《计算机组成与设计》课程设计报告

—— 简单模型机设计（微程序实现与硬布线实现）

班级：18 级计科一班

姓名：王志睿

学号：201800130069

完成日期：2020/12/4

目录

1. 指令系统设计 1

1.1 指令组 1

1. 2 指令格式设计 6

1. 3 指令类型与寻址方式 7

2. 总体结构与数据通路 11

2. 1 总体结构与数据通路框图 11

2. 2 部件逻辑设计 12

3. 微程序实现的控制部件 CU 37

3.1 CU 框图 37

3.2 设计原理图 38

3.3 节拍安排 44

3.4 指令执行流程设计 44

3.5 微指令格式 44

3.6 ROM 内的微程序及地址 48

3.7 汇编程序测试 49

4. 硬布线实现的控制部件 CU 53

4.1 CU 框图 53

4.2 指令执行流程 54

4.3 原理图设计流程 55

4.4 汇编程序测试 62

5. 课程设计总结 64

5.1 遇到的问题和解决的方法 64

5.2 收获与体会 66

1. 指令系统设计

1.1 指令组

（1）微程序实现的指令组以及各条指令的功能如下：

指令	操作码	功能解释
lui rt,imm	00001	rt = imm
add rt,rs	00010	rt = rt + rs
sub rt,rs	00011	rt = rt - rs
and rt,rs	00100	rt = rt & rs
or rt,rs	00101	rt = rt | rs
xor rt,rs	00110	rt = rt ^ rs
addi rt,imm	00111	rt = rt + imm
lw rt,rs,imm	01000	rt = memory[rs+imm]

sw rt,rs,imm	01001	memory[rs+imm] = rt
jr rs	01010	PC = rs
beq rt,rs,imm	01011	PC = (rs==rt)?imm:PC
bne rt,rs,imm	01100	PC = (rs!=rt)?imm:PC
slli rt,imm	01101	rt = rt << imm
srli rt,imm	01110	rt = rt >> imm
mul rt,rs	01111	rt = rt * rs
div rt,rs	10000	rt = rt / rs
subi rt,imm	10001	rt = rt - imm
swap rt,rs	10010	rt <-> rs
push rs	10011	sp = sp - 1
	10100	rs -> stack top

pop rt	10101	stack top -> rt
	10110	sp = sp + 1
sll rt,rs	10111	rt = rt << rs
srl rt,rs	11000	rt = rt >> rs
mov ss,rs	11001	ss = rs
mov sp,rs	11010	sp = rs
call func	11011	push PC
	11100	PC=func
ret	11101	pop PC
	11110	

其中 rt 为目的通用寄存器、rs 为源通用寄存器，imm 为立即数、func 为子程序地址

（2）硬布线实现的指令组以及各条指令的功能如下：

指令	操作码	功能解释
lui rt,imm	00001	rt = imm
lw rt,rs,imm	00010	rt = memory[rs+imm]
sw rt,rs,imm	00011	memory[rs+imm] = rt
add rt,rs	00100	rt = rt + rs
sub rt,rs	00101	rt = rt - rs
and rt,rs	00110	rt = rt & rs
or rt,rs	00111	rt = rt | rs
xor rt,rs	01000	rt = rt ^ rs
mul rt,rs	01001	rt = rt * rs
div rt,rs	01010	rt = rt / rs

addi rt,imm	01011	rt = rt + imm
slli rt,imm	01100	rt = rt << imm
srli rt,imm	01101	rt = rt >> imm
jr rs	01110	PC = rs
beq rt,rs,imm	01111	PC = (rs==rt)?imm:PC
bne rt,rs,imm	10000	PC = (rs!=rt)?imm:PC

其中 rt 为目的通用寄存器、rs 为源通用寄存器，imm 为立即数

1. 2 指令格式设计

基本字长

存储器容量为 256×16 ， 基本字长定为 16 位

指令格式

op(5)	rt(2)	rs(2)	imm(7)

其中，op 字段为操作码字段共 5 位，最多可设置 32 条指令，rt 字段为目的通用寄存器字段共 2 位，rs 字段为源通用寄存器字段共 2 位，rt、rs 字段均可寻址 4 个通用寄存器，imm 字段为指令的立即数字段共 7 位，既可作为操作数字段又可作为寻址字段。

1. 3 指令类型与寻址方式

各条指令所对应的功能及操作码的编码已在上一部分的表格中列出，这一部分重点讲解指令的类型与寻址方式：

存取指令

(1) lui rt,imm

将 imm 字段的内容存入 rt 所指定的通用寄存器中，寻址方式为立即数寻址+寄存器寻址

(2) lw rt,rs,imm

将编号为(rs+imm) 的内存单元中的内容存入 rt 所指定的通用寄存器中，寻址方式为寄存器相对寻址（变址寻址）

(3) sw rt,rs,imm

将 rt 所指定的通用寄存器中的内容存入编号为(rs+imm)的内存单元，寻址方式为寄存器相对寻址（变址寻址）

运算指令

(1) add/sub/and/or/xor/mul/div/sll/srl/swap rt,rs

将 rt、rs 中的内容取出进行运算，结果存入 rt，寻址方式为寄存器寻址

(2) addi/subi/slli/srli rt,imm

将 rt 中的内容取出与 imm 字段进行运算，结果存入 rt，寻址方式为立即数寻址+寄存器寻址

跳转指令

(1) jr rs

无条件跳转到 rs 中的内容所指定的地址，寻址方式为寄存器寻址

(2) beq rt,rs,imm

有条件（rs==rt）跳转到 imm 所指定的地址，寻址方式为寄存器寻址+立即数寻址

(3) bne rt,rs,imm

有条件（rs!=rt）跳转到 imm 所指定的地址，寻址方式为寄存器寻址+立即数寻址

栈操作相关指令

(1) push rs

首先修改栈顶指针，然后将 rs 寄存器中的内容存入栈顶指针所指向的内存单元，寄存器寻址+直接寻址

(2) pop rt

首先将栈顶指针所指向的内存单元中的内容存入 rt 寄存器中，然后修改栈顶指针，寄存器寻址+直接寻址

(3) mov ss,rs

初始化栈段寄存器，ss=rs，寻址方式为寄存器寻址

(4) mov sp,rs

初始化栈顶指针寄存器，sp=rs，寻址方式为寄存器寻址

函数调用相关指令

(1) call func

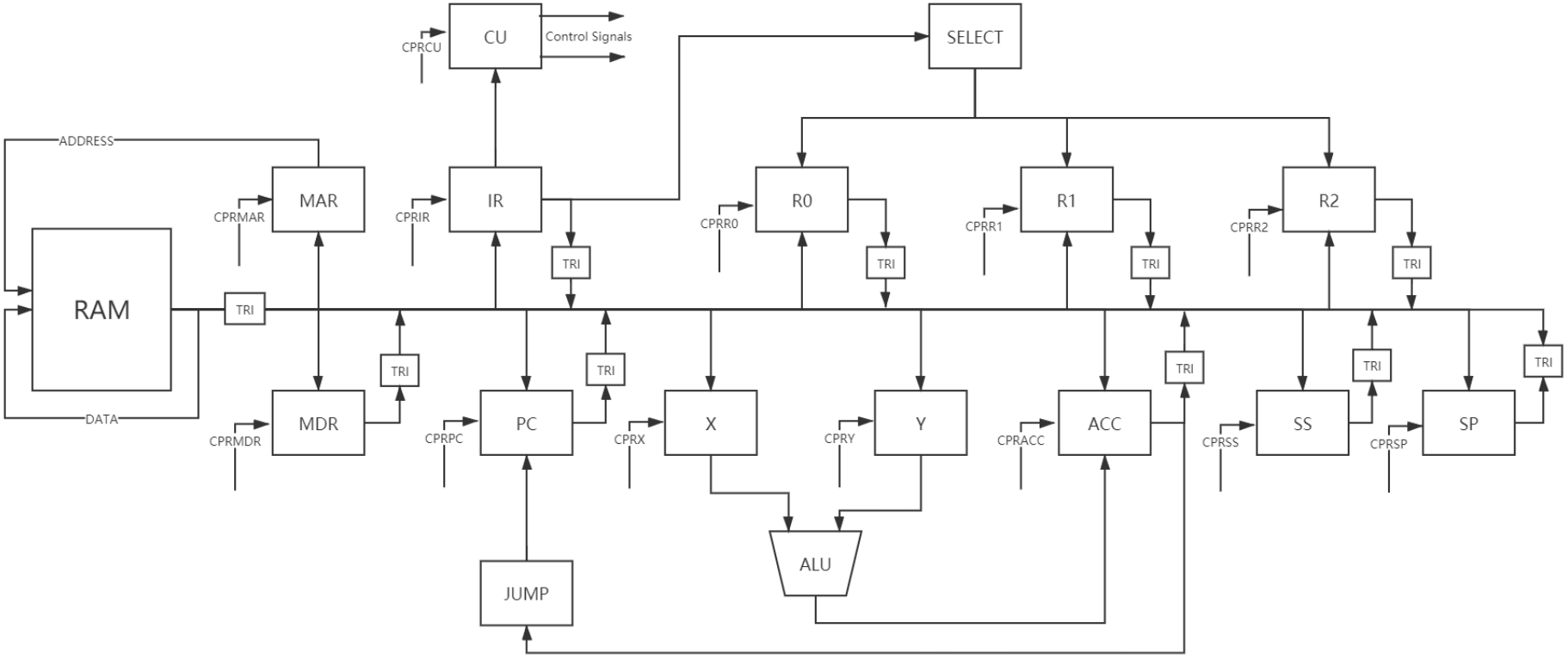
首先将当前程序计数器中的内容压栈，然后再将程序计数器置为 func，寻址方式为寄存器寻址+直接寻址

