Shandong University

School of Computer Science and Technology

计算机组成原理课程设计报告

Course Design of Principles of Computer Organization

姓名　　：施政良　*Zhengliang Shi*

班级　　：*2019*级 四班

学号　　：*201900130133*

完成日期：2021-11-30

指导教师：赵梦莹　*Mengying Zhao*

摘　要

本学期课程设计主要分为四个阶段，分别是微程序控制的运算器设计、微程序控制的存储器读写系统设计、基于微程序的简单模型机设计以及基于硬布线的简单模型设计。其中前三个阶段主要围绕微程序控制方式及进行展开，最后一个阶段侧重于使用组合逻辑方式进行模型机的内核设计。

本次实验中的指令系统在基础要求上进行扩展，并且为了支持条件跳转、在指令集中设计了条件跳转指令 BNE（bound not equal） 以及无条件跳转指令 JR。考虑到循环和递归等常见操作，在指令集中引入了压栈（PUSH）和弹栈（POP）指令，并最终实现了**递归程序**的计算。

实验的总体结构主要包含逻辑运算单元、通用寄存器、指令寄存器、地址寄存器、栈指针寄存器、存储器、时钟发生器等，其中存储器细分为 RAM、SRAM 分别用于存放指令数据和充当**递归栈**。

微程序实现方式和硬布线实现方式的主要差异体现在模型机的控制器设计。微程序方式下需要将指令划分为若干微操作，并为每一个微操作分配对应的节拍。而硬布线方式下需要根据具体的组合逻辑表达设计相应的电路。

为了测试指令集的正确性，在实验中设计了不同的测试程序，分别用于测试运算指令（加、减、**乘、除**以及各种逻辑运算）、跳转指令。同时，为了测试模型机对递归程序的执行效果，我们根据指令格式编制了相应的汇编代码以及机器代码，最终实现并验证了**等比数列**$\boldsymbol{a_n = 2a_{n-1}}(n = 1,2,3\ldots)$的递归计算。

关键词：微程序控制　硬布线　　指令系统　　模型机内核设计

Abstract

The course design of this semester is mainly divided into four stages:microprocessor controlled calculator design, memory reading and writing system design, simple model machine design based on microprogram and hardwiring model design. The first three stages mainly focus on the microprograming control mode and, and the last stage focuses on the design of model mach-ine kernel using combinatorial logic mode.

The instruction system in this experiment is expanded on the basic requirements, and in order to support conditional jumps, a conditional jump instruction BNE and an unconditional jump instruction JR are designed in our work. Taking into account common operations such as loops and recursion, push stack **(PUSH)** and pop stack **(POP)** instructions are introduced in the instruction set. And finally realized the calculation of the recursive program.

The overall structure of the experiment mainly includes logic operation unit, general register, instruction register, register, stack pointer register, memory, clock generator, etc. the memory is subdivideed into RAM and SRAM, which are used to store instruction data and act as recursive stack respectively.

Two control modes have something different.In microprograming mode, the instruct-ions need to be divided into several micro operations, and each micro operation is assigned a corresponding beat. In the hard wir-ing mode, the corresponding circuit needs to be designed accord-ing to the specific combinational logic expression.

In order to test the correctness of instruction set, different test programs are designed. At the same time, in order to test the execution effect of the model machine on the recursive program, we compiled the corresponding assembly and machine code according to the instruction format, realizing **recursive calculation** which is define as $a_n = 2a_{n-1}$ finally.

Key words: microprogram control hardwiring instruction system model

Content

1 课程设计概述

1.1 设计目的

本学习的课程设计以深入了解计算机组成原理为核心，通过理论和分析和实际的操作加深对知识的理解和掌握。同时，应用实际的电路设计，制作简易模型机。设计步骤包括：

- 拟定指令系统: 指令系统将涉及到 指令字长、指令格 式、指令的编码、指令种类、寻址方式。
- 确定总体结构: 包含确定各部件设置 以及它们之间的数 据通路结构
- 具体逻辑设计：各部件的逻辑设计和部件之间的连接确定控制方式()
- 设计逻辑电路并进行调试：波形仿真以及实际电路测试

1.2 设计要求

阶段一：

实验一：微程序控制的运算器设计

实验二：微程序控制的存储器读写系统设计

阶段二：

实验三：基于微程序的简单模型机设计

实验四：基于硬布线的简单模型机设计

1.3 具体设计步骤

1.3.1 拟定指令系统

拟定指令系统将涉及到基本字长、指令格式 、指令种类、寻址方式等内容。

- 在具体应用中指令格式可有单字长指令和双字长指令两种。在双字长格式中，第二字节一般定义为操作数或 操作数地址；
- 对于指令类型而言，模型机有单操数指令、双操作数指令和无操作数指令
- 寻址方式包括寄存器寻址和立即数寻址两种方式

1.3.2 确定总体结构

总体结构设计包含确定各部件设置 以及它们之间的数据通路结构。数据通路不同，执行指令所需要的操作就不同，计算机的结构也不同。

1.3.3 逻辑设计

逻辑设计指各个原件的大体设计框架，以及原件之间的连接方式等，通常和总体的架构有关，因此需要放在确定总体结构之后。

1.3.4 确定控制方式

控制命令是确定信息的流向，不同的数据通 路需要不同的控制命令。常用的控制方式包括微程序控制方式以及组合逻辑控制（硬布线方式）。

在微程序控制方式下主要涉及以下四个步骤

- 首先确定微程序控制器的结构，下地址形成方式、微程序控制时序等
- 其次需要遍址指令流程
- 在上述基础上编址微程序，
- 最后进行调试和验证。

在硬布线实现方式下，主要涉及以下五个步骤

- 首先对指令的步骤和功能进行划分
- 设计节拍发生器
- 根据指令执行流程和数据通路，并设计控制信号的列表
- 根据列表设计控制信号的逻辑表达式
- 最终设计完整电路并实际调试

2 微程序控制的运算器设计

2.1 设计概述

2.1.1 设计目的

1. 熟悉简单运算器的结构
2. 熟悉微命令的产生和时序
3. 熟悉运算器的功能测试
4. 实践操作，设计相应的指令集并完成测试

2.1.2 设计简述

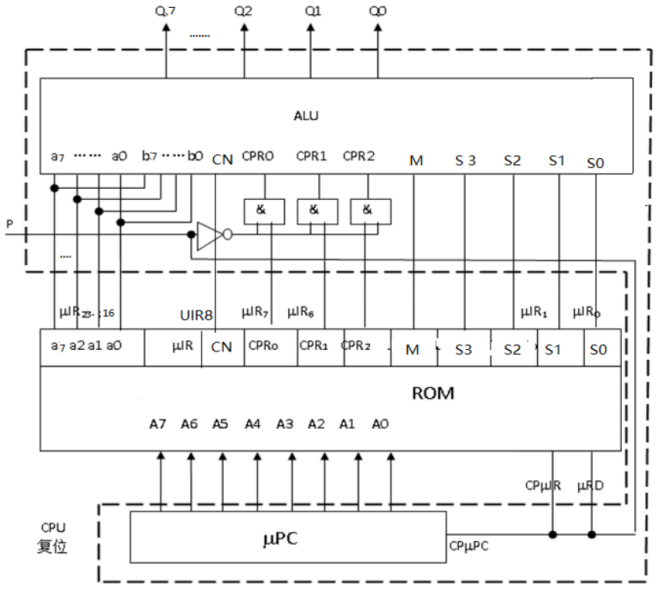
（1） 设计一个八位算法逻辑运算单元 ALU

（2） 要求两个操作数有八位寄存器 R0 和 R1 提供，并将运算结果保存在 R2 中具体何种操作数可以有微命令任意指定，例如 55+AA->R2 （其中 55 和 AA 均为十六进制数）

2.2 结构设计

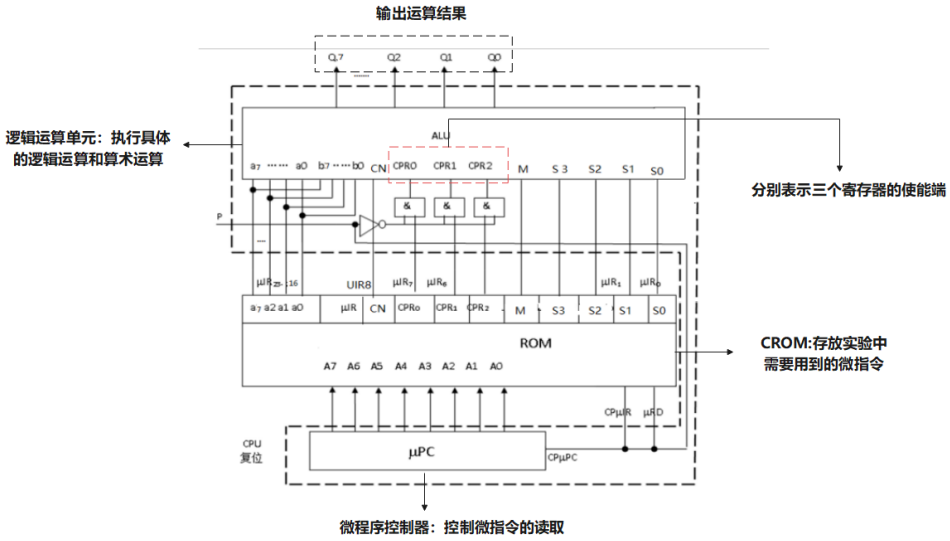
2.2.1 设计框架概述

根据实验内容可知，本实验的核心任务是设计运算逻辑单元 ALU，同时需要设计微程序控制器控制指令的执行。因此总体的设计框架如下所示：



其中 ALU 用于进行算数和逻辑运算，CROM 作为存储控制单元，作用为保存实验中用到的所有微指令， μPC 为微指令控制器，作用类似于 PC，用于控制微指令的读取。

综上，每一部分的作用如下图所示：



2.2.2 运算器设计思路

实验中采用 74LS181 进行运算器的设计，通过芯片的引脚S_0, S_1, S_2, S_3 以及引脚M控制不同类型的运算。具体引脚以及对应的运算种类如下图所示：

方式	M=1 逻辑运算	M=0 算术运算	
S3 S2 S1 S0	逻辑运算	CN=1（无进位）	CN=0（有进位）
0 0 0 0	F=/A	F=A	F=A 加 1
0 0 0 1	F=/(A+B)	F=A+B	F=(A+B)加 1
0 0 1 0	F=(/A)B	F=A+/B	F=(A+/B)加 1
0 0 1 1	F=0	F=负 1	F=0
0 1 0 0	F=/(AB)	F=A 加 A(/B)	F=A 加 A/B 加 1
0 1 0 1	F=/B	F=(A+B)加 A/B	F=(A+B)加 A/B 加 1
0 1 1 0	F=A⊕B	F=A 减 B 减 1	F=A 减 B
0 1 1 1	F=A/B	F=A(/B)减 1	F=A(/B)
1 0 0 0	F=/A+B	F=A 加 AB	F=A 加 AB 加 1
1 0 0 1	F=/(A⊕B)	F=A 加 B	F=A 加 B 加 1
1 0 1 0	F=B	F=(A+/B)加 AB	F=(A+/B)加 AB 加 1
1 0 1 1	F=AB	F=AB 减 1	F=AB
1 1 0 0	F=1	F=A 加 A	F=A 加 A 加 1
1 1 0 1	F=A+/B	F=(A+B)加 A	F=(A+B)加 A 加 1
1 1 1 0	F=A+B	F=(A+/B)加 A	F=(A+/B)加 A 加 1
1 1 1 1	F=A	F=A 减 1	F=A

由于本实验中需要实现的运算包括直接传送、加减算术运算，逻辑与或非以及逻辑异或运算，因此对于上述芯片的功能表可以简化为：

M	$S_3S_2S_1S_0$	CN	操作
算术运算 ：0	1001	1	A加B
	0110	0	A减B
	0000	0	A加1
	1111	1	A减1
	1001	0	A加B加1
逻辑运算： 1	1011	×	AB
	1110	×	A+B
	0110	×	A异或B
	1111	×	A直传
	1010	×	B直传
	0000	×	A逻辑非
	0101	×	B逻辑非

考虑到本实验中的操作数为 8 位，而 74181 芯片的操作数只有四位，因此需要将两片 74181 芯片进行级联，实现八位操作数的运算。具体设计图详见实验原理图。

2.2.3 指令集设计思路

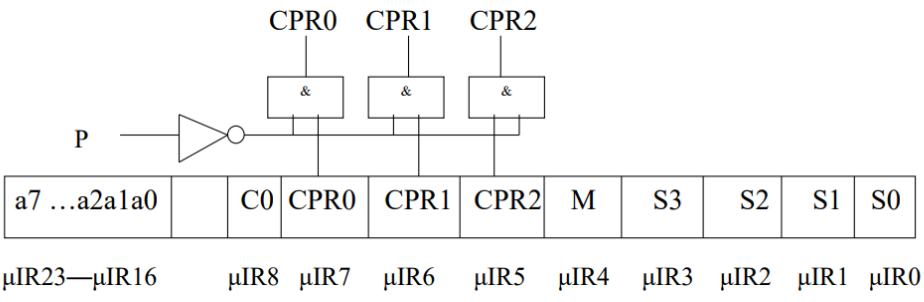
根据实验结构设计可知，在 μPC 的控制下将 CROM 中的指令打入 ALU，ALU 通过对指令的“操作码”等字段进行相应的工作。因此，需要设计相应的微指令格式。根据 74181 芯片的功能表以及上述的架构设计，实验中指令集的设计需要考虑的如下因素：

（1） 如何控制 ALU 进行不同种类的运算

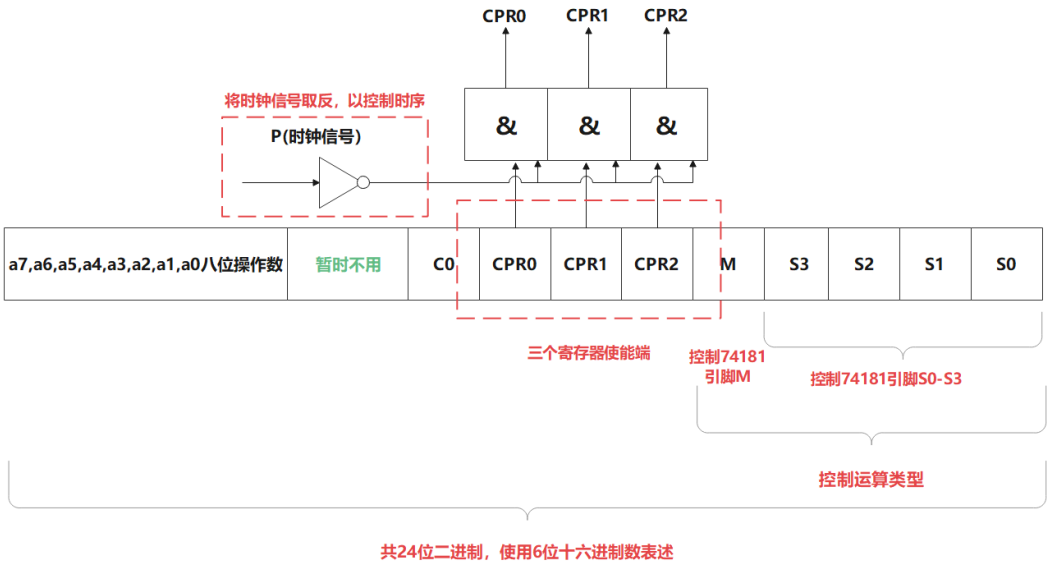
（2） 如何控制三个寄存器的使能端

（3） 操作数的位数

因此，综合以上三个因素，可以采用如下 24 位的指令集设计



各个位的作用如下所示：



其中最后五位M,S_0,S_1,S_2,S_3以及C_0控制运算类型，CPR_0，CPR_1，CPR_2控制寄存器是否使能，最开始的八位$a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7$表示输入的八位操作数大小。

且M,S_0,S_1,S_2,S_3和C_0需要根据 74181 功能表设计，CPR_0，CPR_1，CPR_2需要根据运算中涉及到的寄存器来设计。

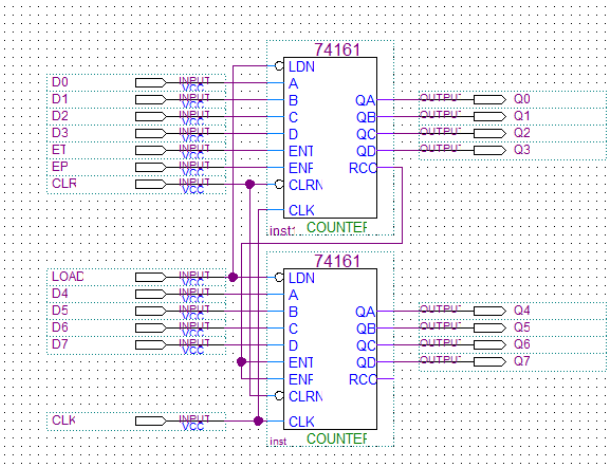
2.3 各元件设计

通过上述设计概述和设计思路可知，对于微程序控制的运算器的实现，需要使用到如下原件：

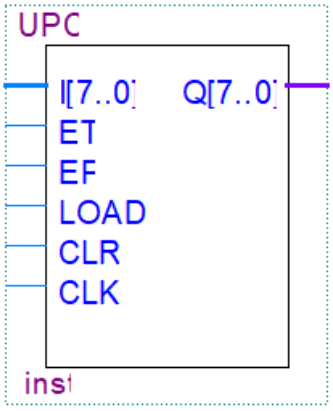
1. uPC：微程序计数器
2. CROM 微程序存储器
3. IR：指令寄存器
4. ALU：逻辑运算单元

2.3.1 微程序计数器 uPC

μPC 的功能类似于指令执行时的 PC（程序计数器），CROM 根据 μPC 的值选中相应的内存单元并打入 ALU，同时每取出一条指令，μPC 都需要自增 1。具体设计可以将两片 74161 芯片级联进行实现。如图所示：

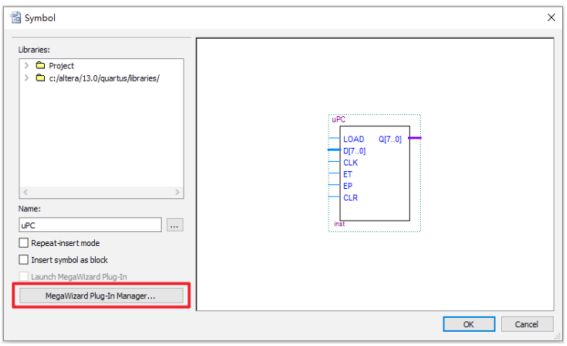


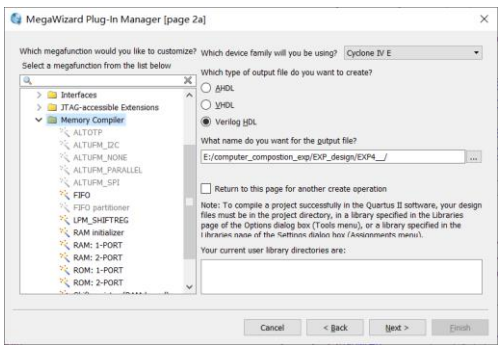
完成设计之后，为了便于之后的使用，可以将其封装为一个独立的原件，如下所示：



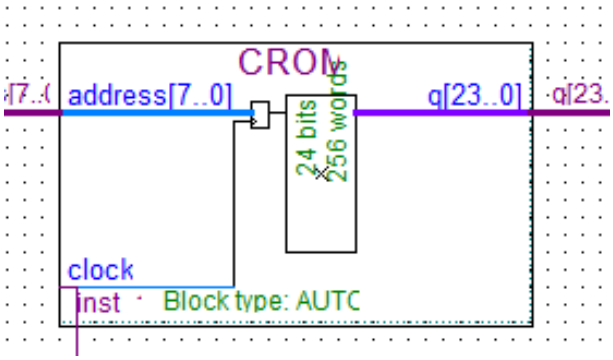
2.3.2 微程序存储器 CROM

微程序存储器用于存储具体的微指令，当 CROM 接收到具体的地址输入时，将会输出对应单元的指令。在 Quartus 平台下，1-port 的 CROM 设计流程如下所示：





