山东大学
SHANDONG UNIVERSITY

计算机组成原理课程设计报告

题目：基于微程序实现及硬布线实现的简单模型机设计

学院：泰山学堂

专业：计算机取向

学号：201605130116

姓名：杜洪超

目录

1 课程设计目的与要求

1.1 设计目的

通过该课程设计的学习，总结计算机组成原理课程的学习内容，运用计算机原理知识，设计一台模型机，从而巩固课堂知识、深化学习内容、完成教学大纲要求，学好这门专业基础课。

1.2 设计要求

学生独立设计指令集，利用所学知识设计出具有一定特色的模型机。课程设计分为两个阶段：第一阶段为微程序实现的模型机内核，第二阶段为硬布线实现的模型及内核，最终达到对计算机的总体设计、基本构成、基本原理有一个清楚的认识并能建立一个清晰的整机概念，从而扎实地掌握一种数字系统的设计方法。

2 课程设计环境

2.1 硬件环境

课程设计整体基于 JYS 开放式计算机组成原理与结构实验系统开发箱，使用现场可编程逻辑阵列（FPGA），通过 EDA 设计技术，设计和下载电路实现数字逻辑实验电路。具体硬件配置如下：

1) FPGA 电路：超大规模集成电路 FPGA 芯片 EP2C8Q208C8
2) 时钟：多频率连续时钟发生器和单脉冲信号发生器
3) RAM 存储器：静态的 256×8 位的 RAM，用于存放指令和数据
4) ROM 存储器：用于存放系统的微程序，微指令 24bit 长，共 256×24 位
5) 发光二极管指示灯及开关：4 组 70 只发光二极管和 24 只拨动开关，其中 L23~L0 用于显示微指令，其余二极管和全部开关可供用户使用
6) 复位信号输入：单片机复位和 CPU 复位

2.2 软件环境

本课程设计使用计算机 CPU 为 Intel(R)Core(TM)i7-600 CPU@3.40GHz，操作系统为 Windows 8，具体开发环境如下：

1) Quartus II

Altera 公司的 QuartusⅡ软件提供了可编程片上系统（SOPC）设计的一个综合开发环境，是进行 SOPC 设计的基础，包括系统级设计、嵌入式软件开发、可编程逻辑器件设计、综合、布局和布线、验证和仿真等。

2) 计算机组成原理与系统结构.exe

负责与硬件实验平台电路调试、RAM 及 ROM 写入等。

3 课程设计具体内容

3.1 模型机设计步骤

1) 拟定指令系统
2) 确定总体结构
3) 逻辑设计
4) 确定控制方式
5) 编制指令流程（微指令）/节拍发生器的设计（硬布线）
6) 编制微程序（微指令）/时序控制信号形成部件的设计（硬布线）
7) 调试

3.2 确定指令系统

指令系统将涉及到指令字长、指令格式、指令类型、指令编码、寻址方式，具体内容见附录。

1) 指令字长

ROM 容量为 256×8 位，故指令字长为 8 位

2) 指令格式

指令格式包括单字长指令和双字长指令，在双字长格式中，第二字节定义为操作数或操作数地址；

3) 指令类型

指令分为单操作数指令、双操作数指令和无操作数指令，整体设计为 Load/Store 型指令集，即访存行为只通过 Load 指令和 Store 指令完成。采用不定操作码位长设计，非 Load/Store 指令操作码为 6 位，Load 和 Store 指令操作码为 4 位。6 位操作码指令包括算术运算加法、减法、乘法，逻辑运算与、或、非和异或，寄存器间赋值 MOV0、MOV1，寄存器跳转 B0，立即数跳转 B1 以及停机指令。

4) 指令编码

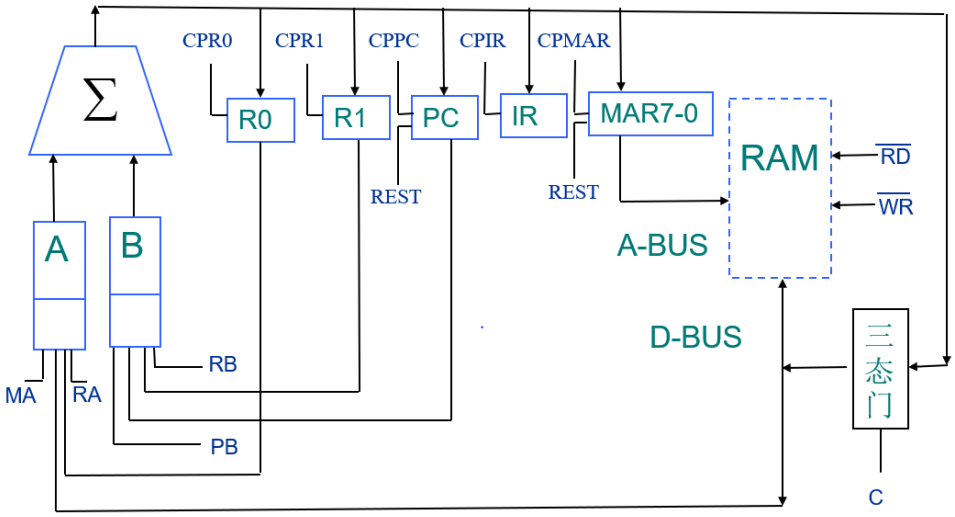
6 位操作码指令因为不涉及访存操作，指令较短，故不包括停机指令在内的 11 条指令顺序编码在 ROM 地址空间的 00 区域；Load 和 Store 指令分别编码在 0100 和 0101 区域，停机指令在指令集最后。

5) 寻址方式

寻址方式只在 Load/Store 指令中体现，每条指令分别实现的 8 种不同的数据流动方式；因为设计中包括两个寄存器 R0,R1,分别占用 4 种方式，包括两种寄存器寻址，立即数寻址，间接寻址。

3.3 确定总体结构

1) 总体结构图



R0,R1 为 8 位通用寄存器，IR 为 8 位指令寄存器，PC 为 8 位程序计数器，MAR 为 8 位地址寄存器；A、B 为两个选择器，分别通过 MA、RA 和 PB、RB 来控制是从寄存器还是 PC 和 RAM 中获取数据；选择器将数据传给 ALU，负责执行运算。

2) 数据通路

a) 取指

$$RAM \xrightarrow{MA} \text{选择器 A} \xrightarrow{A直传} \Sigma \rightarrow BUS \xrightarrow{CPIR} \text{IR}$$

b) 取指令地址

$$PC \xrightarrow{PB} \text{选择器 B} \xrightarrow{B直传} \Sigma \rightarrow BUS \xrightarrow{CPMAR} \text{IR}$$

c) PC+1

$$PC \xrightarrow{PB} \text{选择器 B} \xrightarrow{A+B+1(A=0)} \Sigma \rightarrow BUS \xrightarrow{CPPC} \text{IR}$$

d) RAM 到寄存器(R0)

$$RMA \xrightarrow{MA} \text{选择器 A} \xrightarrow{A直传} \Sigma \rightarrow BUS \xrightarrow{CPR0} \text{R0}$$

e) R0 与 R1 运算结果到 R0

$$R0 \xrightarrow{RA} \text{选择器 A} \xrightarrow{运算} \Sigma \rightarrow BUS \xrightarrow{CPR0} \text{R0}$$

$$R1 \xrightarrow{RB} \text{选择器 B} \xrightarrow{运算} \Sigma \rightarrow BUS \xrightarrow{CPR0} \text{R0}$$

f) 寄存器到 RAM(R0)

$$R0 \xrightarrow{RB} \text{选择器 B} \xrightarrow{B直传} \Sigma \rightarrow BUS \xrightarrow{C/WR} \text{RAM}$$

3.4 其它部件逻辑设计

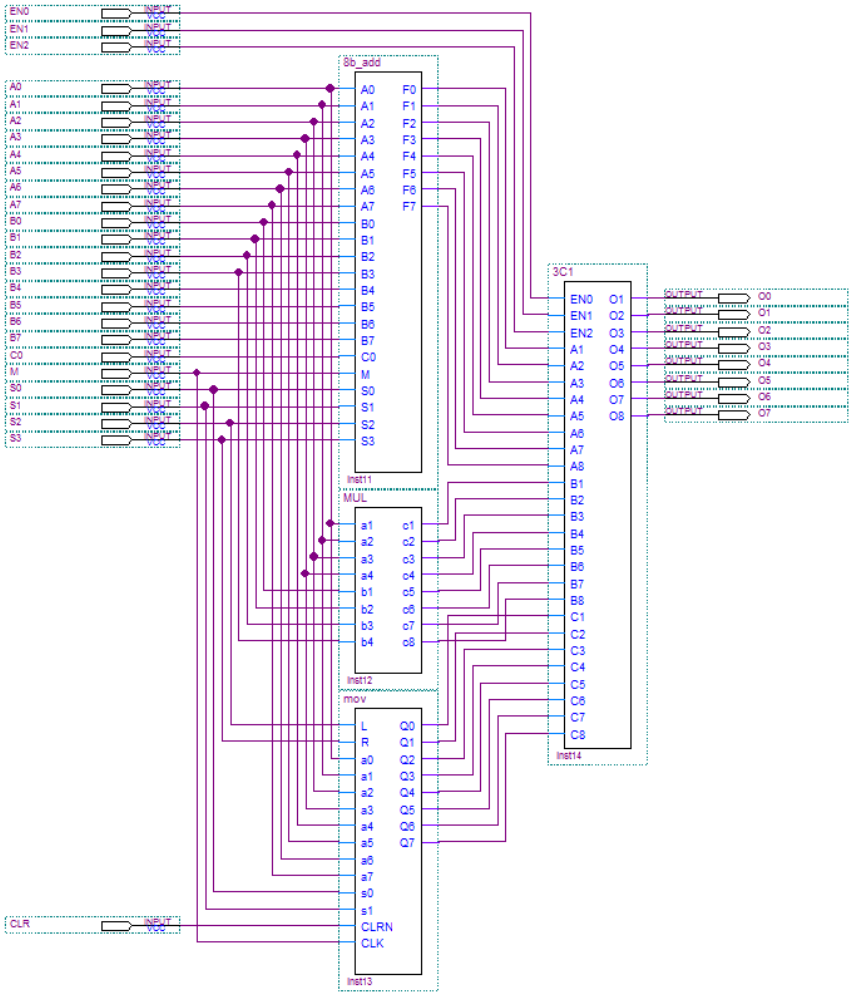
1) ALU 设计

ALU 为算术逻辑单元，所有进行运算的部件的输出通过选择器连接在一起，以有效信号控制具体进行的运算。本次实验使用的指令集只包括加法器和乘法器两部分，可通过二选一选择器和 EN0、EN1 两个控制信号来实现 ALU 的功能。具体电路中采用了三选一选择器，扩展了一个移位寄存器，只是并未在指令中体现。

