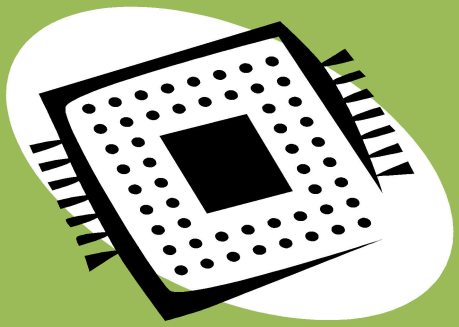
华中科技大学

2018

计算机组成原理 ·实验报告·

专　　业：	物联网工程
班　　级：	IOT1601
学　　号：	U201614897
姓　　名：	潘越
电　　话：	13995718033
邮　　件：	479773533@qq.com
完成日期：	2018-12-18



计算机科学与技术学院

目　录

1 CPU 设计实验

1.1 设计要求

（1）利用 Logisim 平台中现有运算部件以及康奈尔大学开发的寄存器文件，设计实现一个 MIPS 单周期 CPU。

（2）利用微程序控制器设计实现 MIPS 多周期 CPU。

（3）利用硬布线控制器设计实现 MIPS 多周期 CPU。

设计的 MIPS 单周期/多周期 CPU 需要支持的指令有如下 8 条，通过这 8 条指令可以实现内存区域冒泡排序的功能。

表 1.1 需要实现的 CPU 核心指令集描述

#	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]+SignExt16b(imm)　溢出产生异常
4	lw $rt,imm($rs)	R[$rt]←Mem4B(R[$rs]+SignExt16b(imm))
5	sw $rt,imm($rs)	Mem4B(R[$rs]+SignExt16b(imm))←R[$rt]
6	beq $rs,$rt,imm	if(R[$rs] = R[$rt]) PC ← PC + SignExt18b({imm, 00})
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + SignExt18b({imm, 00})
8	syscall	系统调用，这里用于停机

1.2 方案设计

1.2.1 MIPS 单周期 CPU 设计

（1）MISP 单周期 CPU 数据通路设计

参考所给实验材料，寄存器文件选用再次封装的版本，ALU 使用本实验框架自带的版本，设计 CPU 数据通路如下图：

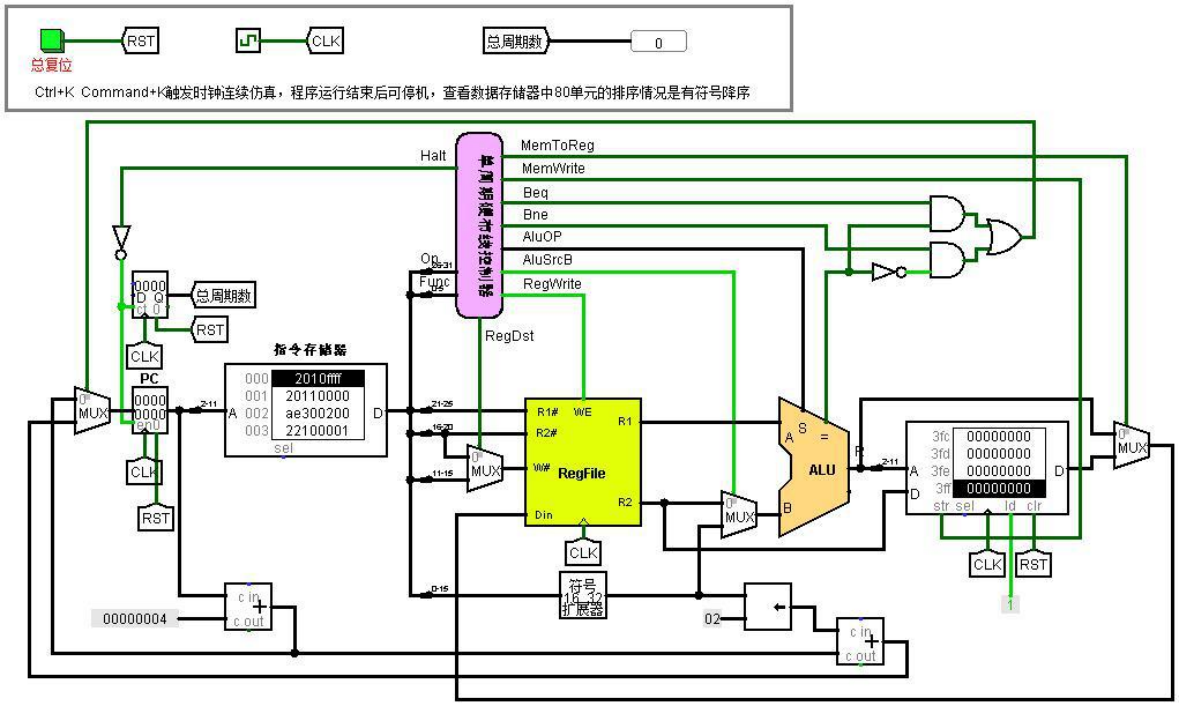


图 1.1 MIPS 单周期 CPU 数据通路图

其中，指令存储器存储要进行运算的指令序列，数据存储器存储排序后的结果，所有的控制信号由单周期硬布线控制器输出，停机信号控制指令计数器 PC 的使能端和周期计数器的使能端。

（2）硬布线控制器电路设计

硬布线控制器的设计如下图：

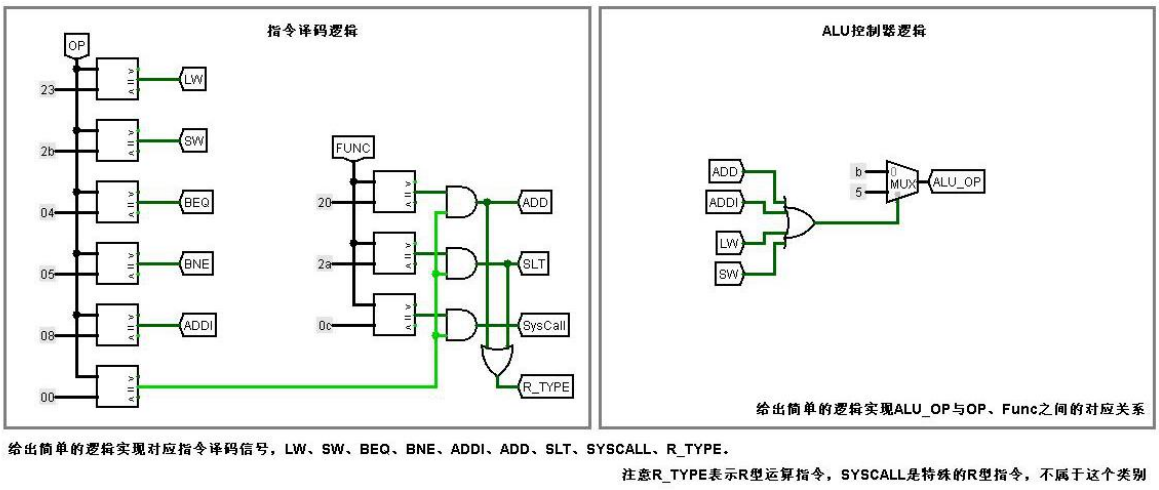


图 1.2 硬布线控制器电路设计

指令逻辑这里实现的比较简单，因为考虑到本实验只需要实现 8 条基本的指令，

所以直接使用了比较器进行比较，根据操作数 OP 的不同来判断是那种指令，如果是 R 型指令，还需要再根据 funct 字段的值来决定是哪种指令。

ALU 控制器逻辑根据这 8 条指令，只需要考虑两种情况，即相加或者判断是否相等，其中需要使用到 ALU 相加的参考表 1.1，共有四条指令，分别是 add、addi、lw 和 sw 四条指令，因此这个逻辑使用一个选择器即可实现。

