0000	0001100001000000000000	30800
SYSCALL	13	1101	0	0	0	11	0	0	0	0	0	0	0	0	0	00	0	1101	0001100000000000001101	3000D

（3） 构建微程序控制器

根据图 2-3 微程序控制器电路图将第（2）步得到的微指令的 16 进制形式的数据加载到微程序控制器中。由于多周期 CPU 的指令类型并没有改变，仍可使用单周期的指令译码逻辑和 ALU 控制器逻辑的电路，电路图如图 2-5 所示：

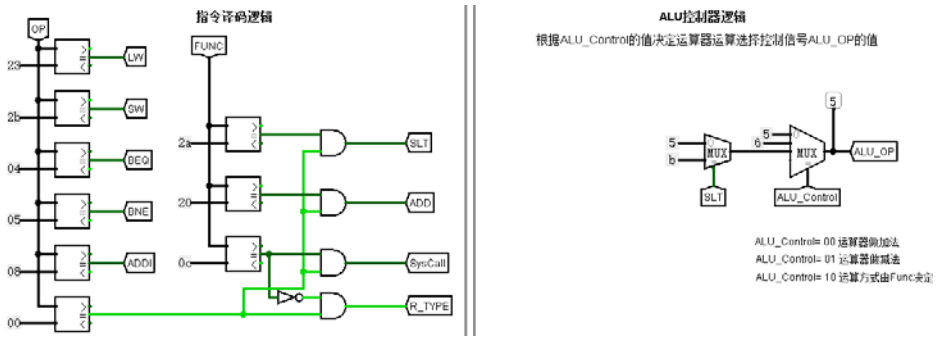


图 2-5 指令译码逻辑和 ALU 控制器逻辑

最后，进行地址转移逻辑的设计，地址的转移逻辑设计如表 2-10 表 2-10 所示。

表 2-10 地址转移逻辑表

R_Type	ADDI	LW	SW	BEQ	BNE	SYSCALL	微程序入口地址（10进制）	S3	S2	S1	S0
1	X	X	X	X	X	X	7	0	1	1	1
X	1	X	X	X	X	X	11	1	0	1	1
X	X	1	X	X	X	X	2	0	0	1	0
X	X	X	1	X	X	X	5	0	1	0	1
X	X	X	X	1	X	X	9	1	0	0	1
X	X	X	X	X	1	X	10	1	0	1	0
X	X	X	X	X	X	1	13	1	1	0	1

根据表 2-9 地址转移关系经过简单的组合逻辑即可得到地址转移逻辑电路。将设计好的微程序封装成一个模块以方便连接到数据通路中，电路图见图 2-6 所示，封装图见图 2-7。