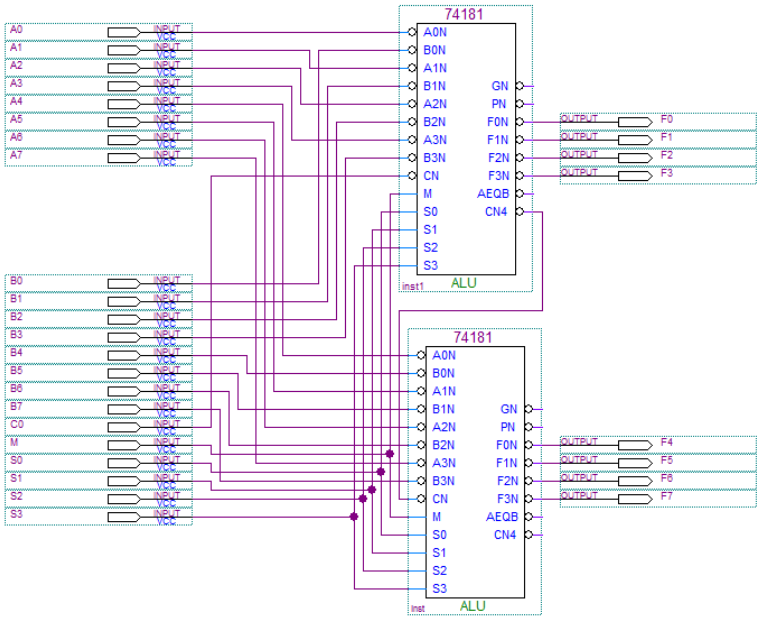
a) ALU 整体电路

包括 8 位加法器，4 位乘法器和一个 8 位移位寄存器，通过三个控制信号

EN0、EN1、EN2 和三选一选择器控制具体执行的运算



b) 8 位加法器电路，通过两片 74181 芯片连接形成

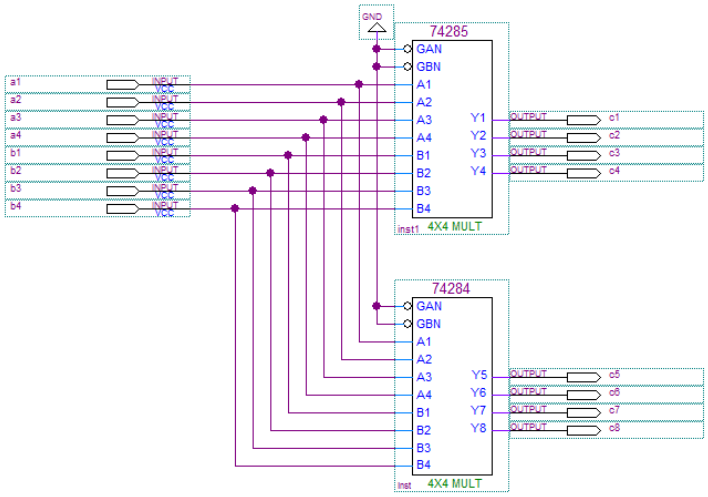


通过 M,CN 和 S0、S1、S2、S3 来控制运算，具体对应关系见下表

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

（上表中的“/”表示求反）

c) 4 位乘法器电路，通过 74285 和 74284 连接形成



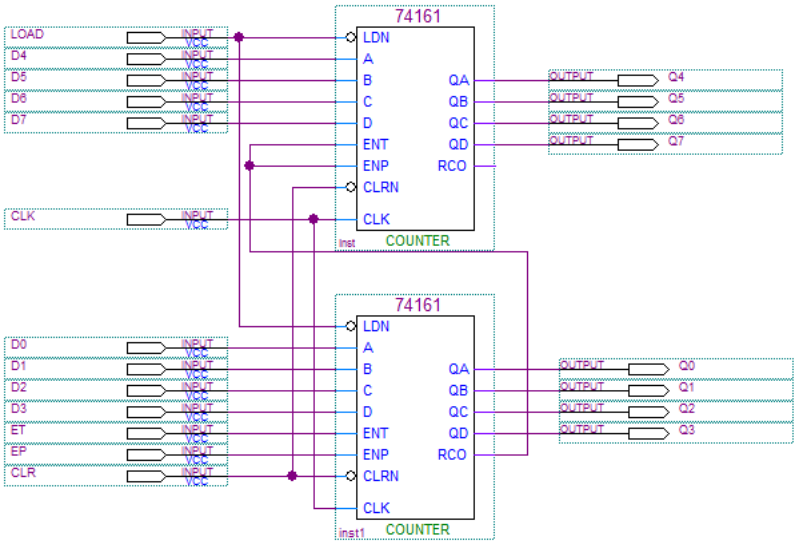
2) μPC 设计

采用两片 74161 连接起来实现 8 位计数器作为 μPC

CLR：清零端，低电平有效；CLR=0 时，Q7Q6Q5Q4Q3Q2Q1Q0=00000000；

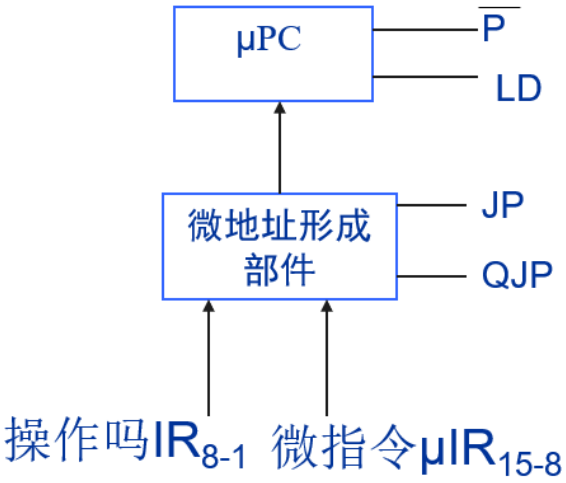
LOAD：置数端，低电平有效；LOAD=0 时，在 CLK 的上升沿，Q7Q6Q5Q4Q3Q2Q1Q0=D7D6D5D4D3D2D1D0；当 CLR=1，LOAD=1，ET=1，EP=1

时，对 CLK 进行增 1 计数。



3) 后继地址形成电路

后继地址形成电路负责形成微程序控制中下一条微指令的地址，实现了三种方式，分别为顺序执行、JP 无条件转移和 QJP 按操作码转移，结构框图如下

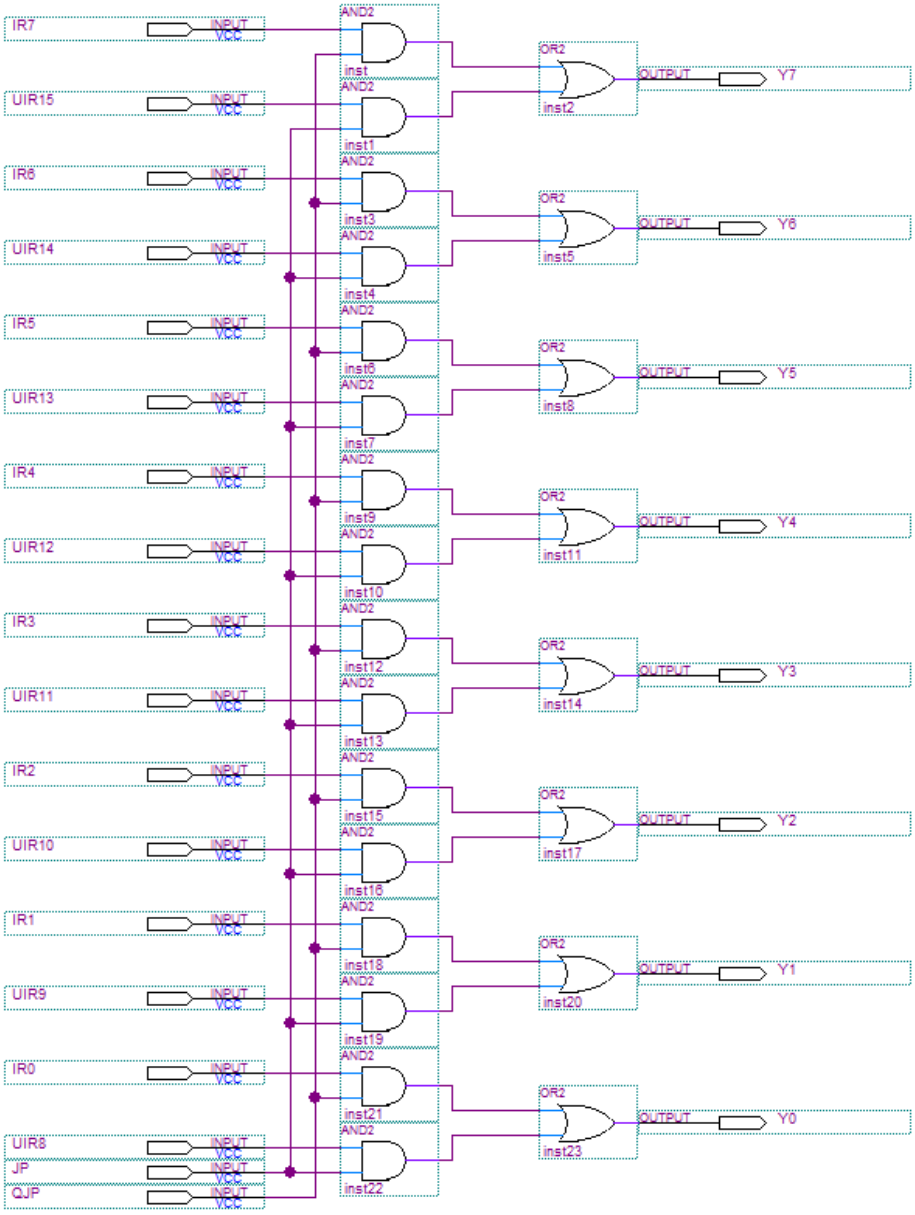


当 LD＝1 时，微程序计数 μPC 执行加 1 操作。

当 LD＝0 时且 JP＝1 时，无条件转移，有微指令的中八位提供转移地址。

当 LD＝0 时且 QJP＝1 时，按操作码转移。

具体电路：

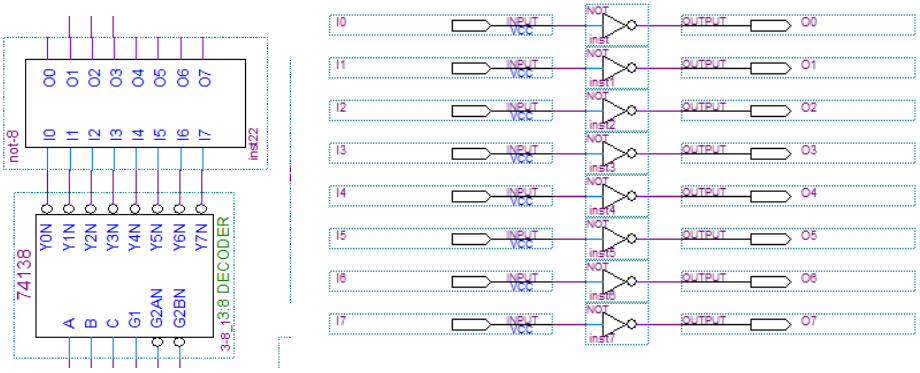


4) 译码器

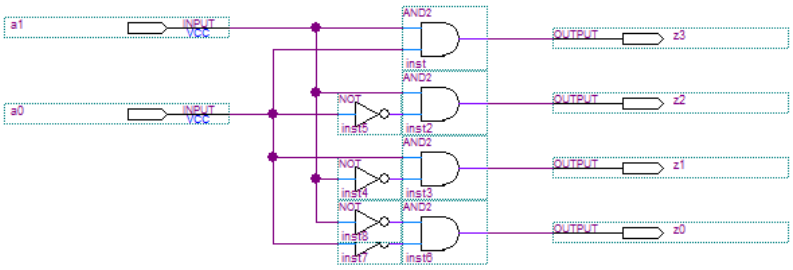
ROM 字长为 24 位，故一条微指令长度为 24 位，为了节约指令位数，对互斥的信号使用译码器形成。根据信号数目不同采用了 3-8 和 2-4 两种译码器