（3）MIPS 单周期 CPU 指令信号分析

下面依次针对单周期 CPU 的数据通路，对要实现的这 8 条指令进行分析，以考虑控制器信号的设计。

① R 型指令：

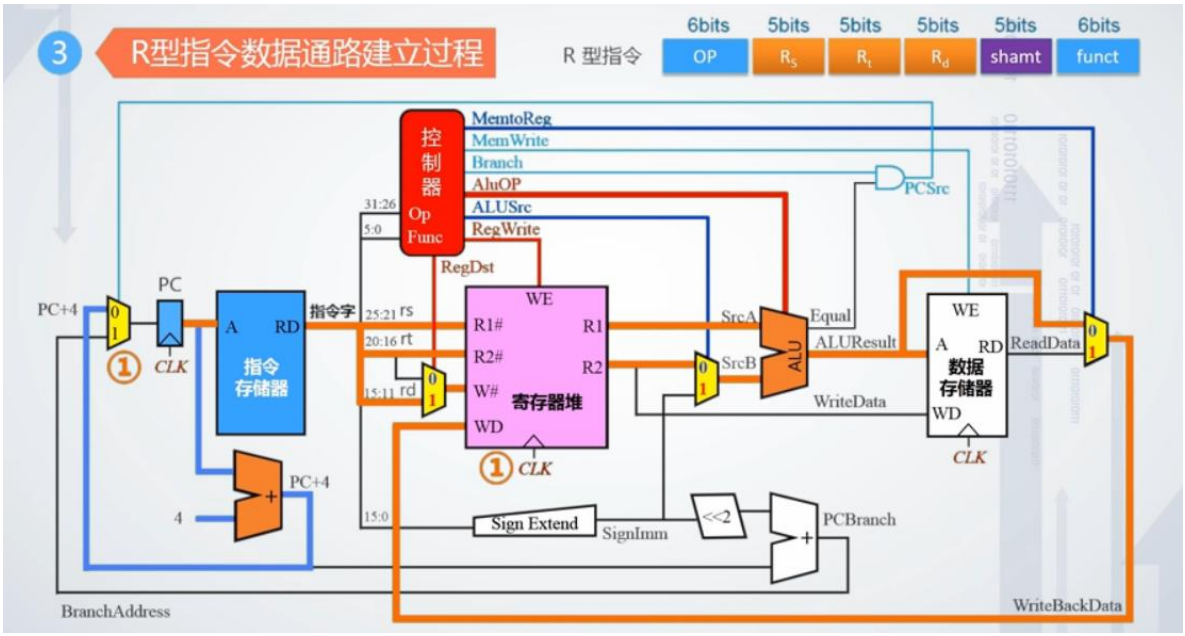


图 1.3 R 型指令数据通路建立过程

R 型指令的数据通路建立过程如图 1.3，其完整的步骤如下：

1.RegDst 信号为 1，将 R_s, R_t, R_d 分别送入寄存器堆中，数据从 R_1R_2 中输出；

2.AluSrcB 信号为 0，将 R_2 送入 ALU 的 SrcB 端，同时 AluOP 执行相应的运算；

3.MemtoReg 信号为 0，将 ALU 运算的结果输送到寄存器堆的 WD 端；

4.RegWrite 信号为 1，将 WD 端的结果写回寄存器堆。

同时在数据通路左侧，PC 自动加 4 以执行下一条指令。

② LW 指令：

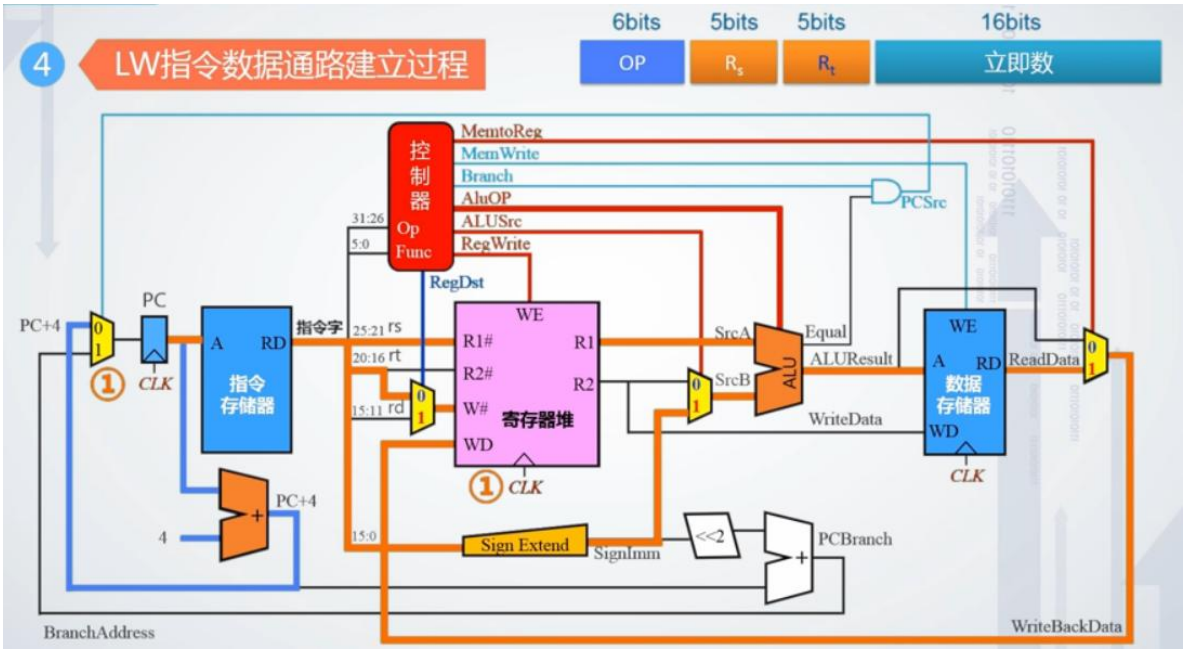


图 1.4 LW 指令数据通路建立过程

LW 指令的数据通路建立过程如图 1.4，其完整的步骤如下：

1.RegDst 为 0，将 R_t 送入寄存器堆的 W#端，代表要写入的寄存器号；

2.AluSrcB 为 1，将 0~15 位的立即数进行符号位扩展之后，和 R1 一起送到 ALU 的两端，ALU 执行相加操作。

3.MemtoReg 信号为 1，根据 ALU 运算的结果作为地址，从数据存储器中读出结果并送入寄存器堆的 WD 端。

4.RegWrite 信号为 1，将 WD 端的结果写回寄存器堆。

同时在数据通路左侧，PC 自动加 4 以执行下一条指令。

③ SW 指令

SW 指令的数据通路建立过程如图 1.5，和 LW 指令很像，其完整的步骤如下：

1.RegDst 信号随意，因为这里并不对寄存器堆写入；

2.AluSrcB 信号为 1，将 0~15 位的立即数进行符号位扩展之后，和 R1 一起送到 ALU 的两端，ALU 执行相加操作，同时将 R_2 的结果送到数据存储器的 WD 端。

