多周期 MIPS CPU 设计步骤

1、 构建数据通路

参考英文教材 389 面多周期 MIPS 处理器数据通路图，在 logisim 中利用相关部件进行构建，注意寄存器文件可以使用 cs3410 库中的寄存器文件，但需要注意的是该寄存器文件体积过大，且引脚分布并不合理，可以利用子电路再次封装成下图中的形状。另外为方便绘图，alu 的引脚也可以适当调整位置。

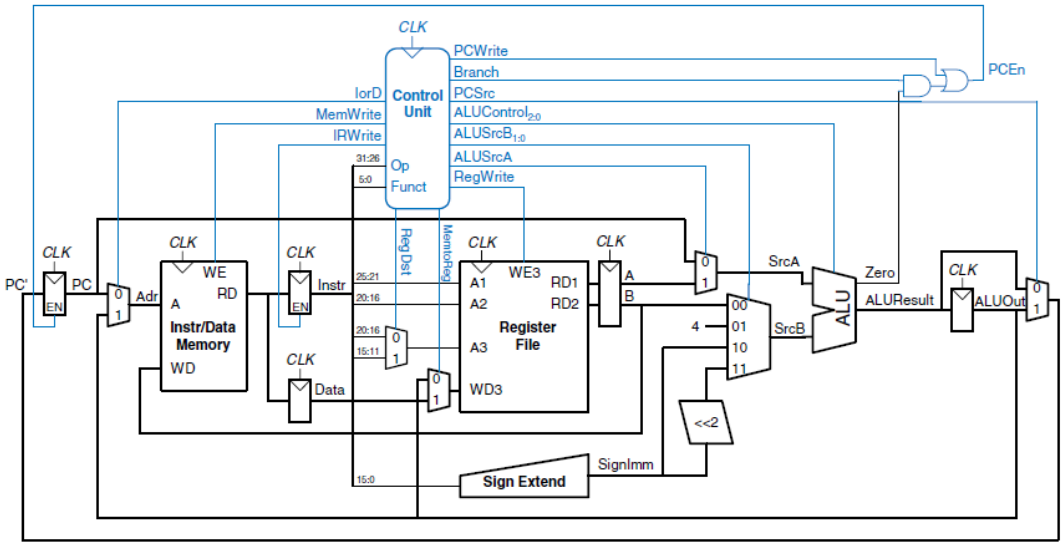
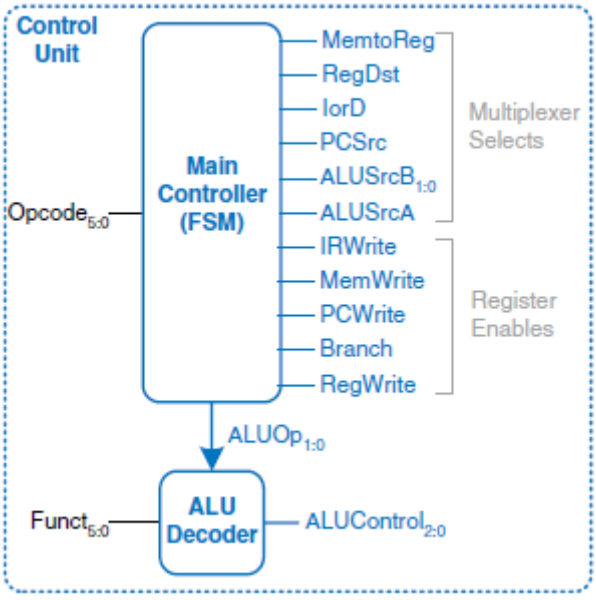