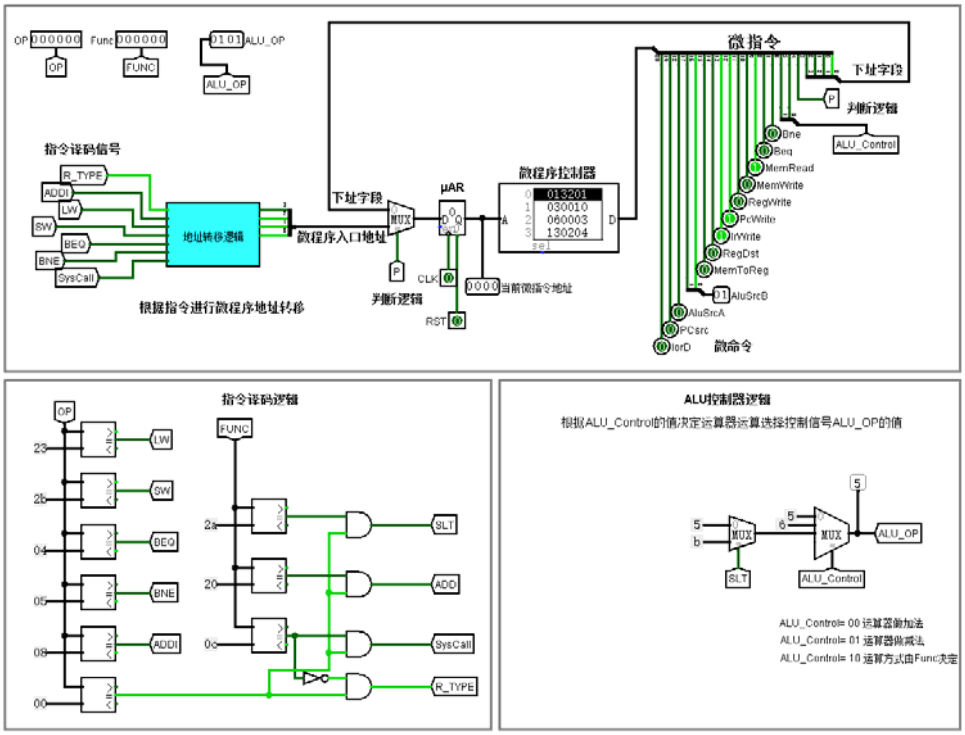


图 2-6 微程序控制器电路图

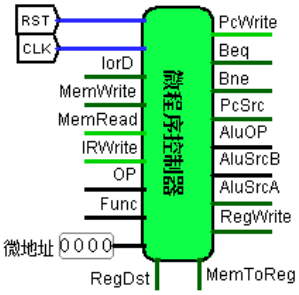


图 2-7 微程序控制器封装图

2.2.3 设计多周期 MIPS CPU（微程序）

将上一步实现的微程序控制器接入到 CPU 数据通路，控制信号和对应的控制端口连接，即可构建多周期 CPU。多周期 CPU 的设计图见图 2-8。

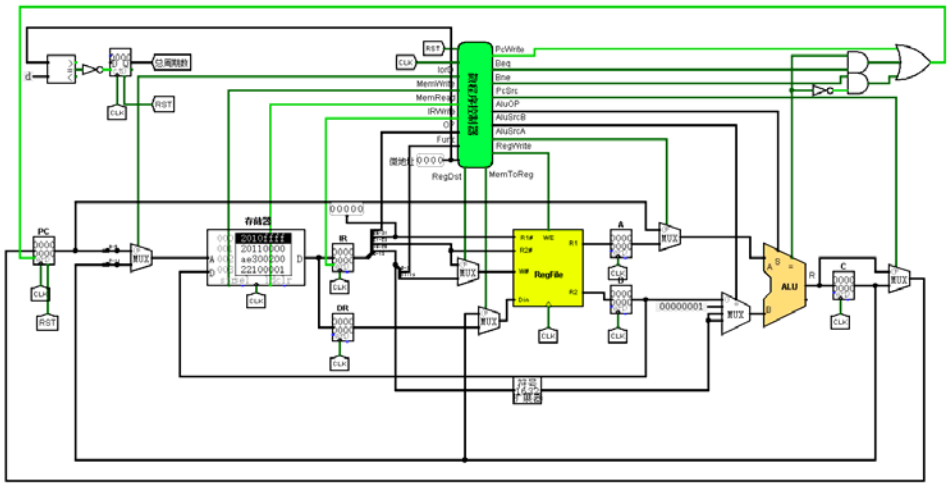


图 2-8 多周期 MIPS CPU(微程序)