3.MemWrite 信号为 1，将 ALU 的计算结果写入到地址为 R_2 的地方。

同时在数据通路左侧，PC 自动加 4 以执行下一条指令。

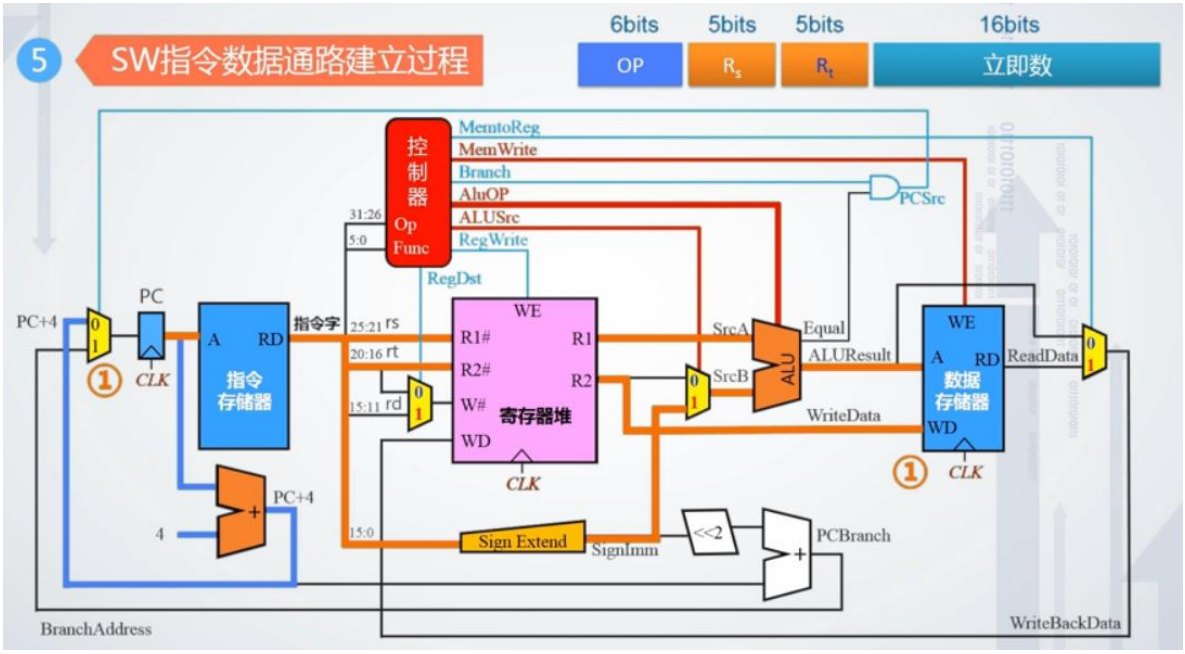


图 1.5 SW 指令数据通路建立过程

④ ADDI 指令

ADDI 指令数据通路建立过程类似 ADD 指令，只不过是和立即数相加，而且 I 型指令和 R 型指令有所区别，简要描述过程如下：

1.RegDst 信号为 1，因为最后要写入 R_t，将 R_t 送入 W#；

2.AluSrcB 信号为 1，将 0~15 位的立即数进行符号位扩展之后，和 R1 一起送到 ALU 的两端，ALU 执行相加操作；

3.MemtoReg 信号为 0，将 ALU 运算的结果输送到寄存器堆的 WD 端；

4.RegWrite 信号为 1，将 WD 端的结果写回寄存器堆。

同时在数据通路左侧，PC 自动加 4 以执行下一条指令。

⑤ 跳转指令

跳转指令这里只有两条，即 beq 和 bne 均为跳转指令，其数据通路建立过程如图 1.6 所示。

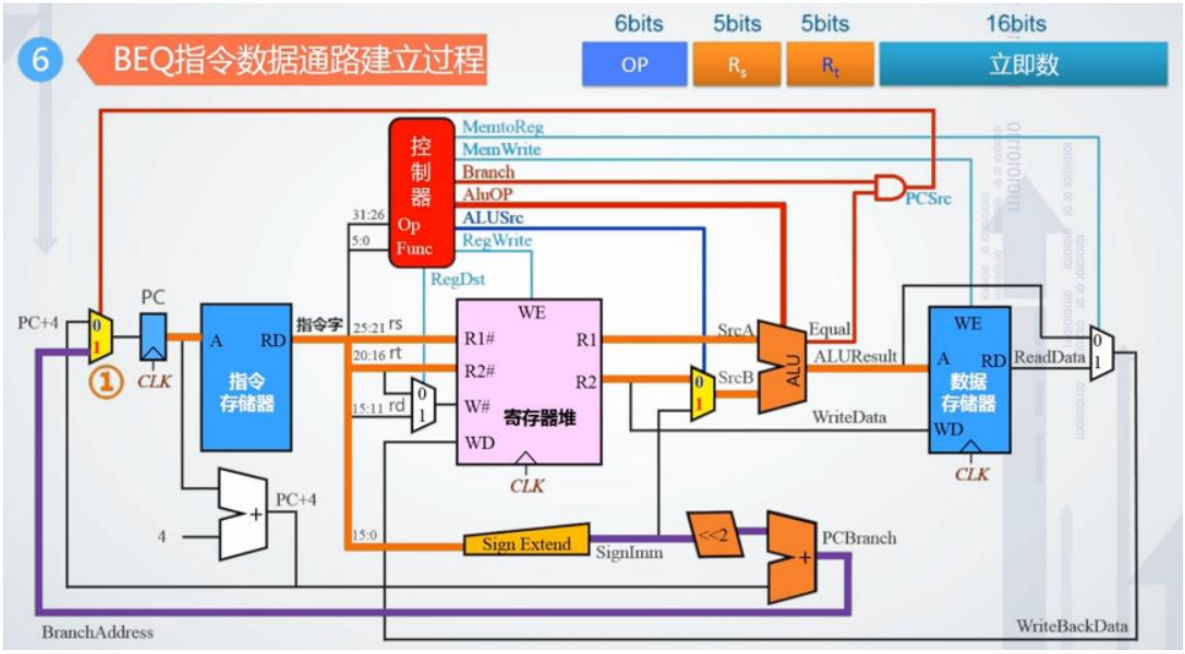


图 1.6 跳转指令数据通路建立过程

上图中只有 Branch 信号表示跳转，我们这里设计的 CPU 中，beq 和 bne 指令是分开的，分别表示相等时跳转和不相等时跳转，其步骤如下：

1.RegDst 信号随意，因为不涉及写入。

2.AluSrcB 信号为 0，将 R_2 送入 ALU 的 SrcB 端，同时 AluOP 执行判断是否相等的操作，Equal 为相等信号输出。

3.根据信号判断是否跳转：当 beq 和 Equal 同时为 1 时或者 bne 为 1，Equal 为 0 时跳转，将 0~15 位的立即数进行符号扩展并左移两位后的结果送入指令计数器 PC 中，实现跳转操作。

⑥ Syscall 指令

停机指令，在译码之后，直接产生一个信号通入指令计数器 PC 的使能端即可。

（4）指令译码信号电路设计

综合（3）中提到的指令信号操作，分析各个信号产生的条件如下表：

表 1.2 控制器各个信号产生的条件

#	控制器信号	产生条件
1	RegDst	R 型指令
2	RegWrite	R 型指令或 ADDI 指令或 LW 指令
3	MemToReg	LW 指令
4	MemWrite	SW 指令
5	AluSrc	ADDI 指令或 LW 指令或 SW 指令
6	Beq	BEQ 指令
7	Bne	BNE 指令
8	Halt	SYSCALL 指令

