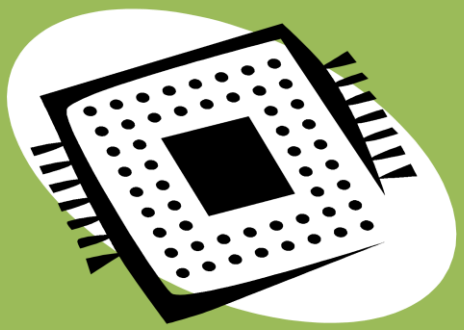
华中科技大学

2019

计算机组成原理 ·实验报告·

专　　业：	计算机科学与技术
班　　级：	CS1706
学　　号：	U201714761
姓　　名：	胡澳
电　　话：	17371232678
邮　　件：	1186767079@qq.com
完成日期：	2019-11-28



计算机科学与技术学院

目　录

1　CPU 设计实验.....2

1.1　设计要求.....2

1.2　方案设计.....2

1.3　实验步骤.....13

1.4　故障与调试.....13

1.5　测试与分析.....14

2　总结与心得.....15

2.1　实验总结.....15

2.2　实验心得.....15

参考文献.....16

1 CPU 设计实验

1.1 设计要求

利用 logisim 平台中的基础组件、在运算器实验中设计的 ALU 以及 CS3410 库中提供的寄存器文件等，构建能够支持 8 条指令的 MIPS CPU。实验中需分别构建单周期和多周期版本的 CPU，其中多周期 CPU 需构建微程序控制器和硬布线控制器。构建的 CPU 可执行 add, slt, addi, lw, sw, beq, bne 和 syscall 这 8 条指令。完成后的 CPU 可正确执行内存区域的冒泡排序程序。上述 8 条指令对应的指令类型、指令功能、操作码以及扩展操作码见表 1-1。

表 1-1 指令类型、功能及操作码

指令	指令类型	指令功能	操作码(hex)	扩展操作码(hex)
add	R	R[rd] = R[rs] + R[rt]	0	20
slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	0	2a
addi	I	R[rt] = R[rs] + SignExtImm	8	无
lw	I	R[rt] = M[R[rs]+SignExtImm]	23	无
sw	I	M[R[rs]+SignExtImm] = R[rt]	2b	无
beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	4	无
bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	5	无
syscall	特殊 R	停机	0	c

1.2 方案设计

1.2.1 单周期 MIPS CPU 数据通路设计

对待实现的 8 条指令，首先实现其中一条指令的数据通路，然后在该数据通路的基础上逐条的实现其余的指令。每次在前面的数据通路中添加新的连线以实现新的指令功能，若出现某一个输入端口在不同的指令中接受来自不同位置的输入，则在该端口处添加多路选择器，同时记录下该指令需要的多路选择器控制信号。完成全部 8 条

指令的数据通路后，可得到如表 1-2 所示的多路选择器控制信号。其中，RegDst、AluSrcB 和 MemToReg 控制信号可直接由控制器给出，而 Branch 信号由于与数据计算结果有关，因此需要在数据通路中进行生成。在我们需要构建的 8 条指令中，跳转指令由 bne 和 beq 两条，当上条指令运行结果使 ALU 将 Equal 信号置为 1 时，beq 指令将产生跳转，当上条指令执行后 ALU 的 Equal 输出为 0 时，bne 指令将产生跳转，根据此关系，可以得到 Branch 的逻辑表达式如下，从而得到生成 Branch 信号的逻辑门电路。

$$\text{Branch} = \text{Beq} \cdot \text{Equal} + \text{Bne} \cdot \overline{\text{Equal}}$$

表 1-2 多路选择器控制信号

输入端口	功能	控制信号	输入数据		指令
寄存器文件 W#	写入寄存器编号	RegDst	0	Rt	非 R 型指令
			1	Rd	R 型指令
ALU_B	运算器输入端 B	AluSrcB	0	寄存器文件输出 R2	R 型指令
			1	I 型指令立即数	I 型指令
寄存器文件 Din	寄存器写入数据	MemToReg	0	ALU 输出	除 lw 外的指令
			1	数据存储器输出	lw 指令
PC 增量	修改 PC 值	Branch	0	0	非跳转指令或跳转条件不满足
			1	I 型指令立即数	跳转指令且满足跳转条件

由于实验中使用的存储器的输入地址为字地址而非字节地址，而在指令中使用的地址为数据的字节地址，因此在连接数据存储器时，应该将 ALU 计算得到的或指令中给出的字节地址转换为字地址后连接到存储器上，又由于实验中使用的存储器的数据位宽为 32 位且地址位宽只有 10 位，因此，在访问数据存储器时，我将地址的 2-11 位分离出来连接到存储器上。同样的，对于指令存储器也存在相同的问题，因此我在做程序计数器 PC 增量时对其直接进行加 1 的操作，从而实现直接的字地址访问。