a) 3-8 译码器

使用 74138 实现，因为输出信号低电平后效，故添加了一个 8 位 NOT 门电路。



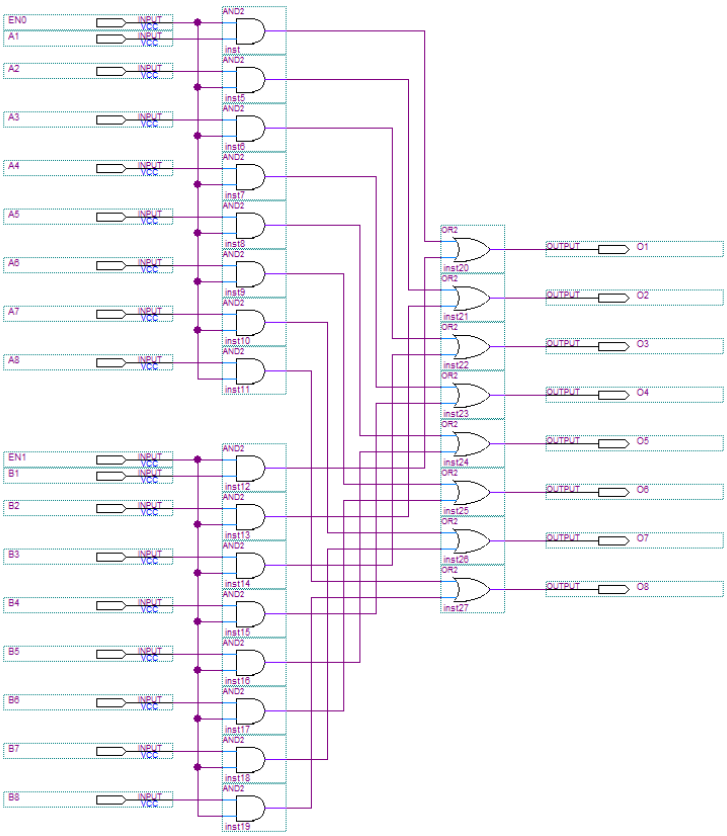
b） 2-4 译码器



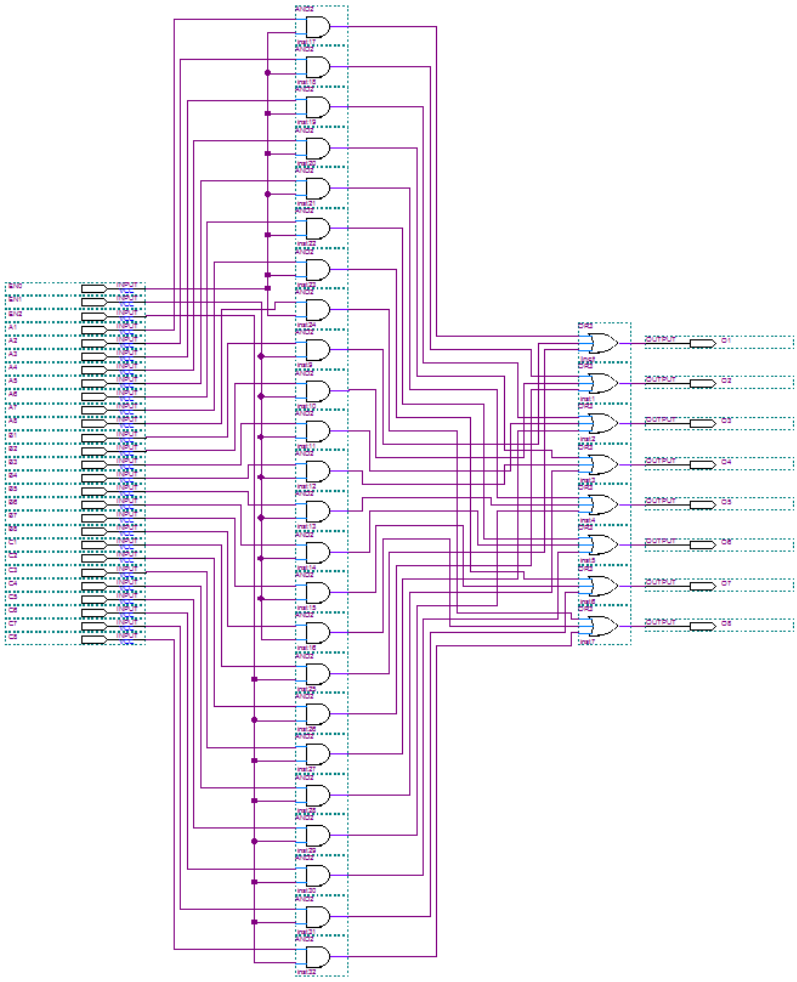
5） 选择器

根据需求不同选择器有二选一和三选一两种

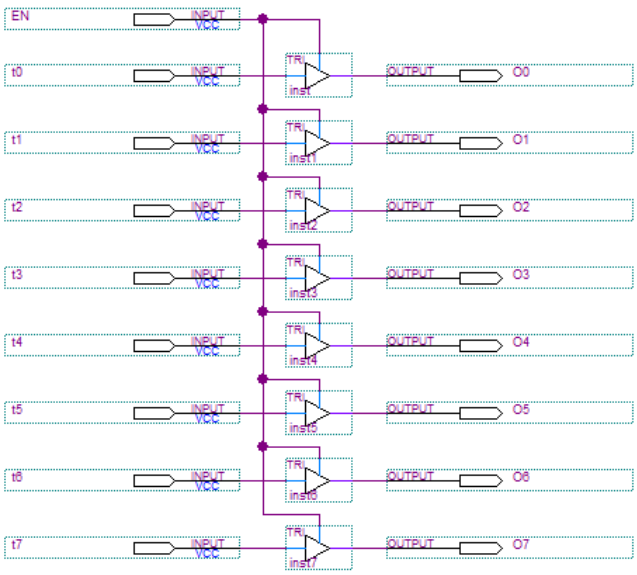
a） 二选一



b） 三选一

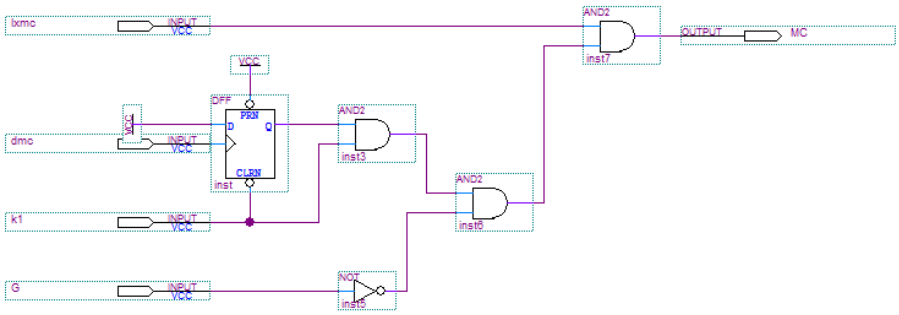


6) 三态门，用于 RAM 读写控制



7） 启停电路设计

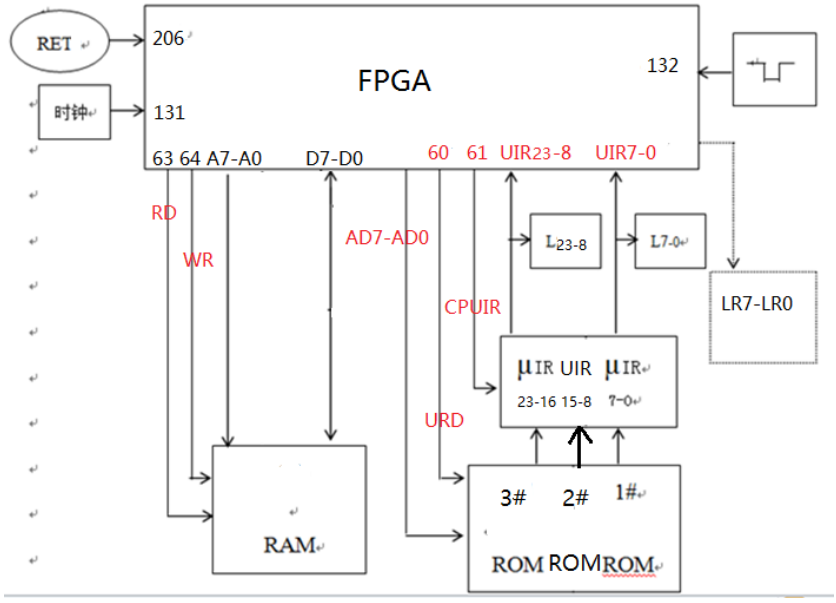
电路使用连续脉冲，设置如下启停电路，其中 K1 为连续脉冲控制开关，G 为停机信号，K1 为 1，摁下单脉冲启动连续脉冲；停机信号 G=1 停止。



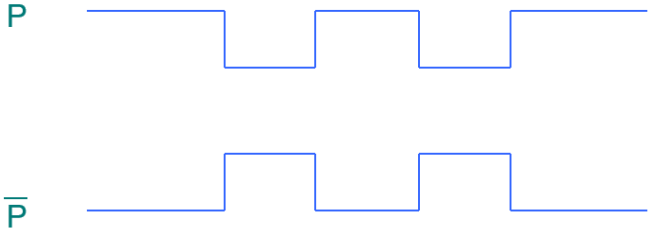
3.5 根据控制方式确定总体电路

3.5.1 微程序控制

1) 微程序的执行方式采用增量、垂直方式,维微程序控制器结构图如下:



2) 微程序控制时序



P 脉冲的低电平用做控制存储器读命令 μRD；

P 脉冲的上升边沿将读出的微指令送 μIR；

脉冲的上升边沿将形成的后继地址送微程序计数器 μPC，同时将运算结果（总线的数据）送指定的寄存器。

3) 微指令格式

微指令字长 24 位即 μIR_{23}～ μIR_0 具体微程序见附录。

a) 微指令字段定义

- ALU 控制： $\mu IR_{21}\cdot\mu IR_{20}\mu IR_{19}\cdot\mu IR_{18}\mu IR_{17}\cdot\mu IR_{16}$