根据上表，设计译码电路如下：

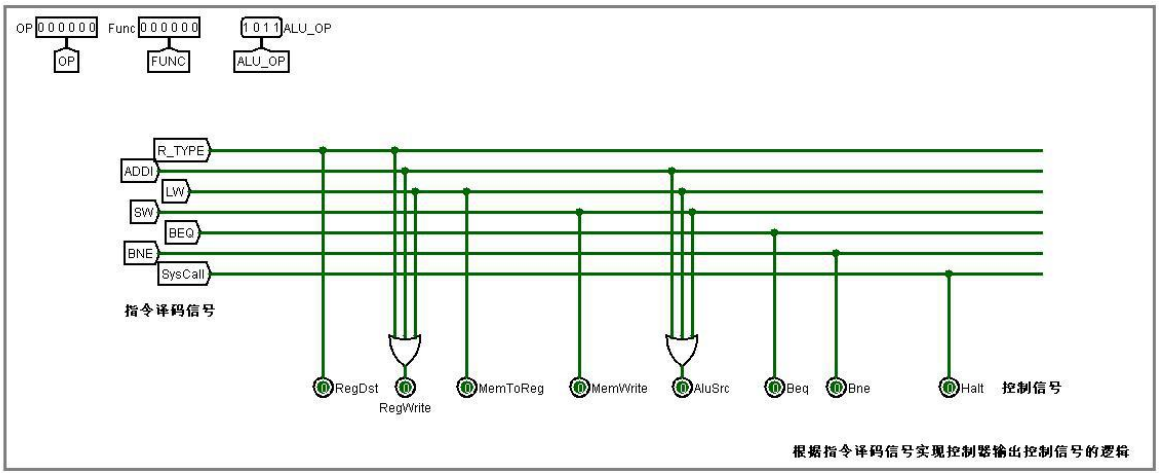


图 1.7 指令译码信号电路设计

至此，这部分的设计完成，可以进行 Logisim 程序的构建。

1.2.2 基于微程序控制器的 MIPS 多周期 CPU 设计

（1）MIPS 多周期 CPU 数据通路设计

参考所给实验材料，寄存器文件选用再次封装的版本，ALU 使用本实验框架自带的版本，设计 CPU 数据通路如下图：

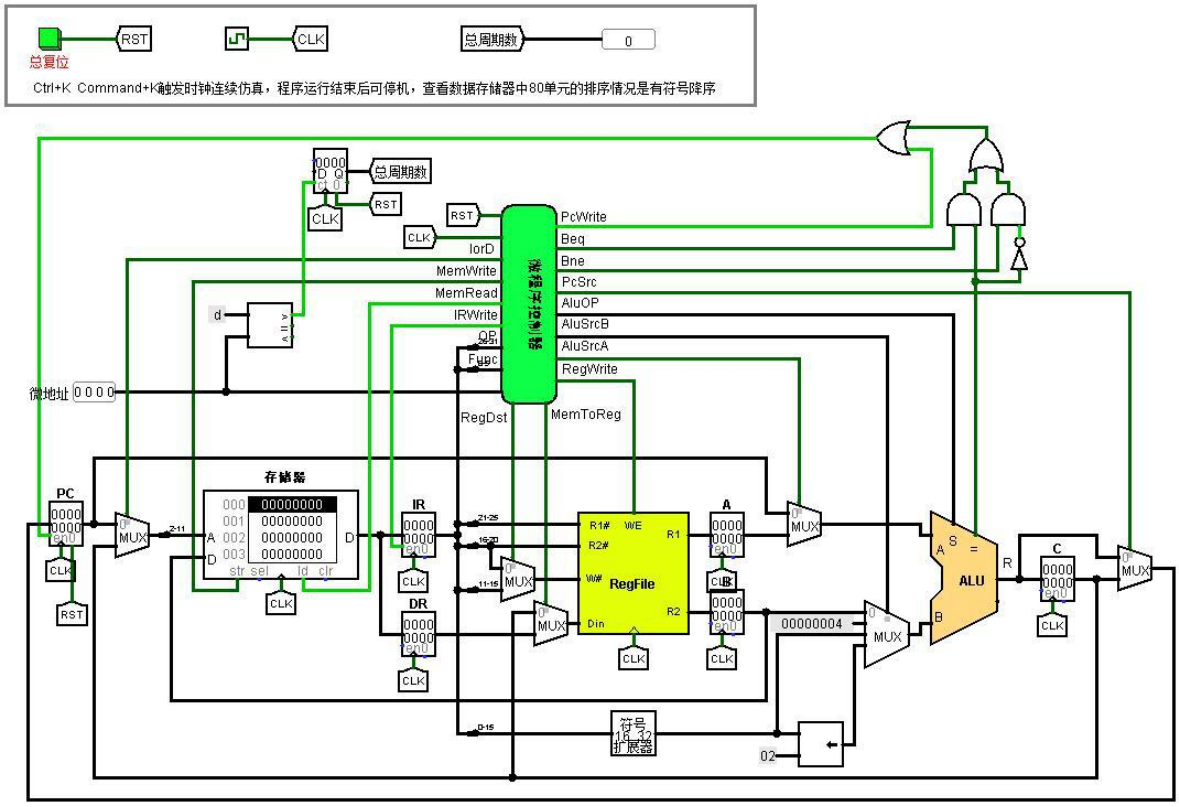


图 1.8 MIPS 多周期 CPU 数据通路图

（2）指令译码和 ALU 控制信号设计

指令译码部分的逻辑和章节 1.2.1 中提到的 MIPS 单周期 CPU 的一致，仍然使用比较器实现，主要的区别在于 ALU 控制信号的设计，如下图所示：

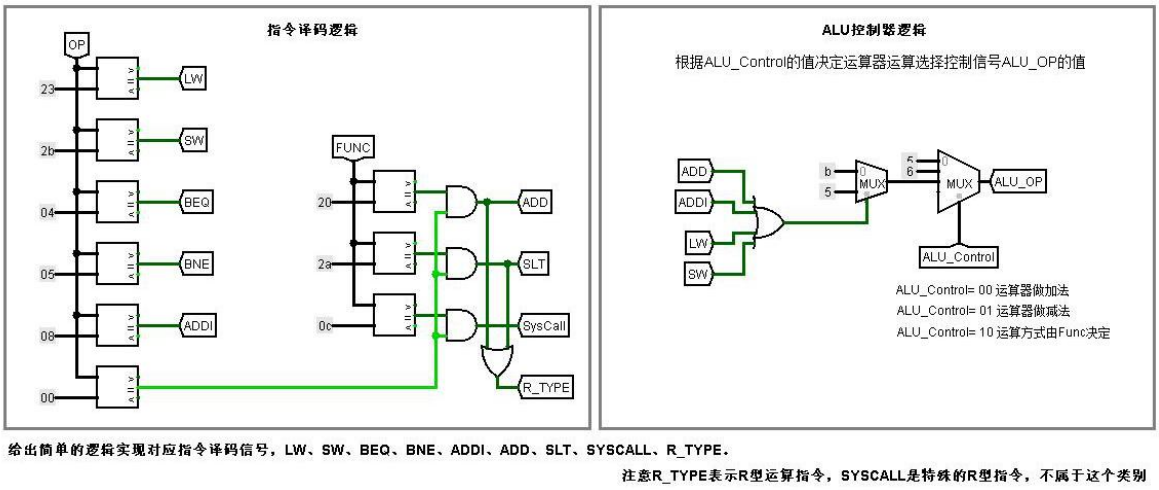


图 1.9 指令译码和 ALU 控制信号设计

ALU_OP 根据微程序中的 ALU_Control 位生成，ALU_Control=00 时，运算器

做加法，ALU_Control=01 时，运算器做减法，ALU_Control=10 时运算方式由 funct 字段决定，因此可以考虑使用两个选择器实现。

（3）指令状态图构建

MIPS 多周期 CPU 中，根据给出的要求实现的 8 条指令，设计如下的状态变换图：

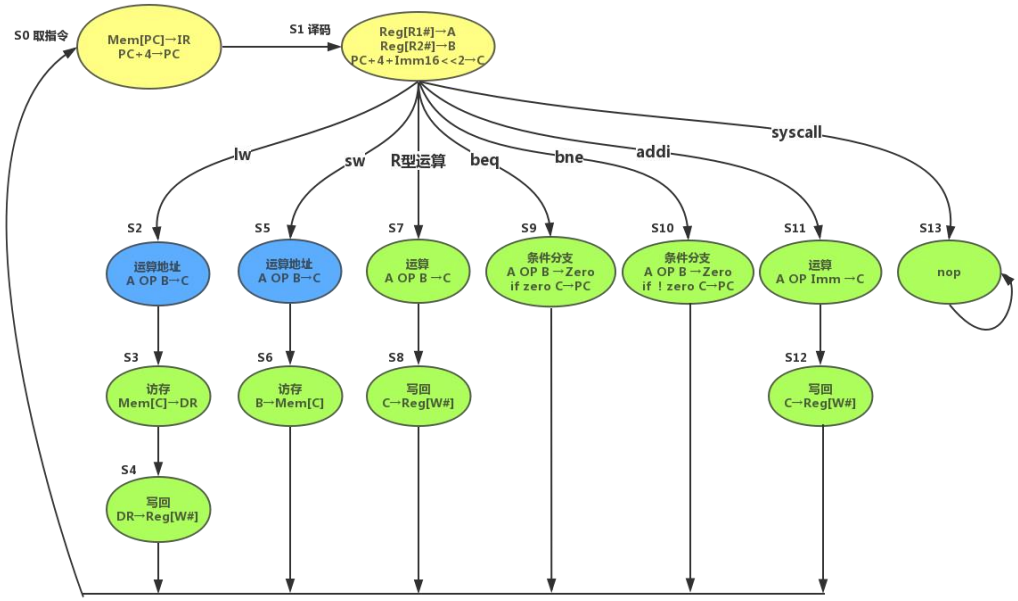


图 1.10 指令状态图

每个指令的执行需要经历如上图所示的几个状态，其中每个状态对应一个微程序，我们只需要合理设计微程序，并设计好微程序的地址转移逻辑，即可实现多周期 CPU 控制器，达到比单周期 CPU 跟高的效率。

（4）MIPS 多周期 CPU 指令信号分析

下面依次针对单周期 CPU 的数据通路，对要实现的这 8 条指令进行分析，以考虑控制器信号的设计。

① 取指令

取指令的数据建立过程如图 1.11，共需要操作，Mem[PC]->IR 和 PC+4->PC，其步骤如下：

1.AluSrcA 信号为 0，选择 PC 的值；

2.AluSrcB 信号为 01，选择 4 和 PC 的值一起送到 ALU 两端，ALU 执行相加操作。

3.PCSrc 信号为 0，将 PC+4 后的值送入寄存器；

4.MemRead 信号为 1，从存储器中读取指令；

5.IRWrite 信号为 1，将指令送入指令寄存器 IR。

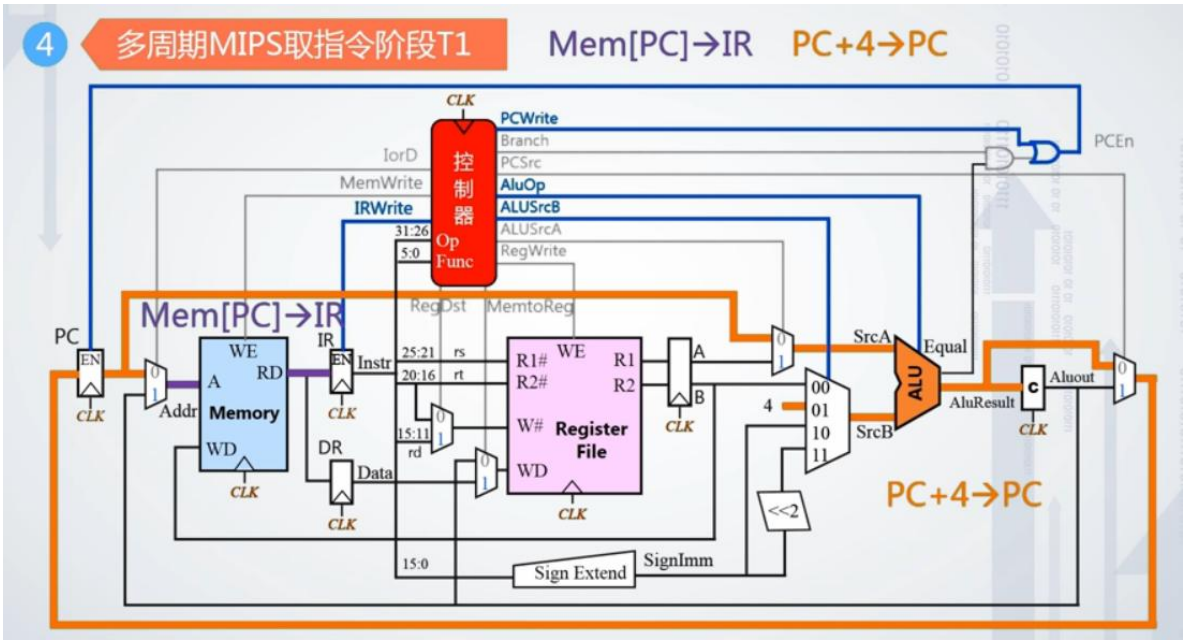


图 1.11 取指令的数据通路建立过程

② 译码

译码部分是取指令之后的第二个操作，如图 1.12 所示：

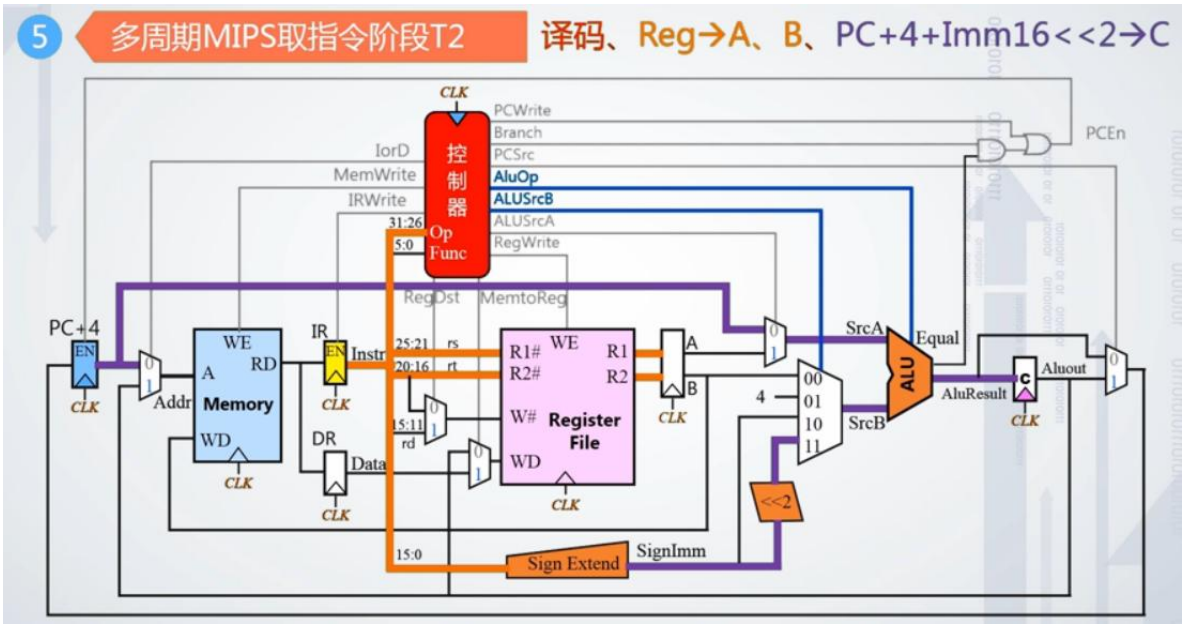


图 1.12 译码的数据通路建立过程

其步骤如下：

1.根据 R_s 和 R_t 从寄存器文件中获取数据 A, B 送入数据寄存器。

2.AluSrcA 信号为 0，AluSrcB 信号为 11，将 PC 的值和 0~15 位的立即数的值经符号位扩展再左移 2 位后的值送入 ALU 两端，ALU 执行相加操作，并把结果送入跳转寄存器中备用。