2.2.4 设计硬布线控制器

（1） 硬布线控制器结构

硬布线控制器的原理图如图 2-9 所示，主要由指令译码电路、FSM 状态机、状态寄存器和硬布线控制器四部分组成。

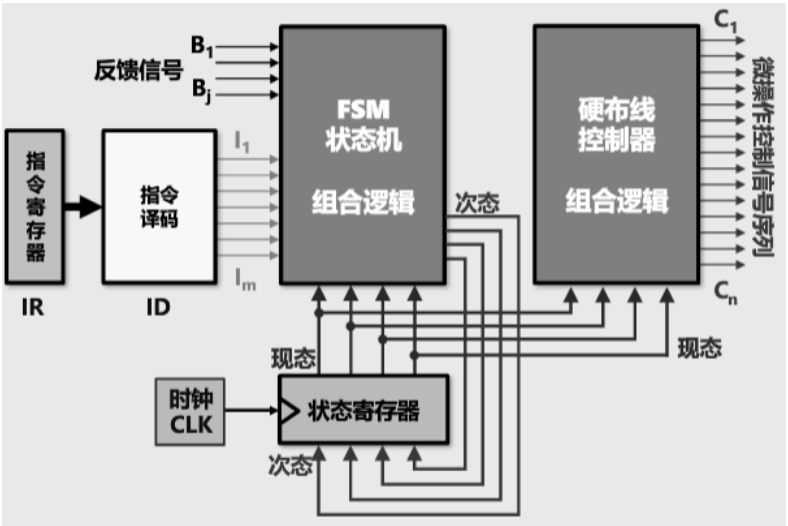


图 2-9 硬布线控制器原理图

（2） 设计硬布线控制器数据通路

与微程序控制器相比，硬布线控制器将每一条微指令对应一种状态，每种状态下给出不同的控制信号，利用状态机可以很容易实现。考虑到使用 Logism 设计状态机较为繁琐，这里硬布线控制器组合逻辑电路将使用微程序的控制存储器代替，其地址就是状态机状态的编号。设计的到硬布线控制器的数据通路如图 2-10 所示：

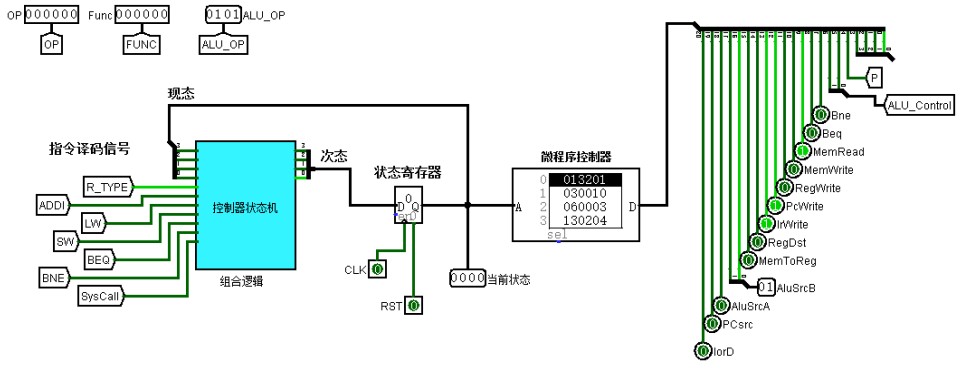


图 2-10 硬布线控制器数据通路电路图

（3） 设计硬布线控制器状态机

根据图 2-4 所示的状态转移图可以确定状态的转移逻辑，填写表 2-11 状态转移真值表，利用最小项表达式和 Logism 自动生成电路即可得到状态转移逻辑电路。

表 2-11 状态转移真值表

S3	S2	S1	S0	R_T	LW	SW	BEQ	BNE	SYS	ADD	其他	最小项表达式	N3	N2	N1	N0
~S3	~S2	~S1	~S0									~S3 ~S2 ~S1 ~S0 +	0	0	0	1
~S3	~S2	~S1	S0	_Type								~S3 ~S2 ~S1 S0 R_Type +	0	1	1	1
~S3	~S2	~S1	S0		LW							~S3 ~S2 ~S1 S0 LW +	0	0	1	0
~S3	~S2	~S1	S0			SW						~S3 ~S2 ~S1 S0 SW +	0	1	0	1
~S3	~S2	~S1	S0				BEQ					~S3 ~S2 ~S1 S0 BEQ +	1	0	0	1
~S3	~S2	~S1	S0					BNE				~S3 ~S2 ~S1 S0 BNE +	1	0	1	0
~S3	~S2	~S1	S0						YSCALL			~S3 ~S2 ~S1 S0 SYSCALL +	1	1	0	1
~S3	~S2	~S1	S0							ADDI		~S3 ~S2 ~S1 S0 ADDI +	1	0	1	1
~S3	~S2	S1	~S0									~S3 ~S2 S1 ~S0 +	0	0	1	1
~S3	~S2	S1	S0									~S3 ~S2 S1 S0 +	0	1	0	0
~S3	S2	~S1	~S0									~S3 S2 ~S1 ~S0 +	0	0	0	0
~S3	S2	~S1	S0									~S3 S2 ~S1 S0 +	0	1	1	0
~S3	S2	S1	~S0									~S3 S2 S1 ~S0 +	0	0	0	0
~S3	S2	S1	S0									~S3 S2 S1 S0 +	0	0	0	0
S3	~S2	~S1	~S0									S3 ~S2 ~S1 ~S0 +	0	0	0	0
S3	~S2	~S1	S0									S3 ~S2 ~S1 S0 +	0	0	0	0
S3	~S2	S1	~S0									S3 ~S2 S1 ~S0 +	0	0	0	0
S3	~S2	S1	S0									S3 ~S2 S1 S0 +	1	1	0	0
S3	S2	~S1	~S0									S3 S2 ~S1 ~S0 +	0	0	0	0
S3	S2	~S1	S0									S3 S2 ~S1 S0 +	1	1	0	1