  M S3 S2 S1 S0 C0

- 三态门控制： μIR6

  0 高阻态 使 C＝0

  1 三态门使能 使 C＝1

- 停机控制： μIR3

  0 G=0，运行

  1 G=1，停机

- A 选择器控制： $\mu IR_{15}\cdot\mu IR_{14}$

  0 0 备用

  0 1 RA

  1 0 MA

  1 1 备用

- B 选择器控制： $\mu IR_{13}\cdot\mu IR_{12}$

  0 0 备用

  0 1 PB

  1 0 RB

  1 1 备用

- 输出分配： $\mu IR_{11}\cdot\mu IR_{10}\cdot\mu IR_9$

  0 0 0 备用

  0 0 1 CPR_0

  0 1 0 CPR_1

  0 1 1 CPPC

1	0	0	CPIR
1	0	1	CPMAR
1	1	0	备用
0	1	1	备用

- 存储器读写控制： $\mu IR_5 \cdot \mu IR_4$

1	0	RD
0	1	WR

- 后继微地址形成方式

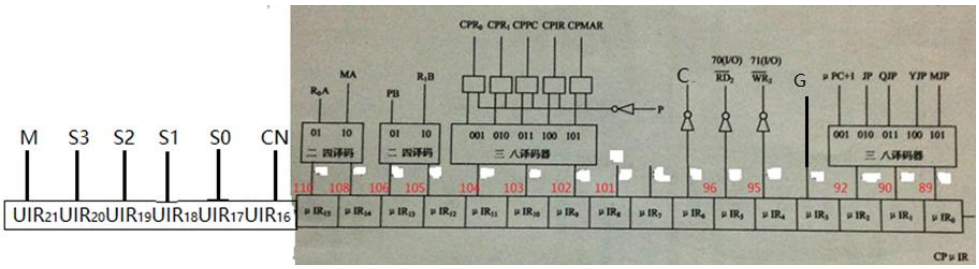
$\mu IR_2 \cdot \mu IR_1 \cdot \mu IR_0$

0	0	0	备用
0	0	1	μPC＋1　顺序执行
0	1	0	JP 无条件转移，地址由 $IR_{15\text{-}8}$ 提供
0	1	1	QJP 高四位按操码转移，低 4 位为 0
1	0	0	YJP 给定高 4 位低 4 位按源寻址方式转移
1	0	1	MJP 给定高 4 位低 4 位按目寻址方式转移
1	1	0	备用
1	1	1	备用

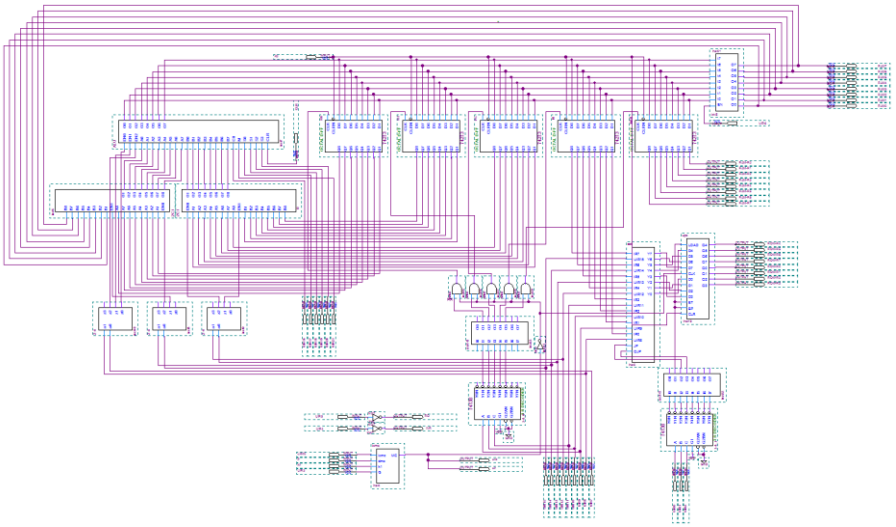
- ALU 选择信号

μIR24 位中 μIR_{22}、μIR_{23}、μIR_7 和 μIR_8 可用于生成 ALU 选择信号，最多可扩展 16 种运算部件，本次实验中使用后两位，经过 2-4 译码器译出，其中 μIR_7 和 μIR_8 初全为 0 则为加法器有效，01 为乘法器有效，10 为移位寄存器有效