译码之后，就可以根据指令的种类跳转到相应的微程序上了。

③ LW 指令

LW 指令的数据通路建立过程有三个阶段，如图 1.13 所示，下面依次叙述每个阶段的具体步骤：

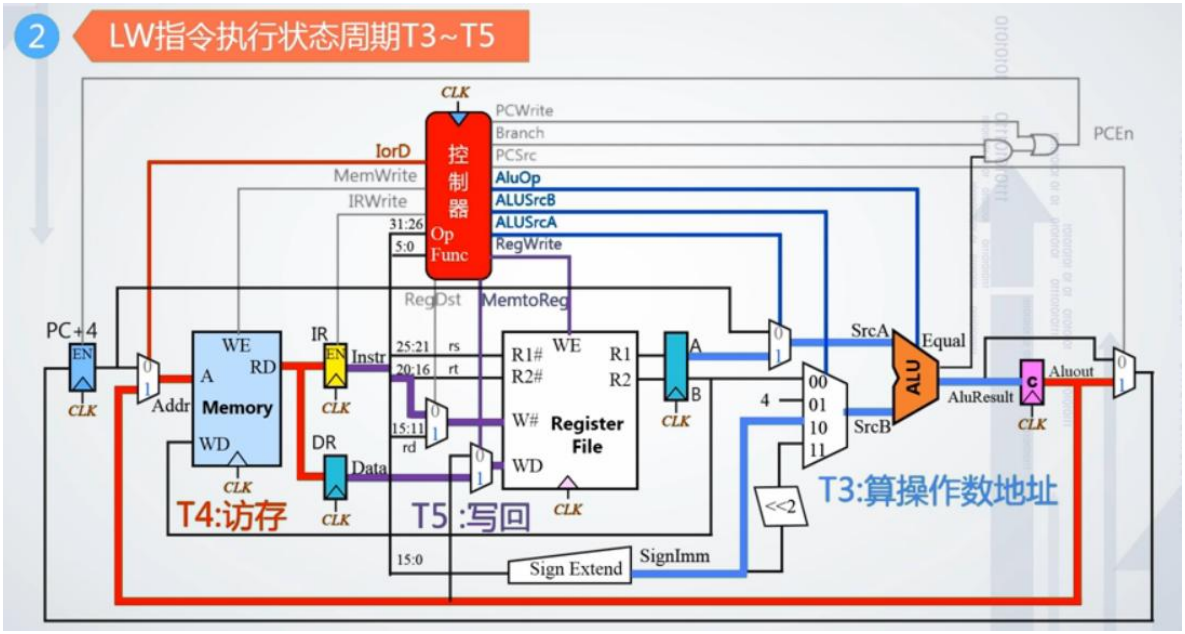


图 1.13 LW 指令的数据通路建立过程

下面依次叙述三个阶段的操作：

1.AluSrcA 信号为 1，AluSrcB 信号为 10，将 R1 的值和和 0~15 位立即数经符号位扩展之后的值送入 ALU 两端，ALU 执行相加操作，把操作数地址存入地址寄存器中；

2.IorD 信号为 1，表明是数据而非指令，MemRead 信号为 1，从存储器中读出数据送入数据寄存器 DR 中。

3.RegDst 信号为 0，MemtoReg 信号为 1，RegWrite 信号为 1，将 DR 中的数据

写回到 R_t 寄存器中。

④ SW 指令

SW 指令和 LW 指令相似，这里不再列出图示，只简要叙述一下步骤。SW 指令分为两个阶段：

1. AluSrcA 信号为 1，AluSrcB 信号为 10，将 R1 的值和和 0~15 位立即数经符号位扩展之后的值送入 ALU 两端，ALU 执行相加操作，把操作数地址存入地址寄存器中；

2.IorD 信号为 1，表明是数据而非指令，MemWrite 信号为 1，将数据写入存储器。

⑤ R 型指令

R 型指令的数据通路建立过程有两个阶段，如图 1.14 所示。

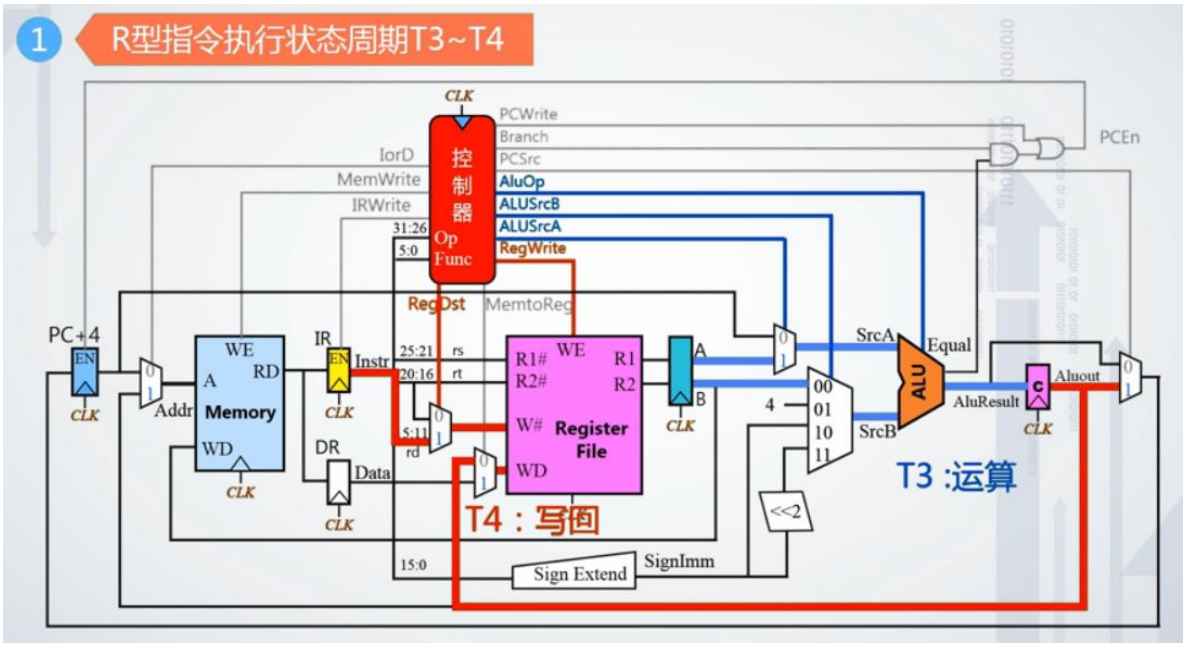


图 1.14 R 型指令的数据通路建立过程

R 型指令的两个阶段如下：

1.AluSrcA 信号为 1，AluSrcB 信号为 00，将 R1 和 R2 送入 ALU 的两端，ALU 执行相加操作，将结果送入寄存器 c 中；

2.RegDst 信号为 1，MemtoReg 信号为 0，RegWrite 信号为 1，将寄存器 c 中的

值写回 R_d 的位置。

⑥ ADDI 指令

ADDI 指令也是运算指令，和 R 型运算类似，只不过区别是对立即数相加，因此，其需要两个阶段，如下：

1. AluSrcA 信号为 1，AluSrcB 信号为 10，将 R1 的值和和 0~15 位立即数经符号位扩展之后的值送入 ALU 两端，ALU 执行相加操作，把结果送入寄存器 c 中；

2. RegDst 信号为 0，MemtoReg 信号为 0，RegWrite 信号为 1，将寄存器 c 中的值写回 R_t 的位置。

⑦ 跳转指令

跳转指令的数据通路的建立如图 1.15 所示，只需要一个阶段。

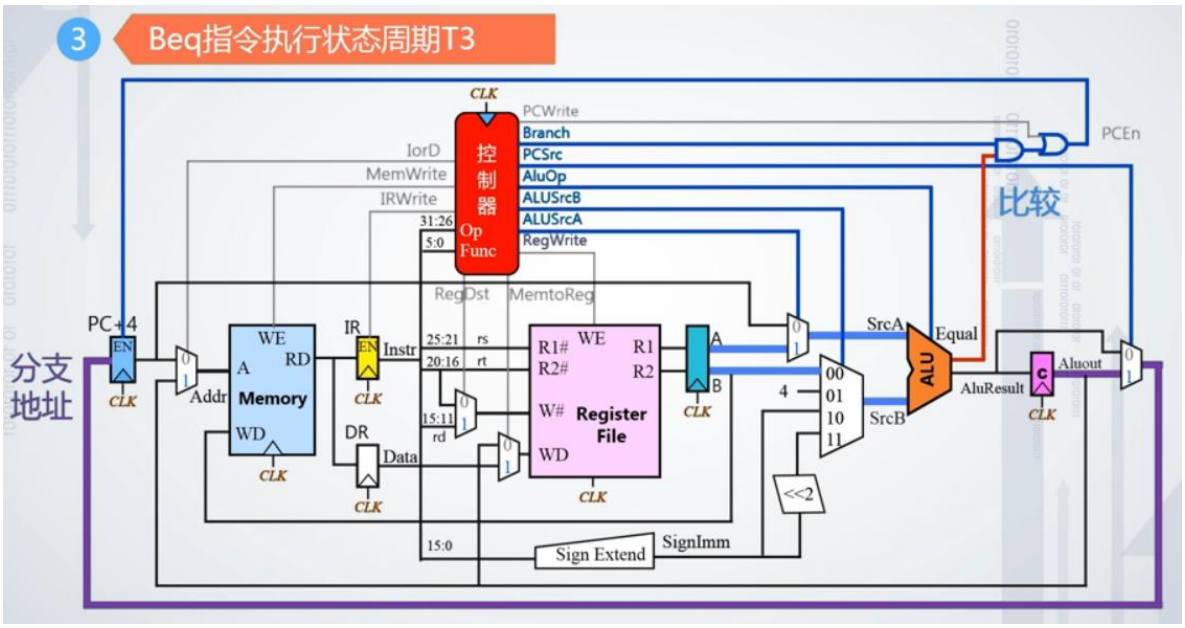


图 1.15 跳转指令的数据通路建立过程

跳转指令只需要让 AluSrcA 信号为 1，AluSrcB 信号为 00，将 R1 和 R2 的值送入 ALU 两端，ALU 进行比较操作，根据信号判断是否跳转：当 beq 和 Equal 同时为 1 时或者 bne 为 1，Equal 为 0 时跳转，将寄存器 c 中的值送入送入指令寄存器 PC 中，完成跳转。

（5）微程序地址转移逻辑设计

微程序的地址转移逻辑比较简单，我们将所有的指令以及下址设计如下表：

表 1.3 控制器各个信号产生的条件

地址	微程序	下址
0000	取指令	0001
0001	译码	0000
0010	LW1	0011
0011	LW2	0100
0100	LW3	0000
0101	SW1	0110
0110	SW2	0000
0111	R 型运算 1	1000
1000	R 型运算 2	0000
1001	Beq	0000
1010	Bne	0000
1011	ADDI1	1100
1100	ADDI2	0000
1101	SYSCALL	1101