（4） 设计硬布线控制器

硬布线控制器没有增加指令，其指令译码电路直接使用微程序控制器的指令译码电路。硬布线控制器的电路图电路图见图 2-11，为了方便接入到 CPU 数据通路中，将其封装成和微程序控制器一样的模块，封装图见图 2-12。

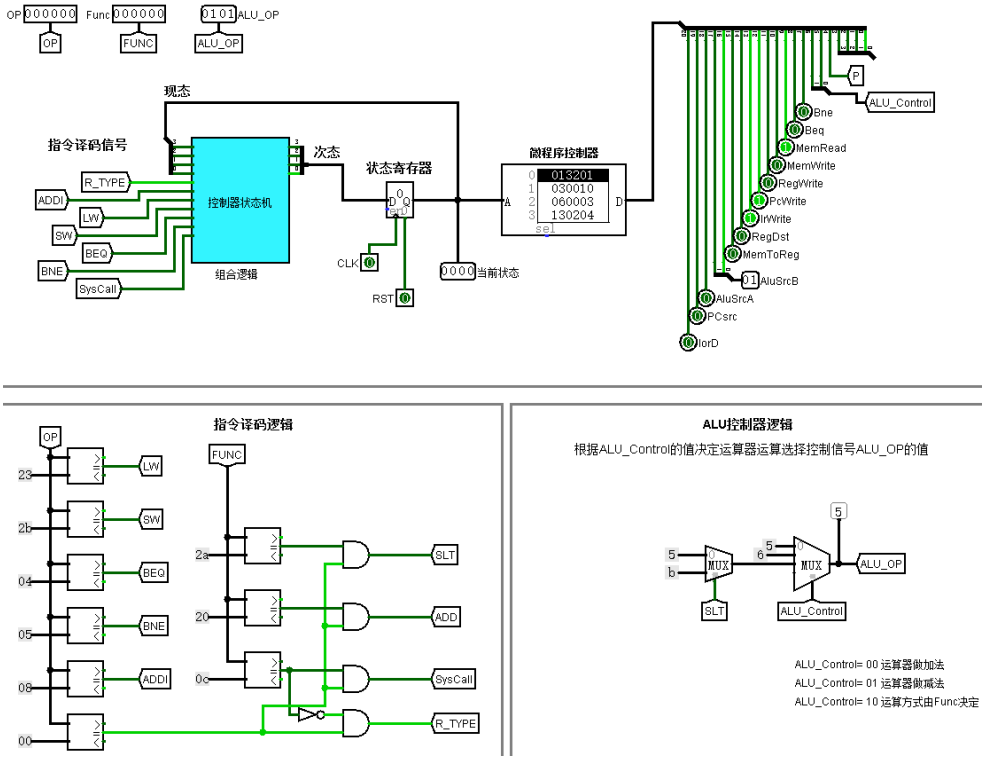


图 2-11 硬布线控制器电路图

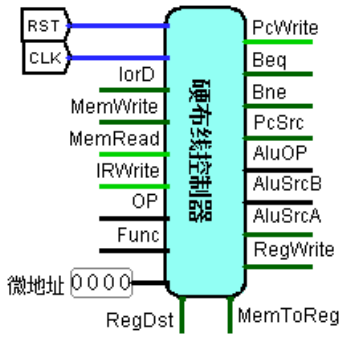


图 2-12 硬布线控制器封装图

2.2.5 设计多周期 MIPS CPU（硬布线）

由于控制器封装图完全一样，将硬布线控制器填换掉微程序控制器即可完成多周期 MIPS CPU（硬布线）的设计，电路图见图 2-13

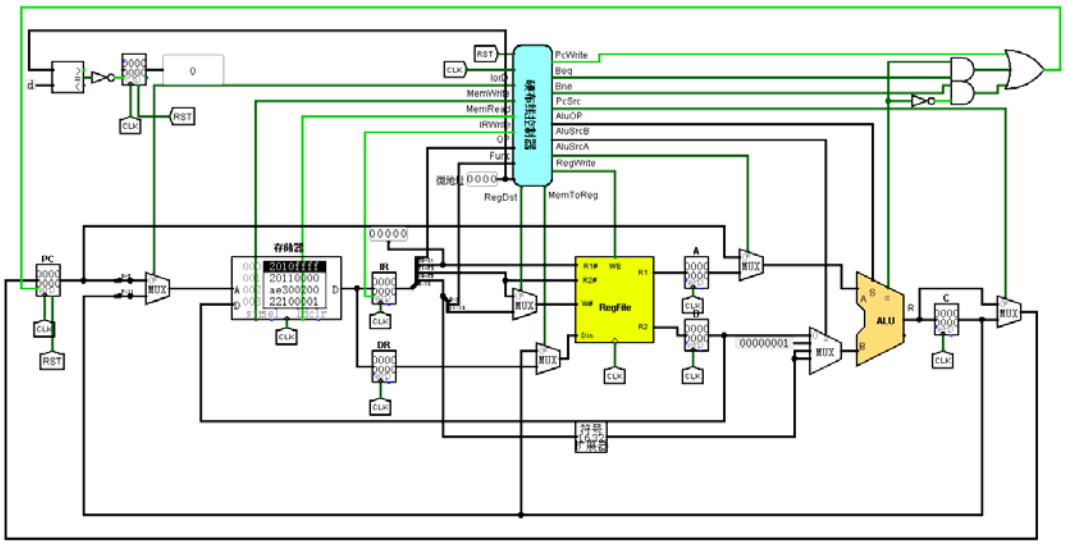


图 2-13 多周期 MIPS CPU(硬布线)

2.3 实验步骤

（1） 根据方案设计，实现多周期 MIPS CPU 的数据通路

（2） 设计微程序控制器

（3） 将封装好的微程序控制器连接到数据通路中，将控制信号接入到对应端口