a)　微命令形成逻辑电路

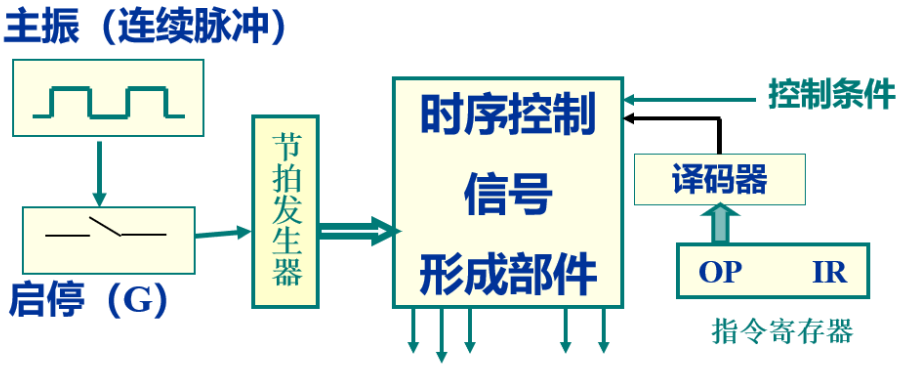


4)　整体电路图



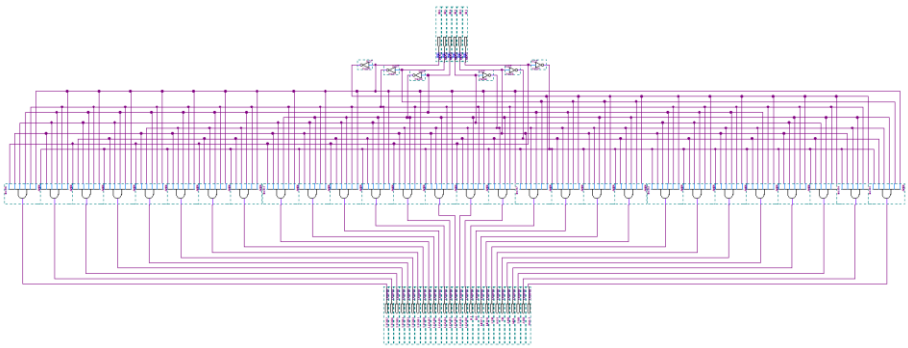
3.5.2 硬布线控制

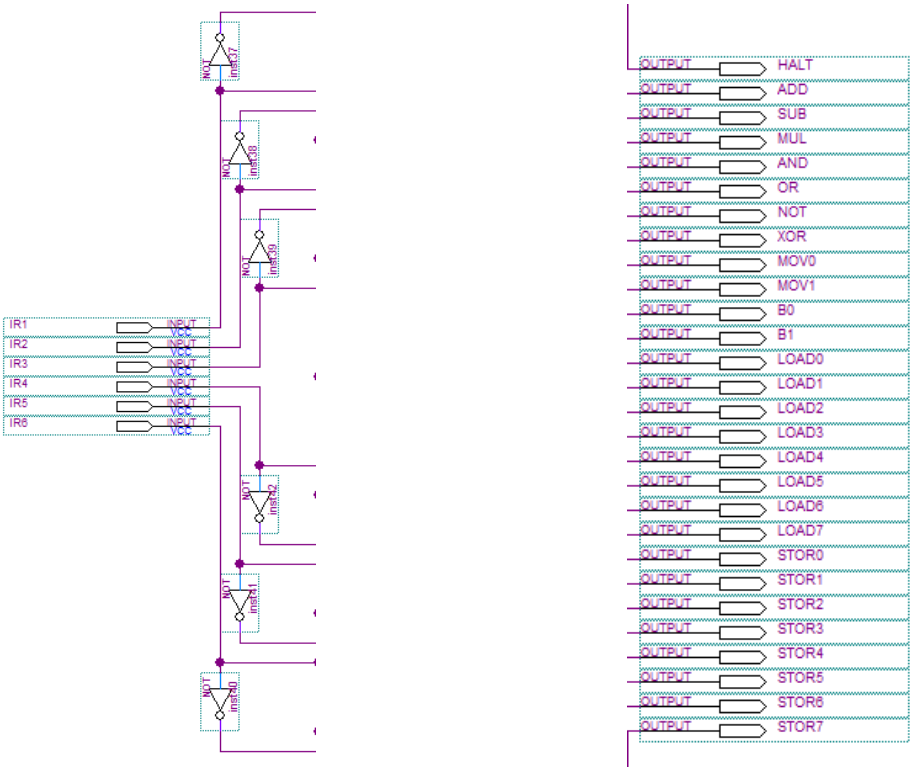
1) 硬布线逻辑电路控制器结构图



2) 操作码 OP 译码器设计

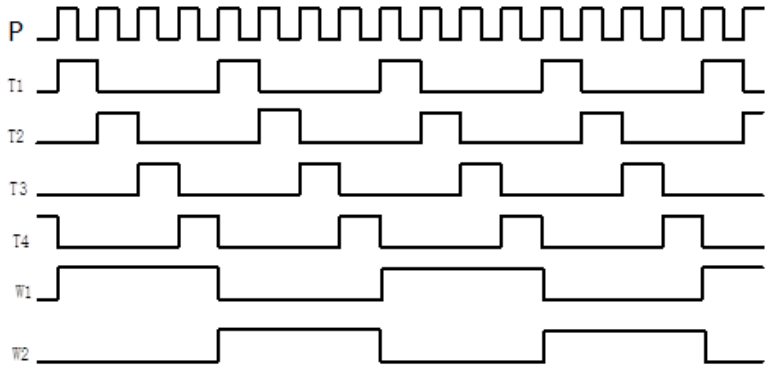
利用指令的操作码，得出逻辑表达式，利用与门、非门设计电路。





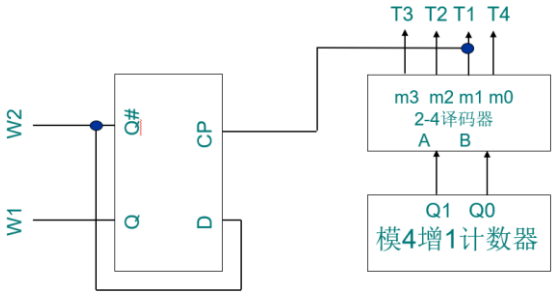
3) 时序与节拍发生器的设计

a) 根据指令执行流程，设计时序为：分取指周期和执行周期，每个周期为 4 节拍，波形图如下：

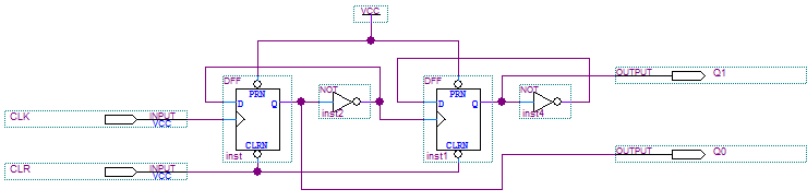


W1 高电平时是取指周期的 4 节拍，W2 高电平时是执行周期的 4 节拍。

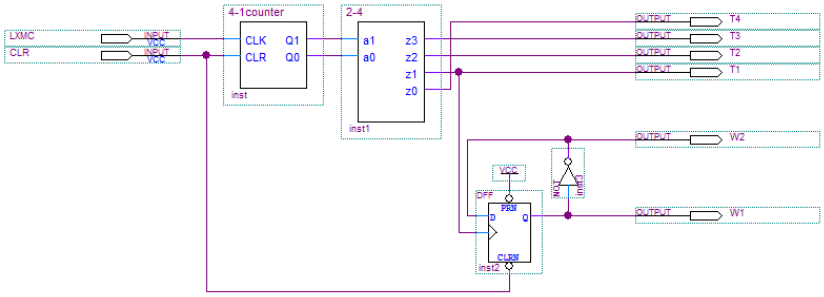
b) 节拍发生器实现电路框图如下：



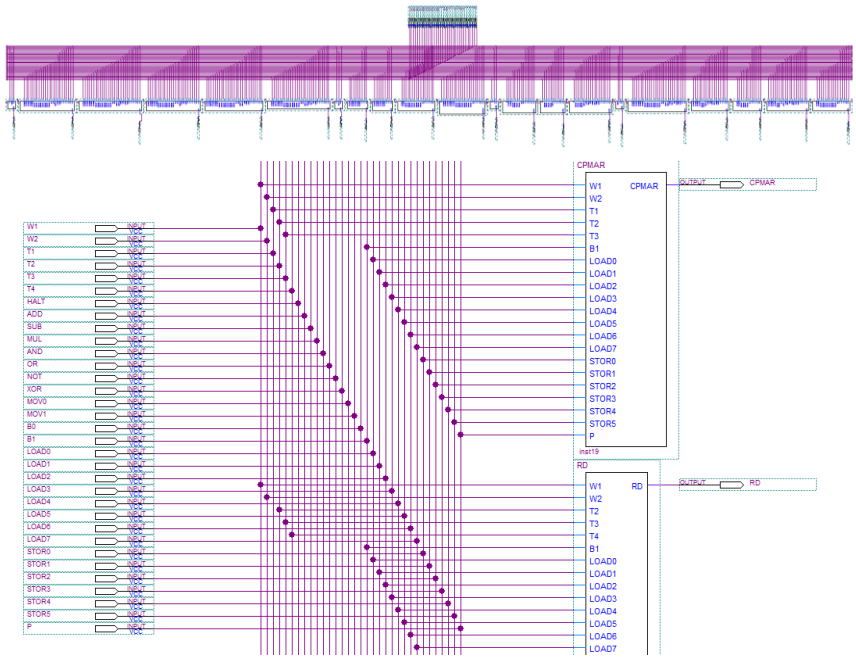
c) 模 4 增 1 计数器



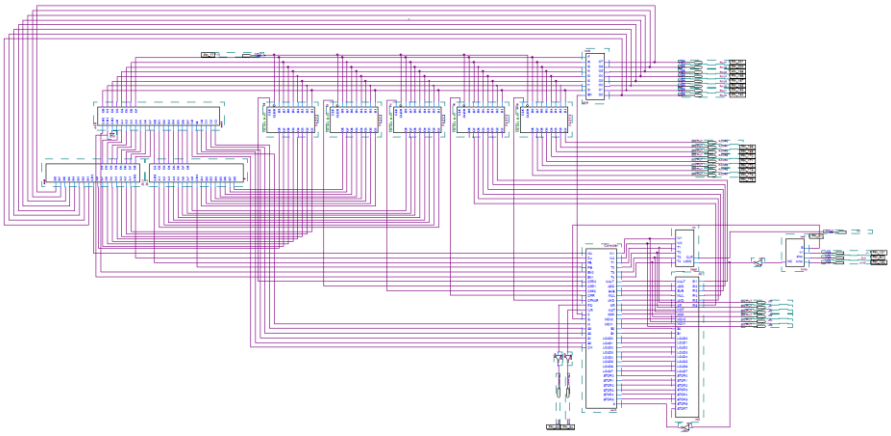
d) 整体电路



4) 根据指令流程和数据通路，写出控制信号的逻辑表达式，利用基本的与或非门实现控制信号，具体部件电路见附录。



5) 整体电路图



4 编写程序并调试

对于本实验设计的指令集，设计程序使其包含所有指令，并将计算结果和寄存器内容存入 RAM，通过 RAM 中的内容测试指令集和电路。程序如下：

LOAD R0 #0A	0100 1000	00001010	48 0A
LOAD R1 #09	0100 1010	00001001	4A 09
STOR #F0 R0	0101 1000	11110000	58 F0
STOR #F1 R1	0101 1010	11110001	5A F1
B	0011 0000		30
LOAD #F0 R0	0100 1100	11110000	4C F0
LOAD #F1 R1	0100 1110	11110001	4E F1
ADD	0000 0100		04
STOR #F2 R0	0101 1100	11110010	58 F2
LOAD #F0 R0	0100 1100	11110000	4C F0
LOAD #F1 R1	0100 1110	11110001	4E F1
SUB	0000 1000		08
STOR #F3 R0	0101 1100	11110011	58 F3

LOAD #F0 R0	0100 1100	11110000	4C F0	
LOAD #F1 R1	0100 1110	11110001	4E F1	
MUL	0000 1100		0C	
STOR #F4 R0	0101 1100	11110010	58 F4	
LOAD #F0 R0	0100 1100	11110000	4C F0	
LOAD #F1 R1	0100 1110	11110001	4E F1	
AND	0000 0100		10	
STOR #F5 R0	0101 1100	11110010	58 F5	
LOAD #F0 R0	0100 1100	11110000	4C F0	
LOAD #F1 R1	0100 1110	11110001	4E F1	
OR	0000 0100		14	
STOR #F6 R0	0101 1100	11110010	58 F6	
LOAD #F0 R0	0100 1100	11110000	4C F0	
NOT	0000 0100		18	
STOR #F7 R0	0101 1100	11110010	58 F7	
LOAD #F0 R0	0100 1100	11110000	4C F0	
LOAD #F1 R1	0100 1110	11110001	4E F1	
XOR	0000 0100		1C	
STOR #F8 R0	0101 1100	11110010	58 F8	
LOAD R0 #50	0100 1000	00111001	48 50	50 00 51 00
MOV1 R1=R0	0010 0100		24	50 50 51 00
LOAD R0 RAM[R1]	0100 0100		44	51 50 51 00
STOR R1 RAM[R0]	0101 0010		52	51 50 51 50
STOR R0 52	0101 1000		58 52	
STOR R1 53	0101 1010		5A 53	

HALT 0110 0000 60

如程序所示，最终实验结果存入从 F0~F8 以及 50~53RAM 单元内，运算所使用的两个操作数分别为 10 进制数 10 和 9，存在 F0 和 F8 单元，F3 到 F8 依次为加、减、乘、AND、OR、NOT、XOR，如图所示：

50	51
51	00
52	00
53	00
54	00

50	51
51	50
52	51
53	50
54	00

F0	00
F1	00
F2	00
F3	00
F4	00
F5	00
F6	00
F7	00
F8	00
F9	00
FA	00
FB	00
FC	00
FD	00
FE	00
FF	00

F0	0A
F1	09
F2	13
F3	01
F4	5A
F5	08
F6	0B
F7	F5
F8	03
F9	00
FA	00
FB	00
FC	00
FD	00
FE	00
FF	00

5 课程设计总结

5.1 问题和解决的方法

1) 指令集设计问题

因为硬件 ROM 容量限制，无法对所有指令实现多种寻址方式，因此设计

Load/Store 型指令集，但仍存在地址空间不足的问题，因此采用软件跳转的方式，将 Load 和 Store 指令的每一段独立的微程序的原位置处设置一条跳转微指令，跳到 ROM 中实际存放微程序的地方；但根据跳转地址的不同，无条件跳转方式有可能会被解析成其它微指令执行，从而导致最终结果错误，因此要合理设计跳转地址，如选择合适的地址从而生成保留的无效信号，从而消除此类影响；对于没有寻址方式的指令，设计为 6 位操作码，从而进一步节省了地址空间；

2) 硬布线中因为指令和控制信号较多，连线十分复杂，采用了独立的模块化方式，对每一个信号单独建立一个模块，最后再把所有模块连接在一起，尽可能地减少了错误的可能；

3) 在测试过程中，经常出现 RAM 内容被修改的问题，经过多方面调试，发现有可能是 RAM 读写信号接错、接反，脉冲信号上升及下降沿接反等等原因，经过修改后修复；但也存在试验台过热或损坏等其它原因，更换试验台可解决。

5.2 收获与体会

通过这次课程设计，总结了计算机组成原理课程的学习内容，把所学的知识与实践结合起来；加深了对计算机组成的理解，巩固了课堂知识；通过对简单模型机的设计，更深刻的掌握了微程序和硬布线两种实现 CPU 控制的方式，并更好地理解了它们之间的差别；同时，此次实验还考验了我们的细心和耐心，以及发现问题解决问题的能力，提高了我们调错的能力和意识，为今后的学习生活留下了宝贵的经验。

6 附录

6.1 指令集

指令	编码	入口地址	操作
ADD	00000100	04	R0=R0+R1
SUB	00001000	08	R0=R0-R1
MUL	00001100	0C	R0=R0*R1
AND	00010000	10	R0=R0&R1

OR	00010100	14	R0=R0||R1
NOT	00011000	18	R0=! R0
XOR	00011100	1C	R0=R0 ^ R1
MOV0	00100000	20	R0=R1
MOV1	00100100	24	R1=R0
B0	00110000	30	PC=R0
B1	00110100	34	PC=*(PC+1)
LOAD0	01000000	40	R0=RAM(R0)
LOAD1	01000010	42	R1=RAM(R0)
LOAD2	01000100	44	R0=RAM(R1)
LOAD3	01000110	46	R1=RAM(R1)
LOAD4	01001000	48	R0=#X
LOAD5	01001010	4A	R1=#X
LOAD6	01001100	4C	R0=RAM(#X)
LOAD7	01001110	4E	R1=RAM(#X)
STOR0	01010000	50	RAM(R0)=R0
STOR1	01010010	52	RAM(R0)=R1
STOR2	01010100	54	RAM(R1)=R0
STOR3	01010110	56	RAM(R1)=R1
STOR4	01011000	58	RAM(#X)=R0
STOR5	01011010	5A	RAM(#X)=R1
STOR6	01011100	5C	RAM(RAM(#X))=R0
STOR7	01011110	5E	RAM(RAM(#X))=R1
HALT	01100000	60	G=1

6.2 微程序

00H:

RAM(MAR)->IR 3E8821

PC+1->PC //取指周期 121601

QJP 000003

04H:

R0+R1->R0 136201

PC->MAR 341A01

JP //ADD 000002

08H:

R0-R1->R0 0C6201

PC->MAR 341A01

JP //SUB 000002

0CH:

RO mul R1->R0 016281

PC->MAR 341A01

JP //MUL 000002

10H:

R0 and R1->R0 366201

PC->MAR 341A01

JP //AND 000002

14H:

R0 or R1->R0 3C6201 341A01 000002

PC->MAR

JP //OR

18H:

! R0->R0 204201 341A01 000002

PC->MAR

JP //NOT

1CH:

R0 xor R1->R0

PC->MAR

JP //XOR 2C6201 341A01 000002

20H:

R0=R1

PC->MAR

JP //MOV0 346201 341A01 000002

24H:

R1=R0

PC->MAR

JP //MOV1 3E6401 341A01 000002

28H:

R0=RO

R0=R0/2

PC->MAR

JP //ROR 264301 240301 341A01 000002

2CH:

R0=R0

R0=R0*2

PC->MAR

JP //LOR 264301 220301 341A01 000002

30H:

R0->PC

PC->MAR

JP //B0 3E6601 341A01 000002

34H:

PC->MAR

PC+1->PC

RAM(MAR)->PC

PC->MAR

JP //B1 341A01 121601 3E8621 341A01 000002

40H:

JP RAM(R0)->R0 00A002 000000

42H:

JP RAM(R0)->R1 00A402 000000

44H:

JP RAM(R1)->R0 00A802 000000

46H:

JP RAM(R1)->R1 00AC02 000000

48H:

JP #X->R0 00B002 000000

4AH:

JP #X->R1 00B802 000000

4CH:

JP RAM(#X)->R0 00C002 000000

4EH:

JP RAM(#X)->R1 00C802 000000

50H:

JP R0->RAM(R0) 00D002 000000

52H:

JP R1->RAM(R0) 00D402 000000

54H:

JP R0->RAM(R1) 00D802 000000

56H:

JP R1->RAM(R1) 00DC02 000000

58H:

JP R0->RAM(#X) 00E002 000000

5AH:

JP R1->RAM(#X) 00E802 000000

5CH:

JP R0->RAM(RAM(#X)) 00F002 000000

5EH:

JP R1->RAM(RAM(#X)) 00F802 000000

60H:

G=1 000008

A0H: RAM(R0)->R0: 3E4A01 3E8221 341A01 000002

R0->MAR

RAM(MAR)->R0

PC->MAR

JP

A4H：RAM(R0)->R1: 3E4A01 3E8421 341A01 000002

R0->MAR

RAM(MAR)->R1

PC->MAR

JP

A8H: RAM(R1)->R0: 3E2A01 3E8221 341A01 000002

R1->MAR

RAM(MAR)->R0

PC->MAR

JP

ACH：RAM(R1)->R1: 3E2A01 3E8421 341A01 000002

R1->MAR

RAM(MAR)->R1

PC->MAR

JP

B0H: #X->R0: 341A01 121601 3E8221 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->R0

PC->MAR

JP

B8H: #X->R1: 341A01 121601 3E8421 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->R1

PC->MAR

JP

C0H: RAM(RAM[#X])->R0: 341A01 121601 3E8A21 3E8221 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

RAM(MAR)->R0

PC->MAR

JP

C8H: RAM(RAM[#X])->R1: 341A01 121601 3E8A21 3E8421 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

RAM(MAR)->R1

PC->MAR

JP

D0H: R0->RAM(R0): 3E4A01 3E4051 341A01 000002

R0->MAR

R0->RAM(MAR)

PC->MAR

JP

D4H: R1->RAM(R0): 3E4A01 342051 341A01 000002

R0->MAR

R1->RAM(MAR)

PC->MAR

JP

D8H: R0->RAM(R1): 342A01 3E4051 341A01 000002

R1->MAR

R0->RAM(MAR)

PC->MAR

JP

DCH: R1->RAM(R1): 342A01 342051 341A01 000002

R1->MAR

R1->RAM(MAR)

PC->MAR

JP

E0H: R0->RAM(#X): 341A01 121601 3E8A21 3E4051 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

R0->RAM(MAR)

PC->MAR

JP

E8H: R1->RAM(#X): 341A01 121601 3E8A21 342051 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

R1->RAM(MAR)

PC->MAR

JP

F0H: R0->RAM(RAM[#X]): 341A01 121601 3E8A21 3E8A21 3E4051 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

RAM(MAR)->MAR

R0->RAM(MAR)

PC->MAR

JP

F8H: R1->RAM(RAM[#X]): 341A01 121601 3E8A21 348A21 342051 341A01 000002

PC->MAR

PC+1->PC

RAM(MAR)->MAR

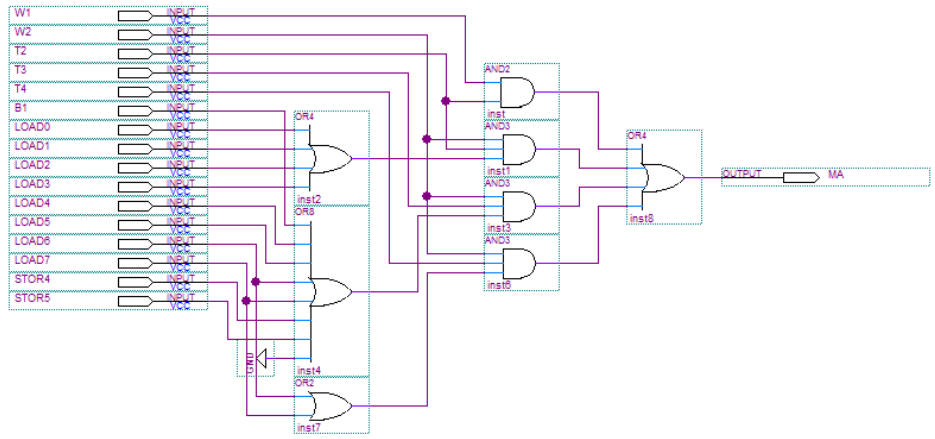
RAM(MAR)->MAR

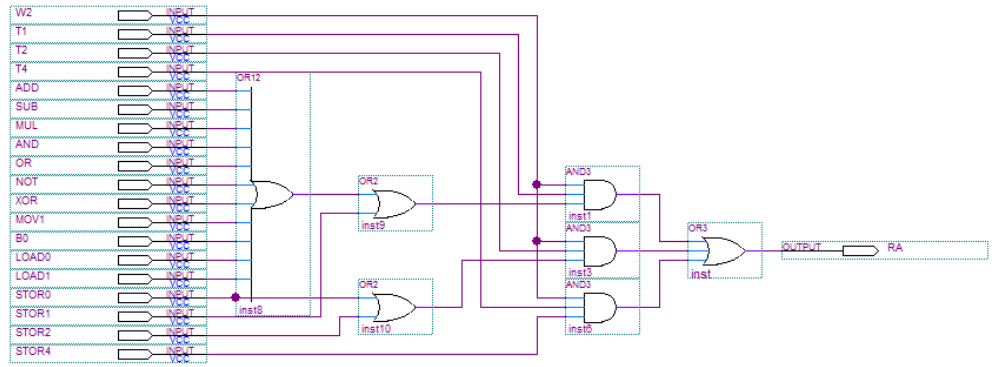
R1->RAM(MAR)