最终将 CROM 的设计如下所示：

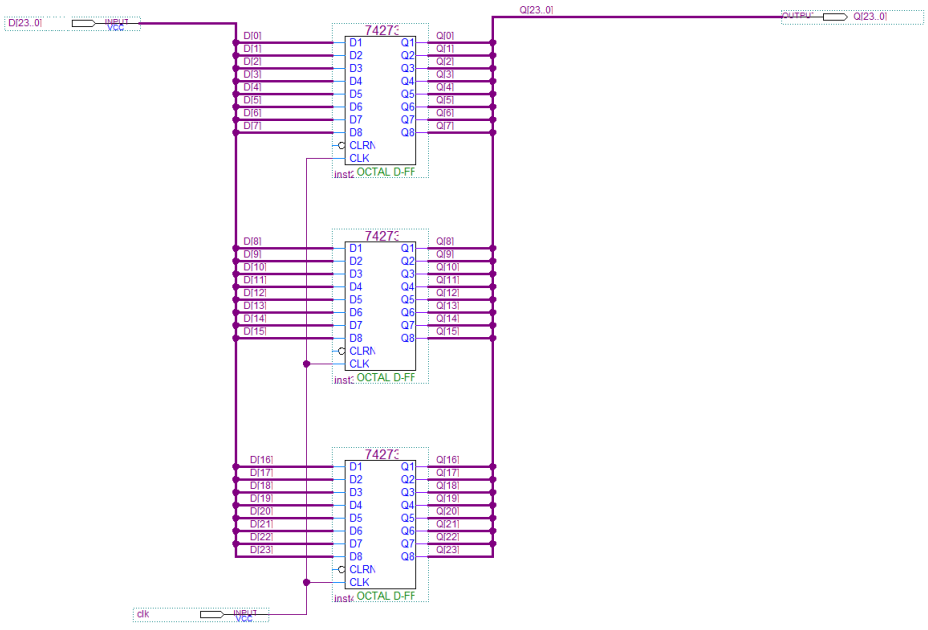


且为了便于之后实验的正确性测试，可以初始化 CROM 如下所示：

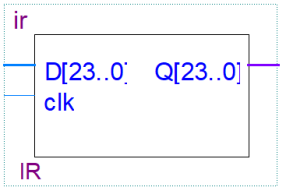
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	550080	AA0040	000129	000026	000020	00012F	000029	00003B
8	00003E	000036	00003F	00003A	000030	000035	000000	000000
16	000000	000000	000000	000000	000000	000000	000000	000000

2.3.3 IR 指令寄存器

指令寄存器 IR 用于在暂存 CROM 的输出，由 24 个 D 触发器构成，在实验中可以使用 74273 芯片进行实现。具体设计如下所示：

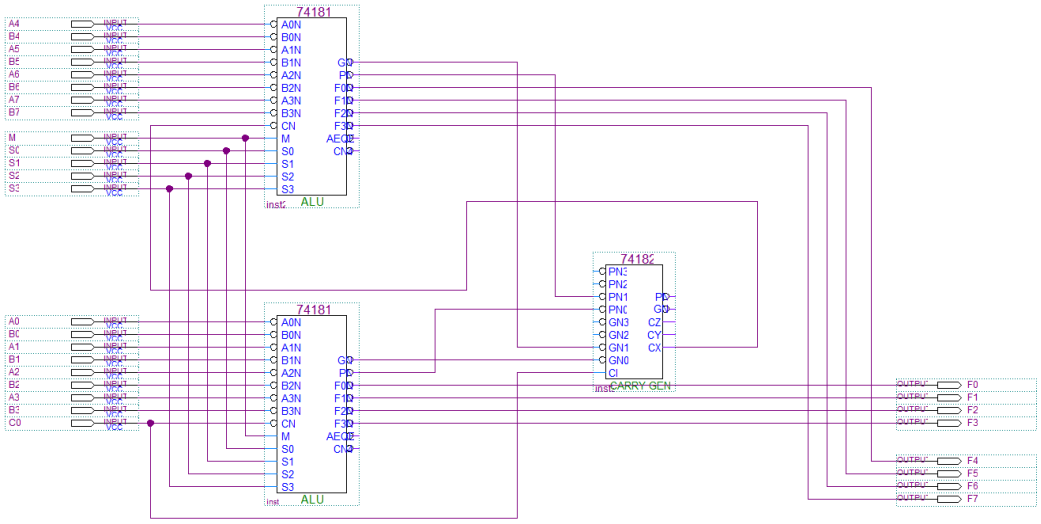


同样，为了方便之后的使用，可以将其封装为独立的原件。

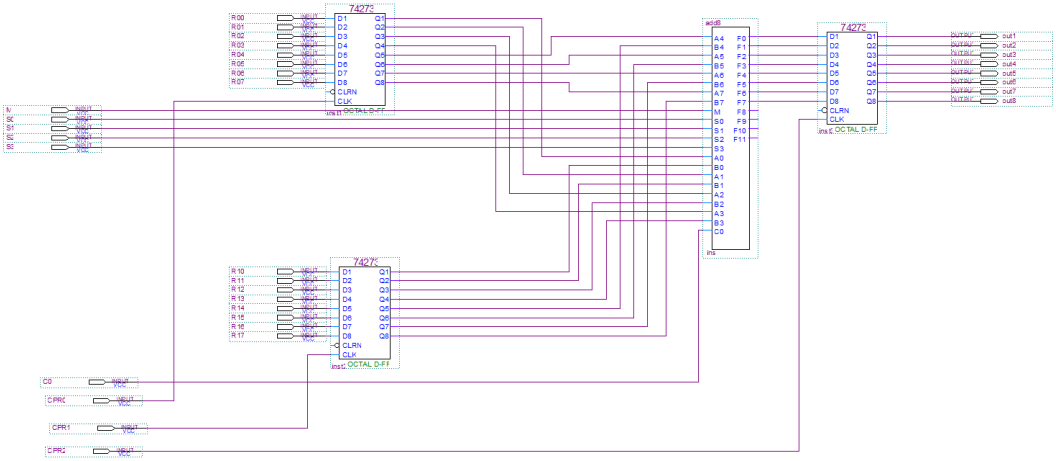


2.3.4 ALU 逻辑运算单元

根据上述设计思路可知，ALU 的设计可以采用 74181 芯片进行实现。为了设计的模块化，可以首先将 74181 芯片进行封装，将其封装为 8 位并行加法器，并在此基础上加入两个寄存器作为实际 ALU 的两个端口用于保存输入和输出。即将 ALU 的实现分为寄存器和加法器两个部分。具体逻辑电路图如下所示：



八位加法器设计



ALU 整体设计

2.4 时序分析和完整电路设计

2.4.1 时序分析

本实验需要实现微指令从读取到执行再到最终显示的过程，由于微指令执

行时存在先后顺序，因此需要控制时序保证指令的正常执行。本实验中的时序设计需要考虑一下几点问题：

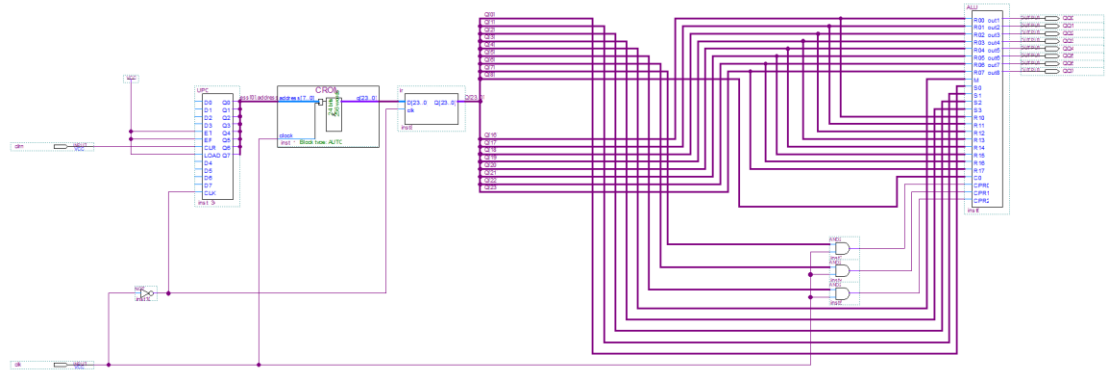
（1） μPC 的时钟应该和 CROM 的时钟相反，以保证从 CROM 中的读取出来的都是最新的 μPC 对应的指令

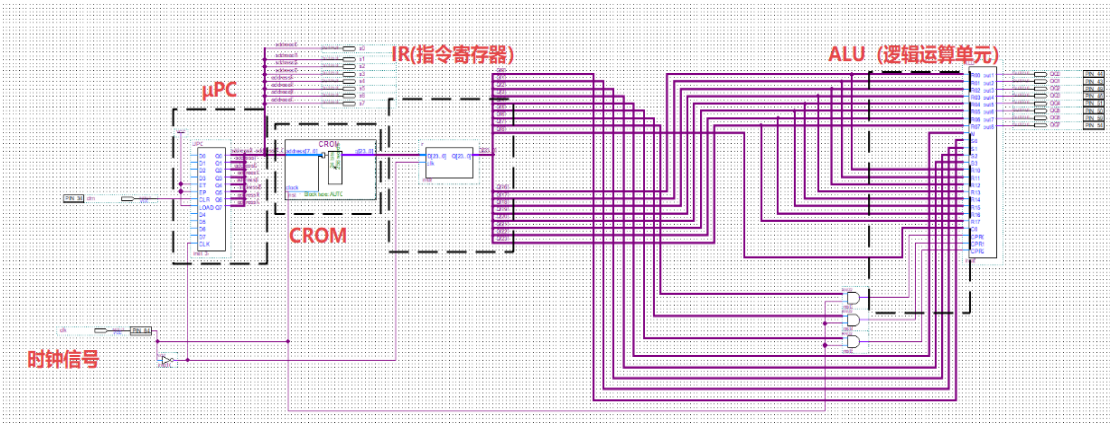
（2） CROM 的时钟应该和指令寄存器 IR 的时钟相反，以保证 IR 中存储的都是最新的 CROM 的输出。

（3） 三个寄存器的使能端应该和指令寄存器的工作时钟相反，以保证在寄存器中存储的是计算后的数据，放置寄存器先读取数据，在打入计算结果导致时序错误。

2.4.2 完整电路设计

在实现上述四个主要元件之后，可以根据指令执行的顺序和流程搭建出整体的电路框架。





2.5 正确性测试

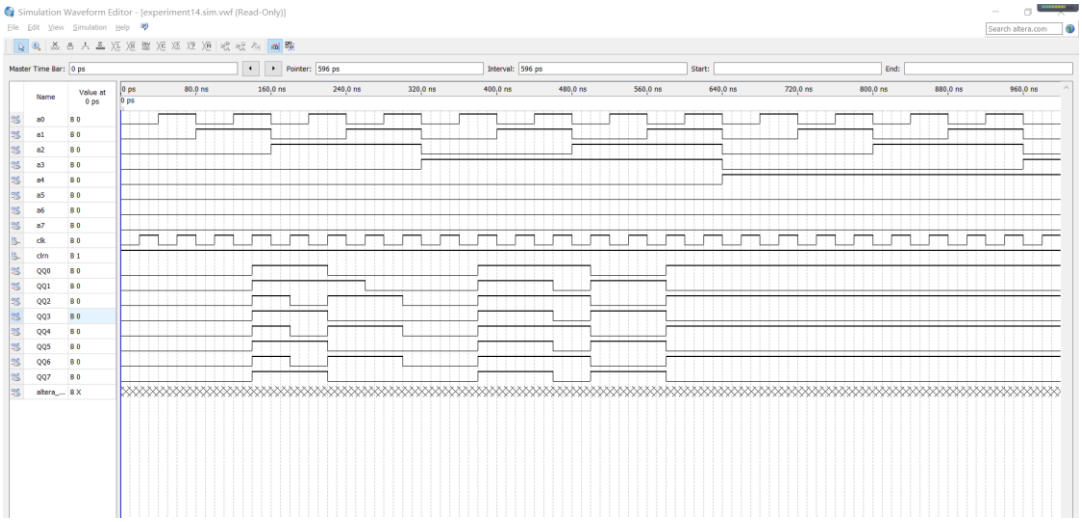
运算器的正确性测试可以分为仿真测试和实际机箱测试两部分。仿真测试

可以直观清晰的看到各个部件的输出，机箱测试可以真实的模拟运算器的输入输出。

2.5.1 仿真测试

仿真测试首先需要对内存进行初始化，此处一共存入十四指令，分别执行加减运算，逻辑运算等命令，内存存储如图所示。

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	550080	AA0040	000129	000026	000020	00012F	000029	00003B
8	00003E	000036	00003F	00003A	000030	000035	000000	000000
16	000000	000000	000000	000000	000000	000000	000000	000000



将仿真的结果和实际结果进行对比，如下表所示：

指令	指令功能	真实结果	实际执行结果
550080	将 55 赋值给R_0寄存器		
AA0040	将 AA 赋值给R_1寄存器		
000129	$R_0 + R_1 \rightarrow R_2$	FF	FF
000026	$R_0 - R_1 \rightarrow R_2$	AB	AB
000020	$R_0 + 1 \rightarrow R_2$	56	56
00012F	$R_0 - 1 \rightarrow R_2$	54	54
000029	$R_0 + R_2 + 1 \rightarrow R_2$	00	00
00003B	$R_0 \& R_1 \rightarrow R_2$	00	00
00003E	$R_0 \mid R_1 \rightarrow R_2$	FF	FF
000036	$R_0 \wedge R_1 \rightarrow R_2$	FF	FF
00003F	$R_0 \rightarrow R_2$	55	55

00003A	$R_1 \rightarrow R_2$	AA	AA
000030	$!R_0 \rightarrow R_2$	AA	AA
000035	$!R_1 \rightarrow R_2$	55	55

根据上表可知，本实验的实验结果正确。

2.5.2 实际电路测试

运算器的实际电路测试可以分为如下步骤

（1）按照实验原理图完成如下图所示的电路输入。主要包括逻辑运算单元 ALU、指令寄存器 IR、微程序控制器 CROM 以及微指令控制器 μPC 等部件。具体实现如上是原理所示。

（2）管脚锁定：完成实验原理图中输入、输出的管脚锁定

clk	Input	PIN_84	5	B5_N0	PIN_84	2.5 V (default)	8mA (default)	
clrn	Input	PIN_34	2	B2_N0	PIN_34	2.5 V (default)	8mA (default)	
QQ0	Output	PIN_44	3	B3_N0	PIN_44	2.5 V (default)	8mA (default)	2 (default)
QQ1	Output	PIN_43	3	B3_N0	PIN_43	2.5 V (default)	8mA (default)	2 (default)
QQ2	Output	PIN_49	3	B3_N0	PIN_49	2.5 V (default)	8mA (default)	2 (default)
QQ3	Output	PIN_46	3	B3_N0	PIN_46	2.5 V (default)	8mA (default)	2 (default)
QQ4	Output	PIN_51	3	B3_N0	PIN_51	2.5 V (default)	8mA (default)	2 (default)
QQ5	Output	PIN_50	3	B3_N0	PIN_50	2.5 V (default)	8mA (default)	2 (default)
QQ6	Output	PIN_59	4	B4_N0	PIN_59	2.5 V (default)	8mA (default)	2 (default)
QQ7	Output	PIN_54	4	B4_N0	PIN_54	2.5 V (default)	8mA (default)	2 (default)

（3）原理图编译、适配和下载：在 Quartus II 环境中选择 EP4CE6/10E 器件，进行原 理图的编译和适配，无误后完成下载

（4）功能测试并记录

（5）实验正确性分析

（6） 生成元件符号

实验电路图具体操作结果（实验箱测试）

A+1结果

A-1结果

直传B结果

A+B结果

通过对比可知，本实验的实验结果正确。

3 微程序控制的存储器读写系统设计

3.1 设计概述

3.1.1 设计目的

1. 了解随机存储器读写系统的结构设计
2. 熟悉随机存储器的读写操作的微程序实现
3. 掌握寄存器、MAR、PC 等部件的设计方法
4. 实践操作，设计电路图并完成测试

3.1.2 设计简述

本实验重点在于设计容量为256 × 8的随机存储器和容量为256 × 24的控制存储器 CROM，并在 RAM 和 CROM 的基础上，设计相应的外围电路和时序对随机存储器进行读写操作。根据具体实验操作本次实验可分为如下几步：

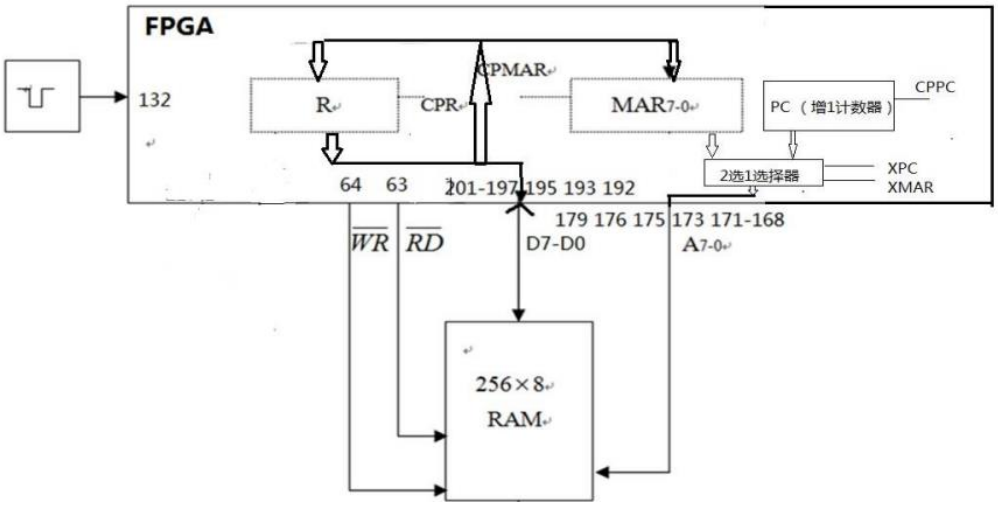
（1） 根据 PC 访问内存，取出地址 Ad1。

（2） 根据 Ad1 访问取出数据 X

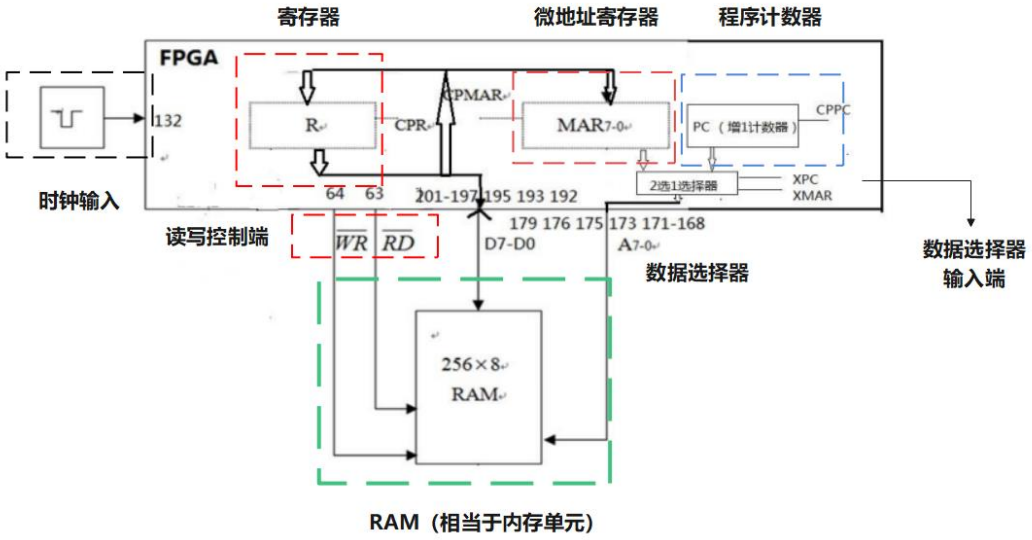
（3） 将 X 保存在 Ad2 地址单元。

3.2 设计架构

根据实验要求可知，本次实验的核心任务是设计存储器读写系统实现控制存储器的设计，因此本实验的实验框架如下所示：



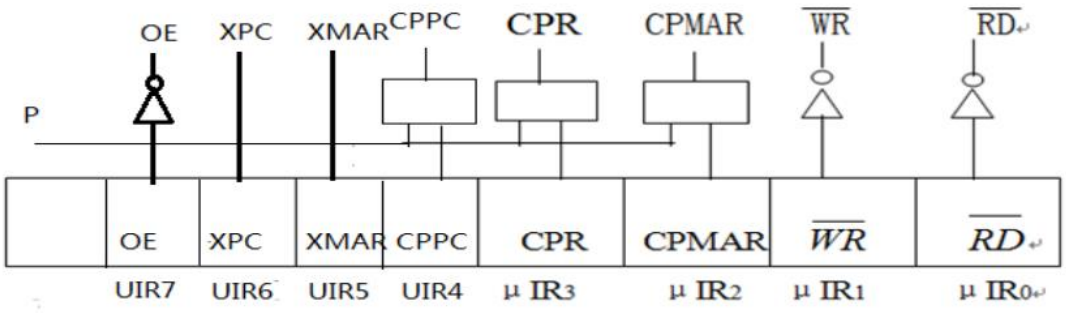
为了实现模块化的设计，可以对实验总体框架进行如下划分：



其中的主要部件有数据选择器，RAM，寄存器，MAR，PC.各种部件的作用如下表所示：

部件名称	具体作用
数据选择器	选择将 MAR 或 PC 的值
RAM	相当于内存
PC	程序计数器，根据 PC 值读取指令
MAR	微指令地址寄存器，根据 MAR 的值在 RAM 中相应地址单元进行读取
寄存器	暂存数据