(2) ret

通过出栈操作，恢复原程序计数器中的内容，寻址方式为寄存器寻址+直接寻址

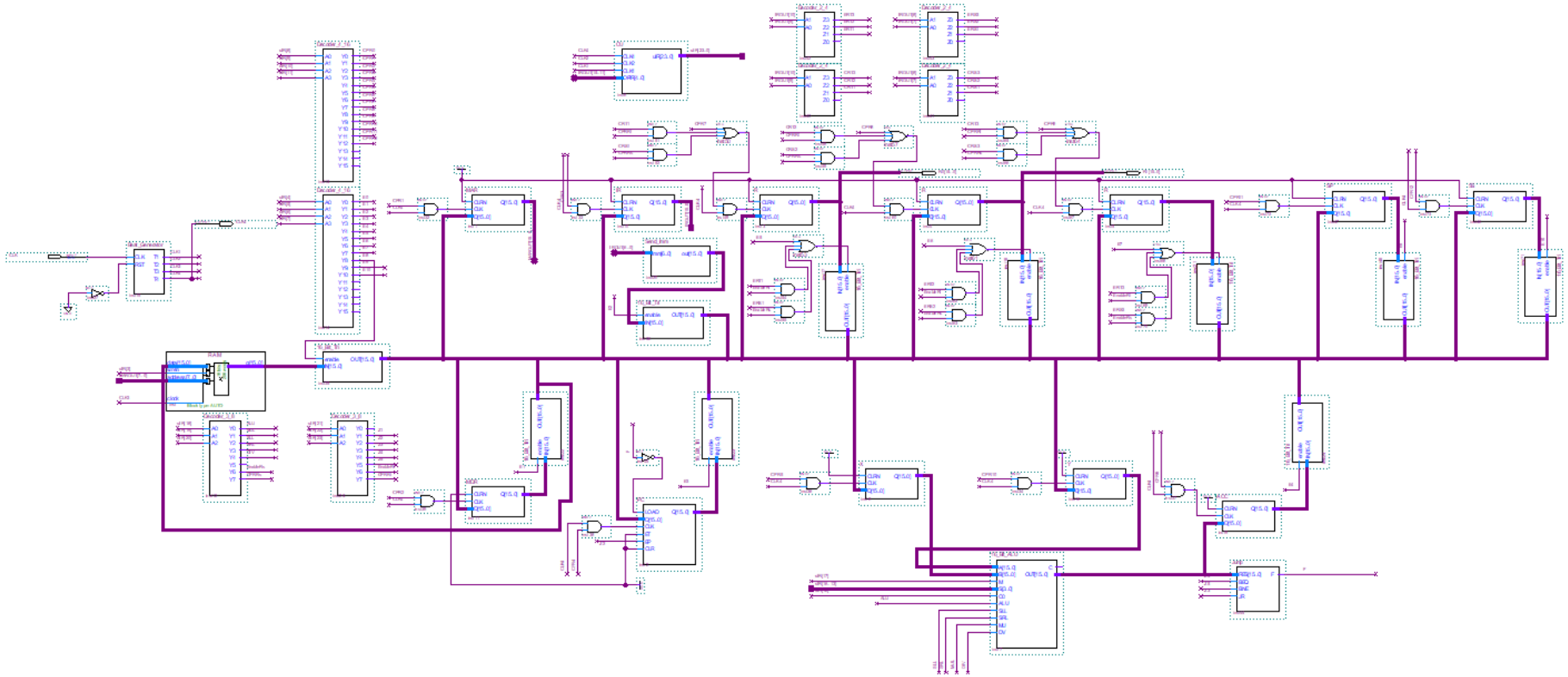
2. 总体结构与数据通路

2.1 总体结构与数据通路框图



2. 2 部件逻辑设计

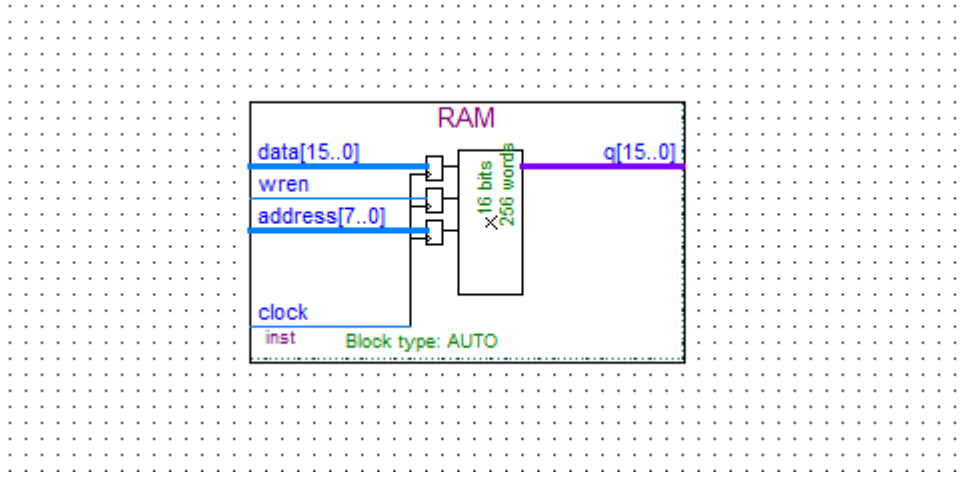
整体结构设计原理图



（1）控制器 CU 的设计流程见下一部分

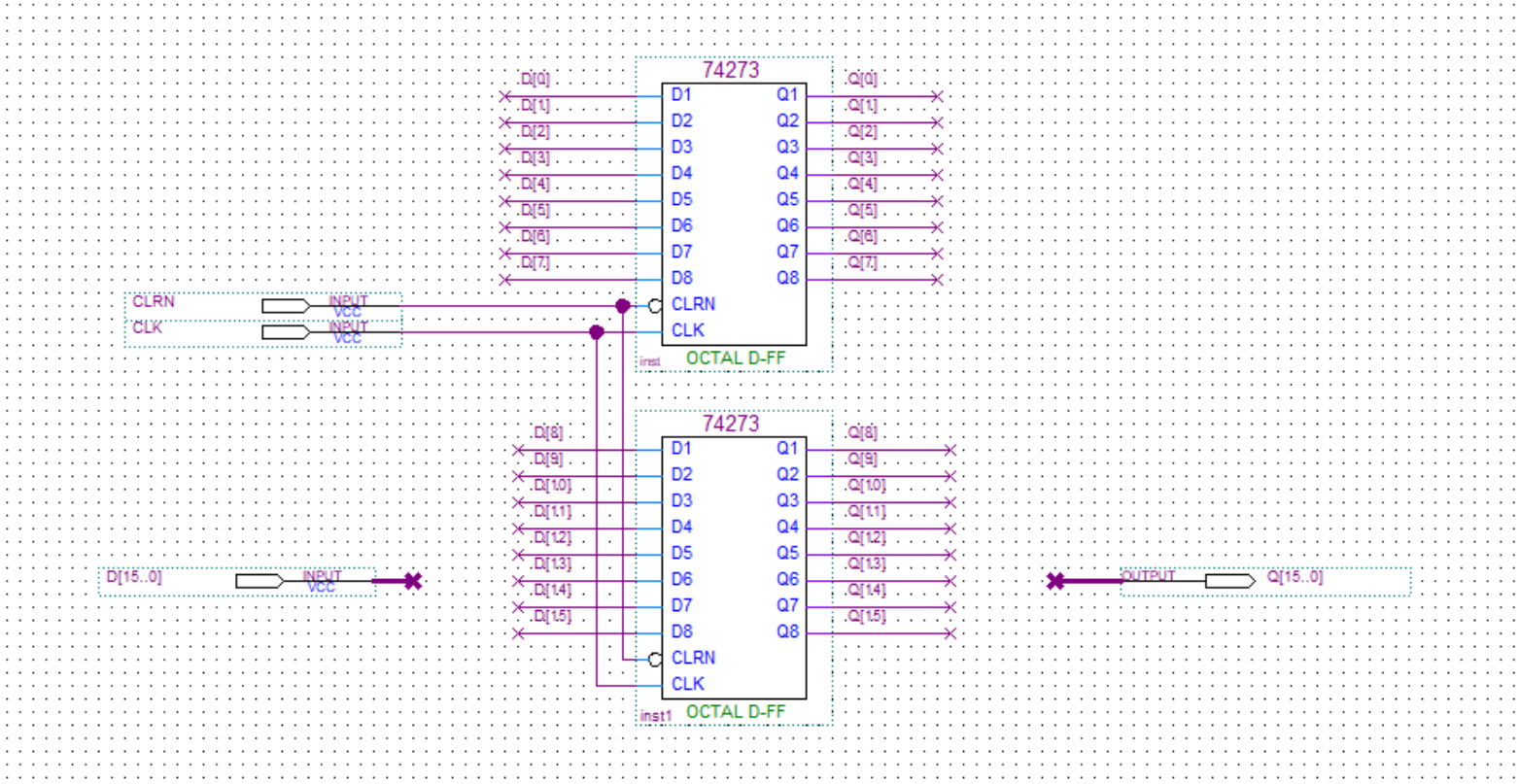
（2）存储器 RAM

容量为 256×16，基本字长为 16 位，用来存储指令和数据，调用 Quartus 器件库中的 RAM:1-PORT 实现

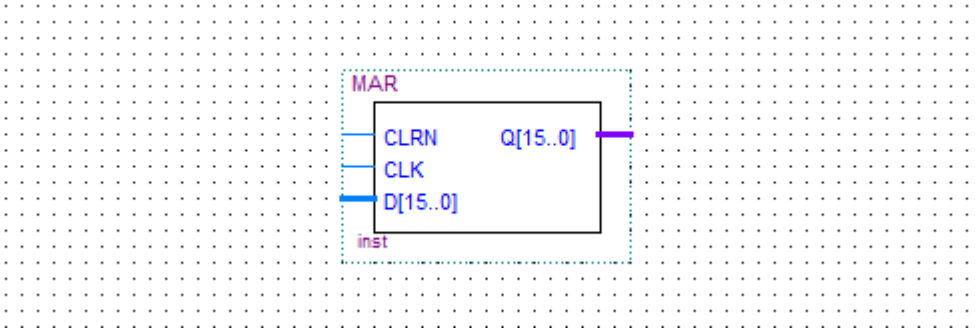


（2）寄存器组

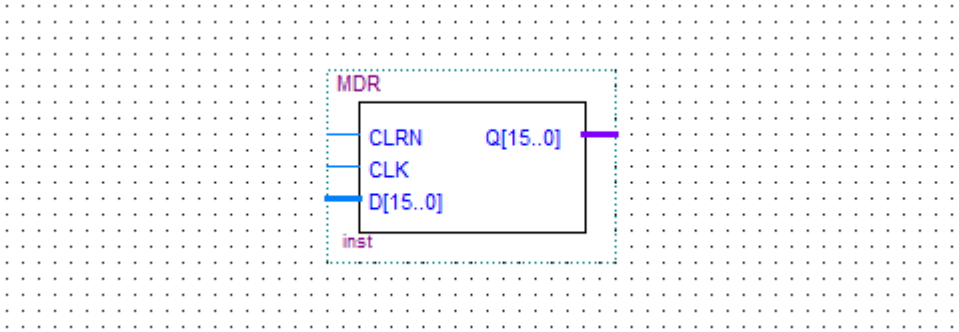
除程序计数器 PC 以外，其他寄存器均采用以下设计（2 片 74273 组合而成）：



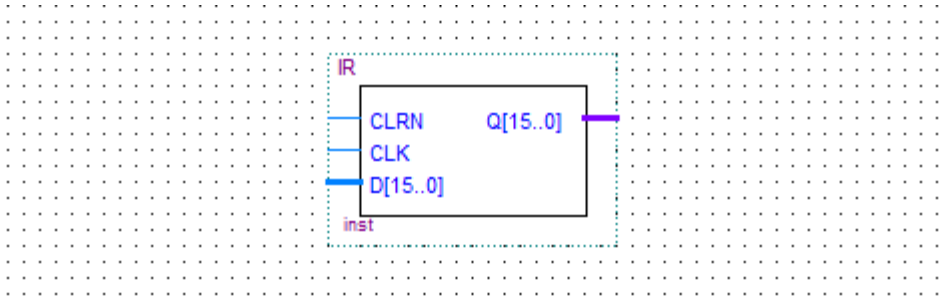
MAR 为存储器地址寄存器（16 位），用来存储指令或数据的内存地址，采用 2 片 74273 实现。



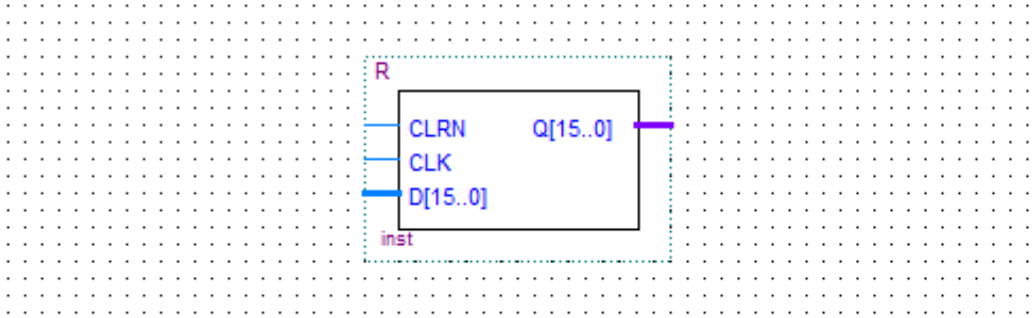
MDR 为存储器数据寄存器（16 位），用来存储要被写入地址单元或者从地址单元读出的数据，采用 2 片 74273 实现。



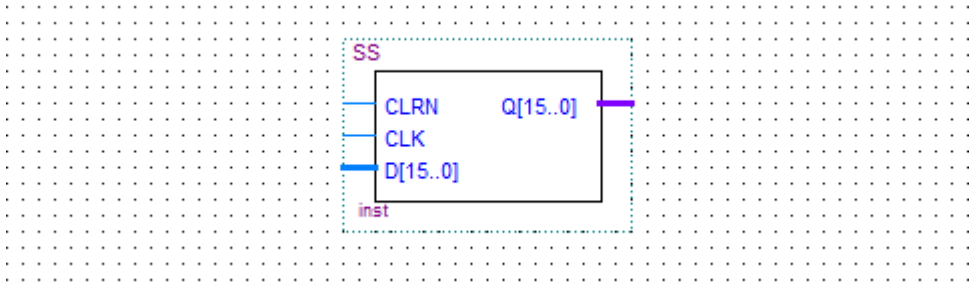
IR 为指令寄存器（16 位），用来存储从内存中取出来的指令，采用 2 片 74273 实现。



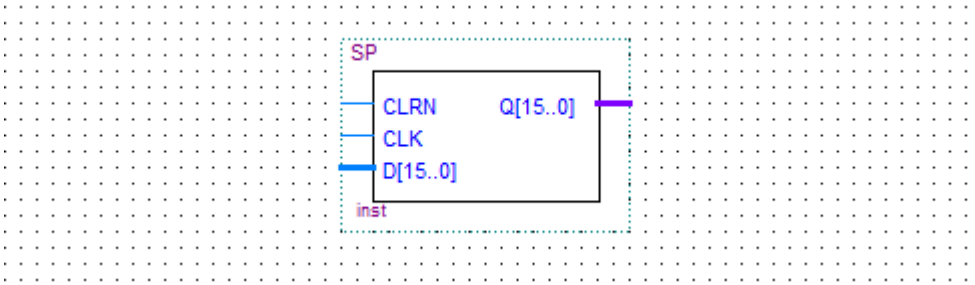
R0、R1、R2 为通用寄存器，16 位，采用 2 片 74273 实现。



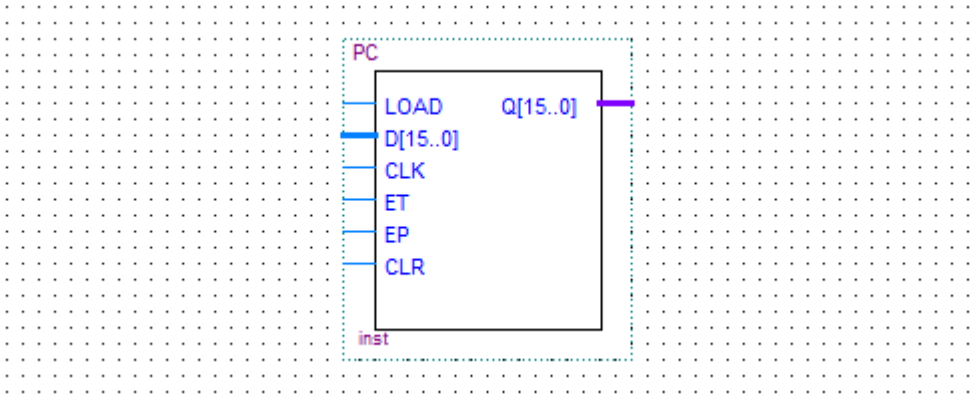
SS 为栈的段地址寄存器，存储自定义的栈段的首地址，16 位，采用 2 片 74273 实现。