另外，在实验中需要记录程序运行的周期数，我使用一个计数器统计 CPU 运行过程中经过的周期数，当程序运行到 syscall 指令后，将计时器使能端置为 0，从而终止

计数。通过上述过程构建得到如图 1-1 所示的完整数据通路。

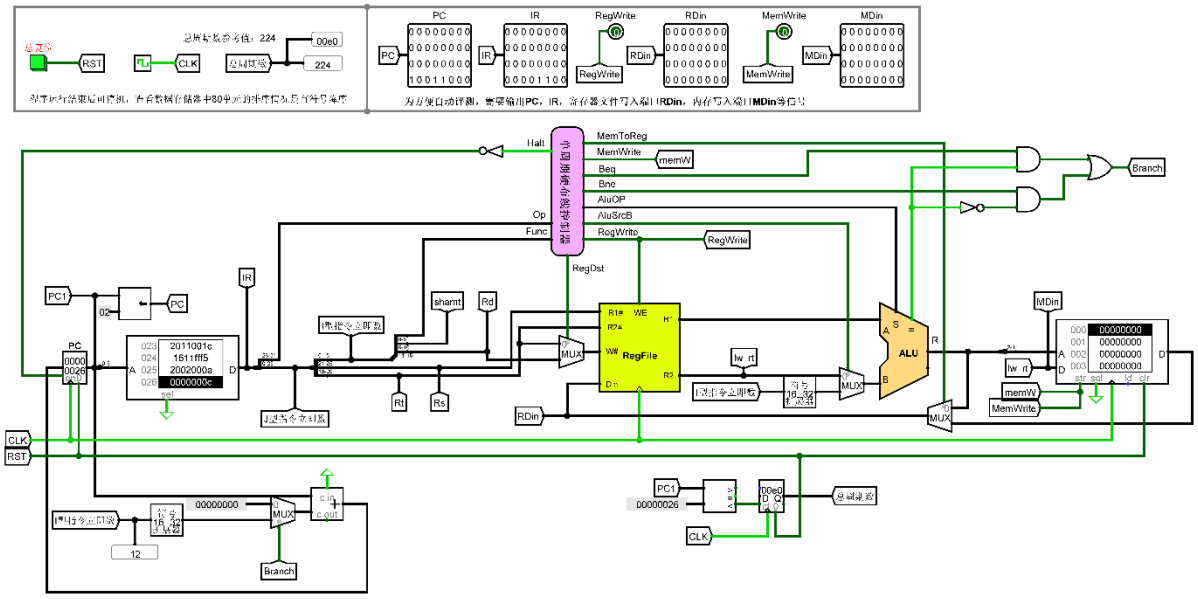


图 1-1 单周期 MIPS CPU 数据通路

1.2.2 单周期硬布线控制器设计

控制器设计包括指令译码逻辑、ALU 控制器逻辑和根据指令译码信号生成控制信号三部分。

1. 指令译码逻辑。该部分根据输入的操作码 OP 和扩展操作码 FUNC 信息确定指令类型并生成相应的指令译码信号。根据表 1-1 中各指令的操作码和扩展操作码，使用比较器将各其与输入的 OP/FUNC 进行比较，即可确定指令类型。由于 syscall 指令为特殊的 R 型指令，其不计入 R_TYPE 中，因此使用简单的组合逻辑对 syscall 信号和(OP == 0)信号进行操作即可得到想要的 R_TYPE 信号。指令译码逻辑的实现如图 1-2 所示。

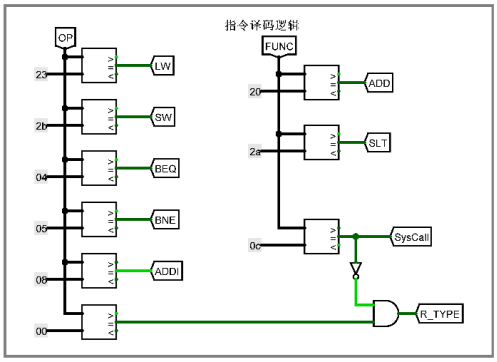


图 1-2 指令译码逻辑

2. ALU控制器逻辑。该部分根据输入的OP/FUNC生成ALU的控制信号ALU_OP。由于实验中需要实现的 8 条指令只会用到 ALU 中的加法功能和有符号比较功能，其运算操作码分别为 0101 和 1011。其中执行 slt 指令时使用有符号比较功能，其余 7 条指令均使用加法功能。使用二路选择器来实现 ALU_OP 的生成，并使用简单的逻辑和比较来生成选择器的选择端输入。当 OP!=0 且 FUNC!=2a，即执行的指令不是 slt 时，选择 0101 输出，否则选择 1011 输出。ALU 控制器逻辑的实现如图 1-3 所示。