对于上述架构，可以使用如下 16 位指令格式进行设计



微程序控制的存储器读写系统指令格式

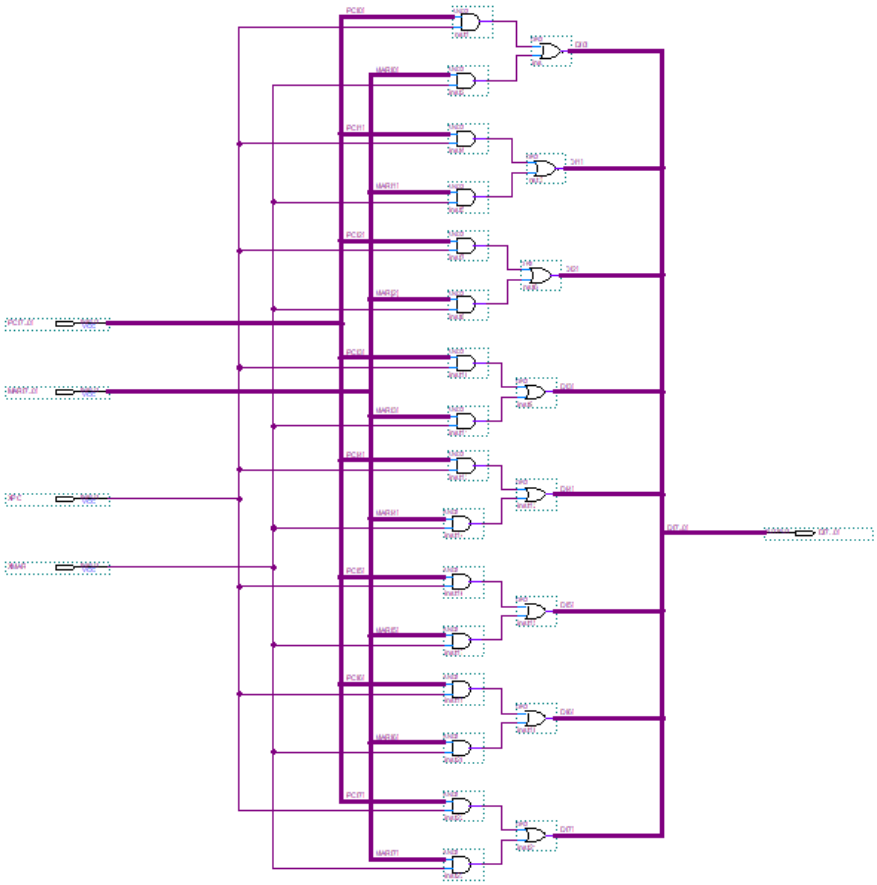
上述架构的整体逻辑是，uPC+CROM+uIR 选出一个指令，然后根据 XPC 与 XMAR 两个信号从数据选择器中选出 PC 中或 MAR 中的地址，然后根据选出的地址对 RAM 进行读取或写入。

3.3 子元件设计

根据上述分析，微程序控制的读写系统中主要用到数据选择器，RAM，寄存器，MAR，PC 四个主要子元件。同时，为了实现时序控制，也需要额外设计时钟发生器。具体设计如下所示。

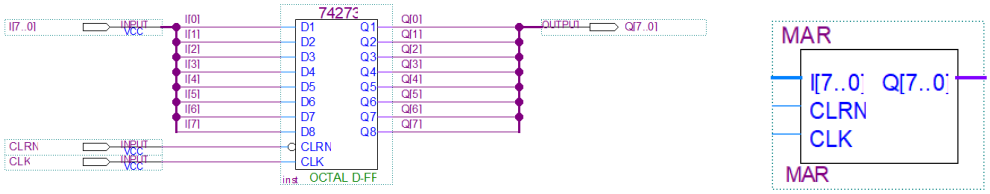
3.3.1 数据选择器结构设计

数据选择器具有两个输入信号端，分别为 XMAR 和 XPC，且 XMAR 和 XPC 中有且仅有一个值为 1,实验中可以用这两个信号实现数据的选择。在实际的设计中，这种选择功能可以通过将变量与目标的输出进行算实现。

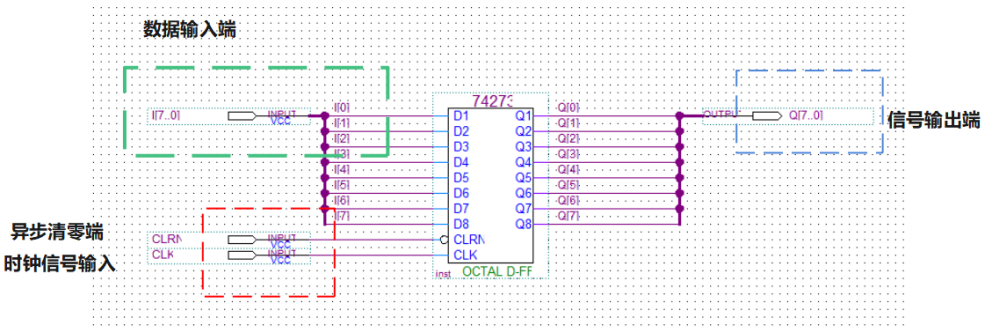


3.3.2 MAR 结构设计

结合实验要求可知，实验操作中需要首先将地址 ad1 送入 MAR 中，并在第二条微指令中将 MAR 中的值输入 RAM，从中选出对应地址单元的。因此，可以通过输入时钟的上升沿和下降沿控制 MAR 是否工作，例如当输入的时钟 CLK 为上升沿时 MAR 工作，当输入的 CLK 为下降沿时，MAR 不工作，从而实现 MAR 时钟的控制。具体电路实现如下图所示：

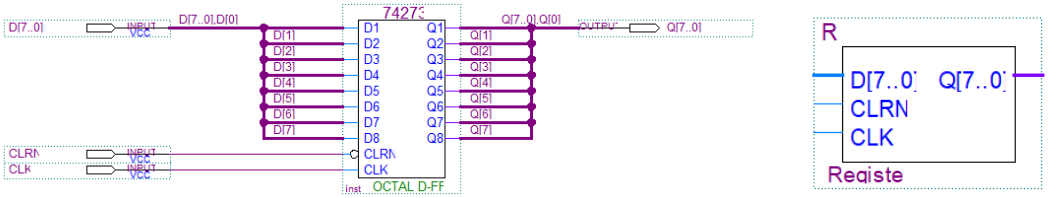


电路中输入输出的含义如下所示：

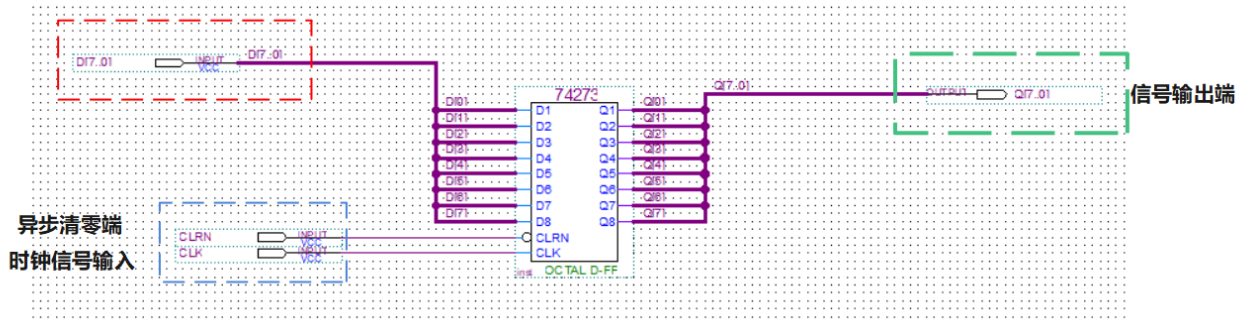


3.3.3 寄存器结构设计

本实验中寄存器的功能较为简单，主要是暂存数据，因此可以采用和 MAR 同样的设计思路，使用 74273 芯片进行实现。具体结构如下所示：具体电路结构如图：

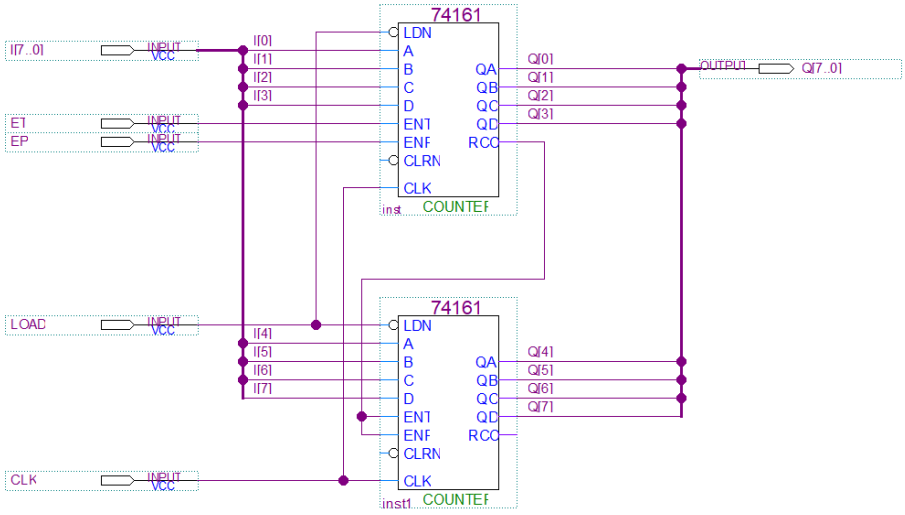


电路中每一部分输入输出含义如下所示：

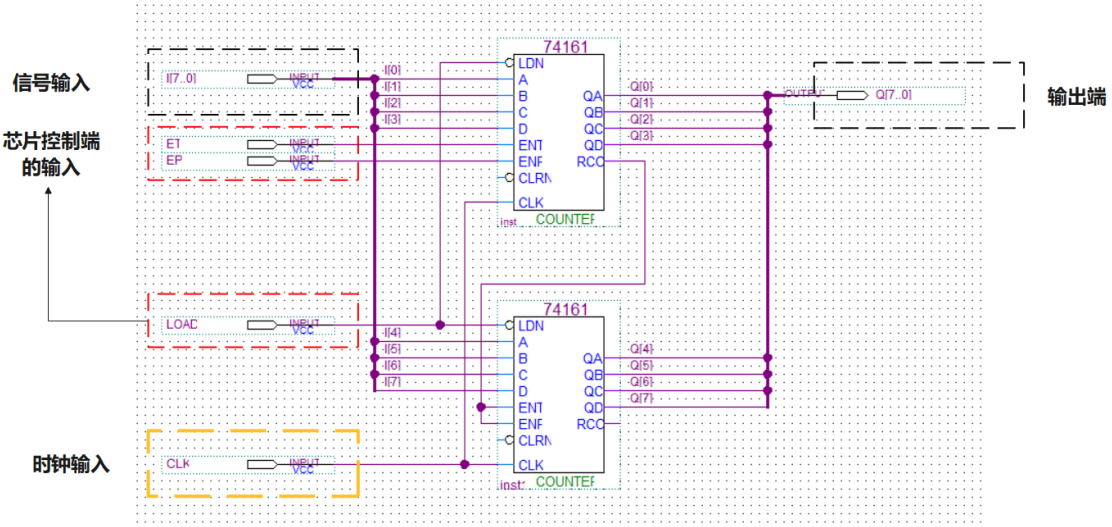


3.3.4 PC 结构设计

在本实验中，程序计数器 PC 的设计和实验一中 μPC 的设计类似，都需要使用 74161 芯片进行设计，且需要实现自增一功能，是 PC 永远指向下一条需要访问的指令。具体设计结构如下所示：

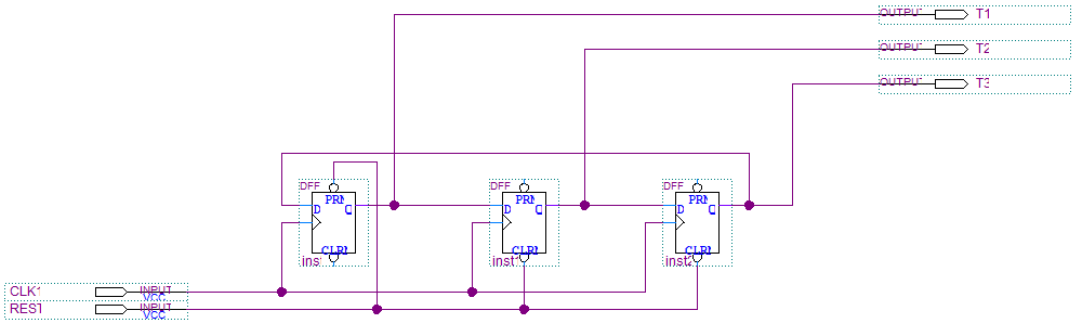


输入输出的含义如下所示：

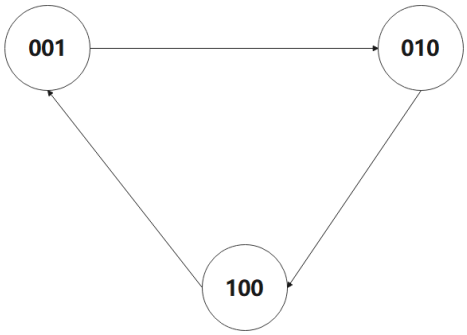


3.3.5 时钟发生器

根据实验要求，结合具体的操作可知，本实验中共需要三个时钟信号分别控制微指令存储器 CROM、μPC 和指令寄存器，RAM 以及三个使能端。为了方便实现时钟的控制，可以设计相应的时钟发生器，将原始输入的时钟 CLK 分成三个不同相位的时钟信号，实现时钟的分时控制。具体的实现可以采用 D 触发器，将三个 D 触发器顺次相连，同时首尾相接构成一个环，在具体工作时将初始打入的 1 循环传递给每一个寄存器。具体设计图如下所示：



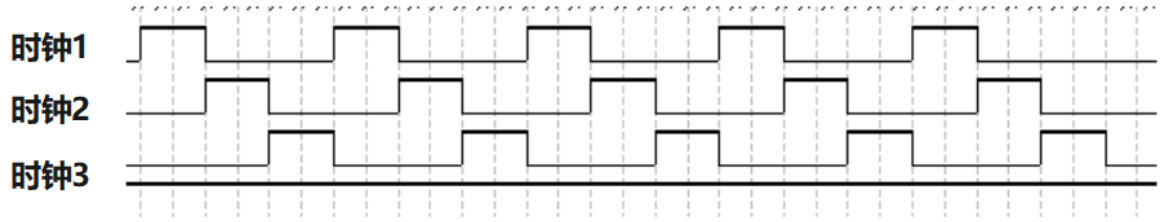
通过上述设计，输出端$T_1T_2T_3$的状态转换为$100 \rightarrow 010 \rightarrow 001 \rightarrow 100 \rightarrow 010$ ...，如此循环，如下所示：



3.4 完整电路图设计和时序分析

3.4.1 时序分析

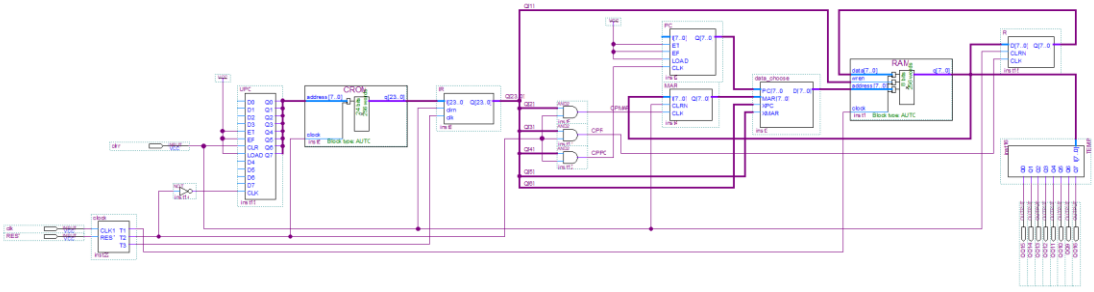
根据上述实验原理分析可知，在本实验中需要控制三组时钟以便实现整体的时序设计。通过 D 触发器可以设计相应的时钟发生器产生相应的时钟信号。具体时钟的波形如下所示：



其中时钟 1 用于控制 μPC，CROM 和指令寄存器 IR，先完成指令的读取。之后，有时钟 2 控制程序计数器、MAR 以及寄存器 R 的使能，最后由时钟 3 控制 RAM 完成内存中的读写操作。

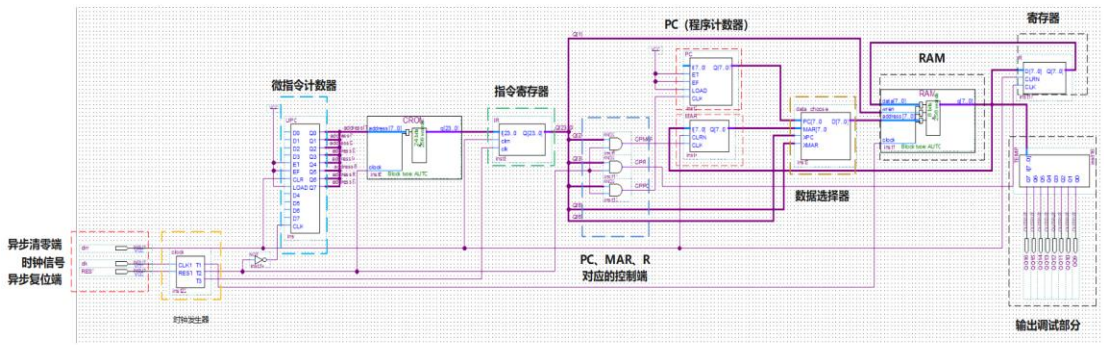
3.4.2 完整电路设计

在上述各个子元件设计完成的基础上，再根据最开始给出的架构设计方案，即可搭建出我们 所要设计的架构——微程序控制的存储器读写系统，具体电路如下图所示。



实验完整逻辑电路图

实验电路图中各部分的名称详解如下图所示：



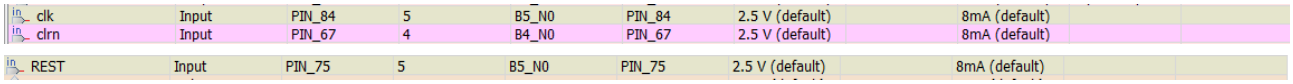
实验完整逻辑电路图详解

3.4.3 实验正确性测试

实际实验中，微程序控制的存储器读写系统可以按照如下步骤进行测试

（1）按照实验原理图完成如下图所示的电路输入。主要包括实验一中设计到的指令寄存器 IR、微程序控制器 CROM、微指令控制器 μPC，以及实验二中新增的 PC、RAM、寄存器 R 以及 MAR 等部件。具体实现如上是原理所示：

（2）管脚锁定：完成实验原理图中输入、输出的管脚锁定具体实现通过参照引脚对照表，分配相应的引脚实现。具体图示如下：



clk	Input	PIN_84	5	B5_N0	PIN_84	2.5 V (default)	8mA (default)
clrn	Input	PIN_67	4	B4_N0	PIN_67	2.5 V (default)	8mA (default)
REST	Input	PIN_75	5	B5_N0	PIN_75	2.5 V (default)	8mA (default)

（3）原理图编译、适配和下载：在 Quartus II 环境中选择 EP4CE6/10E 器件，进行原 理图的编译和适配，无误后完成下载

（4）功能测试并记录：

（5）生成元件符号

3.4.4 实验总结

本实验主要目的是在 RAM 和 CROM 的基础上，设计相应的外围电路和时序实现对随机存储控制器进行读写操作。外围电路主要包括程序计数器，微指令计数器，寄存器等，时序则是指各个部件在沿信号或者电平信号的触发下按照一定的秩序的进行工作。

4 基于微程序的简单模型机设计

4.1 思路

模型机的核心部件为 CU 和 RAM，其中 CU 包括 CROM 和 μPC，其中 CROM 按顺序存储着每条指令的微指令序列，例如取值指令 SELECT、相加指令 ADD 等，μPC 控制对 CU 中微指令的读取。RAM 中存储着我们需要的指令序列，也就相当于我们编写的汇编代码，同时 RAM 也可以存储数据。

每次，我们从 RAM 中读取一条指令 A，根据该指令，我们找到该指令对应的微指令序列在 CU 中的入口地址，这通过设定 μPC 实现，μPC 控制我们读取 CROM 中的哪一条微指令。当我们依次读取指令 A 对应的微指令并执行后，指令 A 就执行完毕，我们将继续执行下一条指令。下一条指令一般是指令 A 相邻的指令，也有可能是其他位置的指令，这需要通过跳转指令实现。

对于每条指令的微指令序列，其包括的微指令条数各不相同。简单来说，每条微指令是 24 位 01 序列，每一位都代表一个控制信号，例如激活寄存器 R0、执行加法运算等，因此每一条微指令都可以执行某些操作，称其为微操作，而一条指令就是通过数条微指令代表的数次微操作相配合，从而执行该条指令代表的操作。

此外，为了实现递归，我们在 RAM 之外增加了栈 SRAM。递归的工作为保护现场，并跳转到目标位置。因此递归指令 CALL 的工作为保存寄存器 R0、R1 的值和当前 PC 的值，放入 SRAM 中。递归子程序执行结束后，返回到断点处继续执行，此时执行返回指令 RETURN，其工作为恢复寄存器 R0、R1 和 PC 的值。

4.2 指令设计

4.2.1 指令格式

- 存储器 RAM 容量为 256*8 位，因此指令基本字长设计为 8 位。
- 指令格式可以有单字长指令和双字长指令。
- 对于单字长指令，8 位数据就代表该指令，也就是操作码。
- 对于双字长指令，前 8 位数据代表该指令，也就是操作码，而后 8 位代表该指令使用到的数据。该数据可以代表操作数，也可能带代表操作数地址。