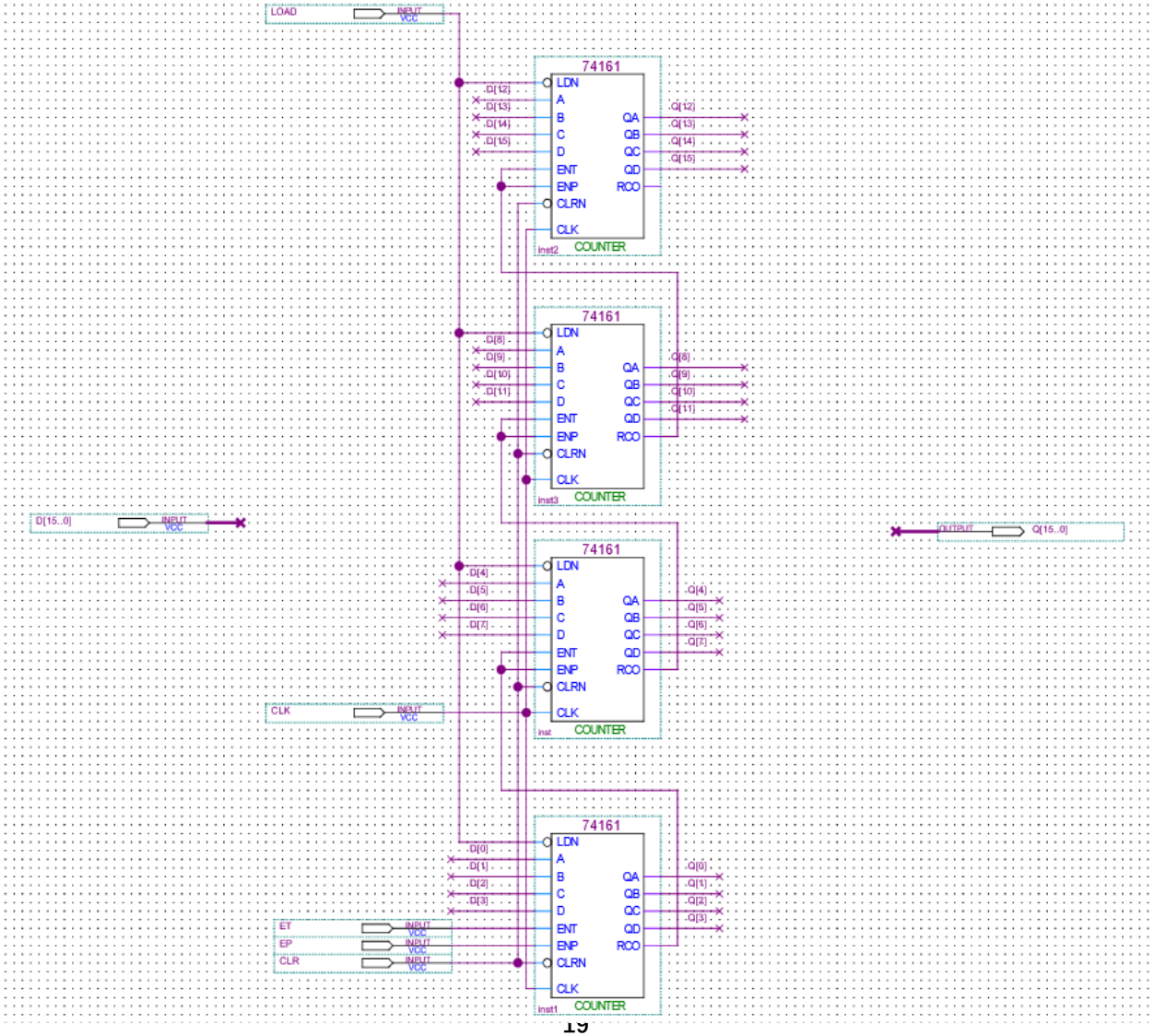


SP 为栈指针寄存器，存取栈顶指针的指向地址，16 位，采用 2 片 74273 实现。



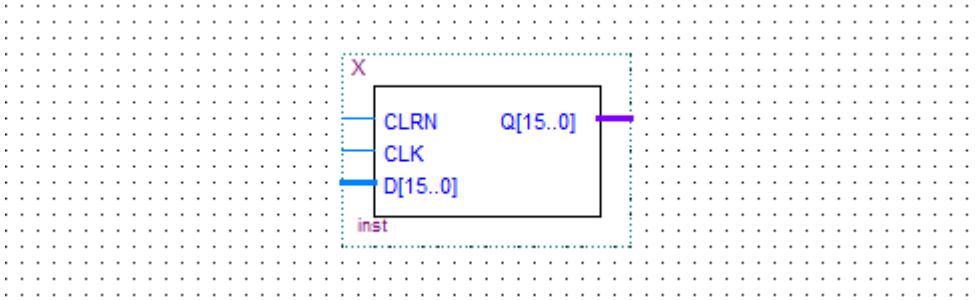
PC 为程序计数器，用来存放当前欲执行指令的地址，16 位，采用 4 片 74161 实现，有保持、+1、清零、置数四种工作状态。



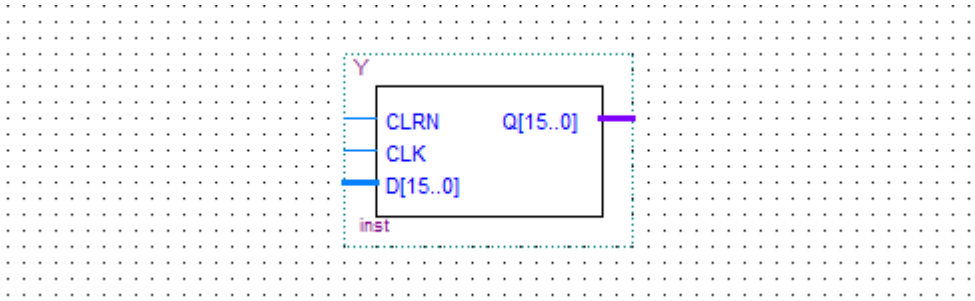

LOAD
INPUT
VCC
74161
LDN
A
B
C
D
ENT
ENP
CLRN
CLK
QA
QB
QC
QD
RCO
COUNTER
inst2
inst3
inst
inst1
D[15..0]
Q[15..0]
OUTPUT
CLK
ET
EP
CLR

（3）运算器 ALU

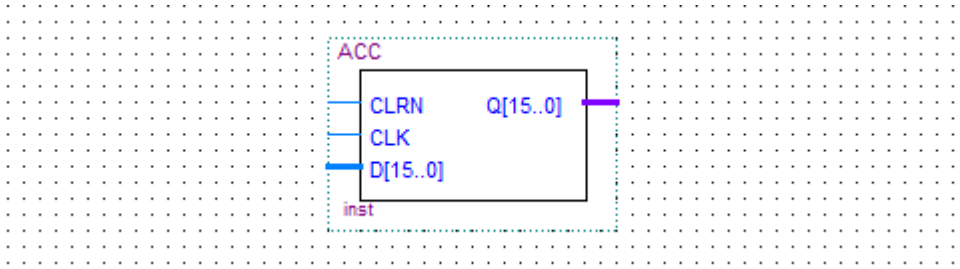
(a) 寄存器 X：存储操作数 1，16 位，采用 2 片 74273 实现



(b) 寄存器 Y：存储操作数 2，16 位，采用 2 片 74273 实现



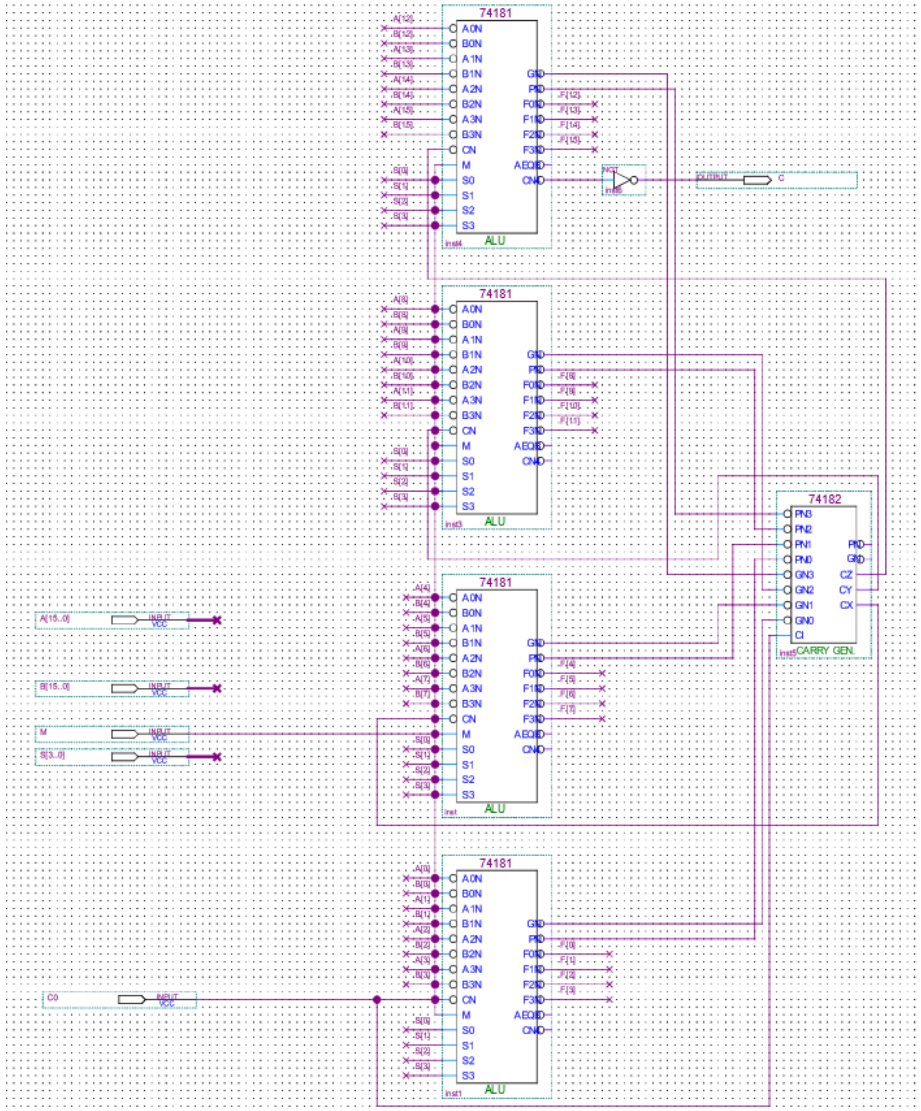
(c) 寄存器 ACC：存储运算结果，16 位。采用 2 片 74273 实现



(d) 运算器

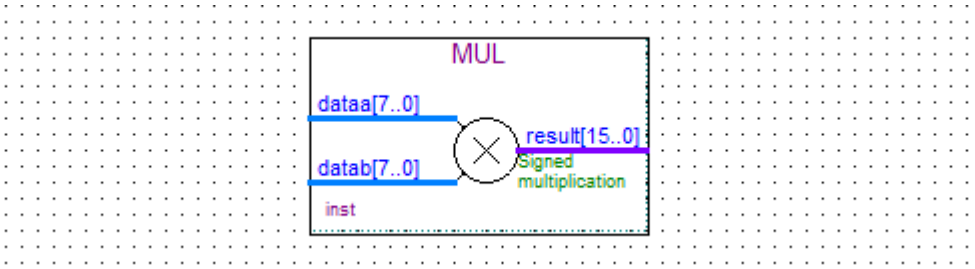
支持基础算术逻辑运算（加、减、与、或、异或）、乘法运算、除法运算、移位运算

(i) 基础算术逻辑运算器件，16 位，采用 4 片 74181 和 1 片 74182 实现。


74181
ALU
74182
CARRY GEN
A0N
B0N
A1N
B1N
A2N
B2N
A3N
B3N
CN
M
S0
S1
S2
S3
GND
PMD
FOMD
F1MD
F2MD
F3MD
AEQMD
CNMD
PN3
PN2
PN1
PN0
GN3
GN2
GN1
GN0
CI
PMD
GND
CZ
CY
CX
A[15..0]
B[15..0]
S[3..0]
CO
INPUT
VCC
C