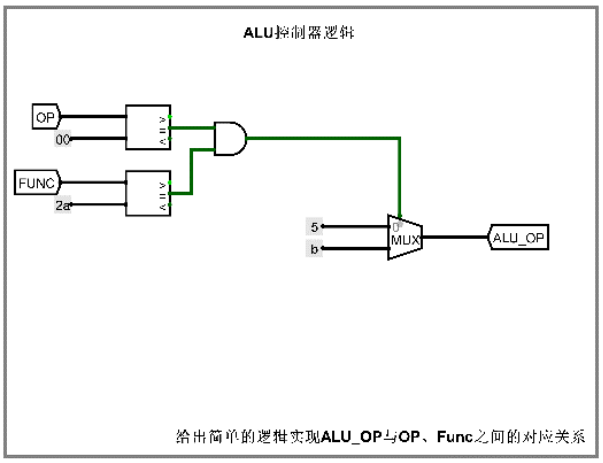


图 1-3 ALU 控制器逻辑

3. 根据指令译码信号生成控制信号。根据 1.2.1 节中设计的数据通路可知各指令需要的控制信号，如表 1-3 所示。根据指令信号和控制信号的对应关系可使用简单的组合逻辑实现控制信号的生成，如图 1-4 所示。

表 1-3 指令译码信号对应的控制信号

指令译码信号	控制信号
R_TYPE(包括 add 和 slt)	RegDst, RegWrite
addi	RegWrite, AluSrc
lw	RegWrite, MemToReg, AluSrc
sw	MemWrite, AluSrc
beq	Beq
bne	Bne
syscall	Halt

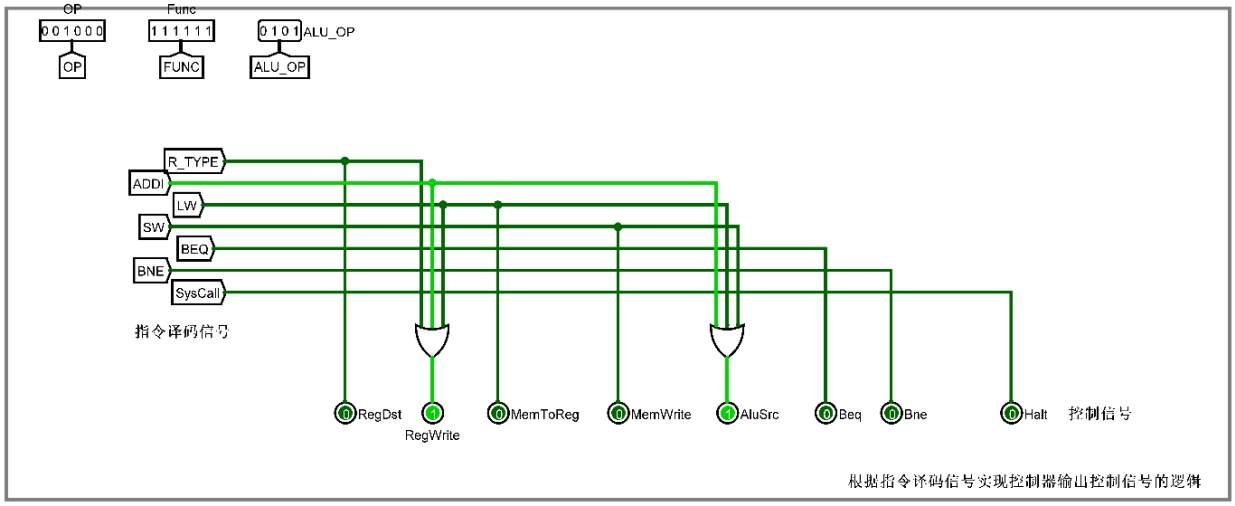


图 1-4 根据指令译码信号生成控制信号逻辑

1.2.3 多周期 MIPS CPU 数据通路设计

与 1.2.1 节中对单周期 MIPS CPU 数据通路的设计一样，多周期 MIPS CPU 的数据通路同样是逐条指令的画出数据通路，并在前一条指令完成后的数据通路的基础上添加新的数据通路，对于有多个数据来源的部件，在其相应的输入端口添加多路选择器来实现多个数据源的输入，同时记录该指令对应的控制信号。完成全部 8 条指令后，可得到如表 1-4 所示的多路选择器控制信号。