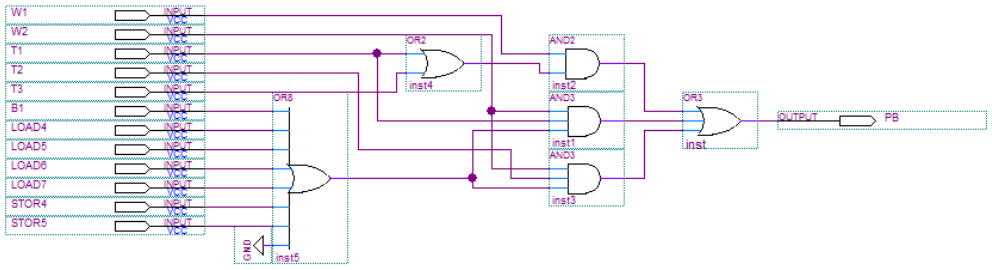
PC->MAR

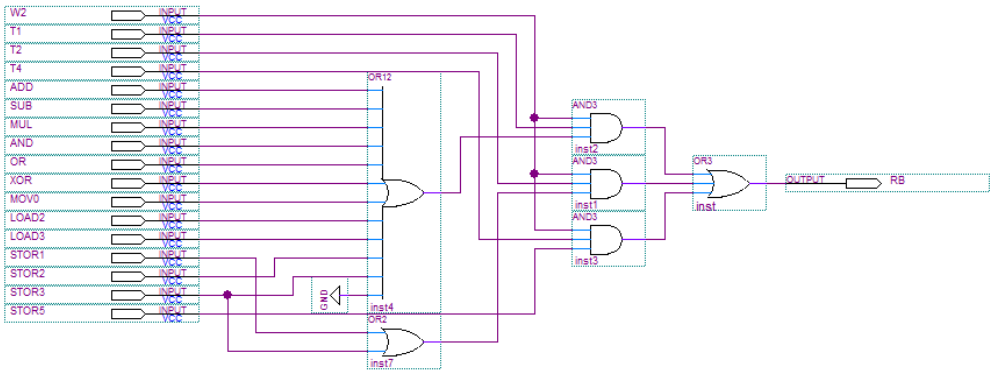
JP

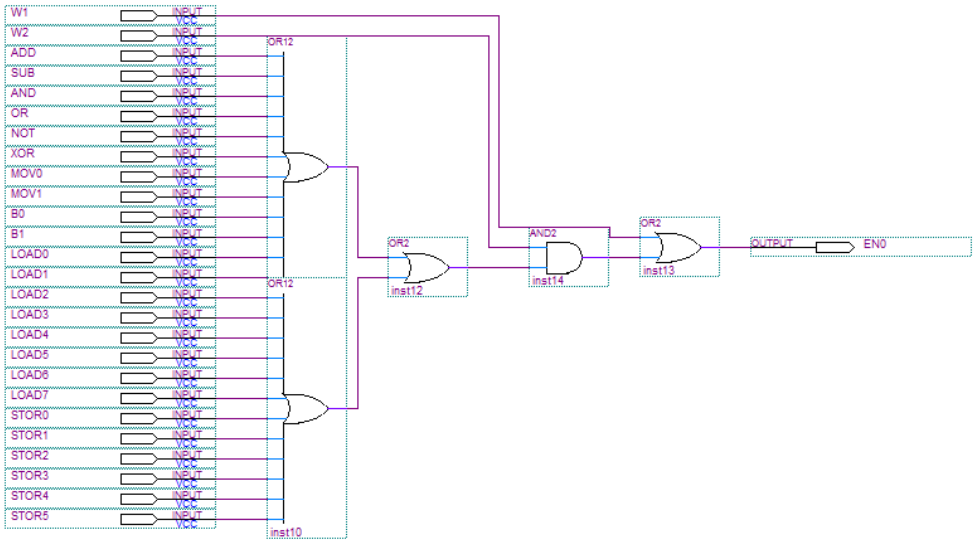
6.3 控制信号生成电路

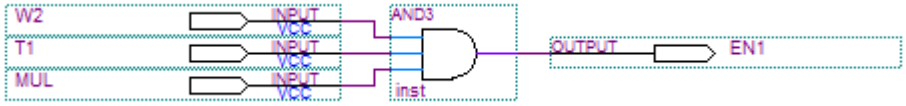


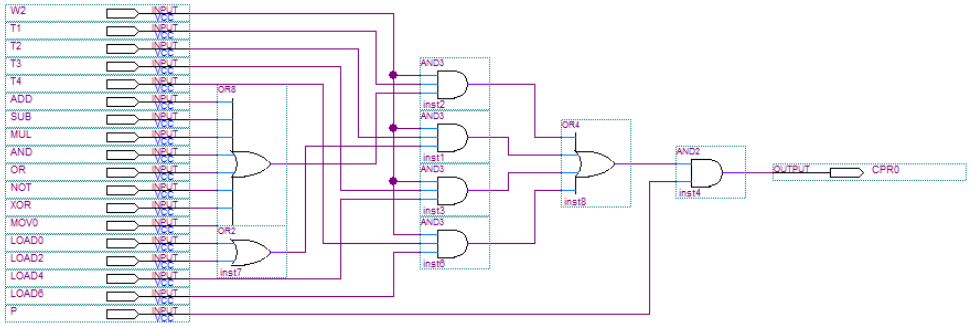


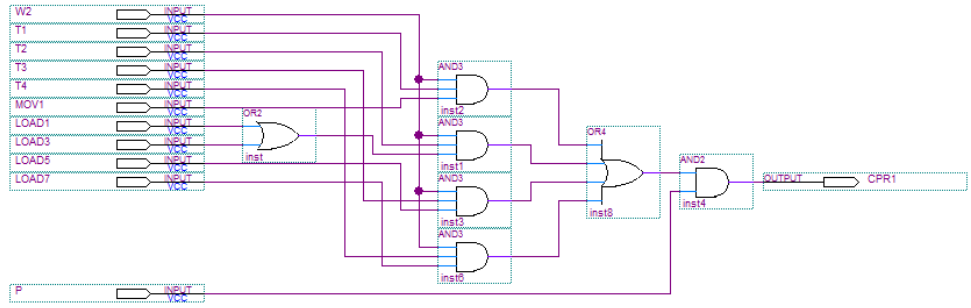


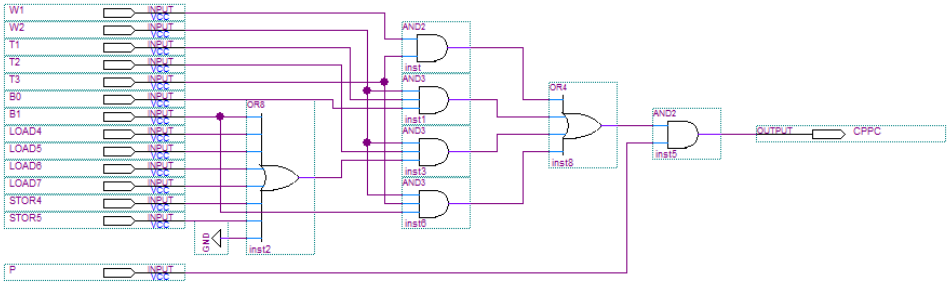

W2
T1
T2
T4
ADD
SUB
MUL
AND
OR
XOR
MOV0
LOAD2
LOAD3
STOR1
STOR2
STOR3
STOR5
INPUT
VCC
OR12
inst4
OR2
inst7
AND3
inst2
inst1
inst3
OR3
inst
OUTPUT
RB

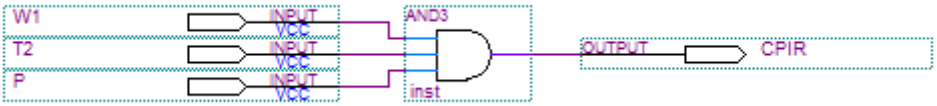

W1
W2
ADD
SUB
AND
OR
NOT
XOR
MOV0
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
STOR3
STOR4
STOR5
INPUT
VCC
OR12
inst10
OR2
inst12
AND2
inst14
inst13
OUTPUT
EN0

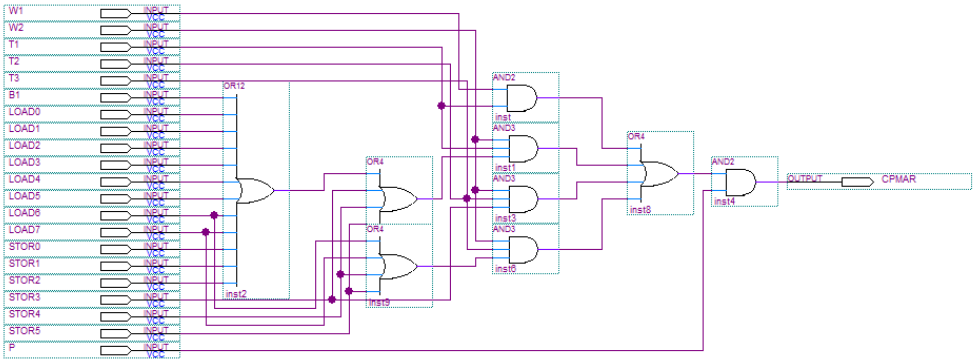

W2
T1
MUL
INPUT
VCC
AND3
inst
OUTPUT
EN1

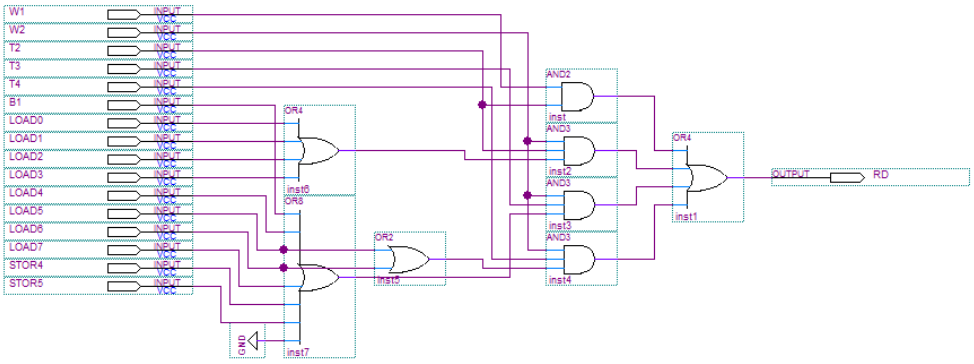

W2
T1
T2
T3
T4
ADD
SUB
MUL
AND
OR
NOT
XOR
MOV0
LOAD0
LOAD2
LOAD4
LOAD6
P
INPUT
VCC
OR8
inst7
AND3
inst2
inst1
inst3
inst5
OR4
inst8
AND2
inst4
OUTPUT
CPR0

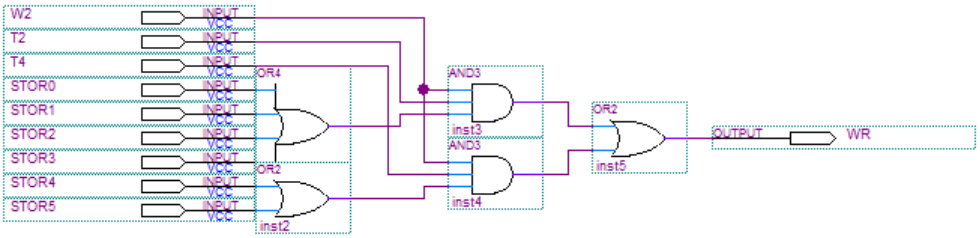

W2
T1
T2
T3
T4
MOV1
LOAD1
LOAD3
LOAD5
LOAD7
P
INPUT
VCC
OR2
inst1
AND3
inst2
inst1
inst3
inst5
OR4
inst8
AND2
inst4
OUTPUT
CPR1

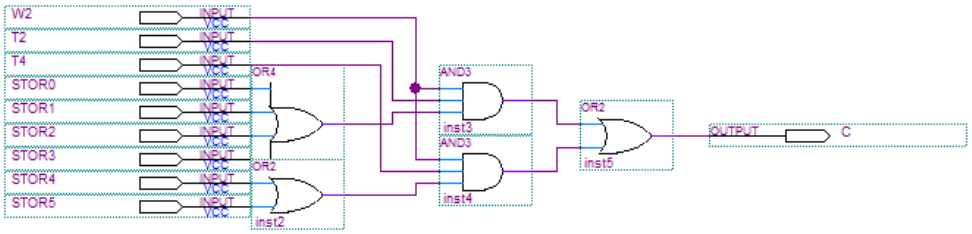

W1
W2
T1
T2
T3
B0
B1
LOAD4
LOAD5
LOAD6
LOAD7
STOR4
STOR5
P
INPUT
VCC
OR8
GND
inst2
AND2
inst
AND3
inst1
inst3
inst6
OR4
inst8
AND2
inst5
OUTPUT
CPPC

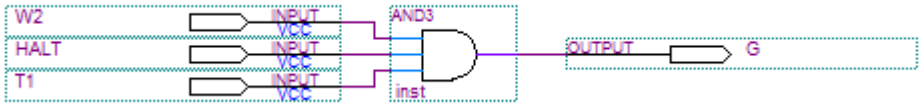

W1
T2
P
INPUT
VCC
AND3
inst
OUTPUT
CPIR

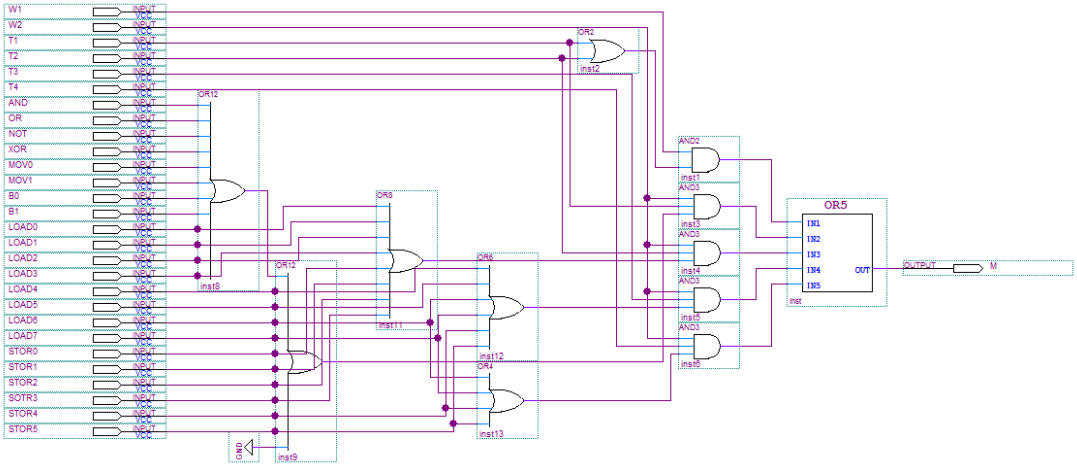

W1
W2
T1
T2
T3
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
STOR3
STOR4
STOR5
P
INPUT
VCC
OR12
inst2
OR4
inst9
AND2
inst
AND3
inst1
inst3
inst6
OR4
inst8
AND2
inst4
OUTPUT
CPMAR

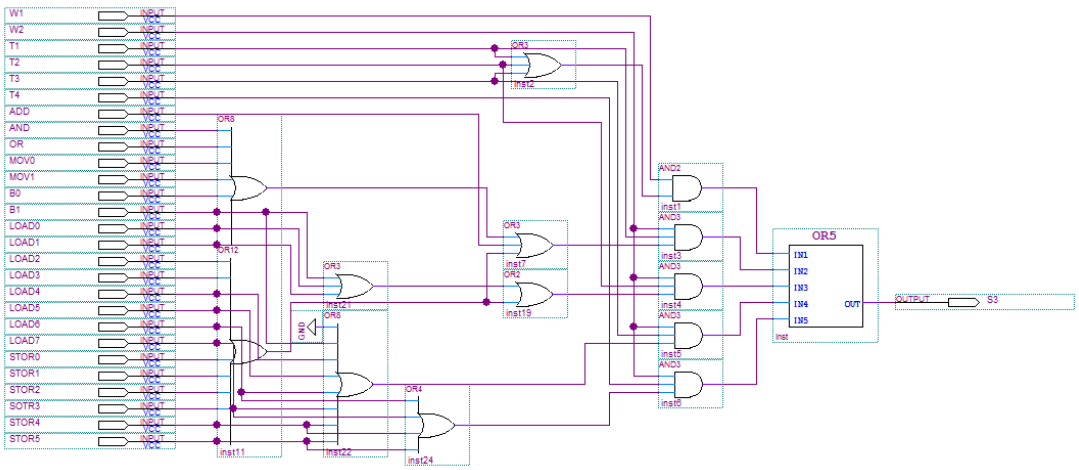

W1
W2
T2
T3
T4
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR4
STOR5
INPUT
VCC
OR4
inst6
OR8
GND
inst7
OR2
inst5
AND2
inst
AND3
inst2
inst3
inst4
OR4
inst1
OUTPUT
RD