Figure 7.27 Complete multicycle MIPS processor



控制器内部示意图

2、 构建控制器子电路框架

构建上图中的控制器子电路，最开始只需要编辑子电路外观，增加输入输出引脚，是的 logisim 中多周期的 CPU 数据通路，以及控制总线连接完整即可。

3、构建有限状态机部分

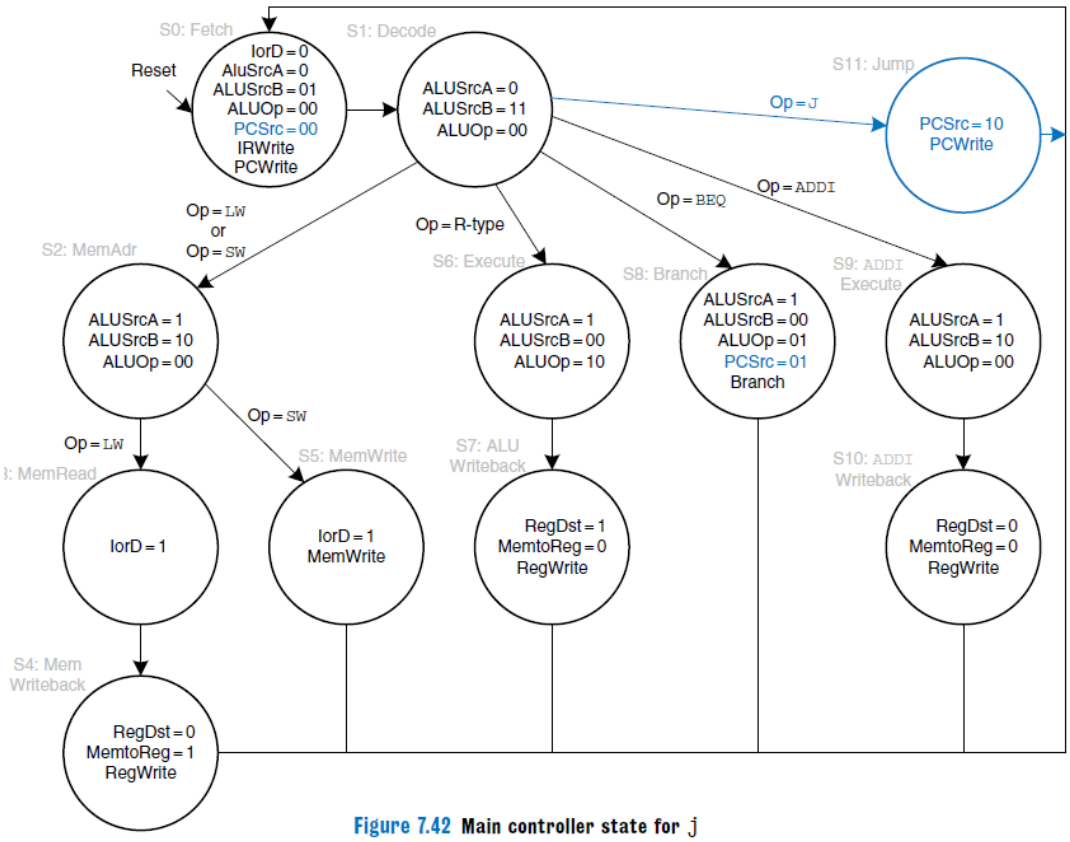
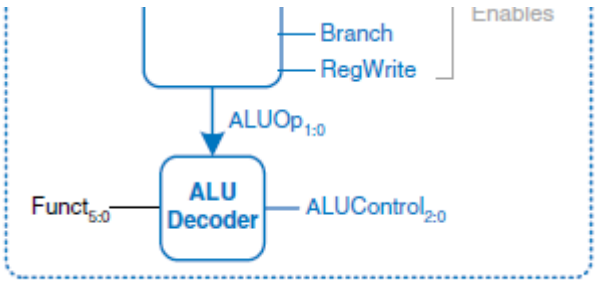


Figure 7.42 Main controller state for j

参考这个图，构建实验所需 8 条指令的状态机，思考一下 syscall 停机指令的状态机是怎样的？R 型指令是否共有一个分支？

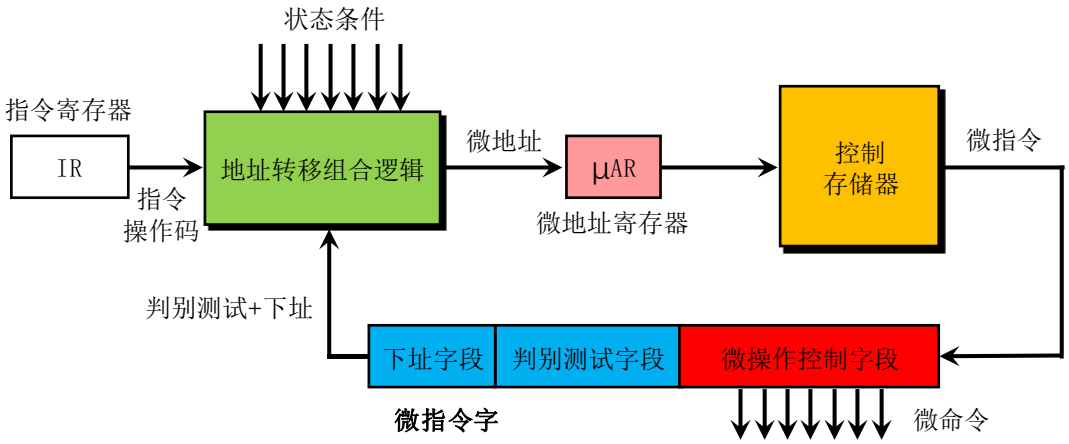
4、构建运算器译码逻辑



设计 ALU Decoder 逻辑，是的 ALUOP 可以生成运算器需要的功能选择信号，实验中 ALUControl 为 4 位。

5、 构建控制器

(1) 构造微程序控制器



按照微程序的逻辑，首先构造控制存储器中的微指令格式，为方便大家编写微程序，这里给出了一个 excel 表，这个表实际就是控制存储器中存放的微程序，每一行是一条微指令，其中第一行是取指令微指令，第二行是指令译码取操作数微指令，其他行是实验中其他指令对应的微程序，可以根据需要进行微调，每条微指令中为 1 的微命令用红色标出，右侧十六进制列自动将微指令转换为 16 进制编码，可以选中后复制到 logisim 中的控制存储器 ROM 中。