由于在多周期 CPU 中，一条指令在多个时钟周期内完成，因此很多数据需要使用寄存器锁存，这些数据包括程序计数器 PC、指令寄存器 IR、数据寄存器 DR、寄存器文件输出 A 和 B 以及 ALU 的输出 C。其中，PC 的值将在第一个时钟周期中实现自增操作，若后续过程中无跳转发生，则 PC 值将一直维持该值，若指令为跳转指令且满足跳转条件，则 PC 的值将再次被修改，因此，需要对 PC 寄存器增加写入控制信号。对于指令寄存器 IR，在第一个时钟周期中完成指令的读取后，在整个指令执行过程中，IR 的值将保持不变，因此也需要对 IR 寄存器添加一个写入控制信号。对于其他的寄存器，当输入发生变化时，可在下一个时钟边缘来临时修改寄存器的值，因此无需添加写入控制。另外，寄存器文件需要添加写使能信号，存储器需要添加读使能和读使能信号来控制对寄存器文件的写入以及对存储器的读写操作。各元件的使能信号如表 1-5 所示。

表 1-4 多周期 CPU 多路选择器控制信号

输入端口	功能	控制信号	输入数据	
存储器地址输入	输入访问存储器的地址	IorD	0	PC
			1	ALU 输出寄存器 C
寄存器文件 W#	写入寄存器编号	RegDst	0	Rd
			1	Rt
寄存器文件 Din	写入寄存器数据	MemToReg	0	ALU 输出寄存器 C
			1	数据寄存器 DR
ALU_A	ALU 输入端口 A	AluSrcA	0	PC
			1	寄存器文件输出寄存器 A
ALU_B	ALU 输入端口 B	AluSrcB	0	1
			1	寄存器文件输出寄存器 B
			2	符号扩展后的 I 型指令立即数
			3	
PC_in	PC 寄存器输入	PcSrc	0	ALU 输出
			1	ALU 输出寄存器 C

表 1-5 多周期 CPU 使能控制

元件	使能端	控制信号
PC 寄存器	写使能	$\mathrm{PCWrite} + \mathrm{Beq} \cdot \mathrm{Equal} + \mathrm{Bne} \cdot \overline{\mathrm{Equal}}$
IR 寄存器	写使能	IRWrite
寄存器文件	写使能	RegWrite
存储器	写使能	MemWrite
	读使能	MemRead

类似于单周期 CPU 的设计，由于设计的 CPU 在没有操作系统的环境下运行，因此没有做字节地址和字地址的转换，又由于实验中存储器字长为 32 位，输入地址为 10 位，因此，当访问存储器时，需要将字节地址左移 2 位从而得到字地址并将字地址送入存储器地址输入端来访问或写入数据。对此，我将每条指令中程序计数器 PC 增量设置为 1，并将 PC 的低 10 位送入存储器即可实现对指令的正确读取。对于指令中给定

或通过 ALU 得到的地址，将其 2-11 位送入存储器即可得到正确的结果。通过上述操作，即可解决由于字地址和字节地址的差异所造成的问题。通过上述操作，即可得到多周期 MIPS CPU 的数据通路如图 1-5 所示。