（4） 加载测试程序，观察运行结果，对执行错误的地方单步调试查找错误原因，然后修改错误直到程序得到正确运行结果。

（5） 设计硬布线控制器

（6） 将微程序控制器替换为硬布线控制器

（7） 加载测试程序，观察运行结果，对执行错误的地方单步调试查找错误原因，然后修改错误直到程序得到正确运行结果。

2.4 故障与调试

2.4.1 AluOP 输出为高阻状态

故障现象：控制器的 AluOP 信号为高阻状态

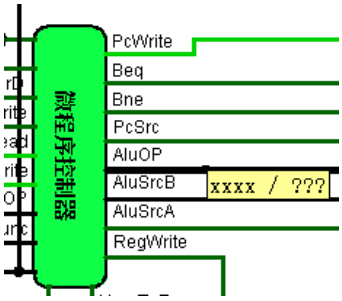


图 2-14 AluOP 输出

原因分析：根据错误现象分析 AluOP 未连接信号，进入微程序控制器检查发现是 AluOP 未连接到隧道的端口

解决方案：将 AluOP 连接到隧道端口，修正结果见图 2-15

图 2-15 AluOP 隧道连接

2.5 测试与分析

测试程序源代码和机器指令见 1.5。

2.5.1 测试结果分析

（1） 多周期 MIPS CPU（微程序）

如图 2-16 所示，程序的最后一条指令为 Syscall，运行完毕后 PC 会停在 027H 的位置。理论的运行周期数为 891，观察运行结果总周期数与预期符合。