4.2.2 指令编码

实验中各条指令编码如下表所示：

指令缩写	指令编码	指令字长	指令含义
SELECT	0000 0000	单字长	取值指令
LW0	0000 1000	双字长	立即数->R0
LW1	0000 1100	双字长	立即数->R1
LW2	0001 0000	双字长	[地址]->R0
LW3	0001 0100	双字长	[地址]->R1
SW0	0001 1000	双字长	R0->[地址]
SW1	0001 1100	双字长	R1->[地址]
BEQ	0010 0000	双字长	符合条件跳转
BNE	0010 0100	双字长	符合条件跳转
JR	0010 1000	双字长	直接跳转
ADD0	0010 1100	双字长	R0+[地址]->R0
ADD1	0011 0000	双字长	R1+[地址]->R1
SUB0	0011 0100	双字长	R0-[地址]->R0
SUB1	0011 1000	双字长	R1-[地址]->R1
DIV0	0011 1100	双字长	R0/[地址]->R0
DIV1	0100 0000	双字长	R1/[地址]->R1
MUL0	0100 0100	双字长	R0*[地址]->R0
MUL1	0100 1000	双字长	R1*[地址]->R1
AND0	0100 1100	双字长	R0&[地址]->R0
AND1	0101 0000	双字长	R1&[地址]->R1
OR0	0101 0100	双字长	R0|[地址]->R0
OR1	0101 1000	双字长	R1|[地址]->R1
XOR0	0101 1100	双字长	R0^[地址]->R0
XOR1	0110 0000	双字长	R1^[地址]->R1
LM0	0110 0100	单字长	R0 左移->R0
LM1	0110 1000	双字长	[地址]左移->R0
RM0	0110 1100	单字长	R0 右移->R0
RM1	0111 0000	双字长	[地址]右移->R0
CALL	0111 0100	双字长	中断，跳转
RETURN	1000 0000	单字长	返回中断处
HALT	1000 1000	单字长	停止

4.2.3 指令功能解析

实验中各条指令所写以及对应功能说明如下表所示：

指令缩写	指令功能
SELECT	根据 PC 从 RAM 取出指令，放入 IR

LW0	数据字段为立即数，放入 R0
LW1	数据字段为立即数，放入 R1
LW2	数据字段为地址，从 RAM 中取出数据，放入 R0
LW3	数据字段为地址，从 RAM 中取出数据，放入 R1
SW0	数据字段为地址，将 R0 数据存入 RAM 中对应部分
SW1	数据字段为地址，将 R1 数据存入 RAM 中对应部分
BEQ	数据字段为地址 A，R0 为 0，则 PC 为 A
BNE	数据字段为地址 A，R0 不为 0，则 PC 为 A
JR	数据字段为地址 A，PC 无条件跳转为 A
ADD0	数据字段为地址 A，R0 加[A]，结果存回 R0
ADD1	数据字段为地址 A，R1 加[A]，结果存回 R1
SUB0	数据字段为地址 A，R0 减[A]，结果存回 R0
SUB1	数据字段为地址 A，R1 减[A]，结果存回 R1
DIV0	数据字段为地址 A，R0 除[A]，结果存回 R0
DIV1	数据字段为地址 A，R1 除[A]，结果存回 R1
MUL0	数据字段为地址 A，R0 乘[A]，结果存回 R0
MUL1	数据字段为地址 A，R1 乘[A]，结果存回 R1
AND0	数据字段为地址 A，R0 与[A]，结果存回 R0
AND1	数据字段为地址 A，R1 与[A]，结果存回 R1
OR0	数据字段为地址 A，R0 或[A]，结果存回 R0
OR1	数据字段为地址 A，R1 或[A]，结果存回 R1
XOR0	数据字段为地址 A，R0 异或[A]，结果存回 R0
XOR1	数据字段为地址 A，R1 异或[A]，结果存回 R1
LM0	R0 数据左移 1 位，结果存回 R0
LM1	数据字段为立即数 A，A 左移 1 位，结果存回 R0
RM0	R0 数据右移 1 位，结果存回 R0
RM1	数据字段为立即数 A，A 右移 1 位，结果存回 R0
CALL	数据字段为地址 A，R0、R1 和 PC 值保存入栈，PC 跳转到 A
RETURN	恢复 R0、R1 和 PC 值
HALT	停机

4.3 微指令

4.3.1 基本思路

每条指令的最终执行效果是由多条微指令的执行效果配合而成。每条指令的操作码就对应了该指令的微操作序列在 ROM 中的首地址，因此可以通过指令操作码（前 8 位）在 ROM 中定位到对应的第一个微指令。

PC 中的值就是我们要读取的指令在 RAM 中的地址，对于任意指令，我们需要根据 PC 的值，从 RAM 中将指令读取并放入 IR，同时设置 μPC 为指令对应的

微操作序列在 CROM 中的地址。这就是取值指令 SELECT 的功能。

我们将取值指令对应的微操作序列放到 CROM 的起始位置，在每条指令执行之前，我们都会执行一次 SELECT 操作用于将指令取出，之后再执行被取出的指令。

4.3.2 微指令每位解析

数位	功能		
inst[23]	0：正常时钟	1：时钟停止	
inst[22]	0：不做操作	1： μPC 清零	
inst[21]	0： μPC 加 1	1： μPC 装入数据	
inst[20]	预留		
inst[19]	000：不做操作	001：激活 IR	010：激活 MAR
inst[18]	011：激活 R1	100：激活 R0	101：激活 SP
inst[17]	110：不做操作	111：不做操作	
inst[16]	00：选择 ALU 结果	01：选择 A_DATA	
inst[15]	10：选择 B_DATA	11：不做操作	
inst[14]	0000：加法操作	0001：减法操作	0010：与操作
inst[13]	0011：或操作	0100：异或操作	0101：乘法操作
inst[12]	0110：除法操作	0111：右移操作	1000：左移操作
inst[11]	其他：不做操作		
inst[10]	A_DATA 部分：		
inst[9]	00：选择 PC	01：选择 RAM/SRAM 数据	10：选择 01
inst[8]	B_DATA 部分		
inst[7]	00：选择 R0	01：选择 R1	10：选择 SP
	11：不做操作		
inst[6]	0：选择 RAM 数据	1：选择 SRAM 数据	
inst[5]	0：读 SRAM	1:写 SRAM	
inst[4]	0：读 RAM	1：写 RAM	
inst[3]	0：不做操作	1：JR 有效	
inst[2]	0：不做操作	1：BEQ 有效	
inst[1]	0：不做操作	1：BNE 有效	
inst[0]	0：不做操作	1：激活 PC 时钟，PC+1 或打入数据	

4.3.3 指令解析

（1）取值指令 SELECT：

SELECT 取出指令 A，并放入 IR 中，并根据指令的操作码（指令前 8 位）得到微指令序列的入口地址。最后一条微指令是将微指令序列入口地址打入 μPC，从而使 CROM 接下来取出 A 的第一条微指令。

指令缩写	微指令代码	微指令对应微操作
SELECT	00000101 00000000 00000000	PC->MAR
	00000011 00000010 00000001	RAM(MAR)->IR PC = PC+1
	00100000 00000000 00000000	指令入口地址->μPC

（2）取数指令 LW：

在每个指令的最后一个微操作为 0->μPC，即将 μPC 的值设置为 0，而这就是取值指令 SELECT 的第一条微指令在 CROM 中的地址，因此读取的下一条微指令就是 SELECT 的第一条微指令，也就是每个指令执行完后，将执行取值指令，取出下一条指令。

指令缩写	微指令代码	微指令对应微操作
LW0	00000101 00000000 00000000	PC->MAR
	01001001 00000010 00000001	RAM(MAR)->R0 PC = PC+1 0->μPC
LW1	00000101 00000000 00000000	PC->MAR
	01000111 00000010 00000001	RAM(MAR)->R1 PC = PC+1 0->μPC
LW2	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
	01001001 00000010 00000001	RAM(MAR)->R0 PC = PC+1 0->μPC
LW3	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
	01000111 00000010 00000001	RAM(MAR)->R1 PC = PC+1 0->μPC

（3）存数指令 SW：

指令所	微指令代码	微指令对应微操作
SW0	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR

	01000000 10000000 00010001	R0->RAM(MAR) PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
SW1	01000000 10000000 10010001	R1->RAM(MAR) PC = PC+1 0-> μPC

（4）跳转指令：

指令缩写	微指令代码	微指令对应微操作
BEQ	00000101 00000000 00000000	PC->MAR
	01000001 00000010 00000101	if R0 == 0 then RAM(MAR)->PC else then PC = PC+1
BNE	00000101 00000000 00000000	PC->MAR
	01000001 00000010 00000011	if R0 != 0 then RAM(MAR)->PC else then PC = PC+1
JR	00000101 00000000 00000000	PC->MAR
	01000001 00000010 00001001	RAM(MAR)->PC 0-> μPC

（5）加法指令：

指令缩写	微指令代码	微指令对应微操作
ADD0	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
	01001000 00000010 00000001	R0 + RAM(MAR)->R0 PC = PC+1 0-> μPC
ADD1	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
	01000110 00000010 10000001	R1 + RAM(MAR)->R1 PC = PC+1 0-> μPC

（6）减法指令：

指令缩写	微指令代码	微指令对应微操作
SUB0	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR

		R0 - RAM(MAR)->R0
	01001000 00001010 00000001	PC = PC+1
		0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
SUB1		R1 - RAM(MAR)->R1
	01000110 00001010 10000001	PC = PC+1
		0-> μPC

（7）除法指令：

指令缩写	微指令代码	微指令对应微操作
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
DIV0		R0 / RAM(MAR)->R0
	01001000 00110010 00000001	PC = PC+1
		0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
DIV1		R1 / RAM(MAR)->R1
	01000110 00110010 10000001	PC = PC+1
		0-> μPC

（8）乘法指令：

指令缩写	微指令代码	微指令对应微操作
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
MUL0		R0 x RAM(MAR)->R0
	01001000 00101010 00000001	PC = PC+1
		0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
MUL1		R1 x RAM(MAR)->R1
	01000110 00101010 10000001	PC = PC+1
		0-> μPC

（9）相与指令：

指令缩写	微指令代码	微指令对应微操作

	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
AND0	01001000 00010010 00000001	R0 & RAM(MAR)->R0 PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
AND1	01000110 00010010 10000001	R1 & RAM(MAR)->R1 PC = PC+1 0-> μPC

（10）相或指令：

指令缩写	微指令代码	微指令对应微操作
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
OR0	01001000 00011010 00000001	R0 | RAM(MAR)->R0 PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
OR1	01000110 00011010 10000001	R1 | RAM(MAR)->R1 PC = PC+1 0-> μPC

（11）异或指令：

指令缩写	微指令代码	微指令对应微操作
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
XOR0	01001000 00100010 00000001	R0 ^ RAM(MAR)->R0 PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000000	RAM(MAR)->MAR
XOR1	01000110 00100010 10000001	R1 ^ RAM(MAR)->R1 PC = PC+1 0-> μPC

（12）左移指令：

指令缩写	微指令代码	微指令对应微操作
LM0	01001000 01000000 00000001	R0 << ->R0 PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001000 10000010 00000000	RAM(MAR)->R0
LM1	01001000 01000000 00000001	R0 << ->R0 PC = PC+1 0-> μPC

（13）右移指令：

指令缩写	微指令代码	微指令对应微操作
RM0	01001000 00111000 00000001	R0 >> ->R0 PC = PC+1 0-> μPC
	00000101 00000000 00000000	PC->MAR
	00001000 10000010 00000000	RAM(MAR)->R0
RM1	01001000 00111000 00000001	R0 >> ->R0 PC = PC+1 0-> μPC

（14）CALL 指令：

指令缩写	微指令代码	微指令对应微操作
CALL	00000000 10000000 00100000	R0->SRAM(SP)
	00001010 00000101 00000000	SP = SP+1
	00000000 10000000 10100000	R1->SRAM(SP)
	00001010 00000101 00000000	SP = SP+1
	00000101 00000000 00000000	PC->MAR
	00001001 00000010 00000001	RAM(MAR)->R0 PC = PC+1
	00000001 00000000 00100000	PC->SRAM(SP)
	00001010 00000101 00000000	SP = SP+1
	01000000 10000000 00001001	R0->PC 0-> μPC

（15）RETURN 指令：

指令缩写	微指令代码	微指令对应微操作
RETURN	00001010 00001101 00000000	SP = SP-1

00000001 00000010 01001001	SRAM(SP)->PC
00001010 00001101 00000000	SP = SP-1
00000111 00000010 01000000	SRAM(SP)->R1
00001010 00001101 00000000	SP = SP-1
01001001 00000010 01000000	SRAM(SP)->R0 0->μPC

（16）HALT 指令：

指令缩写	微指令代码	微指令对应微操作
HALT	10000000 00000000 00000000	停止

4.4 总体结构与数据通路

4.4.1 总述

经过上述对每条微指令对应的微操作分析，可以看到，几乎所有的微操作都是进行数据的转移，例如 PC->MAR 代表将 PC 中的数据存入 MAR，RAM(MAR)->R0 代表将 RAM 中的数据存入 R0 中，可以称其为直接转移微操作。此外还有类似于 R1 + RAM(MAR)->R1 的微操作，该微操作将 R1 与 RAM 中的数据相加，并将结果存入 R1，可以称其为算数运算转移微操作。

4.4.2 数据转移源操作数

分析所有的算数运算转移微操作 A 运算 B -> C，其中注意微操作 SP = SP+1 和 SP = SP-1，SP 为保存栈 SRAM 地址的寄存器，该微操作相加和相减要经过 ALU，因此我们可以分别写为 SP+1 -> SP 和 SP-1 -> SP，SP 为操作数 A，1 为操作数 B。而对于 PC = PC+1 微操作，其相加是通过时钟给出而不是 ALU，因此不能作为 A 运算 B -> C 形式。

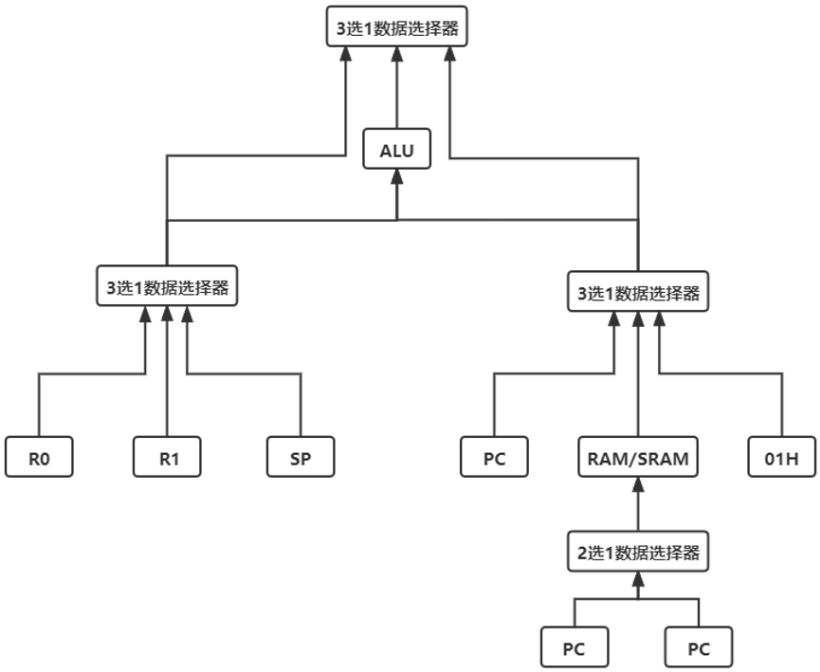
通过分析，我们发现操作数 A 有：R0、R1 和 SP，而操作数 B 有：PC、RAM/SRAM 和 01。我们可以将 R0、R1 和 SP 作为 A_DATA 组，PC、RAM/SRAM 和 01 设置为 B_DATA 组，每次我们只选择 A_DATA 和 B_DATA 中个选择一个数据，并输入 ALU，将计算结果输入对应位置。我们需要设计三选一数据选择器从三个数据中选择一个数据。

对于直接转移微操作，即类似于 PC->MAR 的 A->C 微操作，操作数 A 有 R0、R1、RAM/SRAM、PC，与上述算数运算转移微操作的操作数重复，但是不需要经过 ALU 运算，因此我们可以再增加一个三选一数据选择器，用于从 A_DATA、B_DATA 和 ALU 结果中选择数据。

此外，对于 SRAM 和 RAM，我们不会同时需要它们的数据，因此在使用 RAM

或 SRAM 的数据作为数据转移微操作的源操作数时，我们先将 RAM 和 SRAM 的数据通过一个二选一数据选择器，再将选出来的数据接入接下来的三选一数据选择器。

因此部分数据通路如下图所示：



4.4.3 数据转移目的操作数

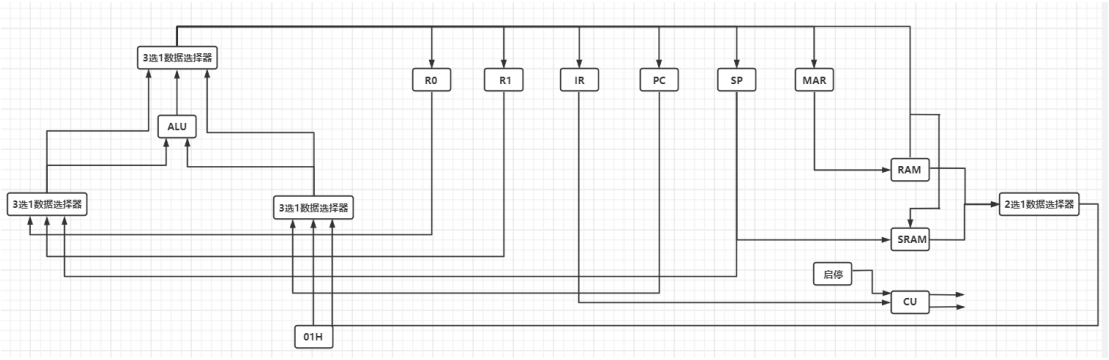
此前我们讨论了数据转移微操作的源操作数，接下来我们分析数据转移微操作的目的操作数。

通过分析所有的数据转移微操作，目的操作数也就是被写入数据的部分有：R0、R1、IR、PC、SP、MAR 以及 RAM/SRAM。对于以上所有部件，当我们希望写入数据时，我们只需要使用时钟信号将部件激活即可，这里通过微指令的部分字段与时钟信号相与，即可在特定时刻激活特定元件。

IR 寄存器中存储从 RAM 中取到的指令前 8 位（操作码），因此 IR 将连接 CU，从而 CU 能够通过 IR 数据输出指令对应的微指令序列。

CU 是控制器，因此启停装置连接 CU，CU 停止，则机器停止。

总体数据通路如下图：



4.4.4 时序安排

讨论模型机整体架构后，我们还需要为模型机安排时序。

对于 CU，其包括 μPC 和 CROM， μPC 需要时钟信号更新数据，CROM 需要时钟信号读取微指令。我们设置 T0 时钟信号控制 CU，其中正时序控制 CROM，逆时序控制 μPC。

按照数据转移指令，由于 SRAM 或 RAM 中的数据也会被转移，因此我们需要先将 SRAM 和 RAM 中的数据取出，以备后续使用。因此 T1 时钟信号将控制 SRAM 和 RAM 的数据读取。

对于 A_DATA 和 B_DATA 的三选一数据选择器，我们使用 T2 时钟信号控制 A_DATA，使用 T3 时钟信号控制 B_DATA。

对于 A_DATA、B_DATA 和 ALU 结果的三选一数据选择器，我们使用 T4 时钟信号控制。

最后，可能有数据写入 SRAM 或 RAM，这需要时钟信号；同时可能有数据写入 R0、R1 等各项寄存器，同样需要时钟信号将寄存器激活，我们使用 T5 时钟信号控制。

最终时序安排见下表：

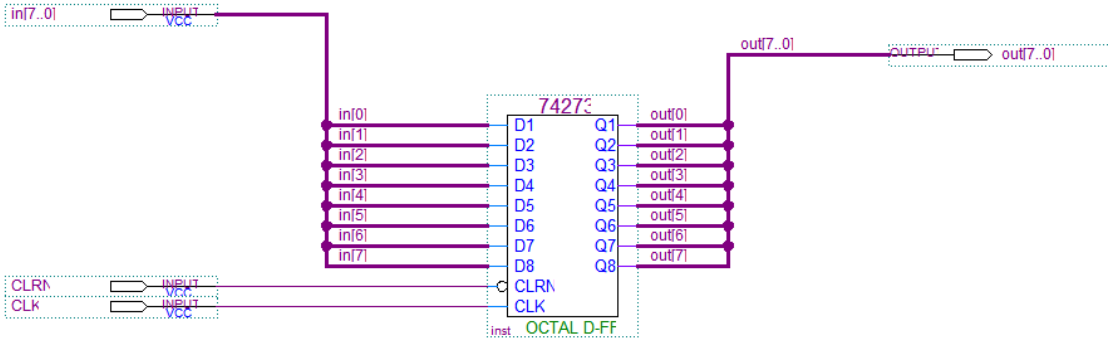
时序	功能
T0	正时序控制 μPC，负时序控制 CROM
T1	控制 RAM/SRAM 数据读取
T2	控制 A_DATA 三选一数据选择器
T3	控制 B_DATA 三选一数据选择器
T4	控制 A_DATA、B_DATA、ALU 结果的三选一数据选择器
T5	控制 RAM/SRAM 数据写入