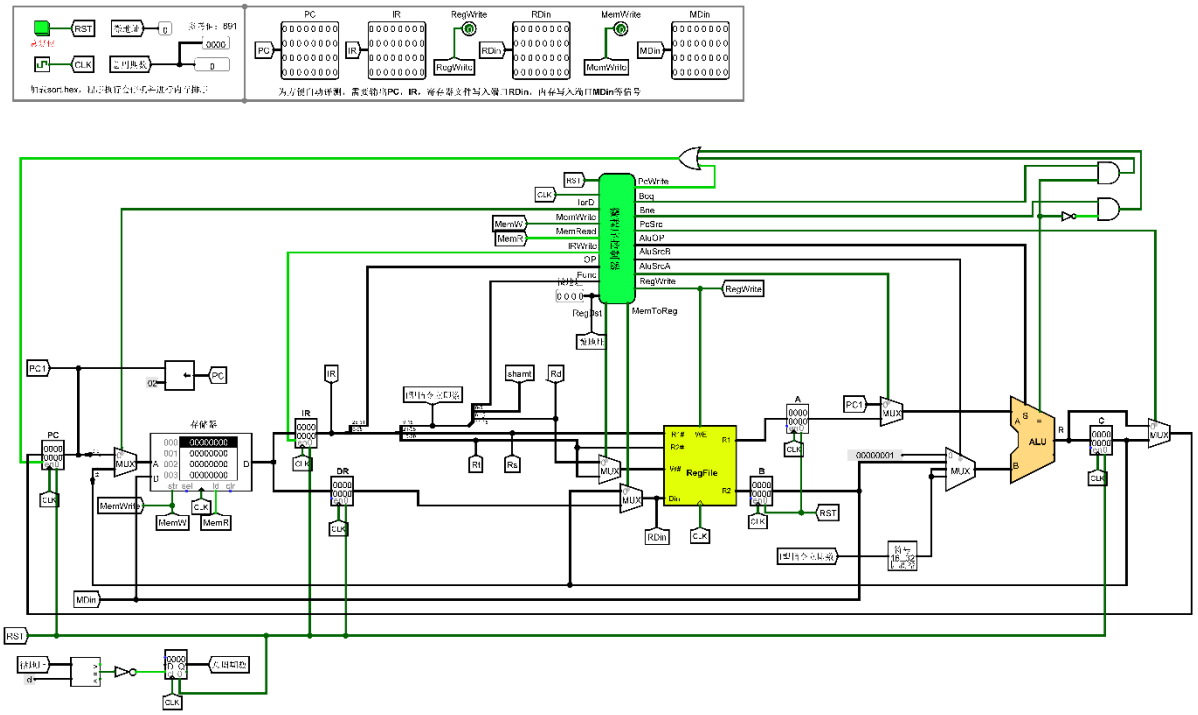


图 1-5 多周期 MIPS CPU 数据通路

1.2.4 多周期 MIPS CPU 控制器设计

与单周期 CPU 的控制器相同，多周期 CPU 的控制器同样由指令译码逻辑、ALU 控制器逻辑和根据指令译码信号生成控制信号三部分组成。本实验中需要分别设计硬布线控制器和微程序控制器两类，这两种控制器中，指令译码逻辑和 ALU 控制器逻辑的设计相同，仅根据指令译码信号生成控制信号的实现不同，对指令译码逻辑、ALU 控制器逻辑、微程序控制器生成控制信号以及硬布线控制器生成控制信号的分析和实现如下。

1. 指令译码逻辑的实现与 1.2.2 中单周期控制器指令译码逻辑的实现相同，直接使用比较器和简单的组合逻辑即可实现指令的译码，得到指令译码信号的输出。其实现如图 1-2 所示。

2. ALU 控制器逻辑。ALU 控制信号 ALU_OP 由微指令中的 ALU_Control 和扩展操作码 FUNC 决定，其关系如表 1-6 所示。使用 ALU_Control 作为多路选择器的控制

信号，并使用 FUNC 确定多路选择器第三个输入端的输入即可实现下表中所示的控制信号输出逻辑。具体的实现如图 1-6 所示。

表 1-6 ALU 控制信号

ALU_Control(bin)	FUNC(hex)	ALU 操作	ALU_OP(hex)
00	无效	加法	5
01	无效	减法	6
10	20	加法	5
	2a	有符号比较	b

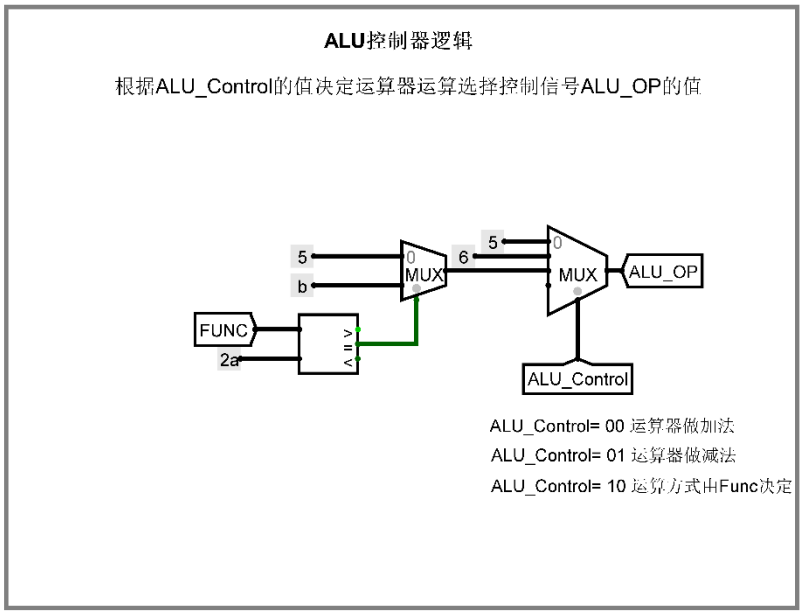


图 1-6 多周期控制器 ALU 控制逻辑

3. 微程序控制器根据指令译码信号生成控制信号。该部分主要由地址转移逻辑和控制存储器组成，地址转移逻辑根据指令译码信号生成相应指令的微程序入口地址，通过该入口地址或微程序的下址字段从控制存储器中读取微指令，使用分线器将微指令分为相应的控制信号输出，其中 ALU 控制信号 ALU_Control 将送往 ALU 控制逻辑部分生成实际的 ALU 控制信号 ALU_OP。为保证控制器正确访问到控制存储器中的微指令，需要添加微指令地址寄存器μPC来存储送入控存的微地址。由于微地址可以来源于地址转移逻辑生成的入口地址，也可以来源于上条微指令的下址字段，因此需要使用多路选择器控制μPC的输入，根据微指令的设计，该多路选择器的控制信号由上条微指令的判别测试字段 P 来实现。