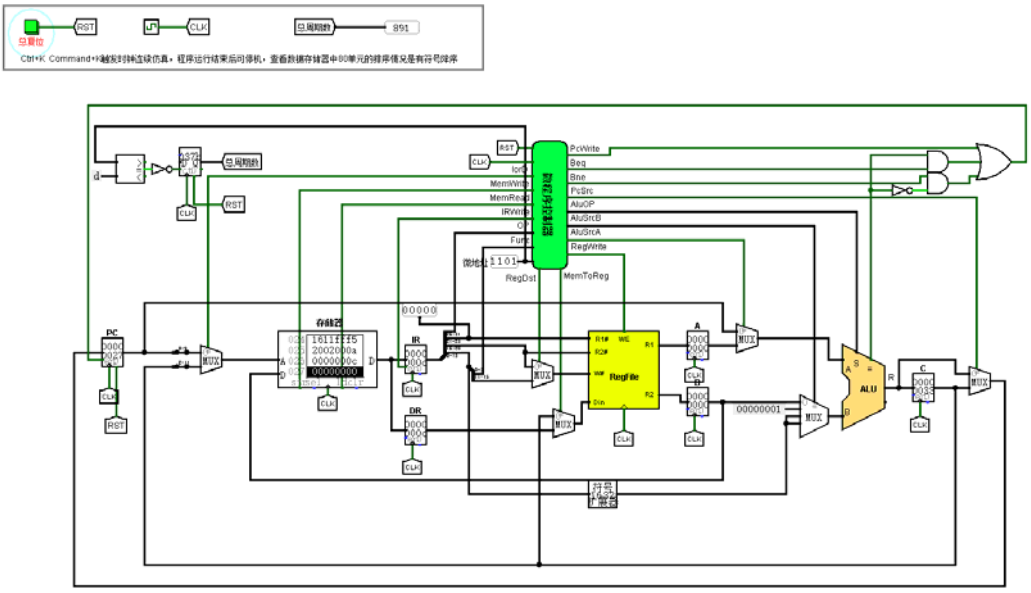


图 2-16 运行结果

如图 2-17，打开数据存储器，理论结果为运行后在 080H 的位置降序排列 6，5，4，3，2，1，0，-1，观察数据存储器可知测试结果与预期相符。

```
000  2010ffff 20110000 ae300200 22100001  22310004 ae300200 22100001 22310004
008  ae300200 22100001 22310004 ae300200  22100001 22310004 ae300200 22100001
010  22310004 ae300200 22100001 22310004  ae300200 22100001 22310004 ae300200
018  00008020 2011001c 8e130200 8e340200  0274402a 11000002 ae330200 ae140200
020  2231fffc 1611fff8 22100004 2011001c  1611fff5 2002000a 0000000c 00000000
028  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
030  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
038  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
040  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
048  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
050  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
058  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
068  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
070  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
078  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
080  00000006 00000005 00000004 00000003  00000002 00000001 00000000 ffffffff
```

图 2-17 执行后数据存储器的内容

（2） 多周期 MIPS CPU（硬布线）

如图 2-18 所示，程序的最后一条指令为 Syscall，运行完毕后 PC 会停在 027H 的位置。理论的运行周期数为 891，观察运行结果总周期数与预期符合。