4.4.5 寄存器介绍：

各数据寄存器功能如下表所示：

寄存器名称	功能
R0	8 位数据通用寄存器
R1	8 位数据通用寄存器
IR	8 位指令寄存器，存储单字长指令或双字长指令前 8 位
MAR	8 位地址寄存器，存储 RAM 地址
SP	8 位栈指针寄出去你，存储 SRAM 地址
PC	带置数的 8 位指令计数器，存储 RAM 的地址

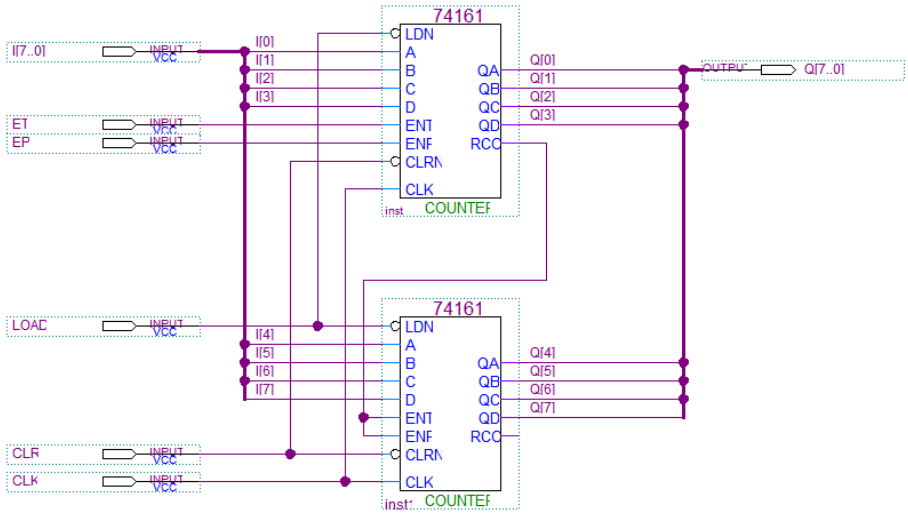
数据寄存器均为 8 位数据长度，可直接在 74273 基础上封装得到。具体结构如下图所示：



对于 PC，同样是 8 位长度，可以存储数据，但是要实现置数功能，即可以通过接收时钟信号实现加 1 或置数功能。

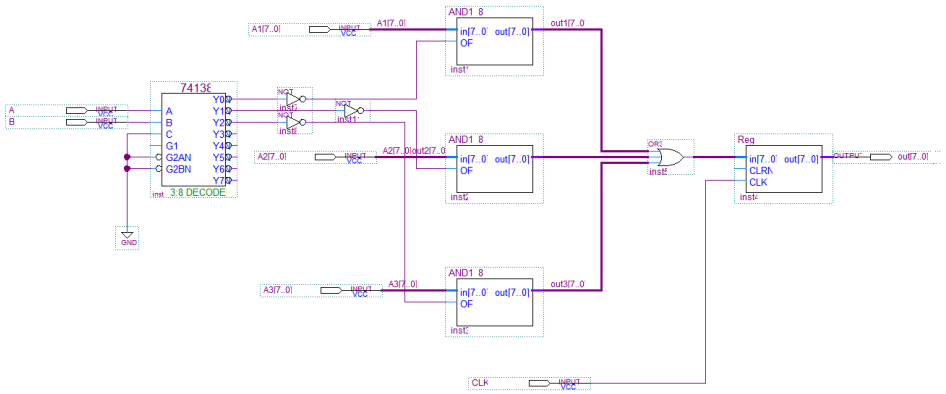
由于 PC 在接收时钟信号时，可以加 1 也可以置数，这里是通过 LOAD 输入确定。当 LOAD 输入为高电位时，PC 接收时钟信号会加 1；当 LOAD 输入为低电位时，PC 接收时钟信号会置数，即打入 I[7..0]数据。

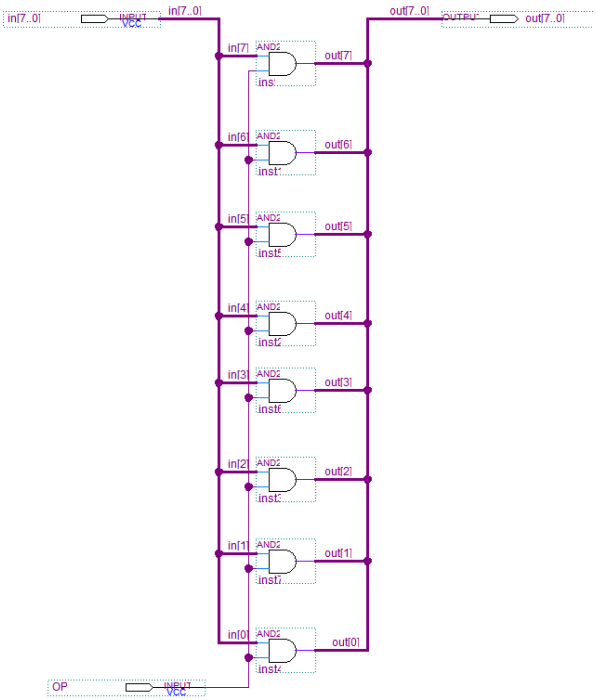
具体结构见下图：



4.4.6 其他部件介绍：

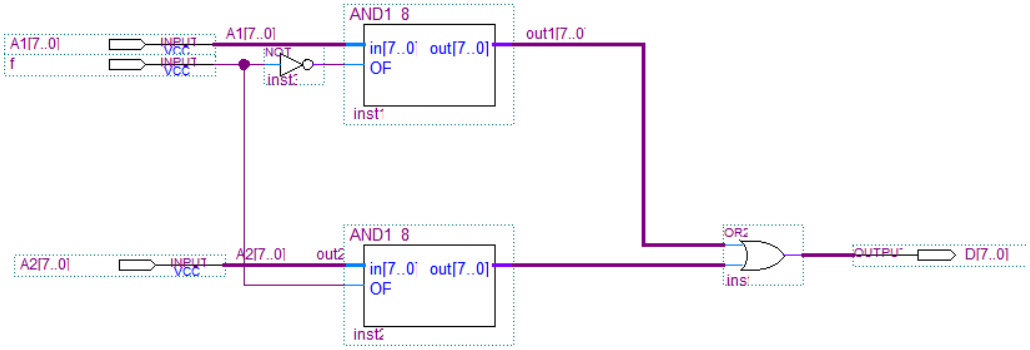
- 三选一数据选择器：





其中，A1、A2 和 A3 是我们需要选择的数据，AB 是控制码，AB=00 时，选择 A1；AB=01 时，选择 A2；AB=10 时，选择 A3。最后的结果将存于 8 位通用寄存器中。

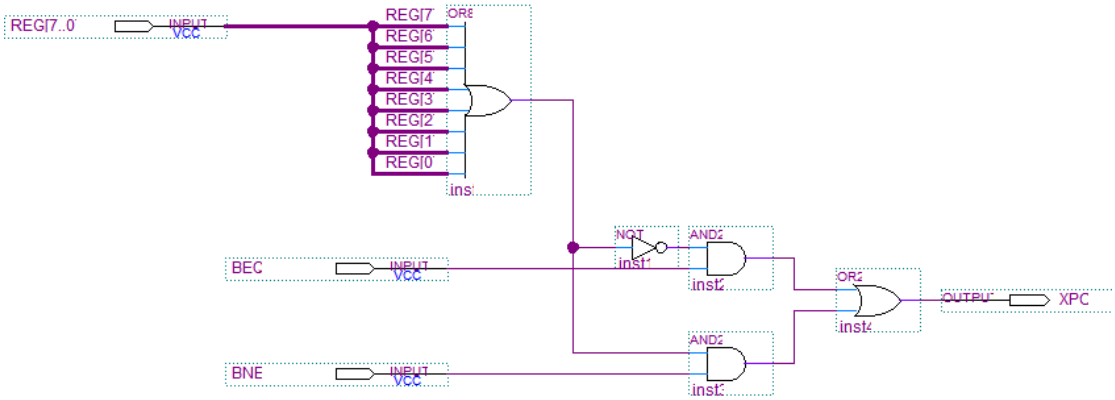
• 二选一数据选择器：



其中，AND1_8 部件见三选一数据选择器部分。

A1、A2 是我们需要选择的数据，f 为控制码，f=0 时，选择 A1，否则选择 A2。

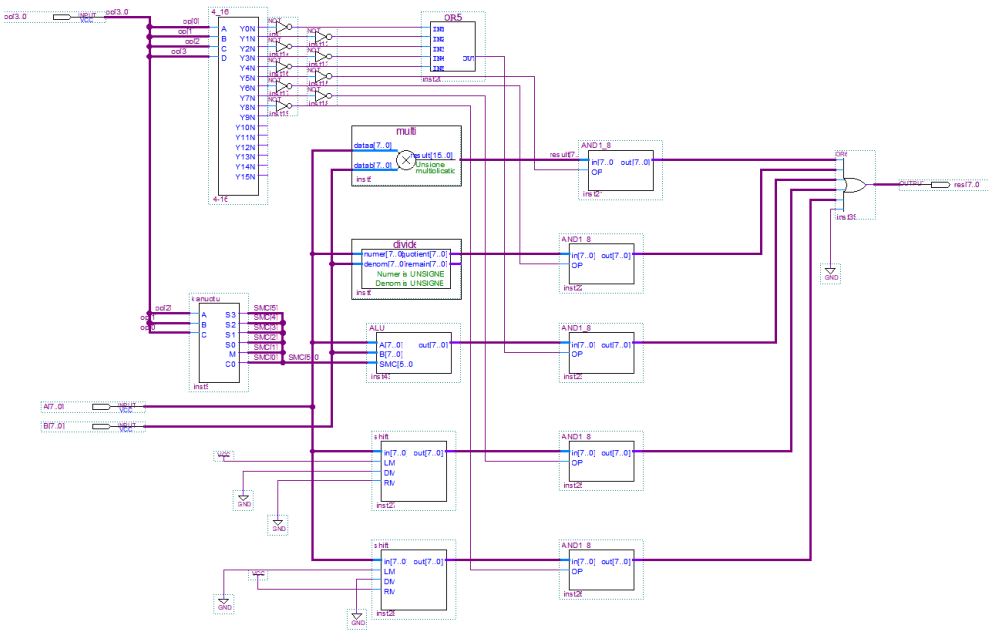
- BNE&BEQ：该部件用于根据 R0 的值实现逻辑跳转。根据此前描述，微指令第 1 位数据控制 BNE，即 inst[1] = 1 时，执行 BNE；微指令第 2 位数据控制 BEQ，即当 inst[2] = 1 时，执行 BEQ。而执行跳转与 PC 部件的 LOAD 控制端有关，当 LOAD=0 时，PC 部件在时钟信号下就会打入数据。该部件结构如下图：



其中，REG 输入 R0 的值，BEG 和 BNE 为控制端，BEG=0，BNE=1 时，说明此时希望执行 BNE。XPC 为最终输出信号，当 XPC=1，并且在外部将其取反输入 PC 部件的 LOAD 端后，就能使 PC 部件打入新数据。

- CU：控制器，能够根据指令类型输出指令对应的微指令序列，其中指令类型（指令前 8 位）存储于 IR 寄存器中。CU 包括 μPC 和 CROM，具体将在后序讲解。

- CACULATOR：运算器，能够实现加法、减法、乘法、除法、左移、右移等模型机所有的算数运算。具体结构如下图所示：

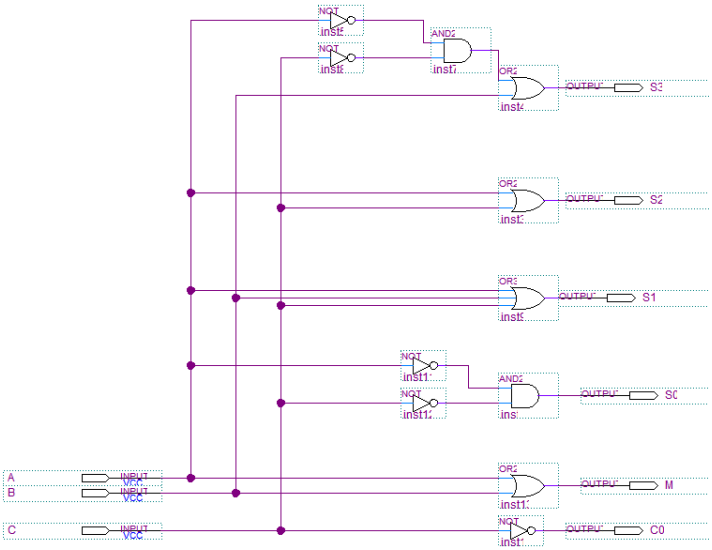


其中，A 和 B 为 8 位操作数数据，op 为操作码，op=0000 时输出加法结果，op=0001 时输出减法结果，op=0010 时输出相与结果，op=0011 时输出相或结果，op=0100 时输出异或结果，op=0101 时输出乘法结果，op=0110 时输出除法结果，op=0111 时输出右移结果，op=1000 时输出左移结果。

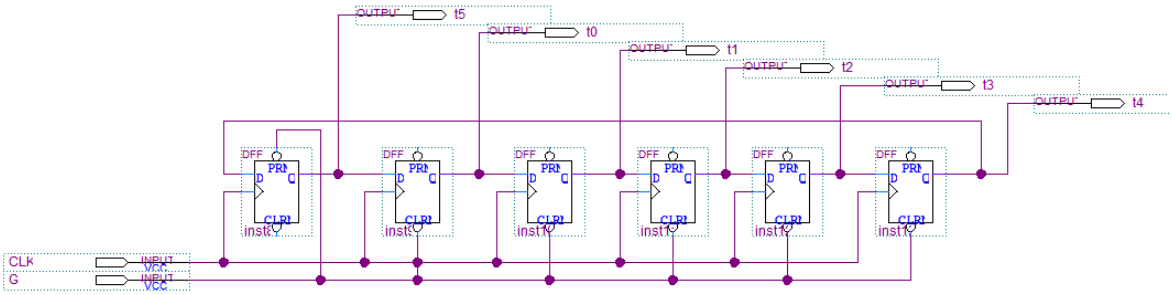
4_16 为 4-16 译码器部件，shift 为 8 位移位器部件，均由自己实现，结构较简单，因此不再展示结构图，AND1_8 在此前三选一选择器部分已有展示。

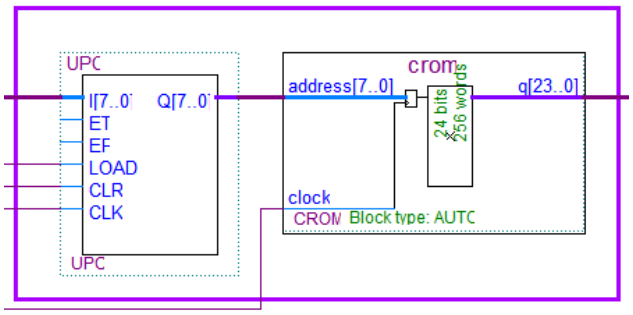
对于 ALU 部件，能够实现加减法、相与、相或、异或操作，为此前实验实现部件，不在此展示结构图。

kanuotu 部件为 ALU 部件的操作码输入部分。由于 ALU 部件能够实现加法、减法、相与、相或和异或五种操作，该五种操作对应 CACULATOR 的操作码最高位均为 0，因此设置逻辑电路将 op 低三位与 ALU 部件的控制操作码进行映射，而这就是 kanuotu 部件的功能。结构如下所示：



• 启停装置：该装置主要为模型机提供时序支持，需要将机器时钟分为不同的六个时序信号，同时能够在终止信号到来时，及时终止。我们使用 D 触发器实现分时，使用 D 触发器的置零端实现时钟停止。具体电路结构如下图：





4.5 CU 模块设计

• 总述：

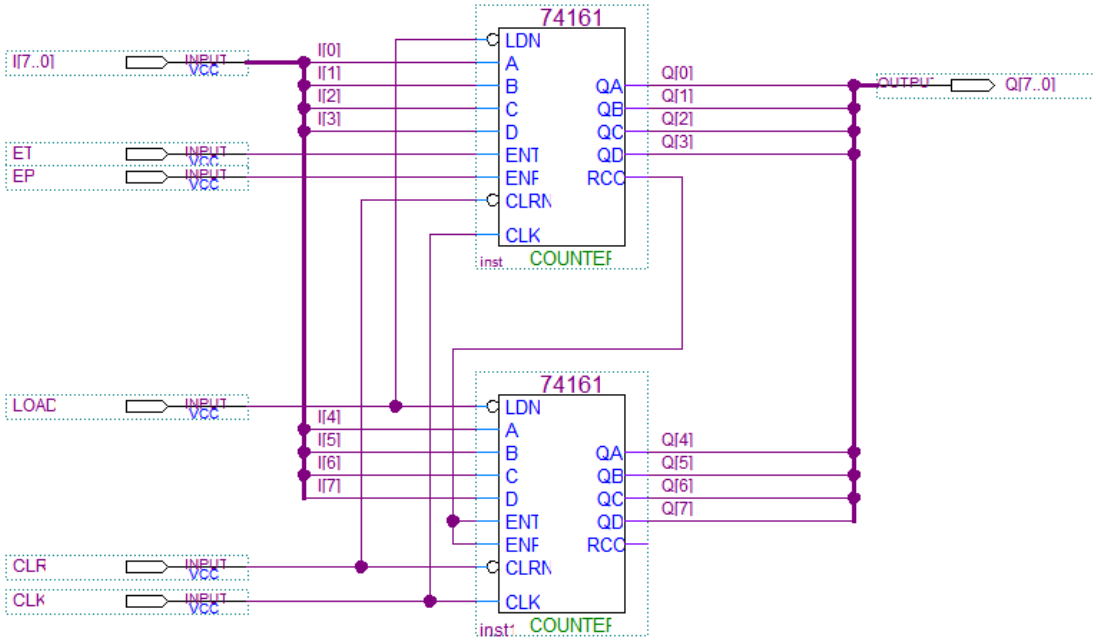
CU 模块是模型机最重要的部分，归根结底模型机靠每一条微指令控制，而微指令就是由 CU 模块产生。

CU 模块由 μPC 和 CROM 构成，其中 CROM 为 256*24bit 的存储器，其中按照顺序存储每条指令的所有微指令序列。μPC 中存储 CROM 的地址，按照 μPC 中的地址，每个周期从 CROM 中读取一条微指令并执行。总体结构如下图所示：

• μPC 部件：

μPC 部件结构与 PC 部件完全相同，是由两个 74161 元件拼接而成的 8 位计数器，接收时钟信号时，当 LOAD 为高电位，μPC 会执行加 1 操作；当 LOAD 为低电位，μPC 执行打入数据的操作，这在“0->μPC”微指令中使用。

具体电路图如下图所示：



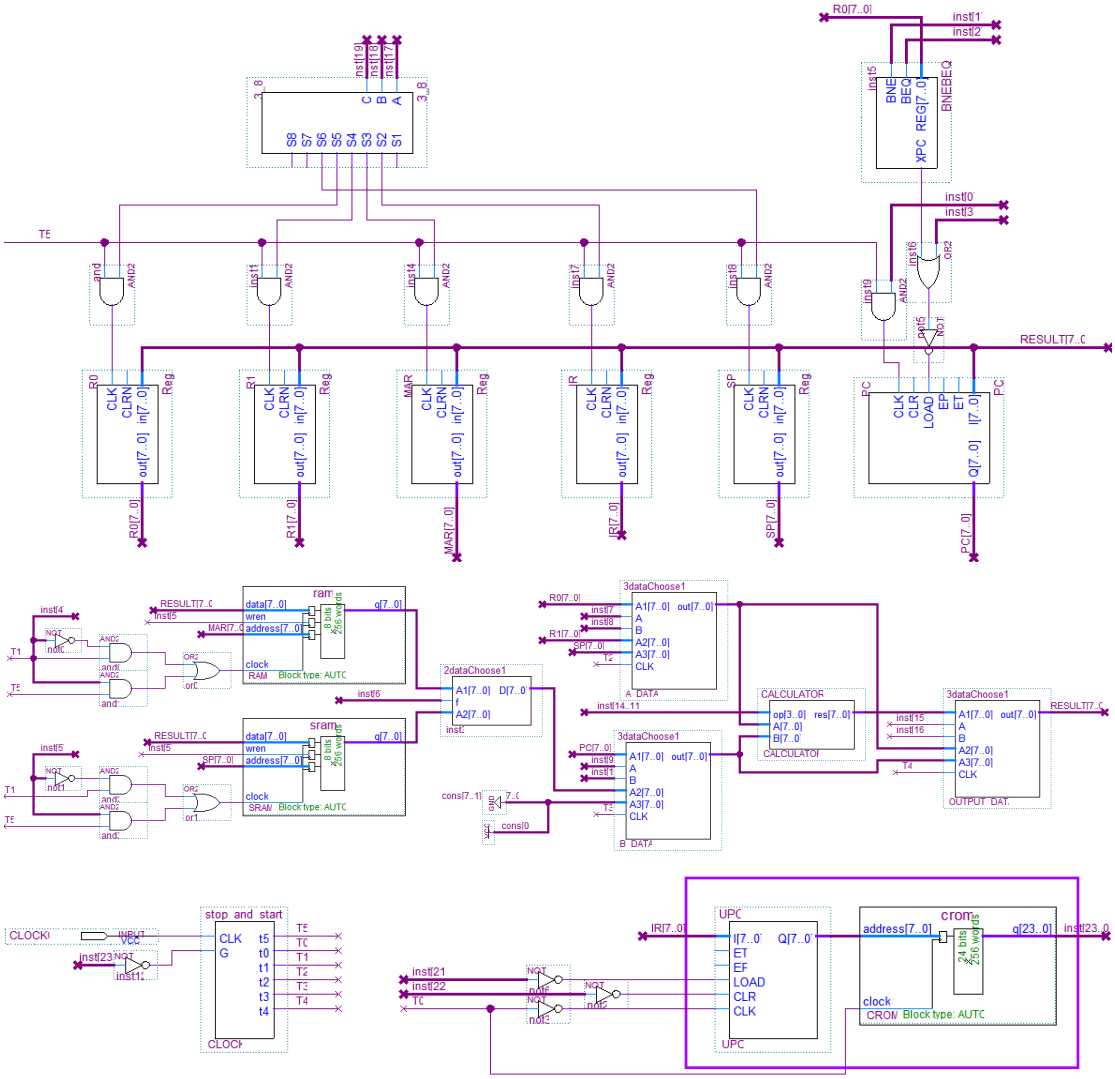
• μPC 地址确定：

μPC 部件本质是一个计数器，记录的是 CROM 地址，即每次从 CROM 中读取微指令的位置。一般情况下，μPC 加 1 即可，代表读取下一条指令。但是读取一条新指令时，μPC 的值要更新为该指令所对应的第一条微指令在 CROM 中的位置。