首先，根据 8 条待实现指令的功能，设计状态以及状态之间的转移关系，得到状态图如图 1-7 所示。本实验中，微指令共包含 16 个字段，分别对应于数据通路中各选择器的选择控制端输入、使能端输入以及用于控制微程序跳转的判别测试字段 P 和下址字段组成，微指令的结构如表 1-7 所示。根据状态图、表 1-4 中实现不同操作时需要的多路选择器控制信号以及表 1-5 中实现不同操作时需要的使能信号，可得到表 1-8 所示的微指令。

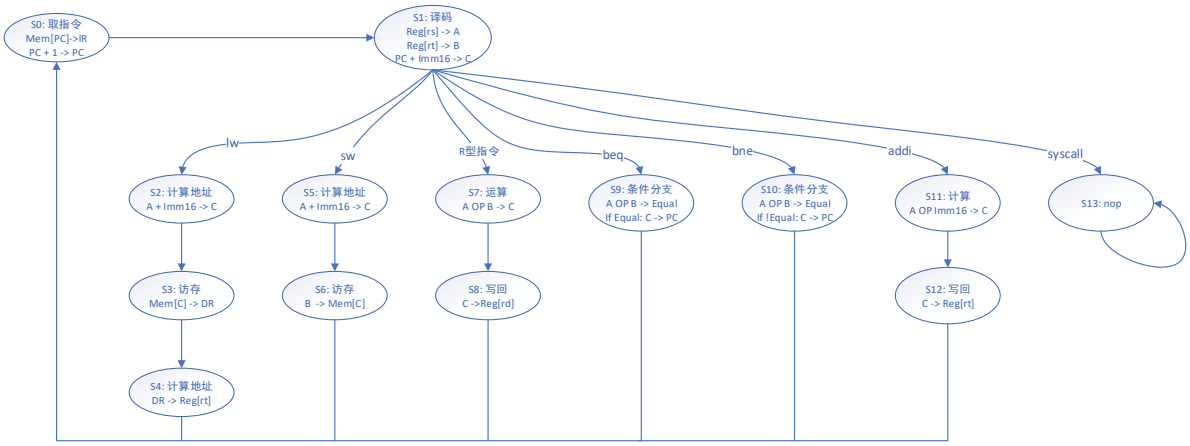


图 1-7 多周期 CPU 控制器状态图

表 1-7 微指令格式

微指令							
IorD	PcSrc	AluSrcA	AluSrcB	MemToReg	RegDst	IrWrite	PcWrite

微指令							
RegWrite	MemWrite	MemRead	BEQ	BNE	AluControl	P	下址字段

表 1-8 微指令及其地址

微指令功能	状态	微指令地址	微指令(bin)	微指令(hex)
取指令	0	0000	00000011001000000001	3201
译码	1	0001	00011000000000010000	30010
LW1	2	0010	00110000000000000011	60003
LW2	3	0011	10000000001000000100	100204
LW3	4	0100	00000110010000000000	C800
SW1	5	0101	00110000000000000110	60006

SW2	6	0110	10000000010000000000	100400
R1	7	0111	00010100000000001001000	50048
R2	8	1000	00000000100001000000	840
BEQ	9	1001	01101000000100000000	D0100
BNE	10	1010	01101000000010000000	D0080
ADDi1	11	1011	00110000000000001100	6000C
ADDi2	12	1100	00000100100000000000	4800
SYSCALL	13	1101	00000000000000001101	D

根据上表中给出的各指令对应的微程序入口地址，使用 Excel 自动生成逻辑表达式可得到地址转移逻辑的逻辑表达式如下。根据逻辑表达式使用 logisim 自动生成即可得到地址转移逻辑的电路。完成上述操作后，即可得到微程序控制器根据指令译码信号生成控制信号的电路图，如图 1-8 所示。