至此，这部分的设计完成，可以进行 Logisim 程序的构建。

1.2.3 基于硬布线控制器的 MIPS 多周期 CPU 设计

MIPS 多周期 CPU 数据通路设计、指令译码和 ALU 控制信号设计以及指令状态变换图构建和章节 1.2.2 中的完全一致，这里不再赘述，仅针对不同的模块进行设计。

（1）状态转移逻辑设计

硬布线控制器的设计原理如下图：

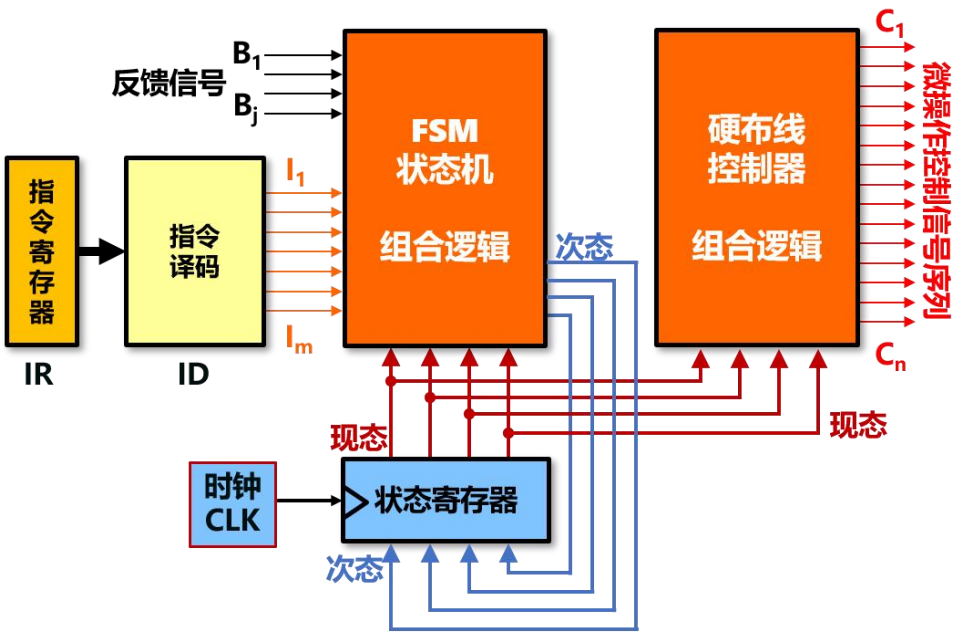


图 1.16 硬布线控制器的设计原理

硬布线控制器完全是由组合逻辑实现，通过设计 FSM 状态机的形式，实现在图 1.10 中的 14 个状态之间的转换，从而完成多周期 CPU，设计的关键是要设计状态之间的次态真值表，有了次态真值表之后，我们就可以根据它进行数字逻辑电路的设计。

根据图 1.10，给出次态真值表如下所示：

表 1.4 控制器状态转换的次态真值表条件

现态(10 进制)	R_Type	LW	SW	BEQ	BNE	SYSCALL	ADDI	次态(10 进制)
0	X	X	X	X	X	X	X	1
1	1	X	X	X	X	X	X	7
1	X	1	X	X	X	X	X	2
1	X	X	1	X	X	X	X	5
1	X	X	X	1	X	X	X	9
1	X	X	X	X	1	X	X	10
1	X	X	X	X	X	1	X	13
1	X	X	X	X	X	X	1	11
2	X	X	X	X	X	X	X	3

续表 1-4：控制器状态转换的次态真值表条件

现态(10 进制)	R_Type	LW	SW	BEQ	BNE	SYSCALL	ADDI	次态(10 进制)
3	X	X	X	X	X	X	X	4
4	X	X	X	X	X	X	X	0
5	X	X	X	X	X	X	X	6
6	X	X	X	X	X	X	X	0
7	X	X	X	X	X	X	X	8
8	X	X	X	X	X	X	X	0
9	X	X	X	X	X	X	X	0
10	X	X	X	X	X	X	X	0
11	X	X	X	X	X	X	X	12
12	X	X	X	X	X	X	X	0
13	X	X	X	X	X	X	X	13