对于每条指令，前 8 位是指令的操作码，同时也是其对应的第一条指令在 CROM 中的地址。当我们取出新指令时，前 8 位会放入 IR 寄存器中，因此我们将 μPC 的数据输入端与 IR 相连，并为 LOAD 端增加逻辑控制部分，从而使得每次读取新指令时，μPC 中的值都能更新为该指令对应第一条微指令的地址。

4.6 总体结构设计

我们将所有数据通路实现，同时加上 CU 部分，并将微指令每一位的功能实现为逻辑电路，最终的电路图如下所示：



4.7 测试程序

由于本模型机能够实现 CALL 和 RETURN，因此我们可以实现递归程序。我们使用递归实现当输入 n 时，输出2^{n-1}的程序。

C++代码如下：

```
1.  int diff(int n){
2.      if(n==1){
3.          return 1;
4.      }
5.      return diff(n-1) + diff(n-1);
6.  }
7.
8.  int main(){
9.      int n=4;  // 可设定 n 的值
10.     diff(n);
```

```
    return 0;
}
```

汇编代码如下：

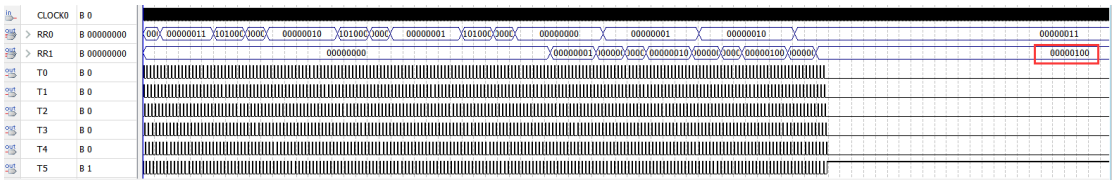
```
main:
    LW0     3           //可装入任意数字
    SW0     (address0#)
    CALL    diff#
    LW3     (address1#)
    HALT

diff:
    LW2     (address0#)     // 初始输入的 x 保存在 R0 中
    SUB0    (address2#)     // address2 中存储 1
    BNE     L1#             // 是否为 0
    LW1     1
    SW1     (address1#)
    RETURN


L1:
    SW0     (address0#)
    CALL    diff#
    LW3     (address1#)
    ADD1    (address1#)
    SW1     (address1#)
    RETURN
```

上述指令中，输入的 n 存储于 R0，最终结果存储于 R1。address0 暂存 R0 的值，address1 暂存 R1 的值，address2 中存储 1，用于执行减法。

4.8. 实验测试

4.8.1 仿真波形测试

为了便与调试，可以先使用 Quartus 提供的波形图进行仿真。对于递归测试程序而言，初始时输入的初值$n = 3$，理论上最终结果应为$2^{3-1} = 4$。实际的仿真测试如下所示：

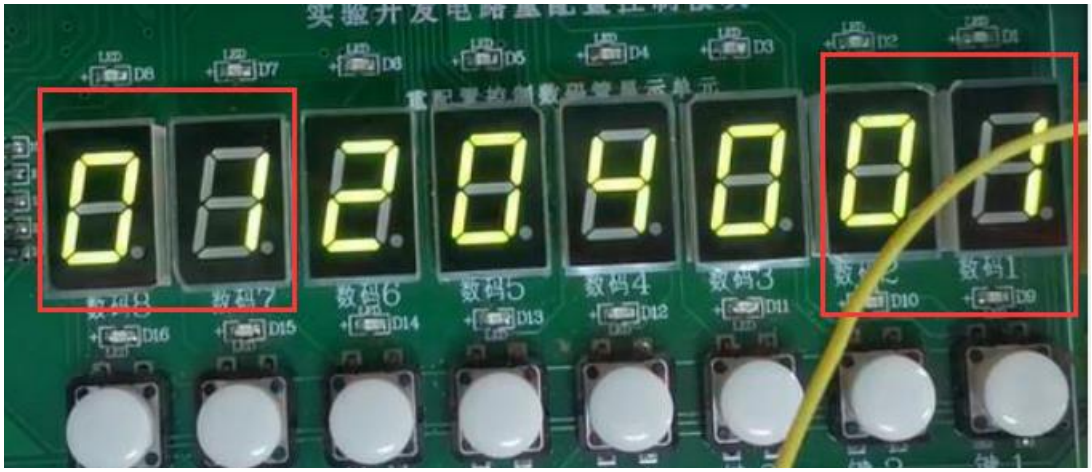


观察可知，实验中波形测试的最终结果为 4，和理论分析的结果相吻合，即仿真测试正确。

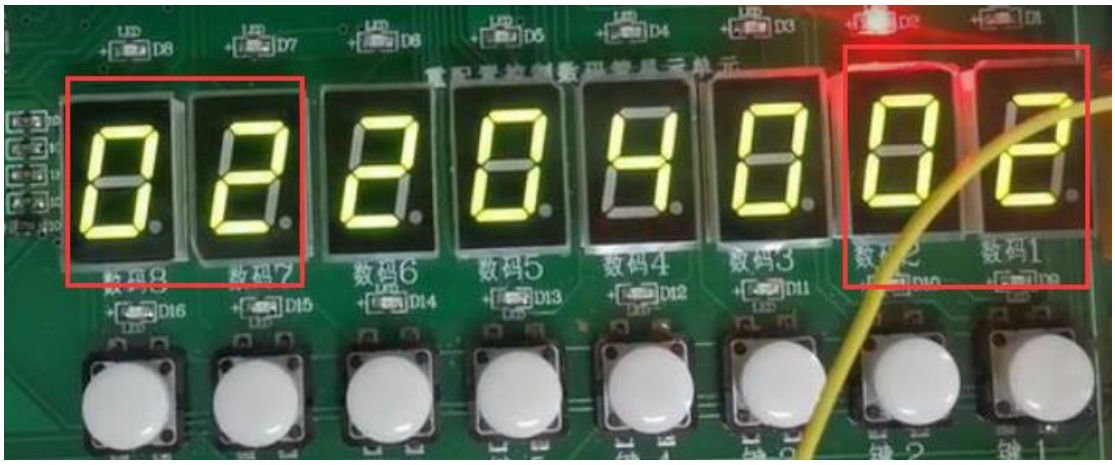
4.8.2 实验箱测试

利用上述汇编代码和模型机的指令架构，可以编制相应的机器代码，并输入模型机用于实验设计的正确性测试。具体测试过程和测试结果如下所示：

对于递归测试，初始装入的数值 n 为 3。根据递归函数的执行过程可知，随着程序的执行，可以依次计算出初值 n=1，n=2 的中间结果，如图所示：

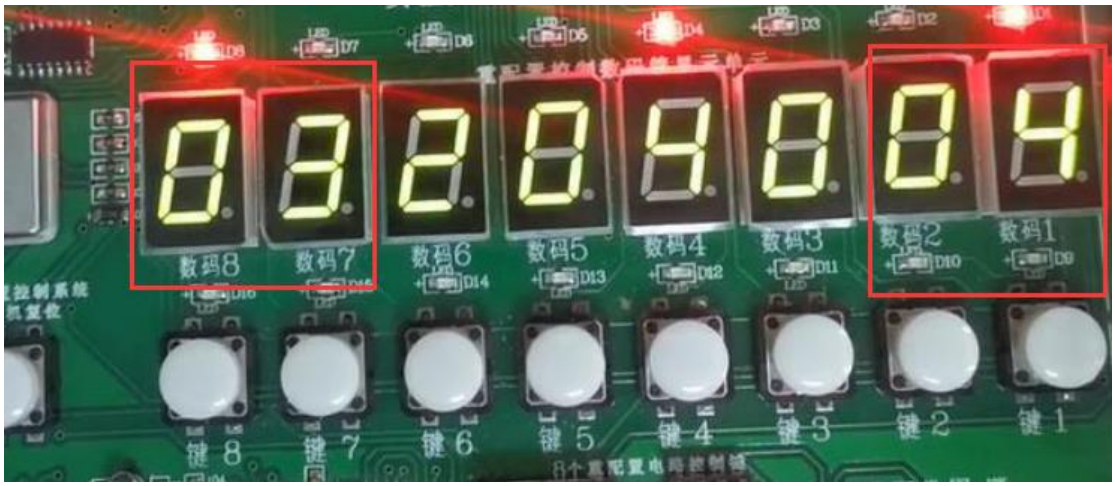


此时计算 n=1 的结果为$2^{1-1}=1$



此时计算 n=2 的结果为$2^{2-1}=2$

最终当计算到 n=3 之后，结果显示为$2^{3-1}=4$, 如下图所示。



根据汇编程序以及实验真实测试可知，当计算完成之后，程序正常退出，模型机停止执行。

综上，汇编程序测试正确，实验中设计的压栈、弹栈等指令正确。

5 硬布线实现的模型机内核

使用硬布线设计方法实现控制单元，思路清晰，简单明了，需要对每一个微操作都需要设计对应的逻辑电路。随着 RISC 指令集（精简指令集）的出现，硬布线设计仍是设计计算机的一种重要方法。

硬布线实现与微程序实现，既有相同点，又有不同点。相同点在于其指令集、指令 编码、指令对应微操作、微操作对应受控门状态均一致，且可以使用相同的测试程序进行正确性检测。不同点在于总体设计、实现方式不相同。

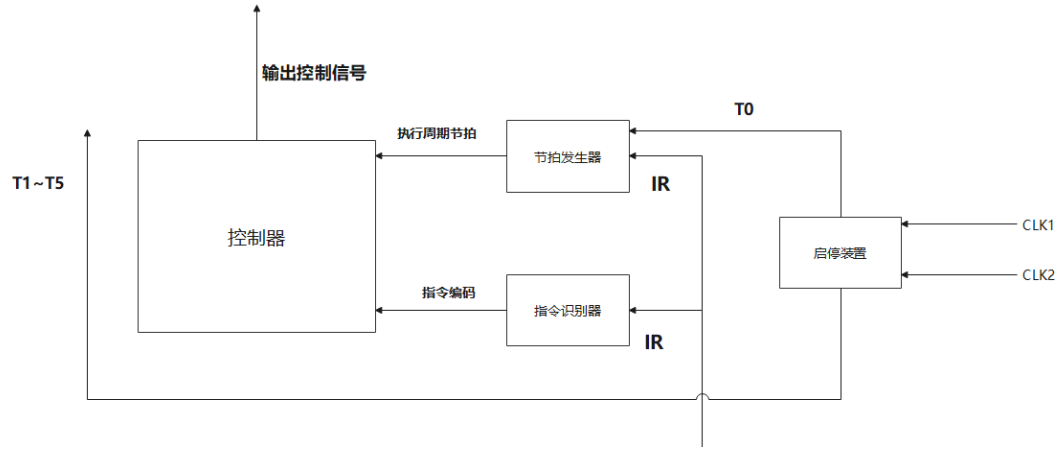
5.1 总体设计

基于硬布线实现的模型机内核主要是由节拍发生器，指令识别器、控制器等部件组成。其中各个部件的作用如下表所示：

器件名称	器件作用
节拍发生器	控制执行的周期和节拍
指令识别器	用于识别不同的输入指令，类似于译码操作
控制器	生成 24 个最终的受控门状态。

此外，和基于微程序实现的模型机内核不同，为了对指令的执行周期进行统一，在硬布线方式下可以直接将取址周期和执行周期合并。由于每一条指令的执行都包括取指和执行周期，因此合并之后对模型机的运行效率不会产生影响，但是可以简化后续的原件设计。

综上，基于硬布线实现的模型机总框图如下所示：



5.2 各元件设计

根据上图所示，基于硬布线实现的控制器主要依赖于两个输入，分别是指令周期的节拍和具体的指令编码。具体的指令有指令识别器产生并输入，而指令的执行周期由节拍发生器产生并输入。

5.2.1 指令识别器

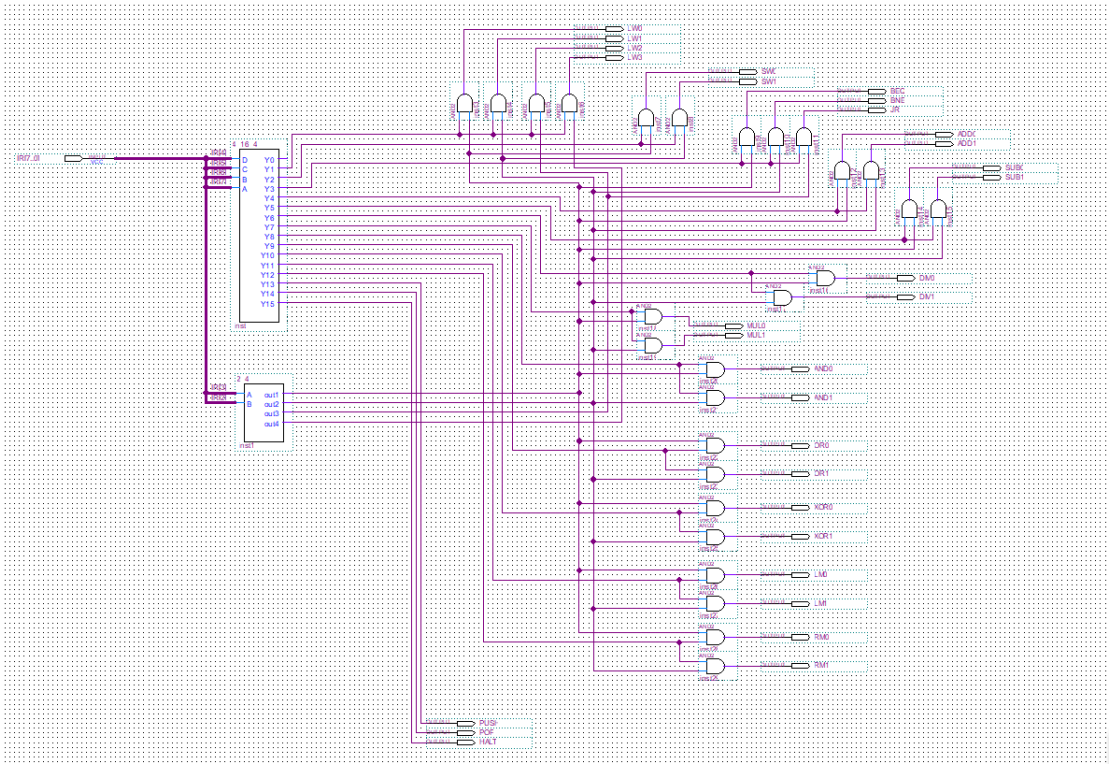
指令识别器主要是针对 IR 的输入，将其识别成对应的指令。在具体实现时，每一种指令都对应一根信号线，且经过识别之后只有指令对应的信号线被置为高电平，其余信号线均保持低电平。例如当 IR 输入为 18H 时，需要将其识别为 LW2 指令，并将 LW2 对应的输出端置 1。

具体实现分为以下两步:

（1）首先使用译码器 74138 对于 IR 的高四位进行识别

（2）对于不同的高四位进行分类识别。

即先使用高四位对指令的类别进行大致的划分，在使用低四位具体识别每一类指令中的每一种指令。具体电路图如下所示：



硬布线方式下控制器中的指令识别器

5.2.2 节拍发生器

节拍发生器主要是为控制器提供执行周期节拍，即告知控制器不同的指令应该执行多少个周期。由于不同执行的执行周期可能存在较大差异（如压栈指令的执行周期需要 11 个节拍，加法指令的执行周期需要 5 个节拍），需要设计与微程序方式中不同的节拍发生器。

分析模型机的各种指令可知，PUSH 指令共有九个微操作，POP 指令有六个微操作，其余指令在执行时大多需要 3 个微操作。由于在实现时将取值周期和执行周期进行了合并，因此对于一条指令，最多需要 11 个节拍，最少需要 5 个节拍。考虑到只有 PUSH 指令和 POP 指令的执行周期较长，因此可以针对 PUSH 指令和 POP 指令的操作码，设计相节拍控制器异步清零端的组合逻辑，从而实现不同的指令，分配不同数目的节拍，避免节拍的浪费。

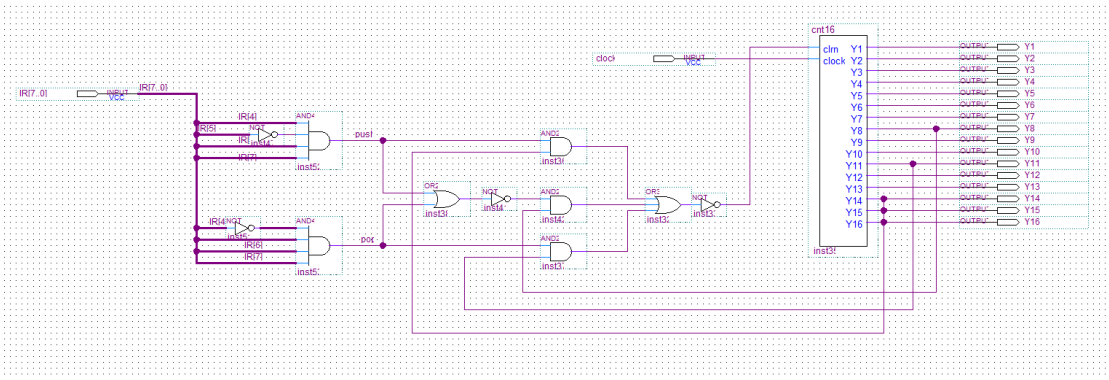
具体指令的节拍分配如下所示：

	取址周期		不同指令的执行周期								
指令节拍分配表	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
指令名称	1	1	1	1	1	0	0	0	0	0	0
LW0	1	1	1	1	1	0	0	0	0	0	0
LW1	1	1	1	1	1	0	0	0	0	0	0
LW2	1	1	1	1	1	0	0	0	0	0	0
LW3	1	1	1	1	1	0	0	0	0	0	0
SW0	1	1	1	1	1	0	0	0	0	0	0
SW1	1	1	1	1	1	0	0	0	0	0	0
BEQ	1	1	1	1	1	0	0	0	0	0	0
BNE	1	1	1	1	1	0	0	0	0	0	0
JR	1	1	1	1	1	0	0	0	0	0	0
ADD0	1	1	1	1	1	0	0	0	0	0	0
ADD1	1	1	1	1	1	0	0	0	0	0	0
SUB0	1	1	1	1	1	0	0	0	0	0	0
SUB1	1	1	1	1	1	0	0	0	0	0	0
DIV0	1	1	1	1	1	0	0	0	0	0	0
DIV1	1	1	1	1	1	0	0	0	0	0	0
MUL0	1	1	1	1	1	0	0	0	0	0	0
MUL1	1	1	1	1	1	0	0	0	0	0	0
AND0	1	1	1	1	1	0	0	0	0	0	0
AND1	1	1	1	1	1	0	0	0	0	0	0
OR0	1	1	1	1	1	0	0	0	0	0	0
OR1	1	1	1	1	1	0	0	0	0	0	0
XOR0	1	1	1	1	1	0	0	0	0	0	0
XOR1	1	1	1	1	1	0	0	0	0	0	0
左移R0(LM0)	1	1	1	1	1	0	0	0	0	0	0
立即数左移放R0(LM1)	1	1	1	1	1	0	0	0	0	0	0
右移R0(RM0)	1	1	1	1	1	0	0	0	0	0	0
立即数右移放R0(RM1)	1	1	1	1	1	0	0	0	0	0	0
PUSH	1	1	1	1	1	1	1	1	1	1	1
POP	1	1	1	1	1	1	1	0	0	0	0
HALT	1	0	0	0	0	0	0	0	0	0	0

因此，对于节拍控制器的异步清零端的组合逻辑表达式如下所示：

$$clear = \overline{S_{12}PUSH + S_9POP + S_5\overline{PUSH + POP}}$$

根据上述逻辑表达式，可以设计出具体的实现电路：



硬布线方式下控制器中的节拍发生器

组合逻辑控制器利用节拍发生器不同的状态组合来区分一条指令不同的执行步骤，且指令执行步骤的接续是通过变换节拍发生器的状态组合完成的。不同于微程序控制器中通过下地址部件给出不同的微指令地址来实现，而这里用节拍发生器取代了原来的下地址形成部件。