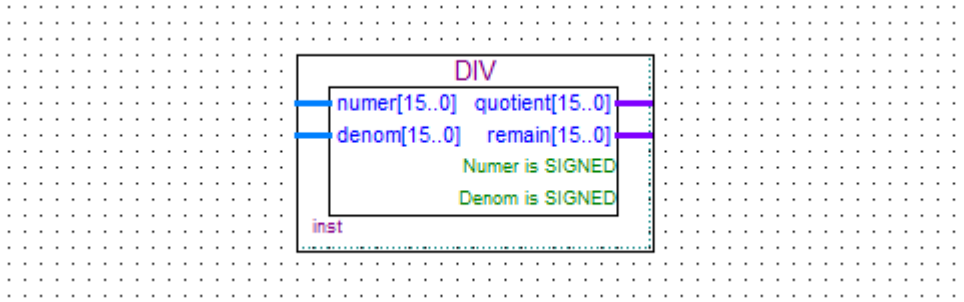
(ii) 乘法器

调用 Quartus 器件库中的 LPM_MULT 实现。



(iii) 除法器

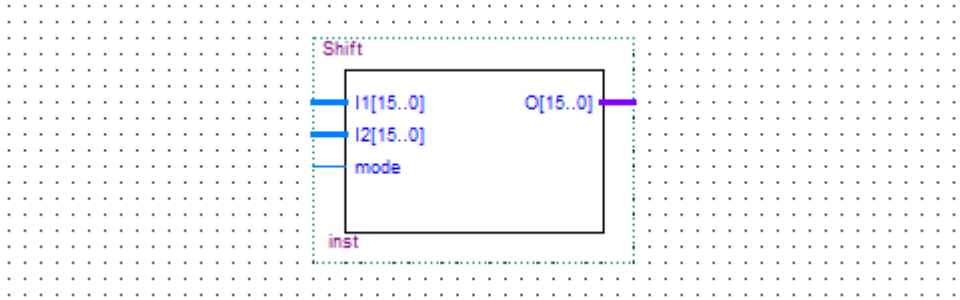
调用 Quartus 器件库中的 LPM_DIVIDE 实现。



(iv) 移位器

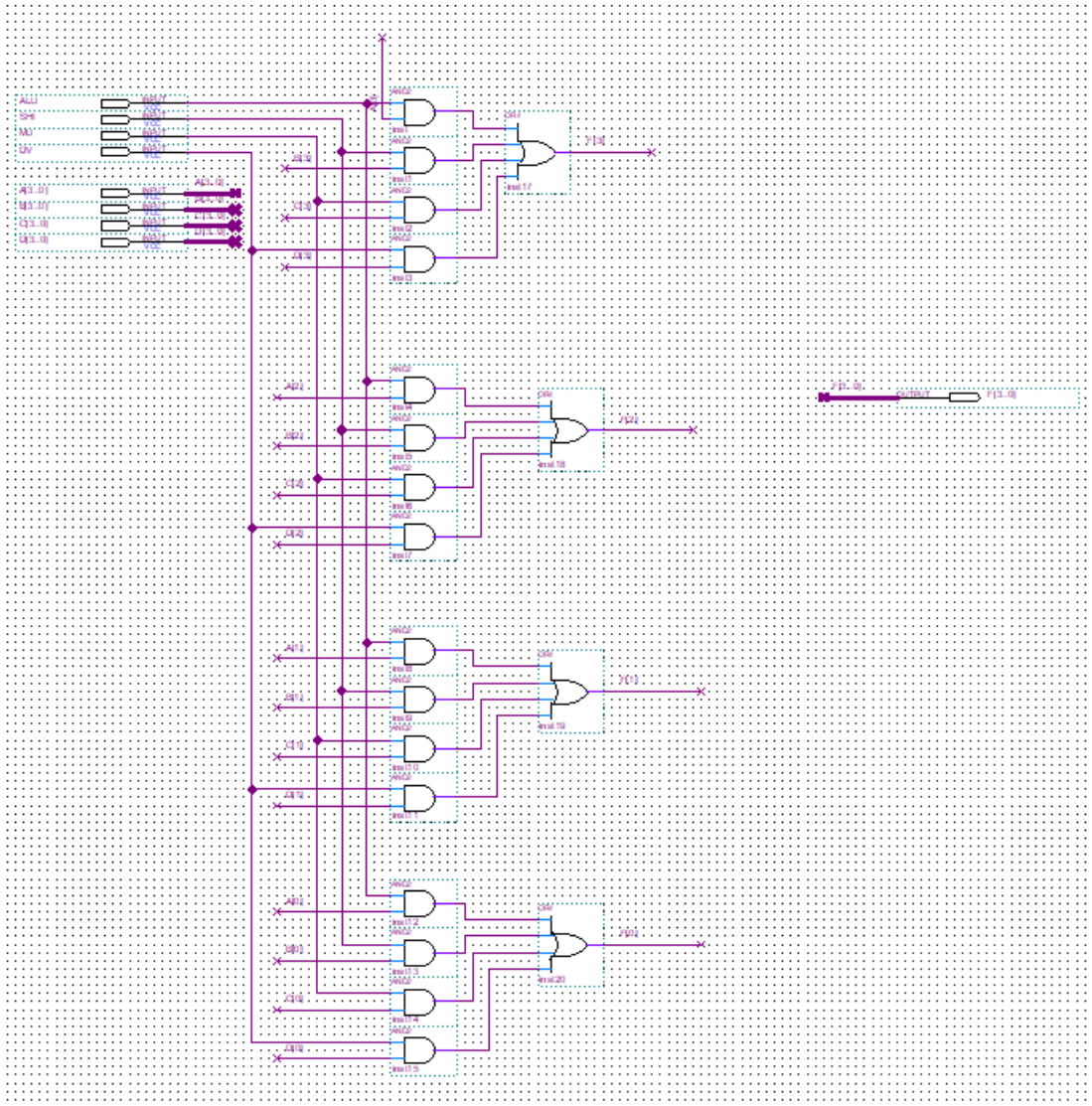
Verilog 编程实现，代码如下：

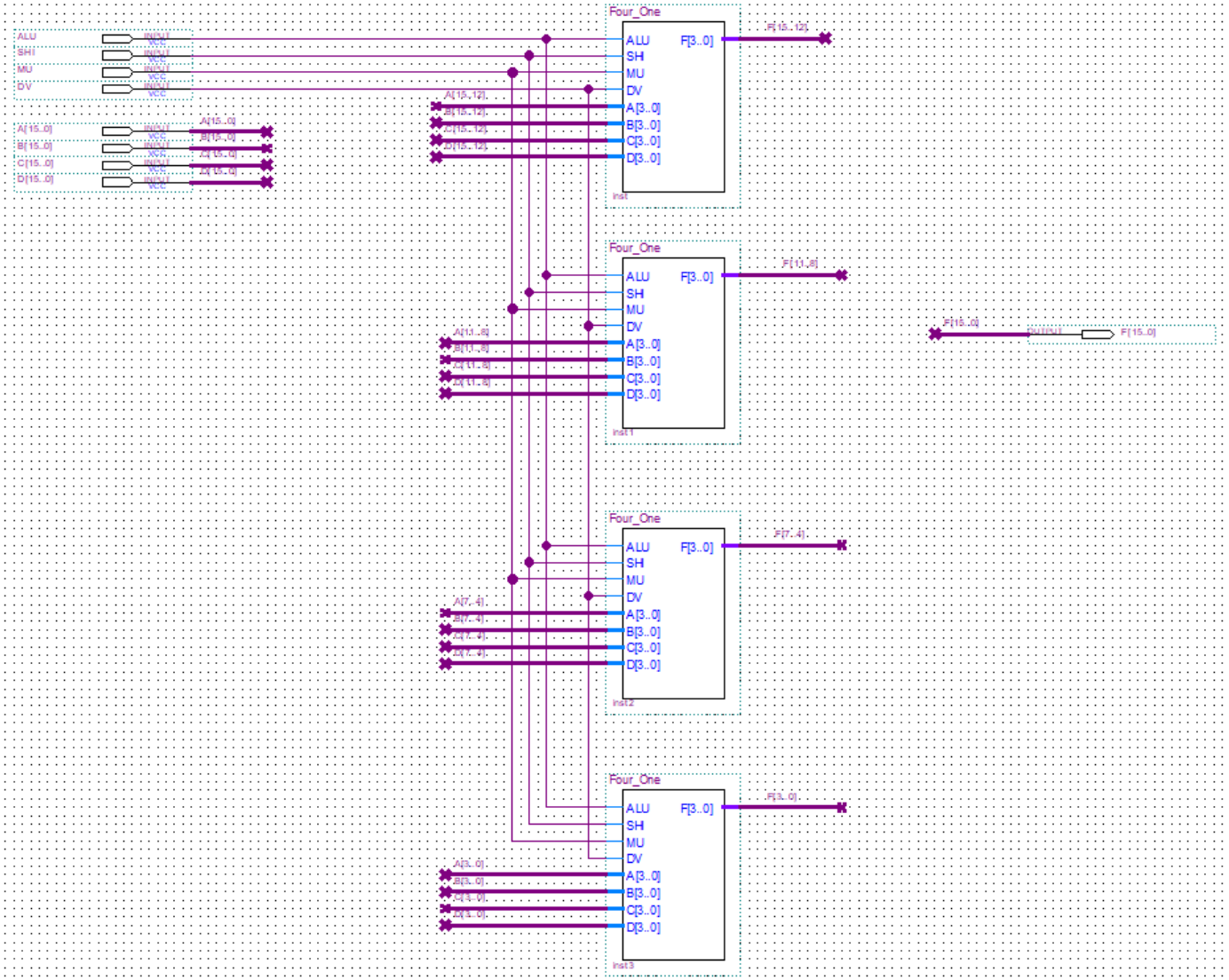
```
module Shift(
input [15:0]I1,
input [15:0]I2,
input mode,
output reg [15:0]O
);
always@(I1,I2,mode)
begin
case(mode)
    0:O=I1<<I2;
    1:O=I1>>I2;
endcase
end
endmodule
```



(v) 运算片选逻辑

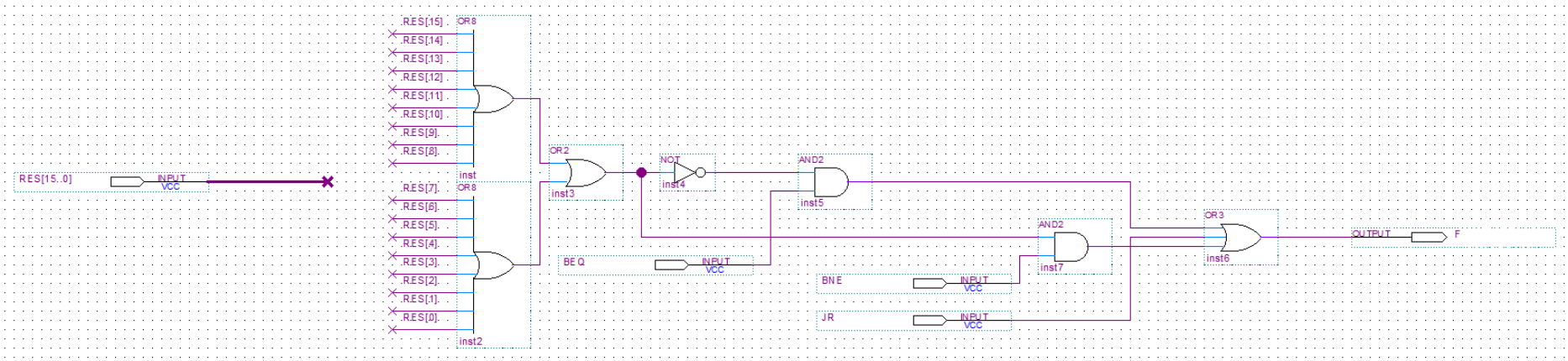
运算片选逻辑为 1 个 16 位的四选一数据选择器，选择运算器的输出数据，首先用基础与门和或门封装 1 个 4 位的四选一数据选择器，然后用 4 片封装好的 4 位四选一数据选择器组装成 16 位四选一数据选择器。




Four_One
ALU
SH
MU
DV
A[3..0]
B[3..0]
C[3..0]
D[3..0]
F[3..0]
inst
inst1
inst2
inst3
F[15..12]
F[11..8]
F[7..4]
F[3..0]
F[15..0]
A[15..0]
B[15..0]
C[15..0]
D[15..0]

（4）跳转逻辑设计 JUMP

本实验实现了 3 种跳转指令，BNE（相当于运算结果不为 0 则跳转），BEQ（相当于运算结果为 0 则跳转），JR（无条件跳转），采用以下逻辑设计实现，首先将运算结果各位相或来判断运算结果是否为 0，然后再与相应的跳转指令信号组合形成跳转逻辑。



（5）通用寄存器片选逻辑设计 SELECT

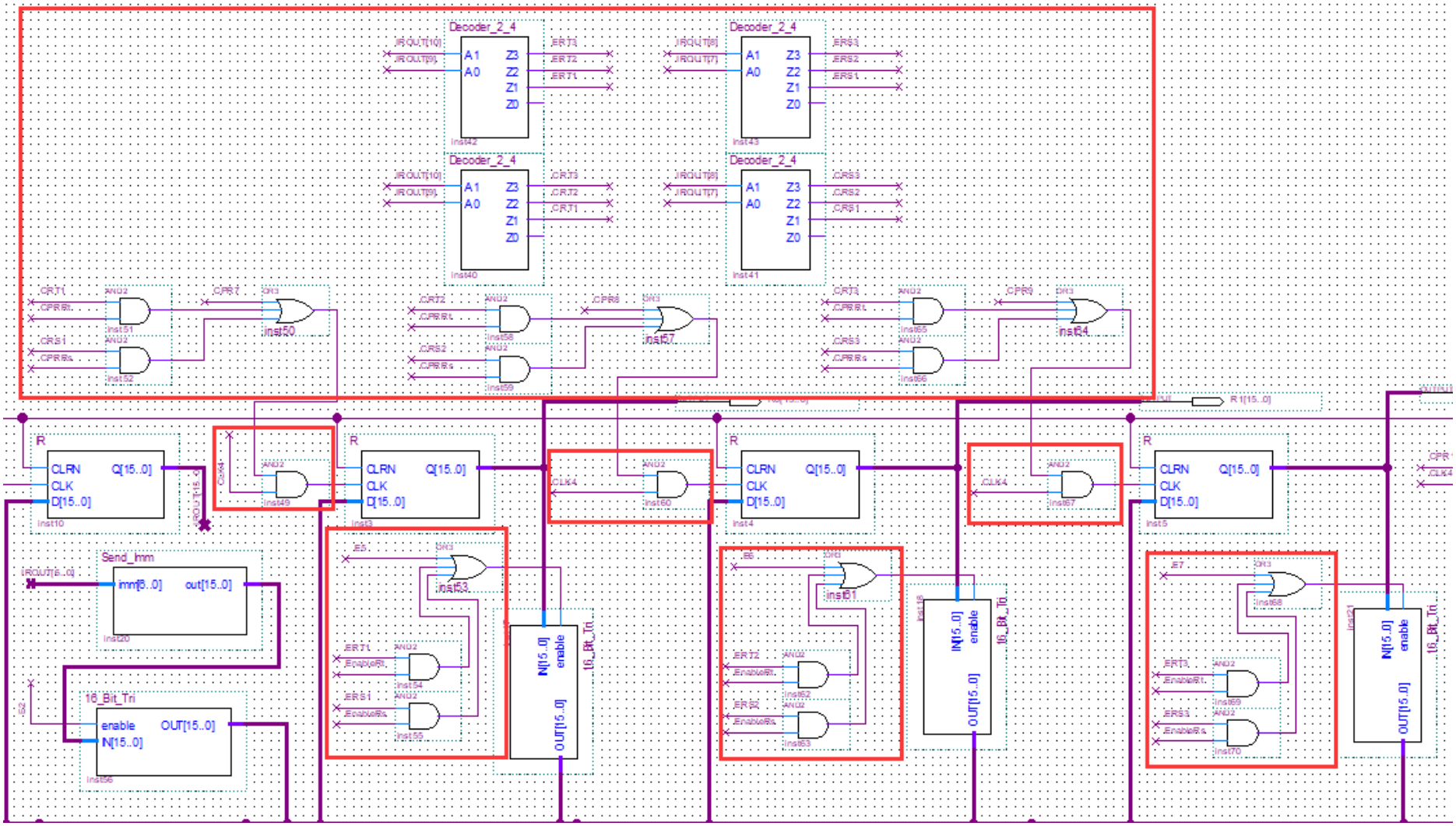
（a）通用寄存器的选择由三部分共同决定：指令、微指令、时钟

（b）首先根据指令中的寄存器寻址字段确定当前指令所要使用的通用寄存器，然后再根据微指令中的寄存器控制信号确定每一个微操作所要使用的通用寄存器，最后再与时钟 / 三态门控制信号相结合，得以下逻辑表达式：

$$CLK - Register = （CPRRi | （CRTi \ \& \ CPRRt）| (CRSi \ \& \ CPRRs)） \ \& \ CLK4$$