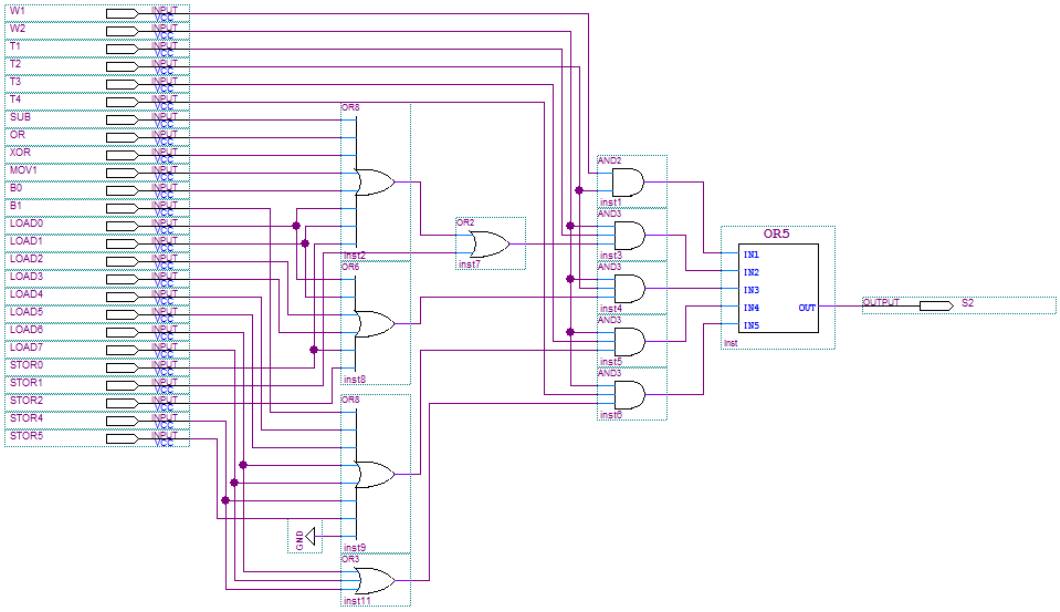

W2
T2
T4
STOR0
STOR1
STOR2
STOR3
STOR4
STOR5
INPUT
VCC
OR4
OR2
inst2
AND3
inst3
inst4
OR2
inst5
OUTPUT
WR

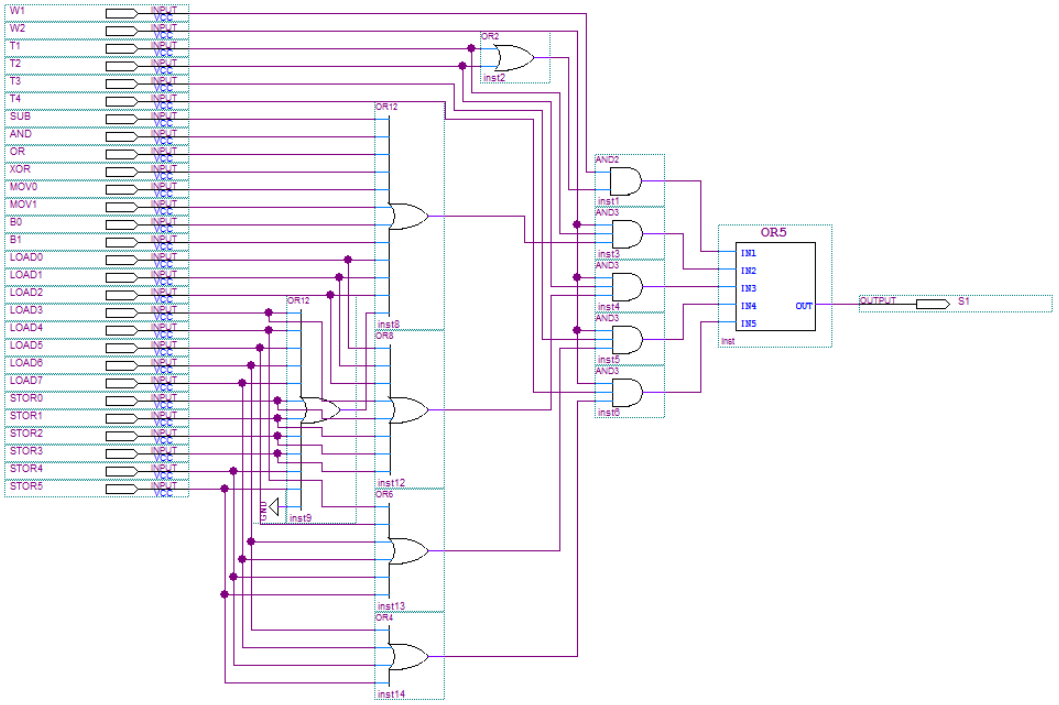

W2
T2
T4
STOR0
STOR1
STOR2
STOR3
STOR4
STOR5
INPUT
VCC
OR4
OR2
inst2
AND3
inst3
inst4
OR2
inst5
OUTPUT
C

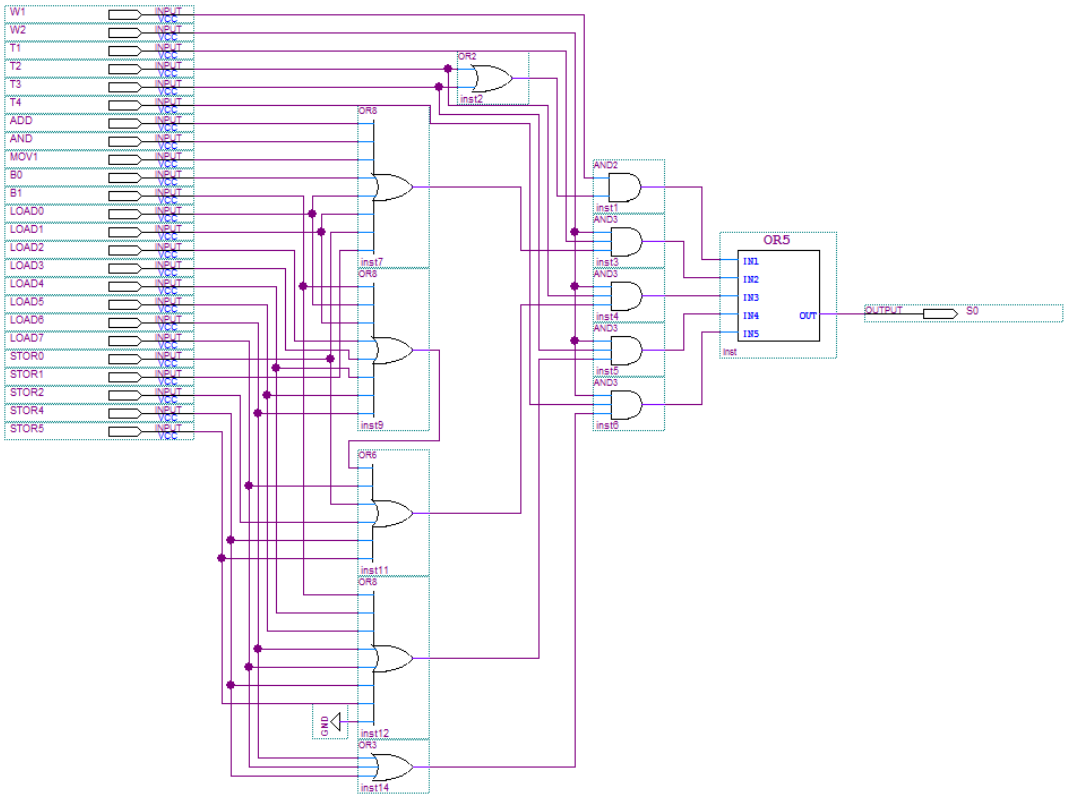

W2
HALT
T1
INPUT
VCC
AND3
inst
OUTPUT
G

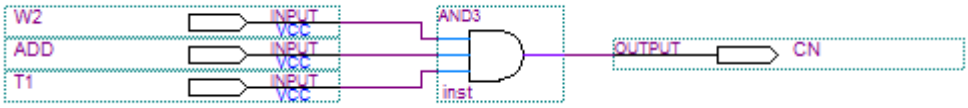

W1
W2
T1
T2
T3
T4
AND
OR
NOT
XOR
MOV0
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
SOTR3
STOR4
STOR5
INPUT
VCC
OR12
OR2
inst2
inst8
OR5
inst11
OR6
inst12
OR4
inst13
GND
inst9
AND2
inst1
AND3
inst3
inst4
inst5
inst6
OR5
IN1
IN2
IN3
IN4
IN5
OUT
inst
OUTPUT
M


W1
W2
T1
T2
T3
T4
ADD
AND
OR
MOV0
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
SOTR3
STOR4
STOR5
INPUT
VCC
OR3
inst12
OR6
OR12
inst11
OR3
inst21
OR6
inst22
GND
OR4
inst24
OR3
inst7
OR2
inst19
AND2
inst1
AND3
inst3
inst4
inst5
inst6
OR5
IN1
IN2
IN3
IN4
IN5
OUT
inst
OUTPUT
S3


W1
W2
T1
T2
T3
T4
SUB
OR
XOR
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
STOR4
STOR5
INPUT
VCC
OR8
OR2
OR6
OR3
GND
inst1
inst2
inst7
inst8
inst9
inst11
AND2
AND3
inst3
inst4
inst5
inst6
OR5
IN1
IN2
IN3
IN4
IN5
OUT
inst
OUTPUT
S2


W1
W2
T1
T2
T3
T4
SUB
AND
OR
XOR
MOV0
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
STOR3
STOR4
STOR5
INPUT
VCC
OR2
inst2
OR12
inst8
OR6
inst12
inst13
OR4
inst14
GND
inst9
AND2
AND3
inst1
inst3
inst4
inst5
inst6
OR5
IN1
IN2
IN3
IN4
IN5
OUT
inst
OUTPUT
S1


W1
W2
T1
T2
T3
T4
ADD
AND
MOV1
B0
B1
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
STOR0
STOR1
STOR2
STOR4
STOR5
INPUT
VCC
OR2
inst2
OR8
inst7
inst9
OR6
inst11
OR3
inst12
inst14
GND
AND2
inst1
AND3
inst3
inst4
inst5
inst6
OR5
IN1
IN2
IN3
IN4
IN5
OUT
inst
OUTPUT
S0


W2
ADD
T1
INPUT
VCC
AND3
inst
OUTPUT
CN