请根据状态图构造微程序，注意下址字段以及 P 字段的设置，最终将微程序 16 进制代码复制到控制存储器中。

备注	ADDR	状态	IorD	PcSrc	AluSrcA	AluSrcB	MemToReg	RegDst	IrWrite	PcWrite	RegWrite	MemWrite	MemRead	BEQ	BNE	AluOP	P	下址字段	微指令	十六进制
取指令	0000	S0	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0001	00000000000000000001	1
译码	0001	S1	0	0	0	00	0	0	0	0	0	0	0	0	0	00	1	0000	00000000000000010000	10
LW1	0010	S2	0	0	1	00	0	0	0	0	0	0	0	0	0	00	0	0000	00100000000000000000	40000
LW2	0011	S3	0	0	0	00	1	0	0	0	0	0	0	0	0	00	0	0000	00000100000000000000	8000
LW3	0100	S4	0	0	0	00	0	0	1	0	0	0	0	0	0	00	0	0000	00000001000000000000	2000
SW1	0101	S5	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
SW2	0110	S6	0	0	0	00	0	0	0	0	1	0	0	0	0	00	0	0000	00000000010000000000	800
R_TYPE1	0111	S7	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
R_TYPE2	1000	S8	0	0	0	10	0	0	1	0	0	0	0	0	0	00	0	0000	00010010000000000000	12000
Beq	1001	S9	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
Bne	1010	S10	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
ADDI1	1011	S11	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
ADDI2	1100	S12	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0
SYSCALL	1101	S13	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	0000	00000000000000000000	0

将控制存储器的输出端利用分线器将所有微命令解析出来连接到对应输出引脚。

构建微地址寄存器

构建地址转移组合逻辑

1、 指令译码，给出 add、addi、slt、bne、beq、lw、sw、syscall 的译码信号，为简化设计，这里可以使用运算器中的比较器生成这些信号。

2、 构造地址转移逻辑，输入为地址译码信号（8 个，也可适当合并，如 add 和 slt 的微程序是一样的），输出为微程序入口地址。这部分可以利用 logisim 自带的真值表生成组合逻辑电路功能自动生成子电路。

3、 构建判断字段逻辑，负责指令译码的微指令判断字段应该为 1，执行这条微指令时微地址寄存器输入应该是地址转移逻辑生成的地址而不是下址字段给出的地址。

跳转到第 6 步进行调试。

(2) 构造硬布线控制器

完成微程序控制器后，我们可以继续挑战一下硬布线控制器设计，首先必须构造 FSM 状态机，根据状态图构造状态机表格，构建状态机真值表。该真值表输入为现态，指令译码信号若干，输出为次态。

现态	输入(指令操作码)/次态						输出(即操作控制信号)
	XXXX	Add	lw	sw	beq	j	
0000(S0)	0001						MemRead =IRWrite= PCWrite=1， IorD=0，ALUSrcA =0, ALUSrcB =01, PCsource=01，ALUop =00
0001(S1)		0110	0010	0010	1000	1001	ALUSrcA =0 , ALUSrcB=11, ALUop =00
0010(S2)			0011	0101			ALUSrcA=1 ,ALUSrcB=10,ALUop=00
0011(S3)		0100					MemRead=1 ,IorD=1
0100(S4)	0000						RegDst=0, RegWrite=1,MemtoReg=1
0101(S5)	0000						MemWrite=1, IorD=1
0110(S6)		0111					ALUSrcA=1, ALUSrcB=00,ALUop =10
0111(S7)	0000						MemtoReg=0,RegWrite= RegDst=1
1000(S8)	0000						PCSource=00,PCWrite=1
1001(S9)	0000						ALUSrcA=1 ,PCWriteCon=1, ALUSrcB=00,PCSource=10,ALUop=01

在 logic Friday 软件中构建对应的真值表，注意 10 多个输入的真值表输入是无法想象的，为简化逻辑，这里可以将构建好的真值表导出为 csv 文件，然后利用 excel 直接编辑，此时输入端可以使用 x 变量，这将大大简化真值表的输入工作，完成后导入 logic Friday 生成逻辑表达式以及电路图，在 logisim 中构建对应电路。

由于控制信号和状态机的状态对应，所以这里状态机的状态可以等效于微程序控制器中的微地址寄存器，直接将实现的状态机电路替换原有的地址转换逻辑，将状态机直接与控制存储器相联，看看系统是否可以正常工作。

如果还有力气，再根据状态机利用逻辑表达式生成最终的微控制信号，这部分比较枯燥了，建议就不做了。

6、 调试

首先加载逐条指令测试程序进行逐条测试，检测各指令功能是否和 Mars 执行结果一致。

最后测试排序测试程序，运行正确内存中 0x80 位置值应该如下，且程序能自动停机。