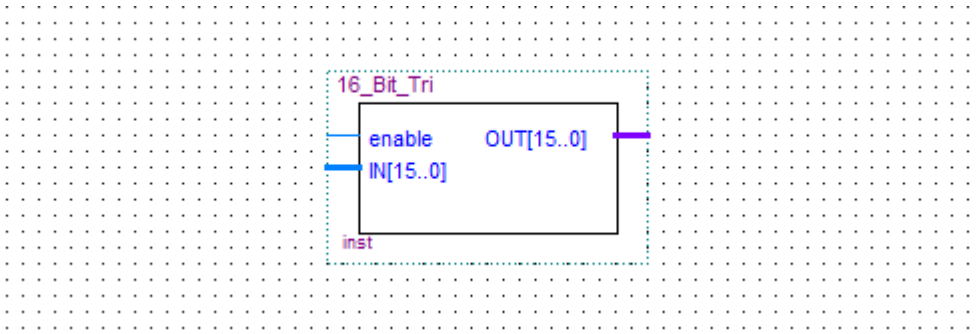
$$Enable - Register = (Ei \mid (ERTi \ \& \ EnableRt) \mid (ERSi \ \& \ EnableRs))$$

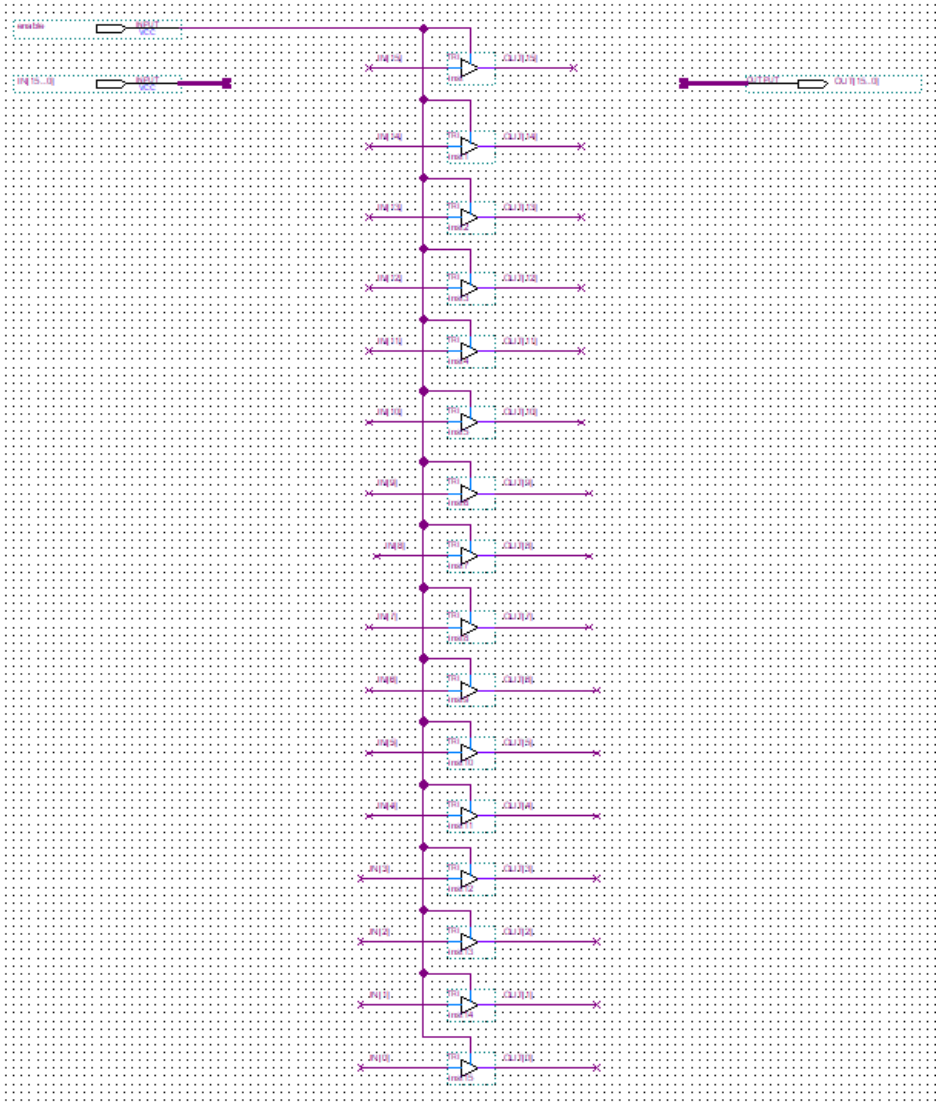
(c) 由上述两个逻辑表达式设计 SELECT 的电路原理图


Decoder_2_4
A1
A0
Z3
Z2
Z1
Z0
inst42
inst43
inst40
inst41
R
CLRN
CLK
D[15..0]
Q[15..0]
inst10
inst4
inst5
Send_Imm
imm[8..0]
out[15..0]
inst20
16_Bit_Tri
enable
IN[15..0]
OUT[15..0]
inst56
inst3
inst16
inst21

（6）三态门设计

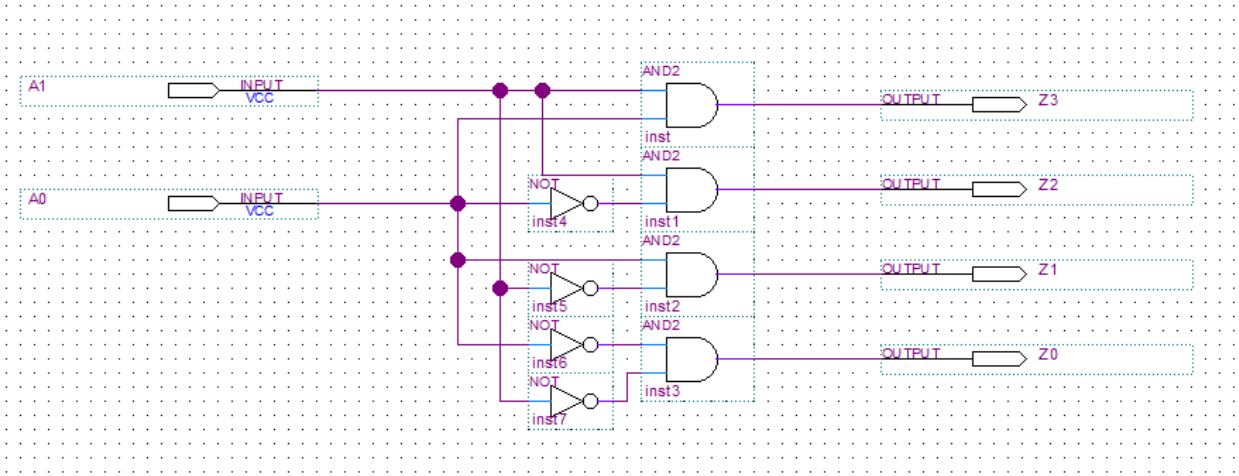
16 位，用来控制总线上的输出，采用 16 个三态门实现



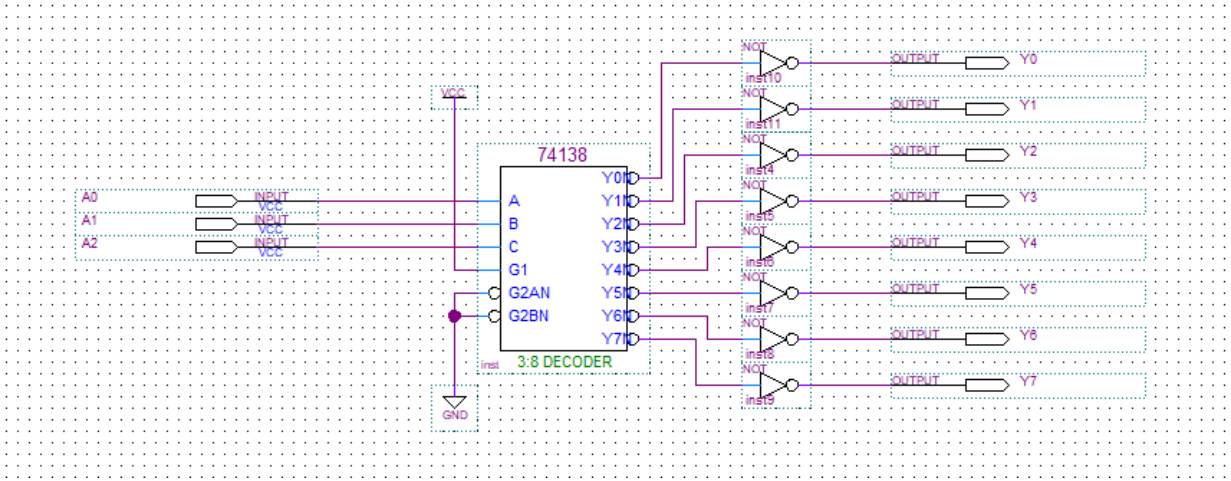


（7）译码器设计

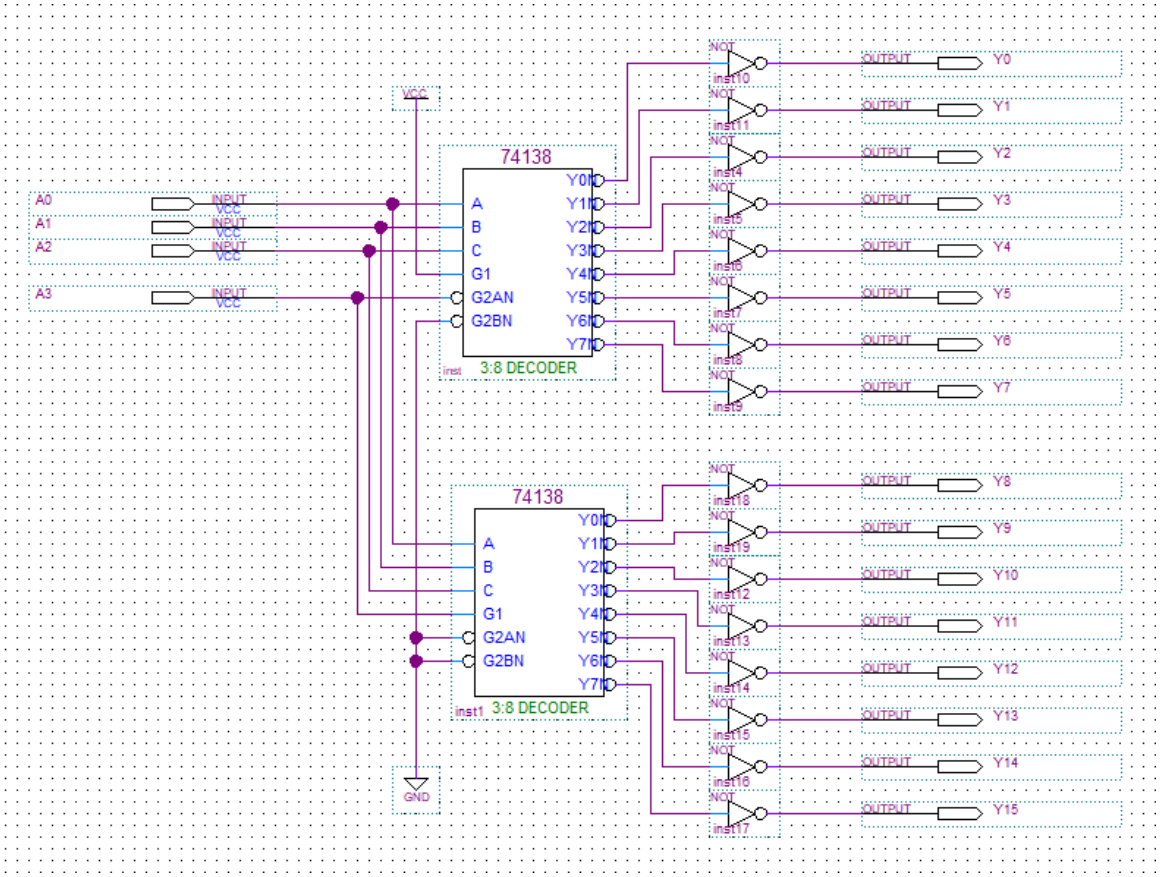
2-4 译码器



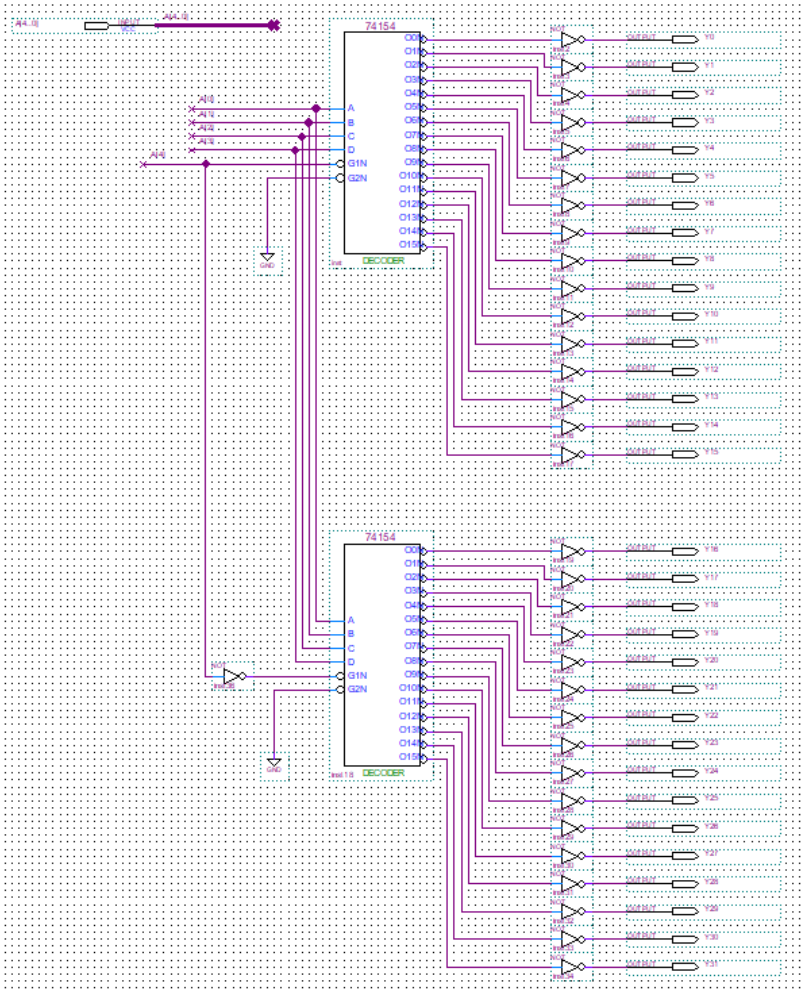
3-8 译码器


74138
A
B
C
G1
G2AN
G2BN
Y0N
Y1N
Y2N
Y3N
Y4N
Y5N
Y6N
Y7N
3:8 DECODER
inst
A0
A1
A2
INPUT
VCC
GND
NOT
inst10
inst11
inst14
inst15
inst16
inst17
inst18
inst19
OUTPUT
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7