根据次态真值表，我们就可以利用 Logisim 自动生成电路，从而完成程序的构建。

1.3 实验步骤

1.3.1 MIPS 单周期 CPU 设计

（1） 数据通路构建

利用 Logisim 构建程序，数据通路最终效果图如下：

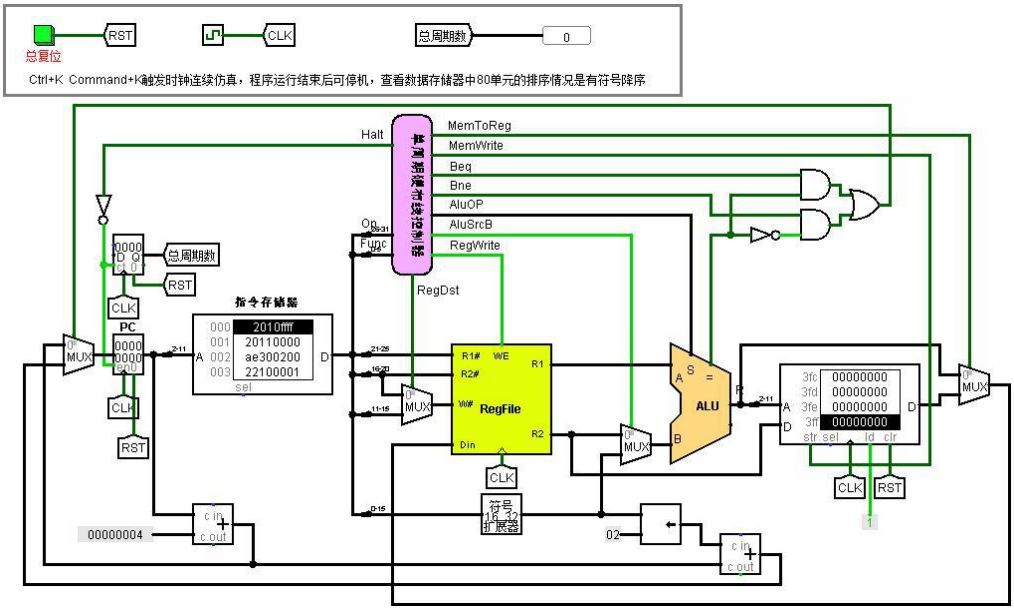


图 1.17 单周期 MIPS（硬布线）

（2） 硬布线控制器构建

利用 Logisim 构建程序，控制器最终效果图如下：

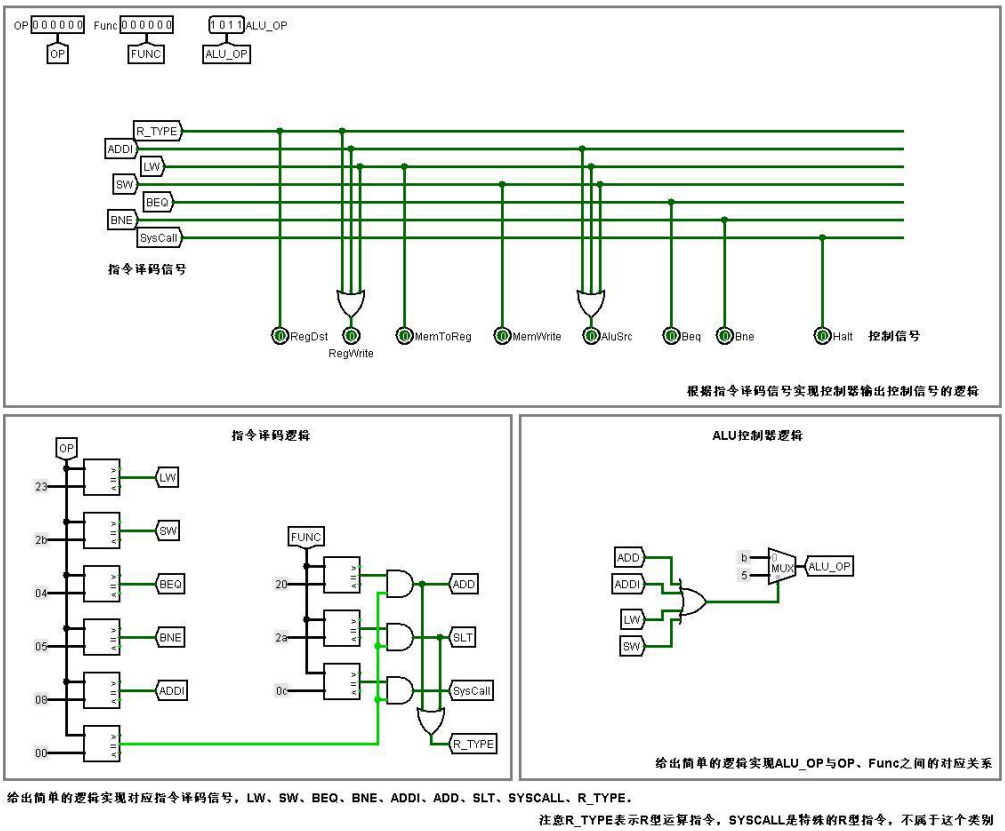


图 1.18 单周期硬布线控制器

1.3.2 基于微程序控制器的 MIPS 多周期 CPU 设计

（1） 数据通路构建

利用 Logisim 构建程序，数据通路最终效果图如下：

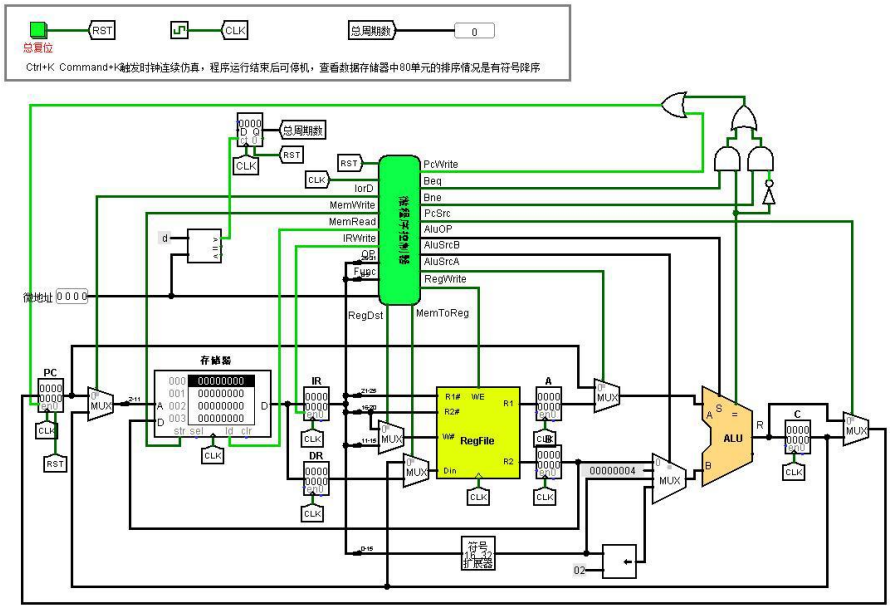


图 1.19 多周期 MIPS

（2） 指令译码和 ALU 控制信号实现

利用 Logisim 构建程序，多周期微程序控制器最终效果图如下：

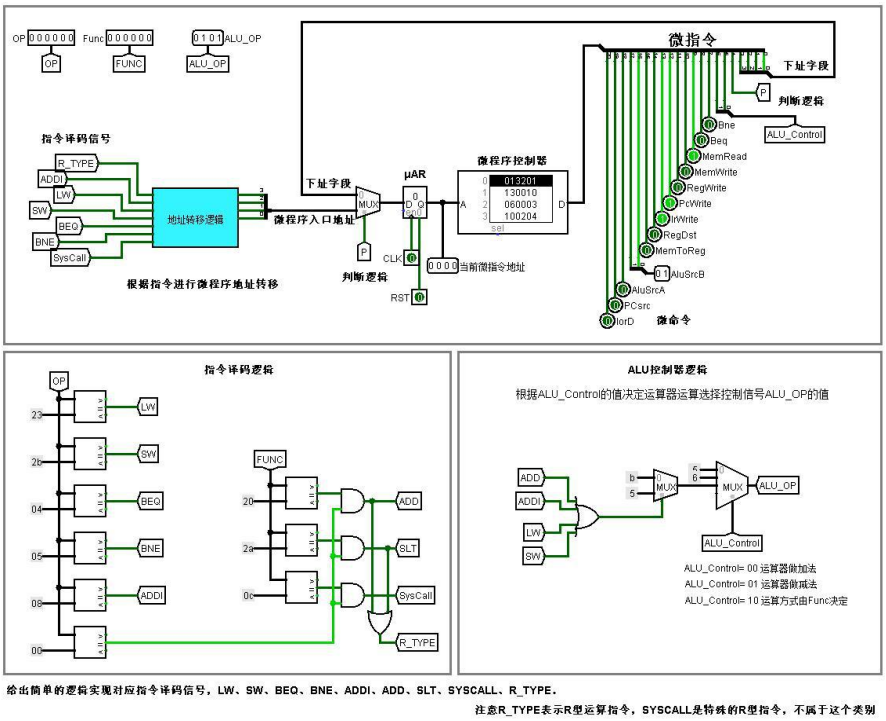


图 1.20 多周期微程序控制器

（3） 微程序实现

微程序的实现利用所给的文件“微指令自动生成.xlsx”生成，填写表格如下图：

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	00001001100100000001	13201
译码	1	0001	1	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000	10011000000000010000	130010
LW1	2	0010	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011	00110000000000000011	60003
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00	0	0100	10000000001000000100	100204
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00	0	0000	00000100010000000000	8800
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110	00110000000000000110	60006
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00	0	0000	10000000001000000000	100400
R型运算1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10	0	1000	00100000000001001000	40048
R型运算2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	00	0	0000	00000100100000000000	4800
Beq	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	10	0	0000	01100000000101000000	C0140
Bne	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	10	0	0000	01100000000011000000	C00C0
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	1100	00110000000000001100	6000C
ADDI2	12	1100	0	0	0	00	0	0	0	0	1	0	0	0	0	00	0	0000	00000000100000000000	800
SYSCALL	13	1101	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	1101	00000000000000001101	D
																			微指令长度	
																			21	

图 1.21 微程序设计

将图中的最右侧一列的十六进制值复制到微程序控制器中即可。

（4） 地址转移电路实现

地址转移逻辑利用所给的文件“微程序地址转移逻辑真值表.xlsx”生成，填写表格如下图：

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

图 1.22 微程序设计

通过手动筛选 S0~S3 全为 1 的情形，生成这四个变量的逻辑表达式，然后利用 Logisim 的分析电路功能即可自动生成地址转移电路如下图：

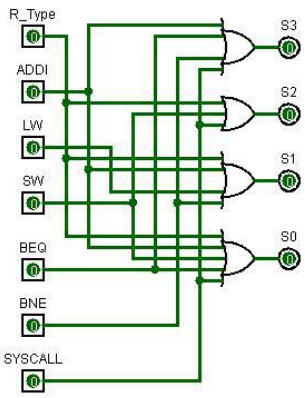


图 1.23 微程序地址转移逻辑

1.3.3 基于硬布线控制器的 MIPS 多周期 CPU 设计

（1） 数据通路构建

数据通路同图 1.17 所示的多周期 MIPS。

（2） 指令译码和 ALU 控制信号实现

利用 Logisim 构建程序，多周期硬布线控制器最终效果图如下：

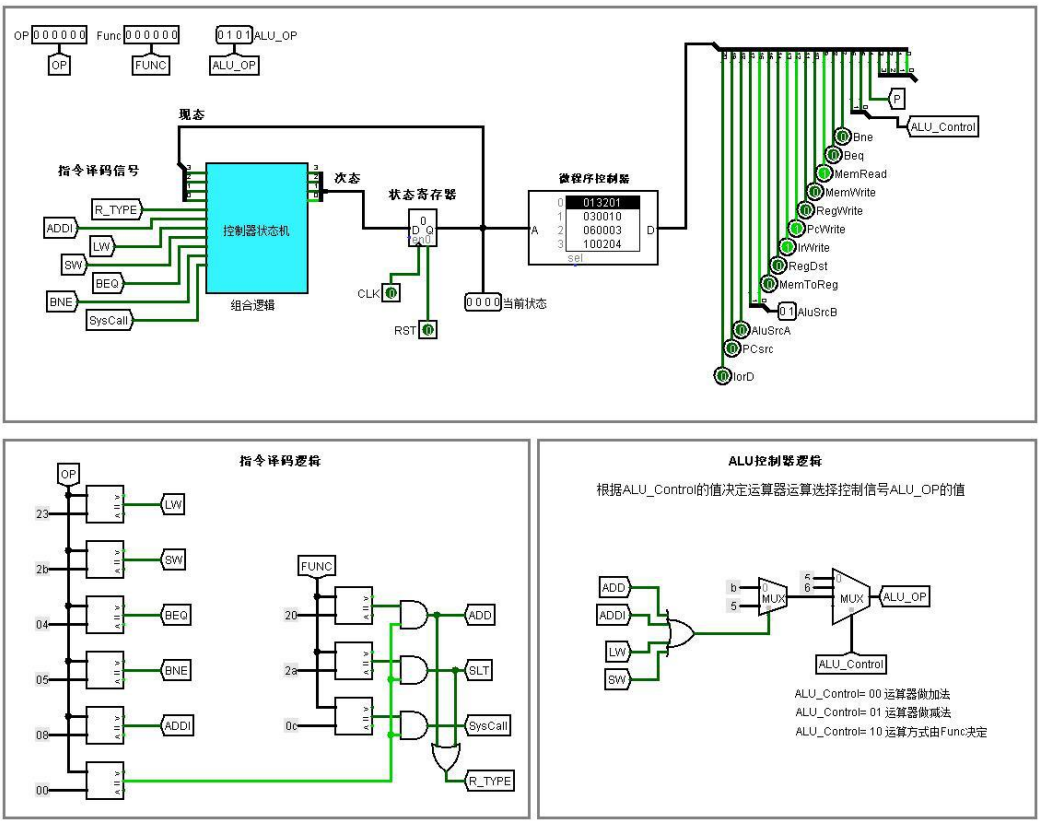


图 1.24 多周期硬布线控制器

（3） 状态转移逻辑实现

状态专利逻辑利用所给文件“状态机真值表管理.xlsx”生成，填上表 1.4 中的内容，通过手动筛选 N0~N3 全为 1 的情形，生成这四个变量的逻辑表达式，然后利用 Logisim 的分析电路功能即可自动生成地址转移电路。

1.4 故障与调试

1.4.1 指令出现死循环

故障现象： MIPS 单周期 CPU 执行时，出现了死循环现象，指令不停的执行，直到遍历指令存储器不停下来。

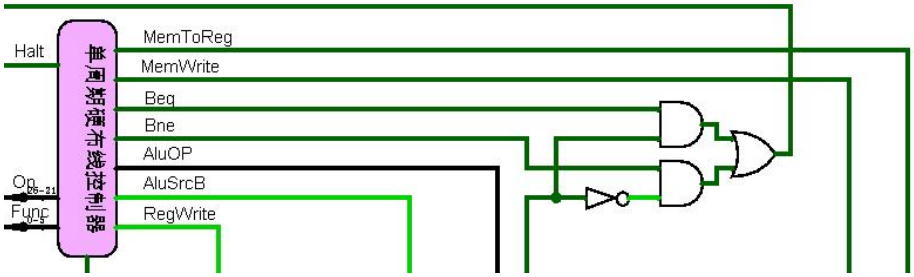


图 1.25 地址跳转逻辑图

原因分析：如图 1.25，跳转逻辑错用成了与门，导致无法成功跳转，因为 beq 和 bne 信号必然不一致。

解决方案：在将图中最右侧的与门改为或门即可。

1.4.2 ALU 功能失效

故障现象：R1 和 R2 的值正常送到了 ALU 两端，但却没有实现加操作。

原因分析：在运行时查看电路的状况，发现 ALU_OP 的值是未定义的 XXXX，沿着通路寻找，发现 ALU 控制器并没有问题，于是寻找接口，发现引脚出的引脚位置不对，设置成了面朝西没有和 ALU_OP 的结果接上。