$$S3 = ADDI + BEQ + BNE + SYSCALL$$

$$S2 = R\_Type + SW + SYSCALL$$

$$S1 = R\_Type + ADDI + LW + BNE$$

$$S0 = R\_Type + ADDI + SW + BEQ + SYSCALL$$

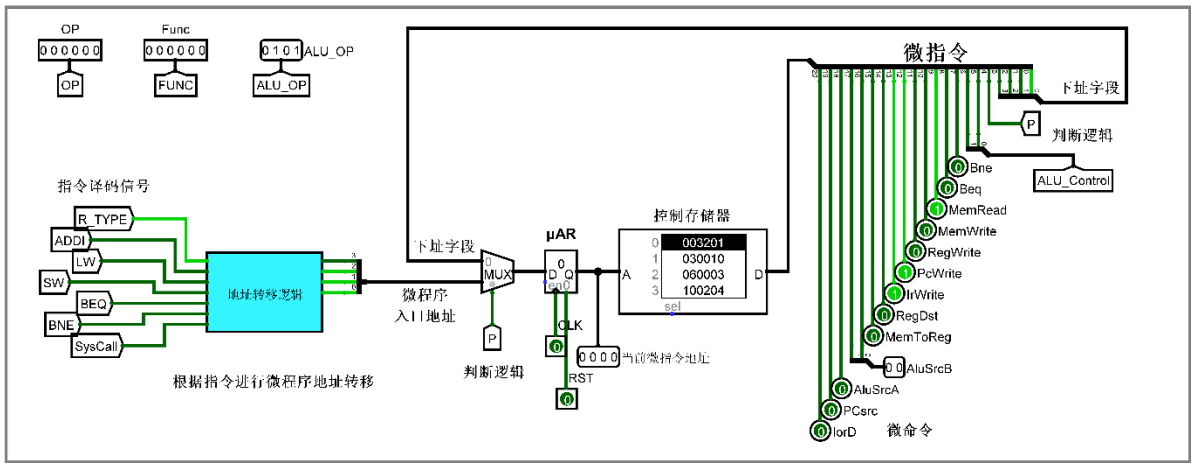


图 1-8 微程序控制器根据指令译码信号生成控制信号

4. 硬布线控制器根据指令译码信号生成控制信号。本实验中并未完全使用硬布线的方法实现控制信号的生成，而是在微指令控制器的基础上，使用硬布线的组合逻辑电路实现状态之间的迁移，而不是使用判别测试字段 P 和下址字段来实现状态迁移。

因此，在硬布线控制器的设计中，仍然使用微程序控制器中构造的微指令来实现控制信号的输出。对于状态迁移，根据图 1-7 状态图可得到如表 1-9 所示的状态迁移表。

表 1-9 状态变迁表

现态(Dec)	输入信号							次态(Dec)
	R_Type	LW	SW	BEQ	BNE	SYSCALL	ADDI	
0								1
1	1							7
1		1						2
1			1					5
1				1				9
1					1			10
1						1		13
1							1	11
2								3
3								4
5								6
7								8
11								12
13								13

根据上表，可使用 Excel 得到由现态 S3S2S1S0 以及输入信号求得次态 N3N2N1N0 的逻辑表达式如下，根据逻辑表达式，使用 logisim 即可直接生成硬布线控制器状态机的组合逻辑电路。完成上述操作后可得到硬布线控制器根据指令译码信号生成控制信号的电路图，如图 1-9 所示。

$$N3 = \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot BEQ + \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot BNE + \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot SYSCALL + \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0 \cdot ADDI + \overline{S3}\cdot S2\cdot S1\cdot S0 + S3\cdot\overline{S2}\cdot S1\cdot S0 + S3\cdot S2\cdot\overline{S1}\cdot S0$$

$$N2 = \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot R\_Type + \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot SW + \overline{S3}\cdot\overline{S2}\cdot\overline{S1}\cdot S0\cdot SYSCALL + \overline{S3}\cdot\overline{S2}\cdot S1\cdot S0 + \overline{S3}\cdot S2\cdot\overline{S1}\cdot S0 + S3\cdot\overline{S2}\cdot S1\cdot S0 + S3\cdot S2\cdot\overline{S1}\cdot S0$$

$$N1 = \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot R\_Type + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot LW + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot BNE + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot ADDI + \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} + \overline{S3} \cdot S2 \cdot \overline{S1} \cdot S0$$

$$N0 = \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0} + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot R\_Type + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot SW + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot BEQ + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot SYSCALL + \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot ADDI + \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} + S3 \cdot S2 \cdot \overline{S1} \cdot S0$$