4-16 译码器（2 片 74138 封装而成）

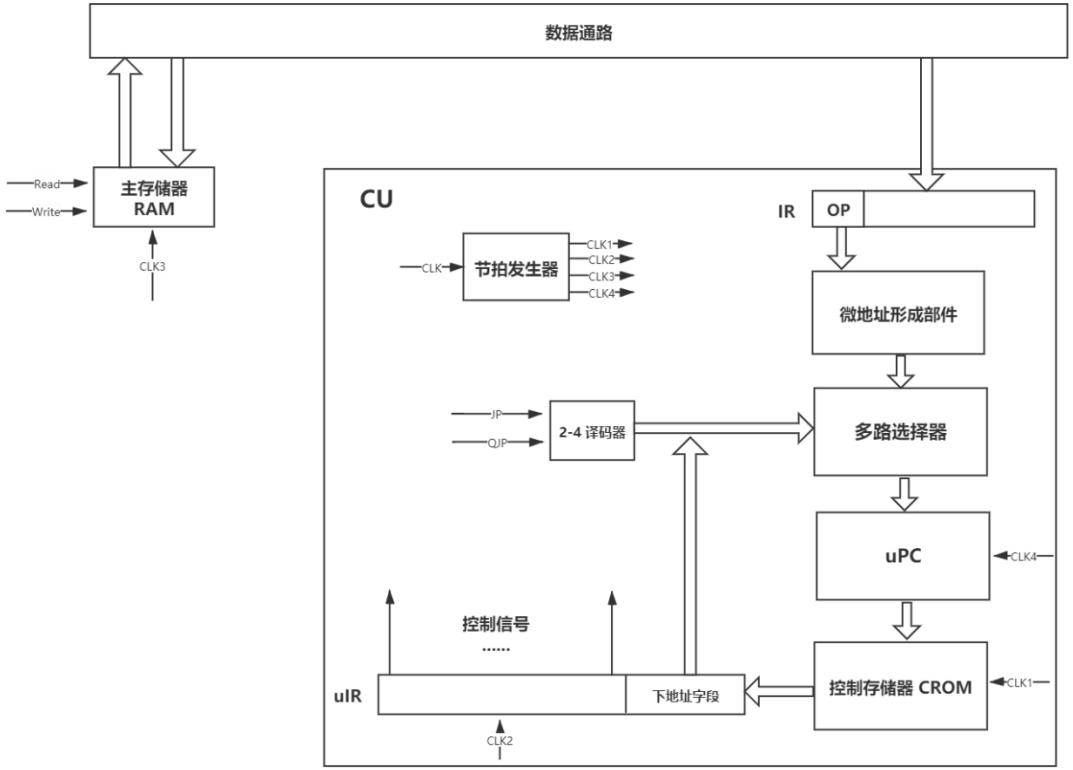


5-32 译码器（2 片 74154 封装而成）



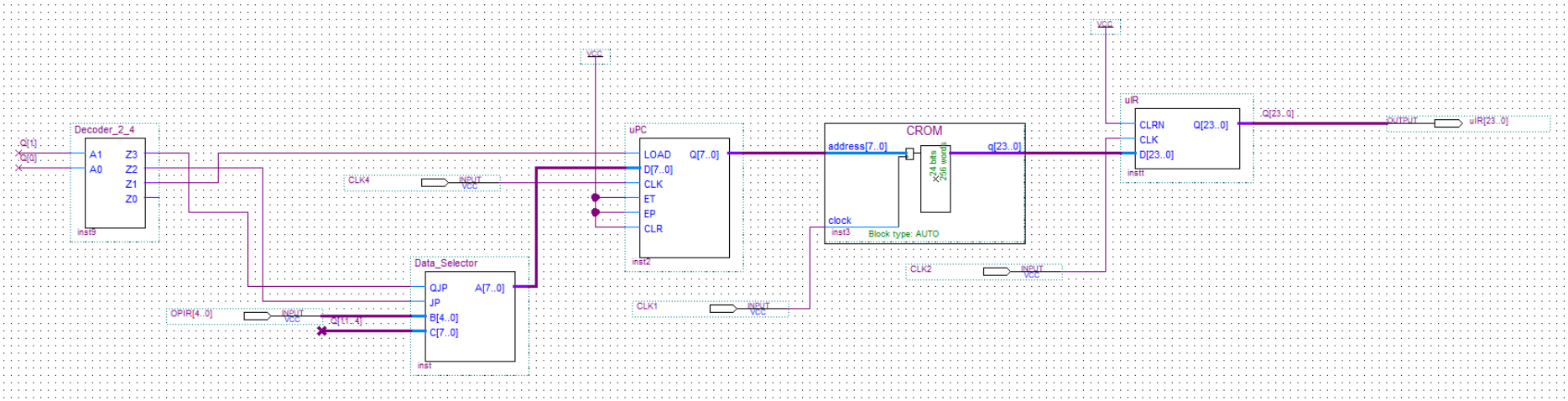
3. 微程序实现的控制部件 CU

3.1 CU 框图



3.2 设计原理图

1. CU 整体原理图

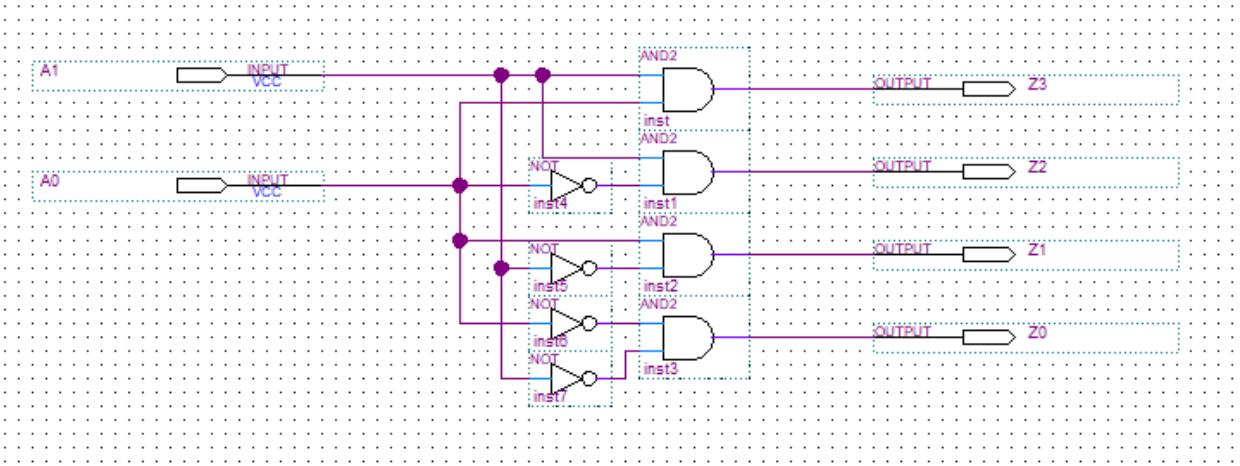


2. 2-4 译码器

01 -> uPC + 1 （微程序计数器加 1）

10 -> JP（无条件跳转，由微指令的 11~4 位决定地址）

11 -> QJP（按操作码转移）



3. 微地址形成电路及多路选择器

（1）微地址形成电路

输入指令的操作码字段（指令的高 5 位）输出微程序的入口地址（8 位）。

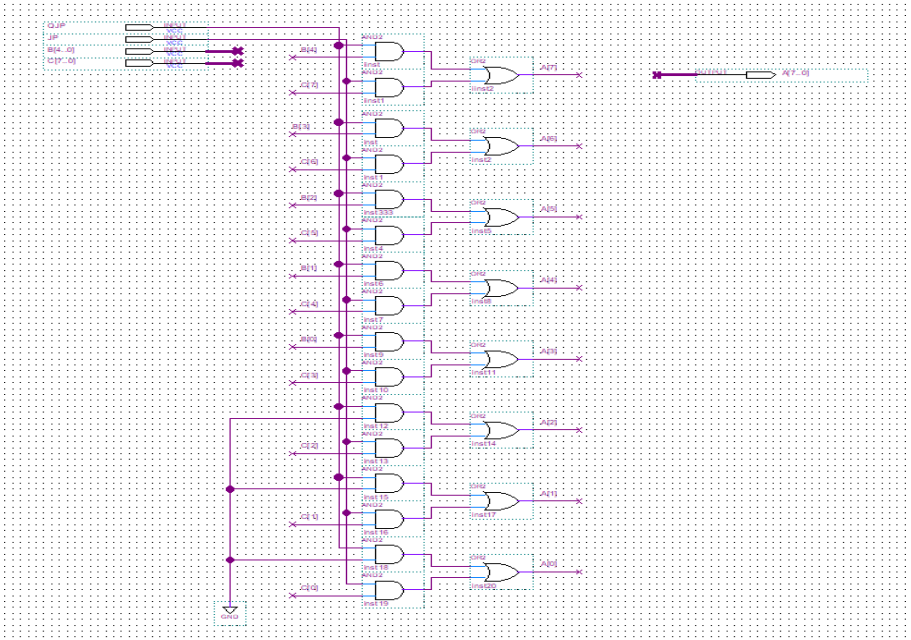
OP（5）	000

（2）多路选择器

根据微指令的后继微地址控制字段选择下一条微指令的地址

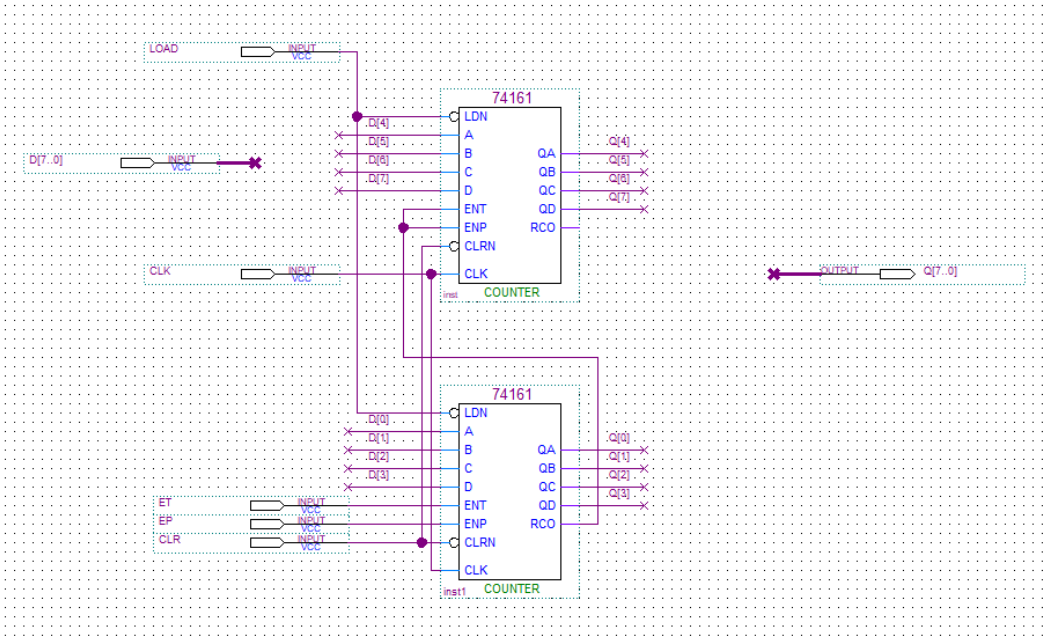
JP = 1, QJP = 0 时地址为 μIR11 μIR10 μIR9 μIR8 μIR7 μIR6 μIR5 μIR4