5.3 硬布线编码

根据指令识别器与节拍发生器，我们可以得到执行周期节拍与指令编码的信息，之后需要根据每个指令微操作所对应的受控门状态来进行硬布线编码，即对于每个受控门设计出对应的组合逻辑电路。

由于模型机的指令长度为 24 位，因此需要 24 个不同的组合逻辑电路，每一个位对应的组合逻辑表达式如下表所示，其中t_1~t_{11}表示节拍，大写指令表示操作。

受控门	组合逻辑
$Inst[0]$	$t_1 + LW0 \cdot t4 + LW1 \cdot t4 + LW2 \cdot t5 + LW3 \cdot t5 + SW0 \cdot t5 + SW1 \cdot t5 + BEQ \cdot t4 + BNE \cdot t4 + JR \cdot t4 + ADD0 \cdot t5 + ADD1 \cdot t5 + SUB0 \cdot t5 + SUB1 \cdot t5 + DIV0 \cdot t5 + DIV1 \cdot t5 + MUL0 \cdot t5 + MUL1 \cdot t5 + AND0 \cdot t5 + AND1 \cdot t5 + OR0 \cdot t5 + OR1 \cdot t5 + XOR0 \cdot t5 + XOR1 \cdot t5 + LM0 \cdot t3 + LM1 \cdot t5 + RM0 \cdot t3 + RM1 \cdot t5 + PUSH \cdot (t8 + t11) + POP \cdot t4$
$Inst[1]$	$BNE \cdot t4$
$Inst[2]$:	$BEQ \cdot t4$
$Inst[3]$:	$JR \cdot t4 + PUSH \cdot t11 + POP \cdot t4$
$Inst[4]$:	$SW0 \cdot t5 + SW1 \cdot t5$

$Inst[5]$:	$PUSH \cdot (t3 + t5 + t9)$
$Inst[7]$:	$SW1 \cdot t5 + ADD1 \cdot t5 + SUB1 \cdot t5 + DIV1 \cdot t5 + MUL1 \cdot t5 + AND1 \cdot t5 + OR1 \cdot t5 + XOR1 \cdot t5 + PUSH \cdot t5$
$Inst[8]$:	$PUSH \cdot (t4 + t6 + t10) + POP \cdot (t3 + t5 + t7)$
$Inst[9]$:	$t1 + LW0 \cdot t4 + LW1 \cdot t4 + LW2 \cdot (t4 + t5) + LW3 \cdot (t4 + t5) + SW0 \cdot t4 + SW1 \cdot t4 + BEQ \cdot t4 + BNE \cdot t4 + JR \cdot t4 + ADD0 \cdot (t4 + t5) + ADD1 \cdot (t4 + t5) + SUB0 \cdot (t4 + t5) + SUB1 \cdot (t4 + t5) + DIV0 \cdot (t4 + t5) + DIV1 \cdot (t4 + t5) + MUL0 \cdot (t4 + t5) + MUL1 \cdot (t4 + t5) + AND0 \cdot (t4 + t5) + AND1 \cdot (t4 + t5) + OR0 \cdot (t4 + t5) + OR1 \cdot (t4 + t5) + XOR0 \cdot (t4 + t5) + XOR1 \cdot (t4 + t5) + LM1 \cdot t4 + RM1 \cdot t4 + PUSH \cdot t8 + POP \cdot (t4 + t6 + t8)$
$Inst[10]$	$PUSH \cdot (t4 + t6 + t10) + POP \cdot (t3 + t5 + t7)$
$Inst[11]$:	$(SUB0 + SUB1 + MUL0 + MUL1 + OR0 + OR1 + RM1) \cdot t5 + RM0 \cdot t3 + POP \cdot (t3 + t5 + t7)$
$Inst[12]$:	$(DIV0 + DIV1 + AND0 + AND1 + OR0 + OR1 + RM1) \cdot t5 + RM0 \cdot t3$
$Inst[13]$:	$(DIV0 + DIV1 + MUL0 + MUL1 + XOR0 + XOR1 + RM1) \cdot t5 + RM0 \cdot t3$
$Inst[14]$:	$LM0 \cdot t3 + LM1 \cdot t5$
$Inst[15]$:	$(SW0 + SW1) \cdot t5 + (LM1 + RM1) \cdot t4 + PUSH(t3 + t5 + t11)$
$Inst[16]$:	$t0 + t1 + (LW2 + LW3) \cdot t5 + (LW0 + LW1 + LW2 + LW3 + SW0 + SW1 + BEQ + BNE + JR + ADD0 + ADD1 + SUB0 + SUB1 + DIV0 + DIV1 + MUL0 + MUL1 + AND0 + AND1 + OR0 + OR1 + XOR0 + XOR1) \cdot (t3 + t4) + (LM1 + RM1) \cdot t3 + PUSH \cdot (t7 + t8 + t9) + POP \cdot (t4 + t6 + t8)$
$Inst[17]$:	$t1 + LW1 \cdot t4 + (LW3 + ADD1 + SUB1 + DIV1 + MUL1 + AND1 + OR1 + XOR1) \cdot t5 + PUSH \cdot (t4 + t6 + t10) + POP \cdot (t3 + t5 + t6 + t7)$

$Inst[18]$:	$t0 + (LW0 + LW1 + LW2 + LW3 + BEQ + BNE + JR + SW0 + SW1 + ADD0 + ADD1 + SUB0 + SUB1 + DIV0 + DIV1 + MUL0 + MUL1 + AND0 + AND1 + OR0 + OR1 + XOR0 + XOR1 + LM1 + RM1) \cdot t3 + (LW1 + LW2 + LW3 + SW0 + SW1 + ADD0 + ADD1 + SUB0 + SUB1 + DIV0 + DIV1 + MUL0 + MUL1 + AND0 + AND1 + OR0 + OR1 + XOR0 + XOR1) \cdot t4 + (LW3 + ADD1 + SUB1 + DIV1 + MUL1 + AND1 + OR1 + XOR1) \cdot t5 + PUSH \cdot t7 + POP \cdot t6$
$Inst[19]$:	$(LW2 + ADD0 + SUB0 + DIV0 + MUL0 + AND0 + OR0 + XOR0 + LM1 + RM1) \cdot t5 + (LM0 + RM0) \cdot t3 + (LW0 + LM1 + RM1) \cdot t4 + PUSH \cdot (t4 + t6 + t8 + t10) + POP \cdot (t3 + t5 + t7 + t8)$
$Inst[20]$:	0(*预留*)
$Inst[21]$:	$t2$
$Inst[22]$:	$(LW0 + LW1 + BEQ + BNE + JR) \cdot t4 + (LW2 + LW3 + SW0 + SW1 + ADD0 + ADD1 + SUB0 + SUB1 + DIV0 + DIV1 + MUL0 + MUL1 + AND0 + AND1 + OR0 + OR1 + XOR0 + XOR1 + LM1 + RM1) \cdot t5 + (LM0 + RM0) \cdot t3 + PUSH \cdot t11 + POP \cdot t8$
Inst[23]	$HALT \cdot t3$