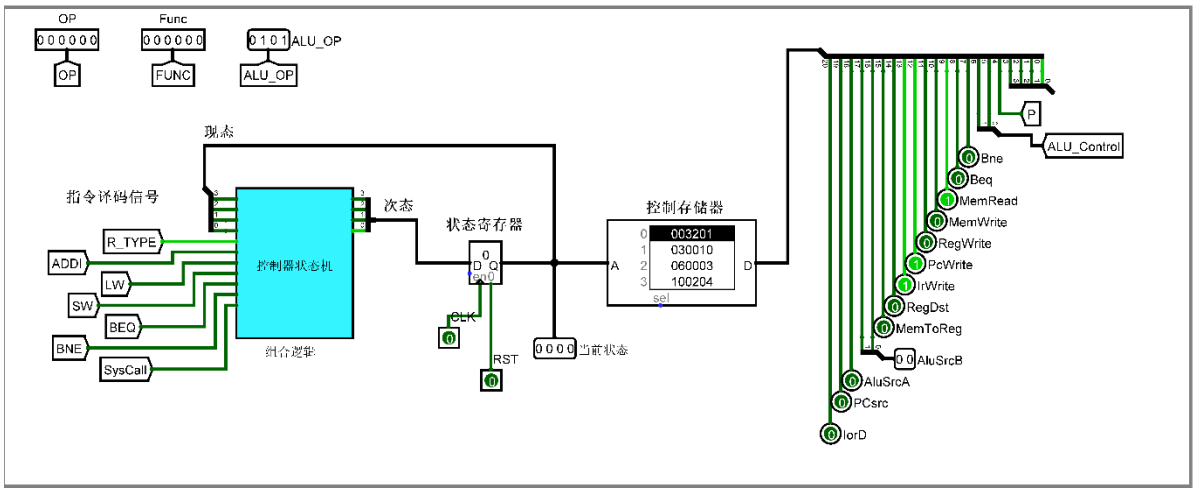


图 1-9 硬布线控制器根据指令译码信号生成控制信号

1.3 实验步骤

（1） 根据 1.2 节方案设计中的设计过程，依次完成 CPU 的数据通路和控制器的设计。

（2） 加载测试程序 sort.hex 到相应的存储器中，运行 CPU 并观察运行结果。

（3） 根据出现的故障对 CPU 进行单步调试、排除故障。

1.4 故障与调试

1.4.1 访问存储器地址错误

故障现象：程序运行过程中不是按照顺序执行，而是每此跳过几条指令执行。

原因分析：实验中按照字节地址设置的 PC 值，因此在没有跳转指令的情况下，每次将 PC 的值自增 4，由于存储器输入地址为按字寻址，导致直接将 PC 的低 10 位输入到存储器中时，每次向前访问 4 个字，而不是 4 个字节，导致访问的数据与预期不相符。

解决方案：该问题可通过两种方法解决，第一种，可以将 PC 的自增值由 4 改为 1，仍将 PC 的低 10 位接入到存储器的地址输入端；第二种方法，可以保持 PC 自增 4 不变，将 PC 的 2-11 位接入到存储器的地址输入端。

1.5 测试与分析

对于单周期 CPU，将测试程序 sort.hex 载入到程序存储器中，启动 CPU，待程序执行结束后，查看数据存储器的值。对于多周期 CPU，将测试程序 sort.hex 载入到存储器开始的位置，启动 CPU，待程序执行完成后，查看存储器中的数据。

对于实验中实现的 3 个 CPU，运行完测试程序后均可看到在相应的存储器的 0x80 地址处开始，依次为 6、5、4、3、2、1、0、-1 八个数的补码，由此可知，CPU 正确执行了测试程序，可以认为成功完成了 8 条指令的 CPU 的设计。

2 总结与心得

2.1 实验总结

本次实验主要完成了如下几点工作：

1） 完成了单周期硬布线控制器、多周期硬布线控制器和多周期微程序控制器的设计。

2） 完成了单周期 CPU 和多周期 CPU 数据通路的设计。

3） 完成了将控制器和数据通路组合成 CPU。

4） 解决了 CPU 设计过程中出现的错误

2.2 实验心得

1） 通过这次设计 CPU 的实验，我对组成原理课上讲的指令系统的内容有了更加深刻的理解。我体会到了指令系统的复杂程度与 CPU 中数据通路以及控制器的设计难度以及电路的复杂度之间直接的关系。另外，实验中设计了 8 条很常用的指令，通过分析指令的执行流程，我对程序在 CPU 上具体的执行过程有了直观的认识，这也让我对计算机执行程序的过程有了更加细致的了解。

2） 虽然本次实验中设计的 CPU 只需要实现 8 条指令，但是 CPU 的数据通路已经很复杂，在画图连线的过程中需要很认真，避免出现错误的连线。

3） 实验中，老师提供了很多已经封装好的元件，通过直接使用这些元件，可以有效的降低我们的工作量，让我们可以更加关注 CPU 控制器和数据通路的设计。后来老师还提供了 MIPS 指令的探针，方便我们对 CPU 进行调试。

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 5 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] 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018 年.

[4] 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.

[5] 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.

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

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: _____