JP = 0, QJP = 1 时地址为 IR15 IR14 IR13 IR12 IR11 000



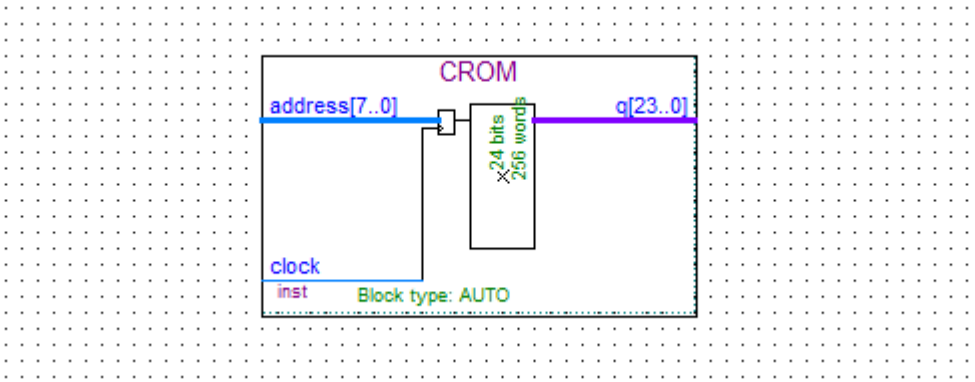
4. 微程序计数器 uPC

uPC 为微程序计数器，用来存放当前欲执行微指令的地址，8 位，采用 2 片 74161 实现，有保持、+1、清零、置数四种工作状态。



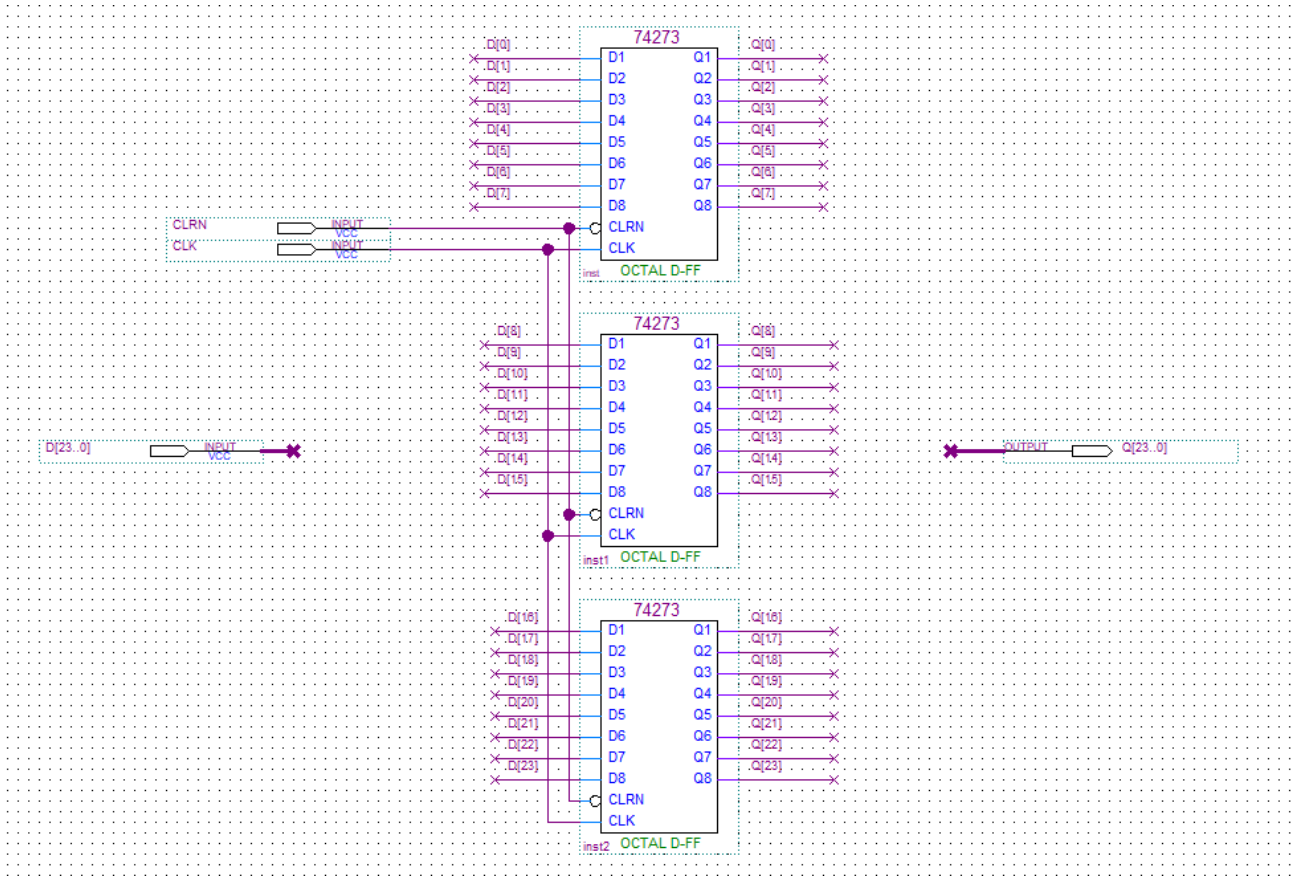
5. 控制存储器 CROM

容量为 256×24，基本字长为 24 位，用来存放微程序，调用 Quartus 器件库中的 ROM:1-PORT 实现



6. 微指令寄存器 uIR

24 位寄存器，采用 3 片 74273 封装而成，用来存放从控制存储器中读取出来的微指令



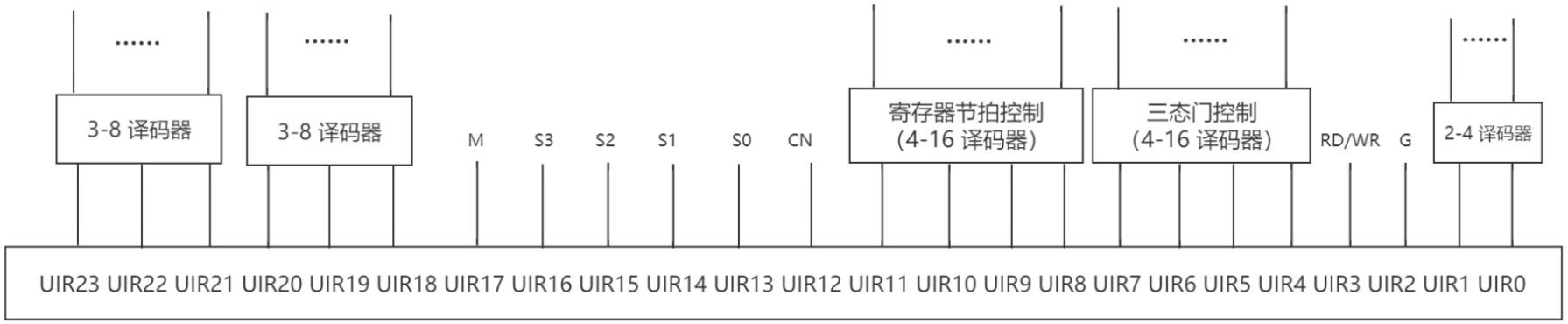
3.3 节拍安排

CLK1 ↑	CLK2 ↑	CLK3 ↑	CLK4 ↑
读取 CROM	输出 uIR	读/写 RAM	寄存器、后继地址送 uPC

3.4 指令执行流程设计

见附录 2

3.5 微指令格式



（1）程序计数器及通用寄存器控制信号

UIR23	UIR22	UIR21	SIGNAL
0	0	0	-
0	0	1	保持
0	1	0	PC=PC+1
0	1	1	JR
1	0	0	BEQ
1	0	1	BNE
1	1	0	EnableRt
1	1	1	CPRRt

（2）运算器及通用寄存器控制信号

UIR20	UIR19	UIR18	SIGNAL
0	0	0	ALU
0	0	1	MUL
0	1	0	SLL
0	1	1	SRL
1	0	0	DIV
1	0	1	-
1	1	0	EnableRs
1	1	1	CPRRs

（3）寄存器控制信号

UIR11	UIR10	UIR9	UIR8	SIGNAL
0	0	0	0	MAR
0	0	0	1	MDR
0	0	1	0	IR
0	0	1	1	PC
0	1	0	0	X
0	1	0	1	ACC
0	1	1	0	R0
0	1	1	1	R1
1	0	0	0	R2
1	0	0	1	Y
1	0	1	0	SP
1	0	1	1	SS
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-
1	1	1	1	-

（4）三态门控制信号

UIR7	UIR6	UIR5	UIR4	SIGNAL
0	0	0	0	MDR
0	0	0	1	IR
0	0	1	0	PC
0	0	1	1	ACC
0	1	0	0	R0
0	1	0	1	R1
0	1	1	0	R2
0	1	1	1	RAM
1	0	0	0	SP
1	0	0	1	SS
1	0	1	0	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-
1	1	1	1	-

（5）微程序跳转逻辑控制信号

UIR1	UIR0	SIGNAL
0	0	-
0	1	uPC+1
1	0	JP
1	1	QJP

3.6 ROM 内的微程序及地址

见附录 3

3.7 汇编程序测试

1. 测试程序一（子程序的调用与返回、所有的运算指令、存取指令）

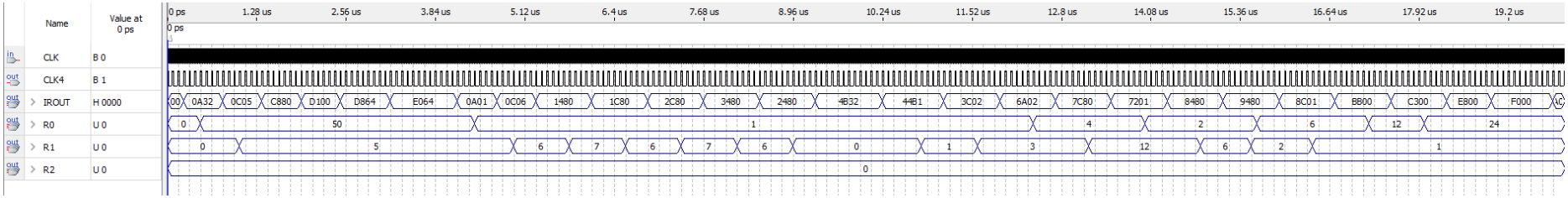
主程序

LUI R0,50	0A32H
LUI R1,5	0C05H
MOV SS,R0	C880H
MOV SP,R1	D100H
CALL1 100	D864H
CALL2 100	E064H
ADDI R0,1	3A01H

子程序（首地址为 100）

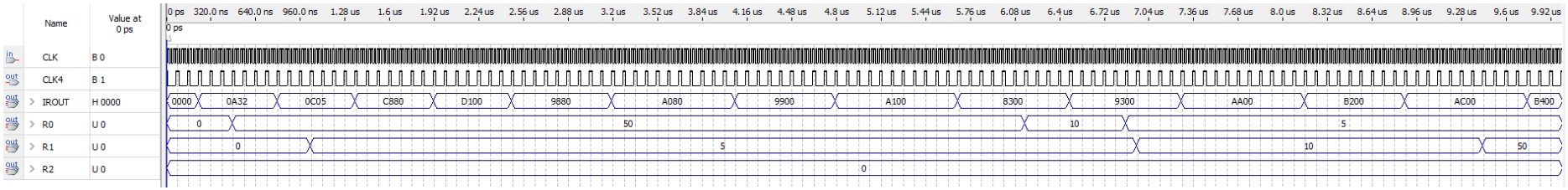
LUI R0,1	0A01H
LUI R1,6	0C06H
ADD R1,R0	1480H
SUB R1,R0	1C80H
OR R1,R0	2C80H
XOR R1,R0	3480H
AND R1,R0	2480H
SW R0,R1,50	4B32H
LW R1,R0,49	44B1H

ADDI R1,2	3C02H
SLLI R0,2	6A02H
MUL R1,R0	7C80H
SRLI R0,1	7201H
DIV R1,R0	8480H
SWAP R1,R0	9480H
SUBI R1,1	8C01H
SLL R0,R1	BB00H
SRL R0,R1	C300H
RET1	E800H
RET2	F000H



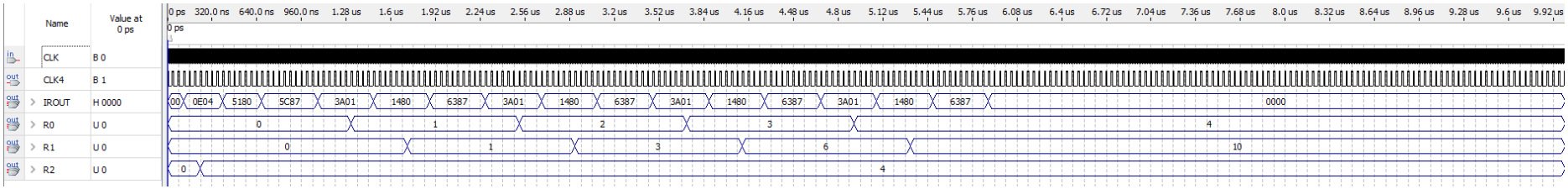
2. 测试程序二（单独测试入栈、出栈）

LUI R0,50	0A32H
LUI R1,5	0C05H
MOV SS,R0	C880H
MOV SP,R1	D100H
PUSH1 R0	9880H
PUSH2 R0	A080H
PUSH1 R1	9900H
PUSH2 R1	A100H
DIV R0,R1	8300H
SWAP R0,R1	9300H
POP1 R0	AA00H
POP2 R0	B200H
POP1 R1	AC00H
POP2 R1	B400H



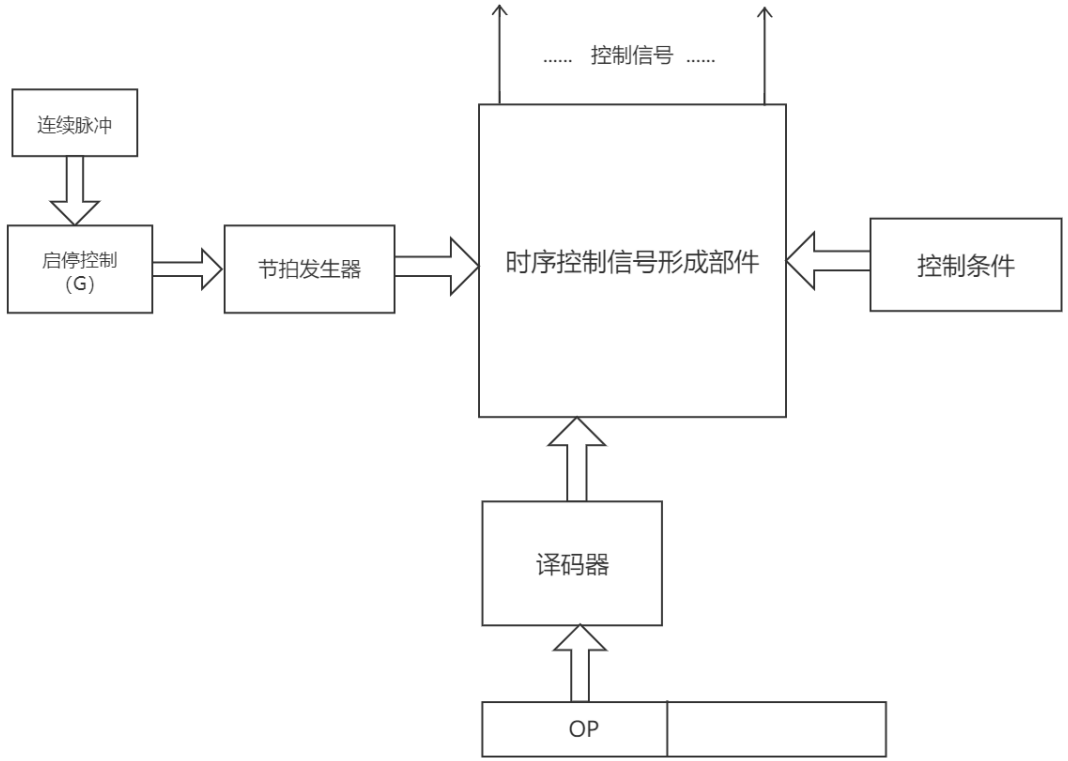
3. 测试程序三（跳转指令测试 1+2+…+n）

LUI R2,4	0E04H
JR R2	5180H
LUI R0,99	0A63H
LUI R1,99	0C63H
BEQ R1,R0,7	5C87H
LUI R0,100	0A64H
LUI R1,100	0C64H
ADDI R0,1	3A01H
ADD R1,R0	1480H
BNE R0,R2,7	6387H