解决方案：将引脚改为南向即可。

1.5 测试与分析

本实验的测试用例为附件中的 sort.hex，其内部内容如下：

```
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
```

图 1.26 测试文件内容

该程序包括了用以实现冒泡排序的 MIPS 指令，右键单击存储器，选择编辑内容，打开 sort.hex 文件，启动时钟仿真开始测试并等待 CPU 停机。

1.5.1 MIPS 单周期 CPU 测试

将时钟频率调到最大（4.1KHz），启动时钟模拟，运行结果如下图：

图 1.27 运行周期数截图

一共运行了 224 个周期停机，对存储器右键，查看内容，可以看到如下的结果：

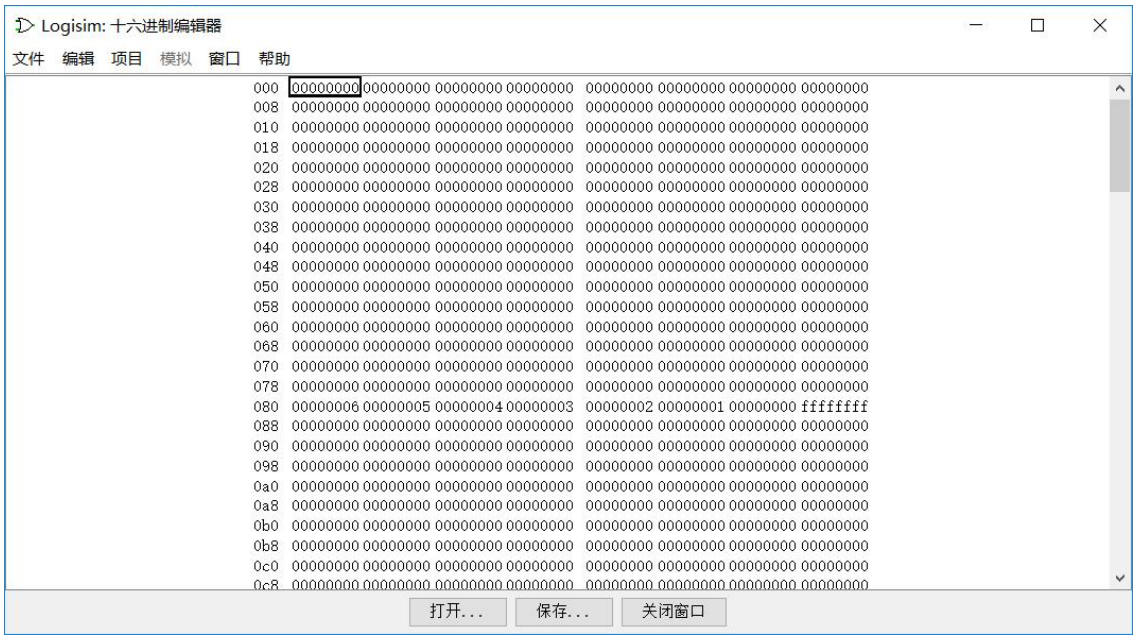


图 1.28 冒泡排序运行结果

从图中可以看到排序已经正确完成，验证我们的 CPU 的实现是正确的。

1.5.2 基于微程序控制器的 MIPS 多周期 CPU 测试

将时钟频率调到最大（4.1KHz），启动时钟模拟，运行结果如下图：

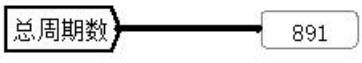


图 1.29 运行周期数截图

一共运行了 891 个周期停机，对存储器右键，查看内容，可以看到如下的结果：

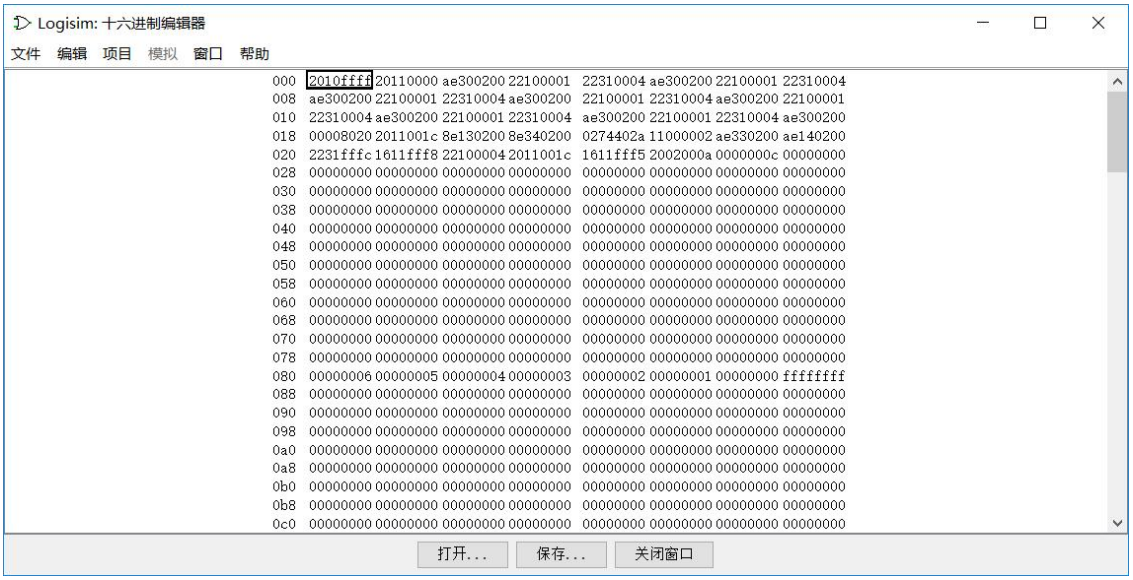


图 1.30 冒泡排序运行结果

从图中可以看到排序已经正确完成，验证我们的 CPU 的实现是正确的。

1.5.3 基于硬布线控制器的 MIPS 多周期 CPU 测试

将时钟频率调到最大（4.1KHz），启动时钟模拟，运行结果如下图：

图 1.31 运行周期数截图

一共运行了 891 个周期停机，对存储器右键，查看内容，可以看到如下的结果：

图 1.32 冒泡排序运行结果

从图中可以看到排序已经正确完成，验证我们的 CPU 的实现是正确的。

2 总结与心得

2.1 实验总结

1） 本次实验设计了 MIPS 单周期 CPU 的数据通路，MIPS 单周期 CPU 硬布线控制器，MIPS 多周期 CPU 数据通路，MIPS 多周期 CPU 微程序控制器，MIPS 多周期 CPU 硬布线控制器

2） 本次实验实现了 add、slt、addi、sw、lw、beq、bne、syscall 八条指令的功能，并根据这 8 条指令完成了硬布线状态转移逻辑和微程序的实现。

3） 本次实验完成了模拟一个简单的冒泡排序在 MIPS 单周期和多周期 CPU 中的运行过程，实现了一个简单的支持 8 条指令的 MPIS 指令集 CPU。

2.2 实验心得

1） 本次实验感觉难度并没有 Cache 实验的难度大，主要还是因为在课堂上有了对 MIPS 单周期和多周期 CPU 的各种指令的数据通路的讲解，一旦熟悉了这些指令数据通路的建立，就很容易根据其设计出微程序或者是硬布线逻辑，相比 Cache 实验容易上手多了。

2） 这次实验最大的收获就是让我好好的整个复习了一遍第六章中央处理器的知识吧，仅仅是上课听了一遍，真的直接动手去做的时候，才发现一开始完全是懵的。为了去做这次实验专门一遍做一遍复习第六章，同时看一下 MOOC 的视频，只有完全熟练了各条指令数据通路的建立，才能很快而且准确的做出这次实验来，而且，这样做出来的程序几乎是没有问题的。我在章节 1.4 故障与调试中仅仅只写了两条，而且都不是什么重要的问题，复习了一遍指令之后做出来的微程序找不出一点错误，这就是这次实验带给我最大的收获吧，即巩固了知识，也减轻了期末复习的负担。

3） 除了知识上的复习以外，另一个就是更加会玩 Logisim 了吧，比如这次实验以前还没怎么用过自动生成电路这种功能等等，同时也玩了很多部件，也为下学期的课设打好了基础。

4） 总之本次实验还是要感谢课程组提供了几个方便的 Excel 表格，给实验减少了很大的负担，真的要让我来设计真值表设计组合逻辑电路，我估计还要打开数字逻辑课本，复习半天相关知识吧

5） 最后还是要感谢组成原理课程组，组原实验是我这学期众多的实验中体验最好的一门，每做一次实验都是对相关课本的一个很好的复习。不只是这一次，比如运算器的实验，如果不熟悉课本上的补码运算器的原理，实验会卡壳很久，同样 Cache 实验也是这样，在完成实验的过程中总是能复习一遍课程，同时实验占用的时间也不长，比较合理，难度适中，时间花费也不是很多。

6） 对实验的建议的话，我觉得还是可以多利用一下实验资源的，我在做四次实验的时候，除了 Cache 实验时时间比较紧张，只做了要求的全相连以外，每个实验都做了所有的部分，收获都不小，建议以后要求检查所有的实验，都做完真的能让同学们有很大的收获，而且也花不了多少时间。

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.

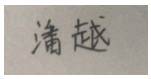
[6] 谭志虎,秦磊华,胡迪青. 计算机组成原理实践教程——从逻辑门到 CPU. 北京：清华大学出版社，2018 年.

• 指导教师评定意见 •

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: _____ **2018-12-24**