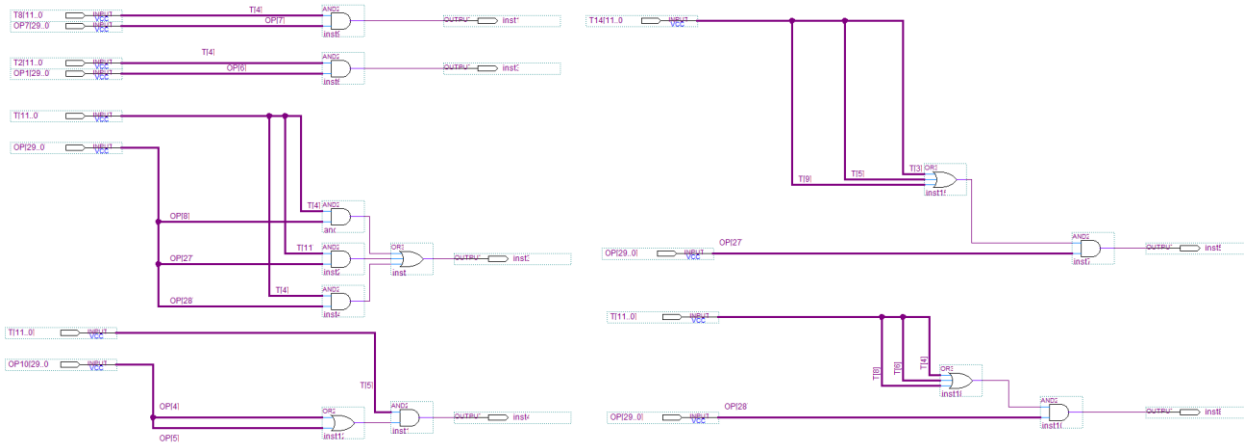
其中受控门 inst[20]被设置为预留位，便于之后的扩展，因此被设置为 0。

5.4 具体电路结构

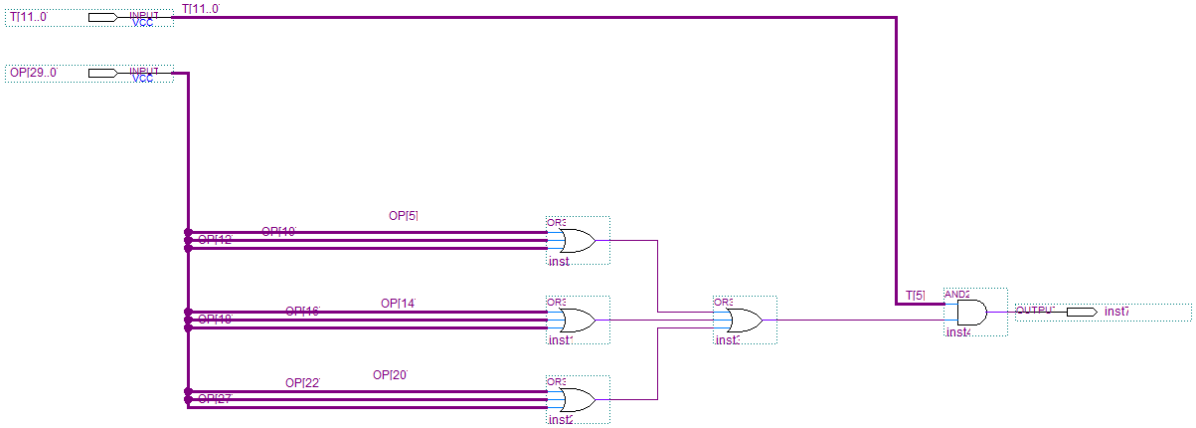
5.4.1 控制器

基于上述的组合逻辑表达式，可以为指令的每一位都设计相应的控制电路。由于本模型机指令字长为 24 位，因此共设计 24 个组合逻辑电路。具体电路见附录

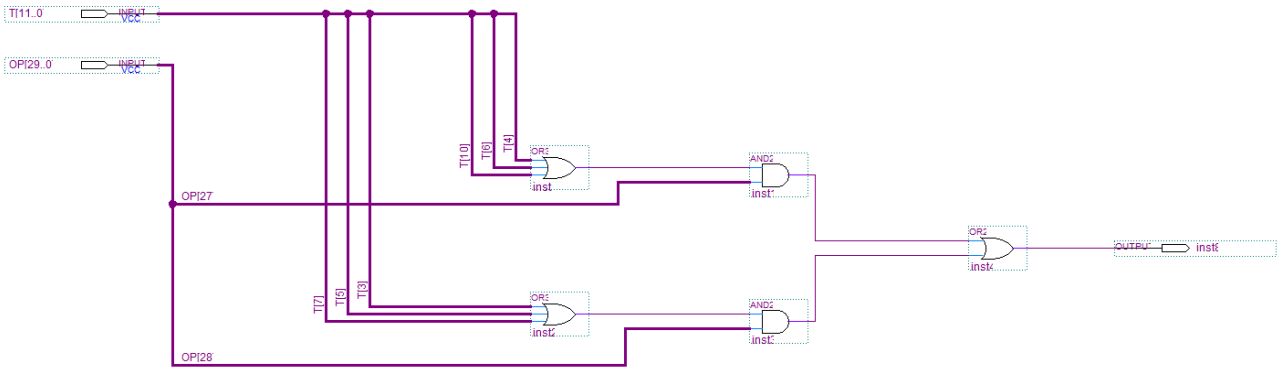
Inst1-inst6 如下图所示：



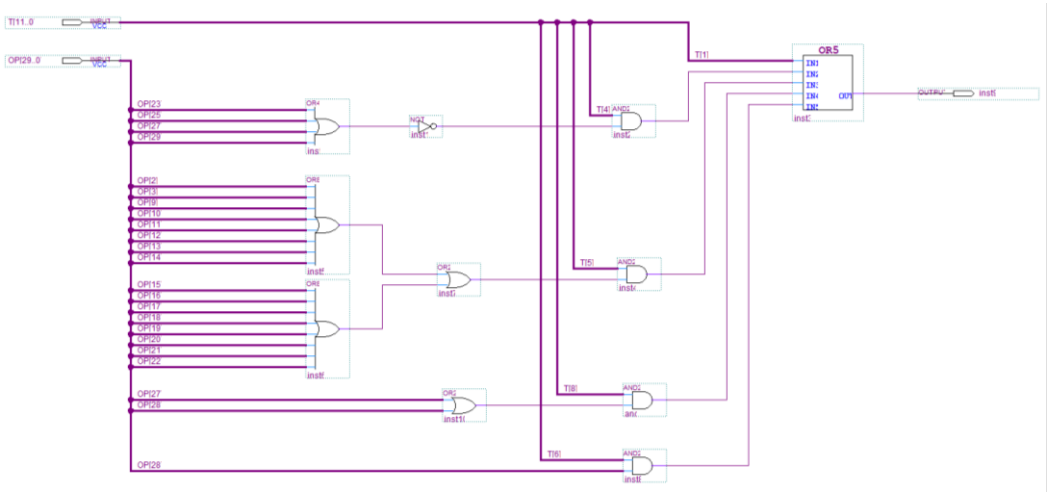
Inst7 如下图所示：



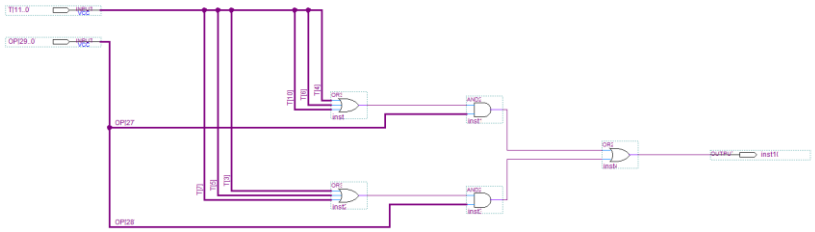
inst8 如下图所示：



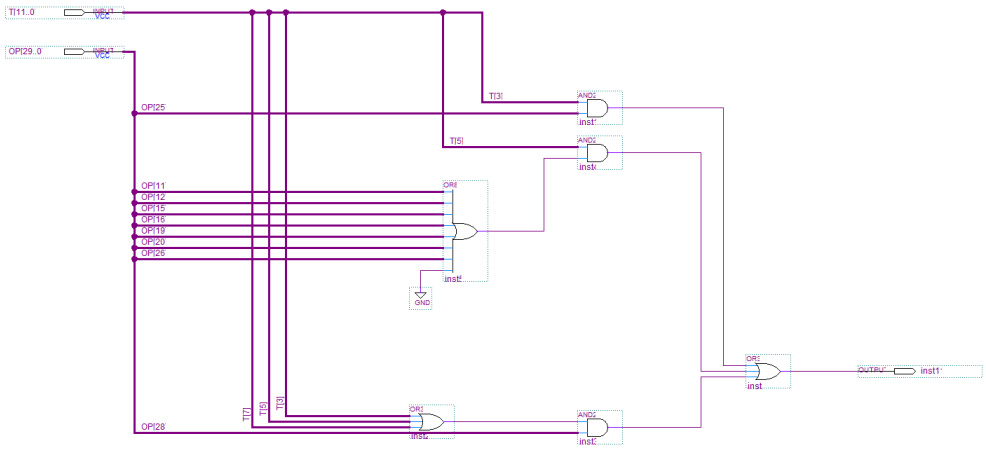
inst9 如下图所示：



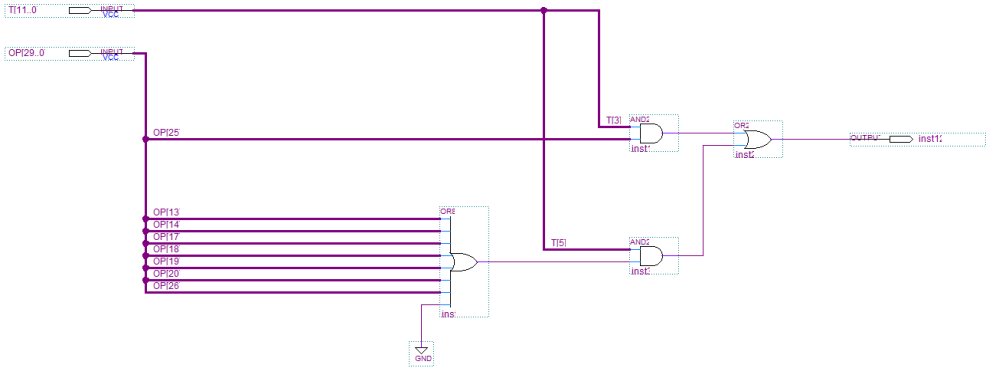
inst10 如下图所示：



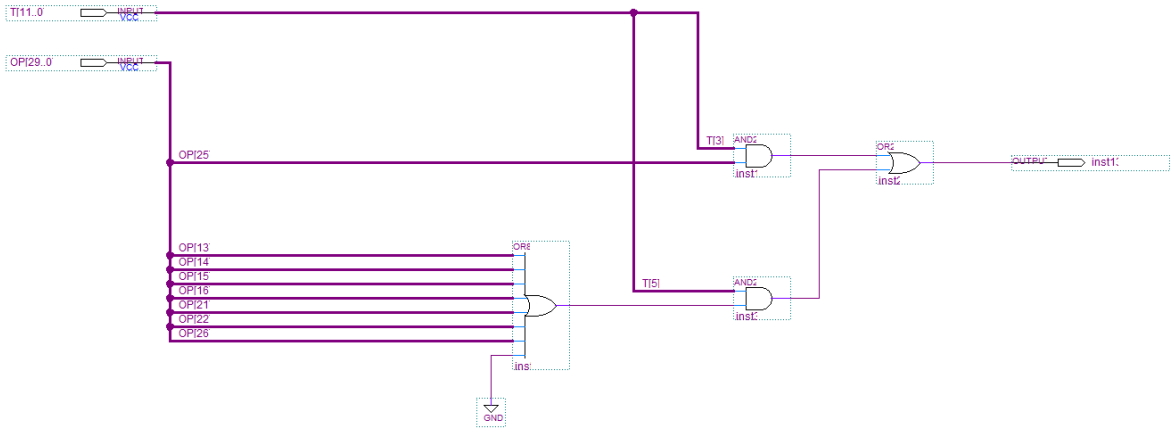
inst11 如下图所示：



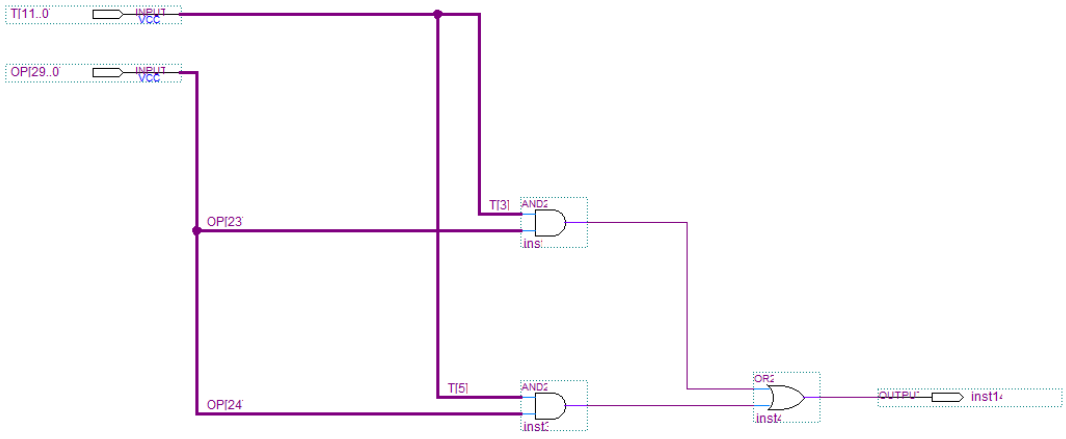
inst12 如下图所示：



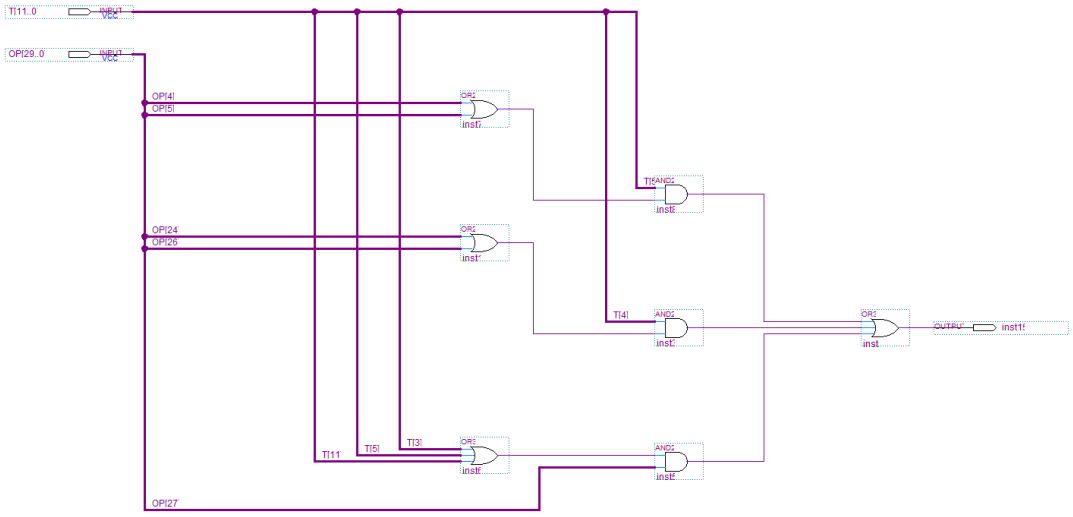
inst13 如下图所示：



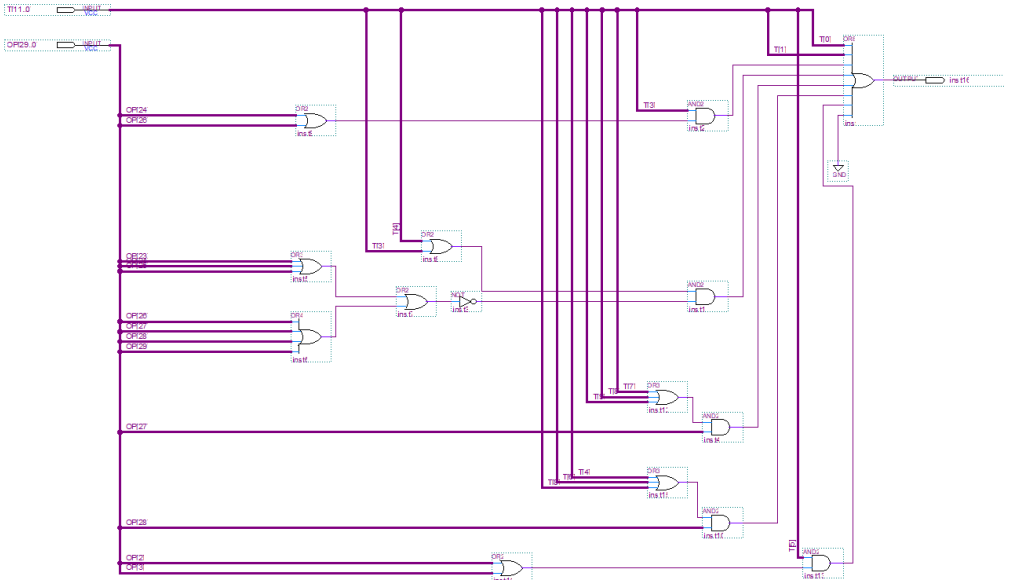
inst14 如下图所示：



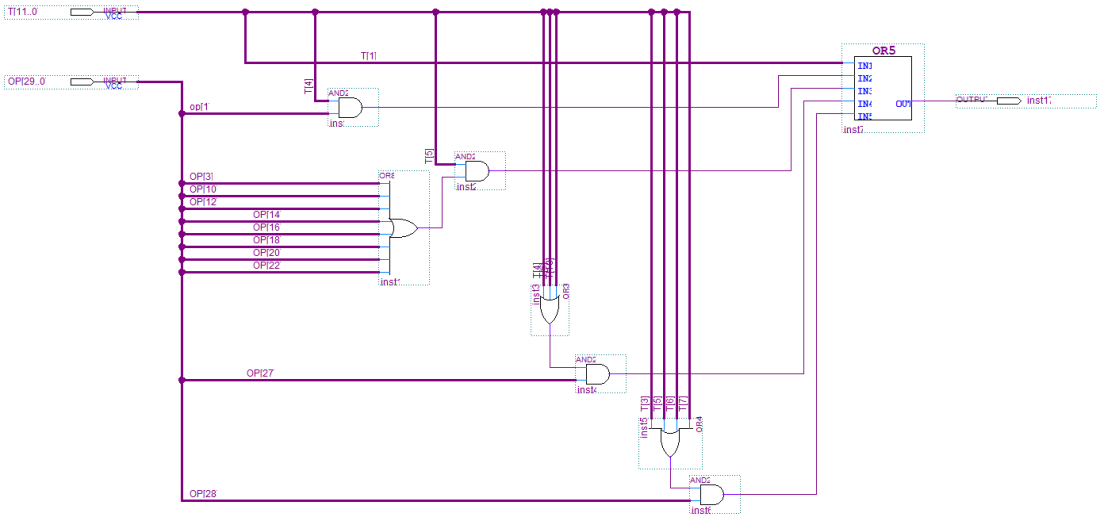
inst15 如下图所示：



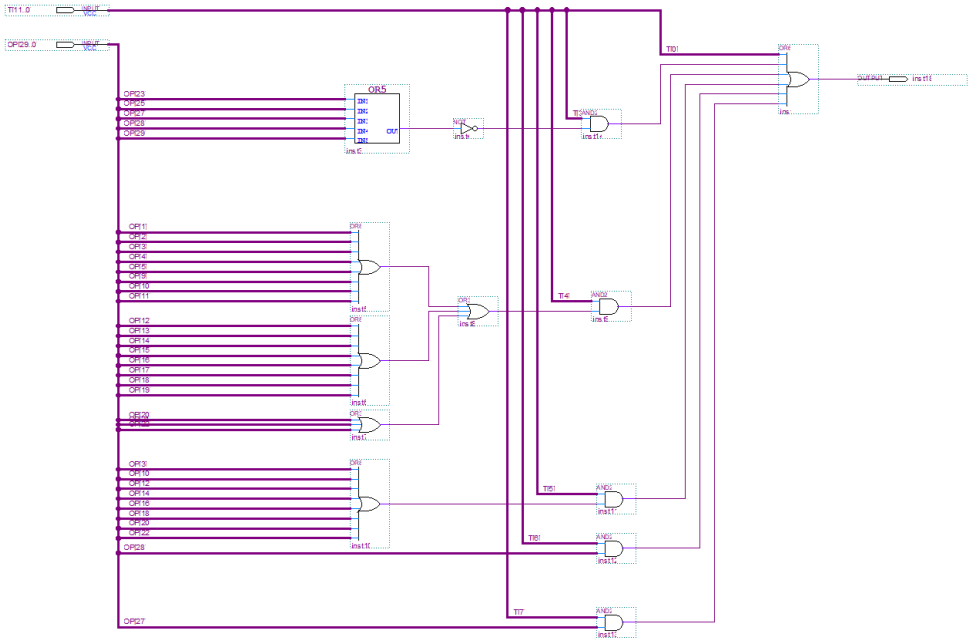
inst16 如下图所示：



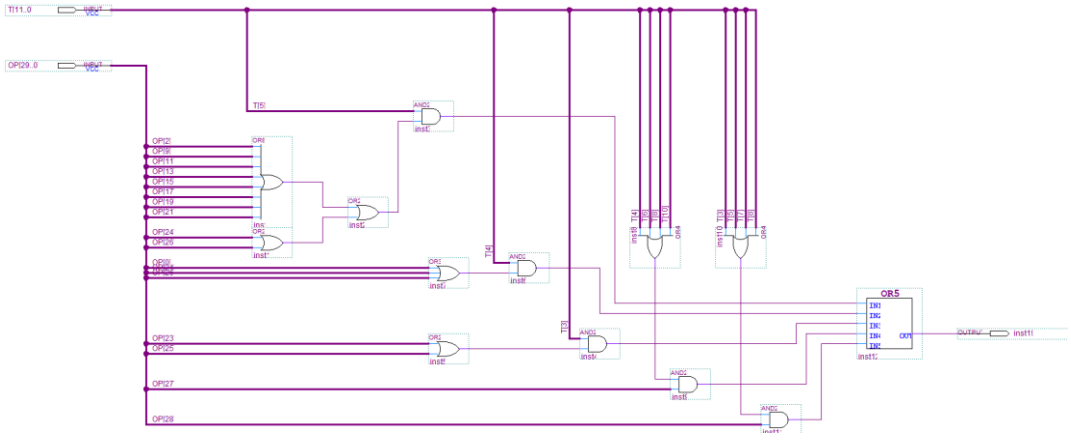
inst17 如下图所示：



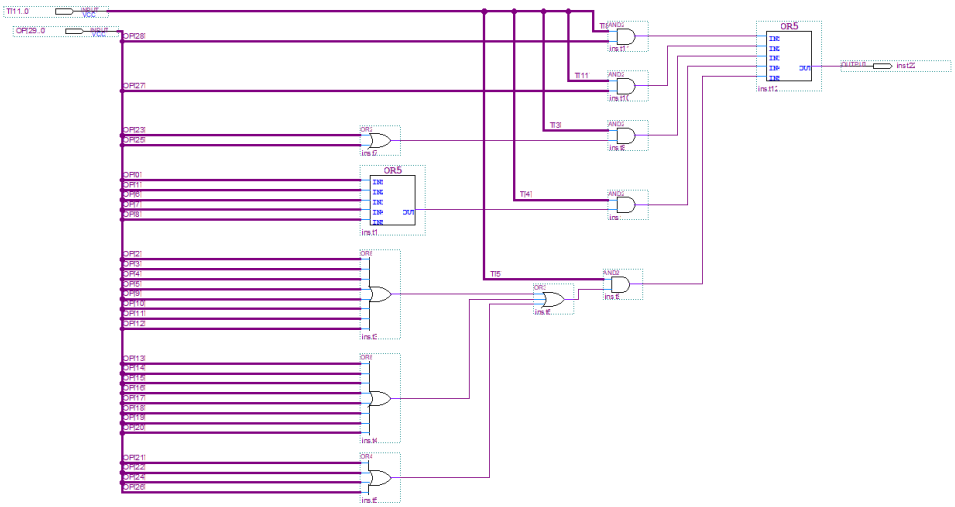
inst18 如下图所示：



inst19 如下图所示：

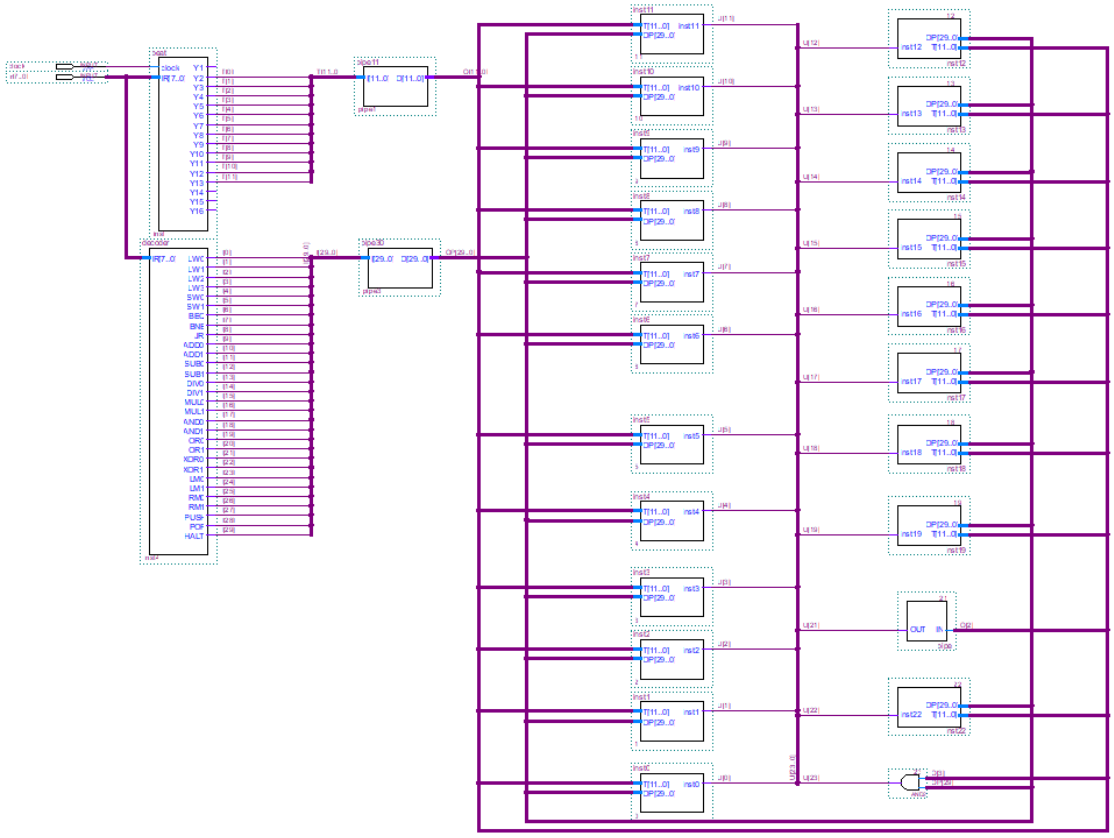


inst22 如下图所示：



5.4.2 完整电路

根据上述分析可知，基于硬布线实现的模型机和基于微程序方式实现的模型机主要区别在于控制器的实现方式不同。由于在实验中采用模块化设计，因此只需要将控制器进行替换即可得到硬布线方式实现的总体电路。如下所示：



8 致谢与展望

本次课程设计将课本上的知识和实际的动手操作相结合，从理论和实践两方面加深对计算机组成原理的理解。在实验过程中，首先需要感谢赵梦莹老师的理论指导，为之后的具体设计提供了明确的方向和思路。同时，需要感谢实验学长的耐心帮助，例如在实验二中通过和学长的交流，我们找到了设计的漏洞，使用节拍发生器将原始的时钟进行分离，达到了预期效果。

在本次实验中除了使用上学期的计算机组成原理的知识之外，还多次使用数字逻辑的相关知识，例如卡诺图的设计，时序的分析，节拍发生器的实现以及同步异步方式的分析等，因此也要感谢数字逻辑课程的吕知辛老师为课程设计作出的知识铺垫。

经过课程设计的四个阶段，我们完成了简单的模型机的设计和实现，并且作为实验的创新之处，我们在基础的要求上引入了压栈、弹栈指令，并设计测试程序实现了等比数列的递归计算。但从更进一步的角度分析，我们的模型机设计仍存在很多问题，例如指令系统设计的规范性、逻辑电路设计的模块化以及各个子元件在实现时的封装性等细节问题。

通过本次课程设计，我从应用的角度加深了对计算机组成原理的认知，希望在之后的学习中可以进一步加强相关领域的学习，更全面的理解计算机的底层原理。

9 参考文献

[1] 唐朔飞. 计算机组成原理 [M]. 高等教育出版社: 北京市,2008:1

10 附录

10.1 实验三实验四中的测试程序汇编代码

测试 JNE JEQ JR 等跳转指令以及 MUL DIV 等算数运算指令（//后为注释）

```
0:   LW0      3          // 3 -> R0
     MUL0     20         // R0 = R0 * [20]
     DIV0     21         // R0 = R0 / [21]
     BNE      30         // if R0 != 0 then PC = 30
     HALT

30:  ADD0     22         // R0 = R0 + [22]
     LW1      1          // 1->R1
     SW0      23         // R0 -> [23]
     LW0      0          // 0 -> R0
     BEQ      50         // if R0 == 0 then PC = 50
     HALT

50:  LW2      23         // [23] -> R0
     MUL0     21         // R0 = RO * [21]
     JR       60         // PC = 60
     HALT

60:  ADD1     21         // R1 = R1 + [21]
     HALT
```

其中：[20] = 6, [21] = 9, [22] = 1

执行完毕，R0 = 27，R1 = 10

10.2 实验三和实验四中的递归测试程序

递归程序数学公式描述、高级语言描述以及汇编代码表示

数学公式表示

通项公式：$\begin{cases} a_n = 2a_{n-1} \\ a_1 = 1 \\ a_2 = 2 \\ a_n = 2^{n-1} \end{cases}$

高级语言表示：

```
int diff(int n){
    if(n==1){
        return 1;
    }
    return 2*diff(n-1);
}

int main(){
    int n=4;
    diff(n);
    return 0;
}
```

汇编语言表示：

```
lw0 2# R0                    # 也可以装入任意一个数字 2,3,4...
sw0 R0 (address0#)
PUSH diff#
lw3 (address1#) R1
HALT

diff:
lw2 (address0#) R0      # 初始输入的 x 保存在 R0 中
SUB0 1# R0              # 先减去 1
BNE L1#                 # 是否为 0
LW1 1# R1
SW1 R1 (address1#)
POP


L1
SW0 R0 (address0#)
PUSH diff
lw3 (address1#) R1
ADD1 (address1#) R1
SW1 R1 (address1#)
POP
```

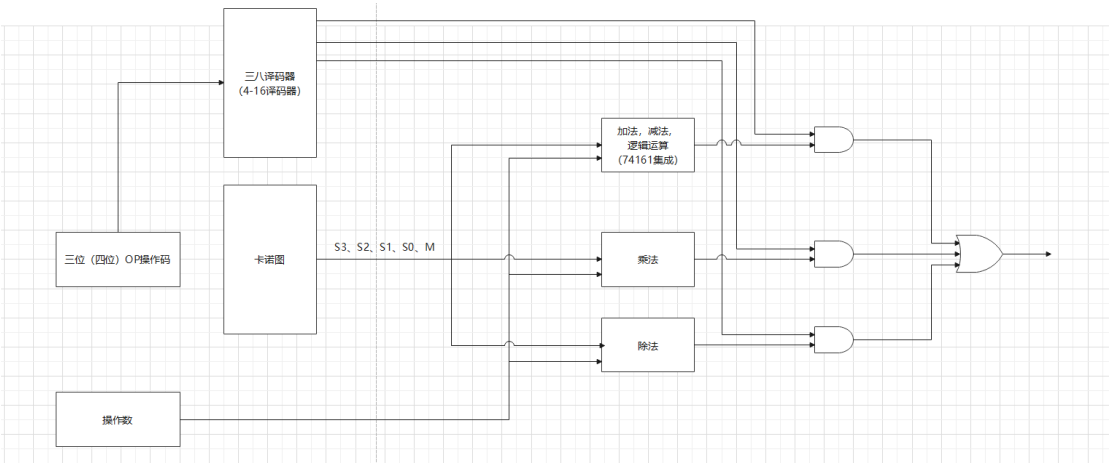
10.3 实验三指令每一位作用

指令位数	作用
23	等于 0，正常时钟，等于 1，时钟停止
22	等于 1， μPC 清零
21	等于 0， μPC 加 1，等于 1，装入数据
20	预留
19	
18	19-17:001IR，010MAR，011R1，100R0，101SP　000 都不选
17	
16	都为 0，选择 ALU 结果，[15]=1,[16]=0，选择 ADATA,[15]=0,[16]=1，
15	选择 BDATA
14	
13	14-11:0000 加，0001 减，0010 与，0011 或，0100 异或，0101 乘，0110
12	除，0111 右移，1000 左移
11	
10	都为 0，选择 PC，[9]=1,[10]=0，选择 RAM/SRAM,[9]=0,[10]=1，选择
9	01　BDATA
8	都为 0，选择 R0，[7]=1,[8]=0，选择 R1,[7]=0,[8]=1，选择 SP　ADATA
7	
6	等于 0，选择 RAM，等于 1，选择 SRAM
5	等于 1，SRAM 写。等于 0，SRAM 读
4	等于 1，RAM 写。等于 0，RAM 读
3	等于 1 时，PC 打入数据，不自增
2	控制 BEQ，该位为 1，当 R0=0，R1->PC
1	控制 BNE，该位为 1，当 R0!=0，R1->PC
0	控制 PC，希望 pc+1 或打入 pc，该位为 1

10.4 实验四指令集设计

指令名称	指令编码	指令名称	指令编码
LW0	0001 0000	**MUL0**	0111 0000
LW1	0001 0100	**MUL1**	0111 0100
LW2	0001 1000	**AND0**	1000 0000
LW3	0001 1100	**AND1**	1000 0100
SW0	0010 0000	**OR0**	1001 0000
SW1	0010 0100	**OR1**	1001 0100
BEG	0011 0000	**XOR0**	1010 0000
BNE	0011 0100	**XOR1**	1010 0100
JR	0011 1000	**LM0**	1011 0000
ADD0	0100 0000	**LM1**	1011 0100
ADD1	0100 0100	**RM0**	1100 0000
SUB0	0101 0000	**RM1**	1100 0100
SUB1	0101 0100	**PUSH**	1101 0000
DIV0	0110 0000	**POP**	1110 0000
DIV1	0110 0100	**HALT**	1111 0000

10.5 实验三中 ALU 整体架构设计示意图



10.6 实验四指令集设计以及时序统计

周期	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1

t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1

t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t5	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
t4	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
t6	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t7	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t8	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
t10	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t11	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
t4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	
t5	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
t6	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
t7	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
t8	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
t0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
t2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10.7 数据通路以及具体电路设计

数据通路以及具体电路设计见前 4.4 总体结构与数据通路 以及 5.4 具体电路结构。