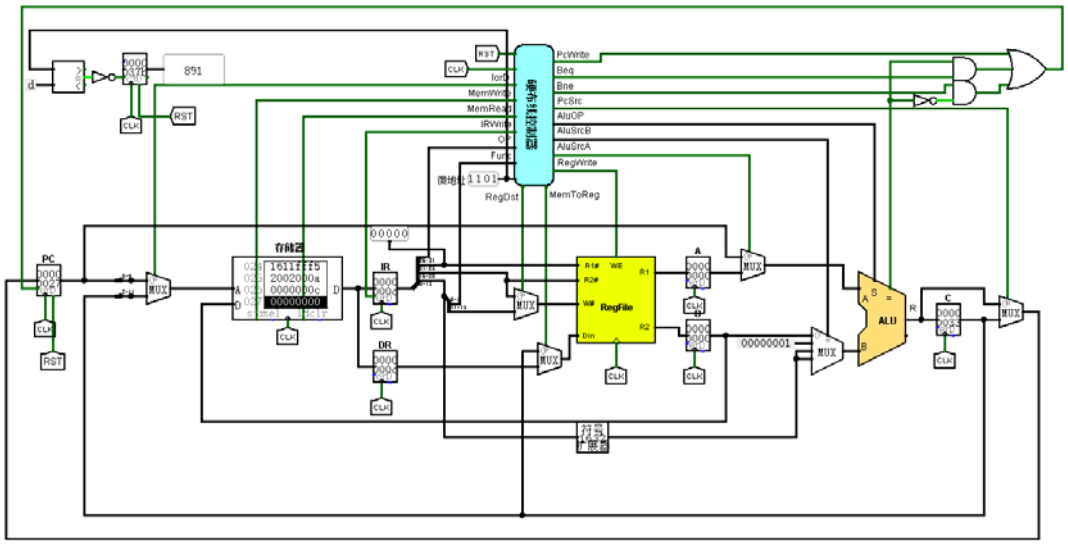


图 2-18 运行结果

如图 2-19，打开数据存储器，理论结果为运行后在 080H 的位置降序排列 6，5，4，3，2，1，0，-1，观察数据存储器可知测试结果与预期相符。

000	2010ffff	20110000	ae300200	22100001	22310004	ae300200	22100001	22310004
008	ae300200	22100001	22310004	ae300200	22100001	22310004	ae300200	22100001
010	22310004	ae300200	22100001	22310004	ae300200	22100001	22310004	ae300200
018	00008020	2011001c	8e130200	8e340200	0274402a	11000002	ae330200	ae140200
020	2231fffc	1611fff8	22100004	2011001c	1611fff5	2002000a	0000000c	00000000
028	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
030	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
038	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
040	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
048	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
050	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
058	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
060	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
068	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
070	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
078	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
080	00000006	00000005	00000004	00000003	00000002	00000001	00000000	ffffffff

图 2-19 执行后数据存储器的内容

3 总结与心得

3.1 实验总结

1) 完成方案总结：

设计并实现了单周期 MIPS CPU 数据通路、单周期 MIPS CPU 控制器、多周期 MIPS CPU 数据通路、多周期 MIPS CPU 微程序控制器、多周期 MIPS CPU 硬布线控制器。

2) 功能总结：

设计的单周期和多周期 MIPS CPU 可以执行 8 条核心 MIPS 指令，并通过排序测试程序。

3) 其他需要总结的内容：

理解了多周期 CPU 的设计和调试。

3.2 实验心得

1） 通过 ALU 运算器的实验我掌握了使用逻辑门执行算术运算的原理。

2） 通过数据传输实验我掌握了字模码显示的原理、CRC 编码的原理和流水传输机制的原理。

3） 通过 Cache 实验我掌握了高速缓存的原理及替代算法。

4） 通过寄存器文件实验我熟悉了 ROM、RAM 的使用和字拓展、位拓展的原理。

5） 通过 CPU 控制器译码电路的设计，深刻掌握了 MIPS 指令的指令格式与分类。

6） 通过多周期 CPU 各类指令数据通路的设计，深刻理解了 CPU 工作原理与进本实现，明白了 CPU 性能和架构的关系，极大提高了知识。

7） 通过 CPU 实验，深刻理解了 CPU 的组成结构和计算机的层次结构，掌握了 CPU 的功能模块划分，明白了汇编指令、机器指令和微指令之间的关系。

8） 实验最大的收获就是让我对课上的知识有了更加深刻的理解，让我明白在上完课以后往往没有真的学懂知识，只有通过做实验，主动去学习，才能加深自己的理解。

9） CPU 的实验我只完成了功能要求，在性能上还有改进空间。

10） 在实验过程中一定要勤保存实验文件，否则程序卡死时就欲哭无泪了。

11） 养成勤备份的习惯，这样在设计出错时可以回退到前一个步骤。

12） 实验建议：希望平衡一下各次实验的实验量，此外希望增加 Cache 的组相连和直接相连实验。

参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

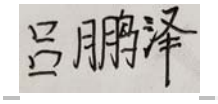
[5] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

[6] 谭志虎，秦磊华，胡迪清. 计算机组成原理实践教程——从逻辑门到 CPU. 清华大学出版社

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字： 

二、对课程实验的学术评语（教师填写）

三、对课程实验的评分（教师填写）

评分项目 （分值）	报告撰写 （30 分）	课设过程 （70 分）	最终评定 （100 分）
得分			

指导教师签字：_____