4. 硬布线实现的控制部件 CU

4.1 CU 框图



4.2 指令执行流程

状态	W1（取指周期）	W2（执行周期）																	W3（存取周期）	
指令	所有指令	lui rt,imm	lw rt,rs,imm	sw rt,rs,imm	add rt,rs	sub rt,rs	and rt,rs	or rt,rs	xor rt,rs	mul rt,rs	div rt,rs	addi rt,imm	slli rt,imm	srli rt,imm	jr rs	beq rt,rs,imm	bne rt,rs,imm	lw rt,rs,imm	sw rt,rs,imm
控制信号																				
PCADDI	T3																			
M							T2	T2	T2											
S3			T2	T2	T2		T2	T2				T2								
S2						T2		T2	T2							T2	T2			
S1						T2	T2	T2	T2							T2	T2			
S0			T2	T2	T2		T2					T2								
CN			T2	T2	T2							T2								
CPRC3	T1		T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1		T1	T1	T0	T0	
CPRC2	T3		T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0	T0 T3	T0 T3			
CPRC1	T1 T2		T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2		T1	T1	T0	T0	
CPRC0	T0 T1 T2		T0 T3	T0 T3	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0		T0	T0	T0	T0	
CPRRt		T0			T3	T3	T3	T3	T3	T3	T3	T3	T3	T3				T1		
CPRRs																				
EC3	T2																	T1		
EC2			T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3						
EC1	T0	T0	T1	T1								T0	T0	T0		T3	T3			
EC0	T0																			
ERt					T1	T1	T1	T1	T1	T1	T1	T1	T1	T1		T1	T1		T0	
ERs			T0	T0	T0	T0	T0	T0	T0	T0	T0				T0	T0	T0			
RD/WR																			T0	

4.3 原理图设计流程

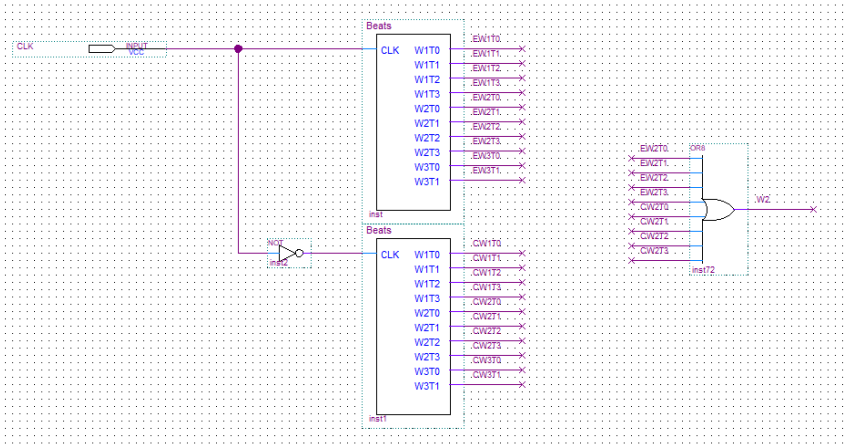
（1） 节拍发生器

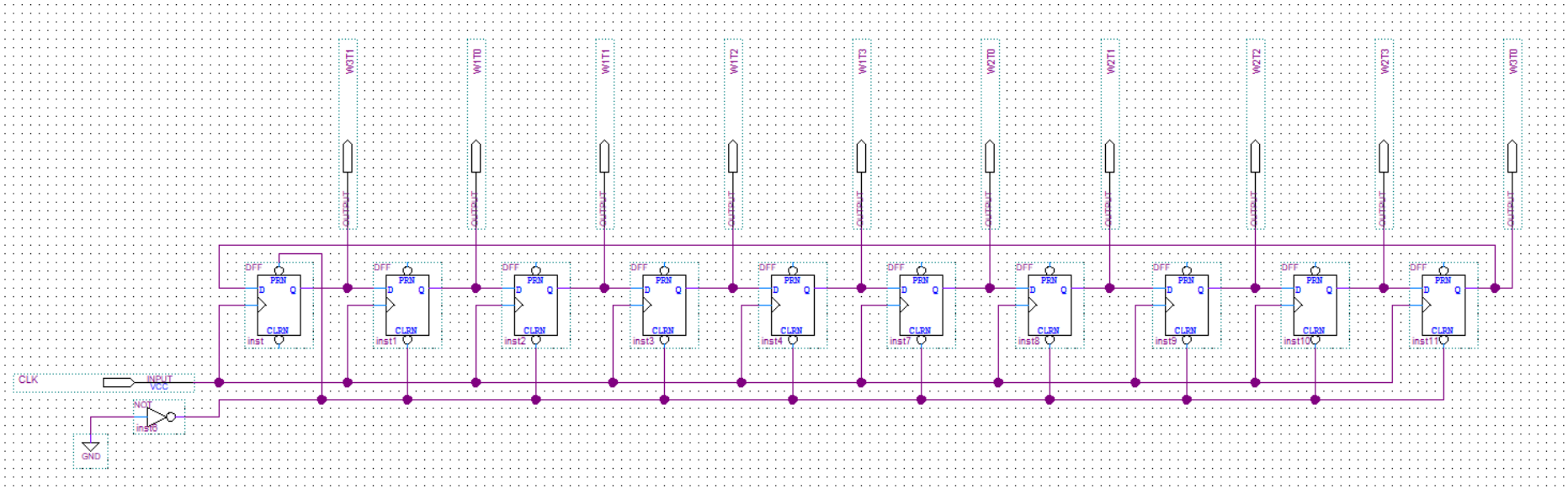
使用节拍发生器不同状态组合来区分一条指令不同的执行步骤以及完成指令执行步骤的接续，本实验的节拍安排如下:

取指周期（W1）：T0、T1、T2、T3

执行周期（W2）：T0、T1、T2、T3

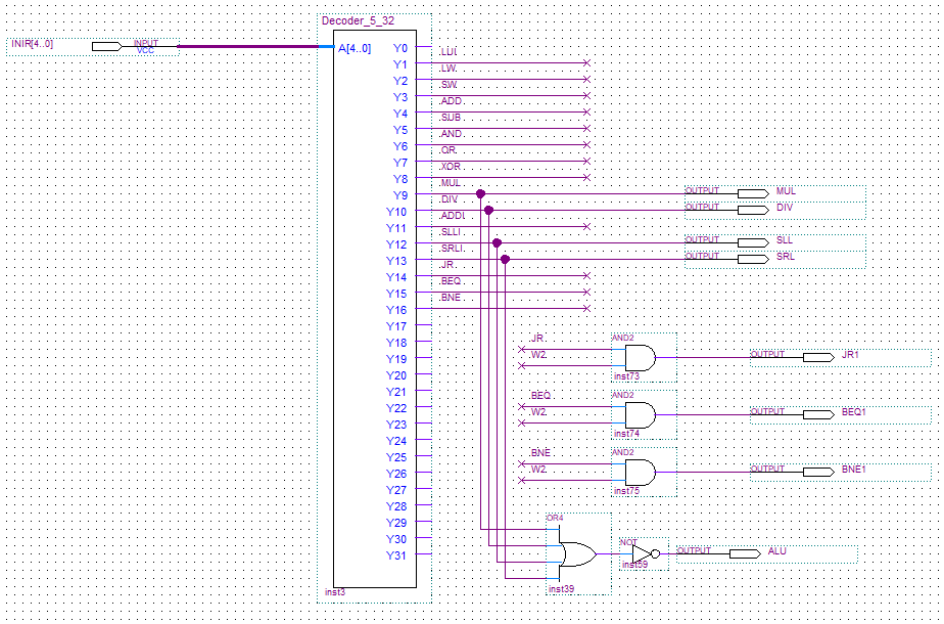
存取周期（W3）：T0、T1


CLK
INPUT
VCC
Beats
CLK
W1T0
W1T1
W1T2
W1T3
W2T0
W2T1
W2T2
W2T3
W3T0
W3T1
inst
NOT
inst2
Beats
CLK
W1T0
W1T1
W1T2
W1T3
W2T0
W2T1
W2T2
W2T3
W3T0
W3T1
inst1
OR8
W2
inst72


W3T1
W1T0
W1T1
W1T2
W1T3
W2T0
W2T1
W2T2
W2T3
W3T0
OUTPUT
DFF
PRN
D
Q
CLRN
inst
inst1
inst2
inst3
inst4
inst7
inst8
inst9
inst10
inst11
CLK
INPUT
VCC
NOT
inst6
GND

（2） 5-32 译码器

利用指令的操作码得出逻辑表达式，并以此为基础设计后续的控制信号电路



（3）时序控制信号产生部件:

(a) 根据指令执行流程和数据通路，列出各控制信号的列表

状态	W1（取指周期）	W2（执行周期）																	W3（存取周期）	
指令	所有指令	lui rt,imm	lw rt,rs,imm	sw rt,rs,imm	add rt,rs	sub rt,rs	and rt,rs	or rt,rs	xor rt,rs	mul rt,rs	div rt,rs	addi rt,imm	slli rt,imm	srli rt,imm	jr rs	beq rt,rs,imm	bne rt,rs,imm	lw rt,rs,imm	sw rt,rs,imm
控制信号																				
PCADDI	T3																			
M							T2	T2	T2											
S3			T2	T2	T2		T2	T2				T2								
S2						T2		T2	T2							T2	T2			
S1						T2	T2	T2	T2							T2	T2			
S0			T2	T2	T2		T2					T2								
CN			T2	T2	T2							T2								
CPRC3	T1		T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1		T1	T1	T0	T0	
CPRC2	T3		T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T0 T2	T3	T0 T2 T3	T0 T2 T3			
CPRC1	T1 T2		T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2	T1 T2		T1 T2	T1 T2	T0	T0	
CPRC0	T0 T1 T2		T0 T3	T0 T3	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0		T0	T0	T0	T0	
CPRRt		T0			T3	T3	T3	T3	T3	T3	T3	T3	T3	T3				T1		
CPRRs																				
EC3	T2																	T1		
EC2			T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3	T3						
EC1	T0	T0	T1	T1								T0	T0	T0		T3	T3			
EC0	T0																			
ERt					T1	T1	T1	T1	T1	T1	T1	T1	T1	T1		T1	T1		T0	
ERs			T0	T0	T0	T0	T0	T0	T0	T0	T0				T3	T0	T0			
RD/WR																			T0	

(b) 根据列表写出各控制信号的逻辑表达式并化简，结果如下:

$$VarA = LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI$$

$$VarB = LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI + BEQ + BNE$$

$$VarC = ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI$$

$$VarD = W3 * T0 * (LW + SW)$$

$$PCADDI = W1 * T3$$

$$M = W2 * T2 * (AND + OR + XOR)$$

$$S3 = W2 * T2 * (LW + SW + ADD + AND + OR + ADDI)$$

$$S2 = W2 * T2 * (SUB + OR + XOR + BEQ + BNE)$$

$$S1 = W2 * T2 * (SUB + AND + OR + XOR + BEQ + BNE)$$

$$S0 = W2 * T2 * (LW + SW + ADD + AND + ADDI)$$

$$CN = W2 * T2 * (LW + SW + ADD + ADDI)$$

$$CPRC3 = W1 * T1 + W2 * T1 * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI + BEQ + BNE) + W3 * T0 * (LW + SW)$$

$$= W1 * T1 + W2 * T1 * VarB + VarD$$

$$CPRC2 = W1 * T3 + W2 * ((T0 + T2) * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI) + T3 * JR + (T0 + T2 + T3) * (BEQ + BNE))$$

$$= W1 * T3 + W2 * ((T0 + T2) * VarB + T3 * (JR + BEQ + BNE))$$

$$CPRC1 = W1 * (T1 + T2) + W2 * ((T1 + T2) * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI + BEQ + BNE) + W3 * T0 * (LW + SW)$$

$$= W1 * (T1 + T2) + W2 * ((T1 + T2) * VarB) + VarD$$

$$CPRC0 = W1 * (T0 + T1 + T2) + W2 * (T3 * (LW + SW) + T0 * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI + BEQ + BNE)) + W3 * T0 * (LW + SW)$$

$$= W1 * (T0 + T1 + T2) + W2 * (T3 * (LW + SW) + T0 * VarB) + VarD$$

$$CPRRt = W2 * (T0 * LUI + T3 * (ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI) + W3 * T1 * LW$$

$$= W2 * (T0 * LUI + T3 * VarC) + W3 * T1 * LW$$

$$CPRRs = 0$$

$$EC3 = W1 * T2 + W3 * T1 * LW$$

$$EC2 = W2 * T3 * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI)$$

$$= W2 * T3 * VarA$$

$$EC1 = W1 * T0 + W2 * (T0 * (LUI + ADDI + SLLI + SRLI) + T1 * (LW + SW) + T3 * (BEQ + BNE))$$

$$EC0 = W1 * T0$$

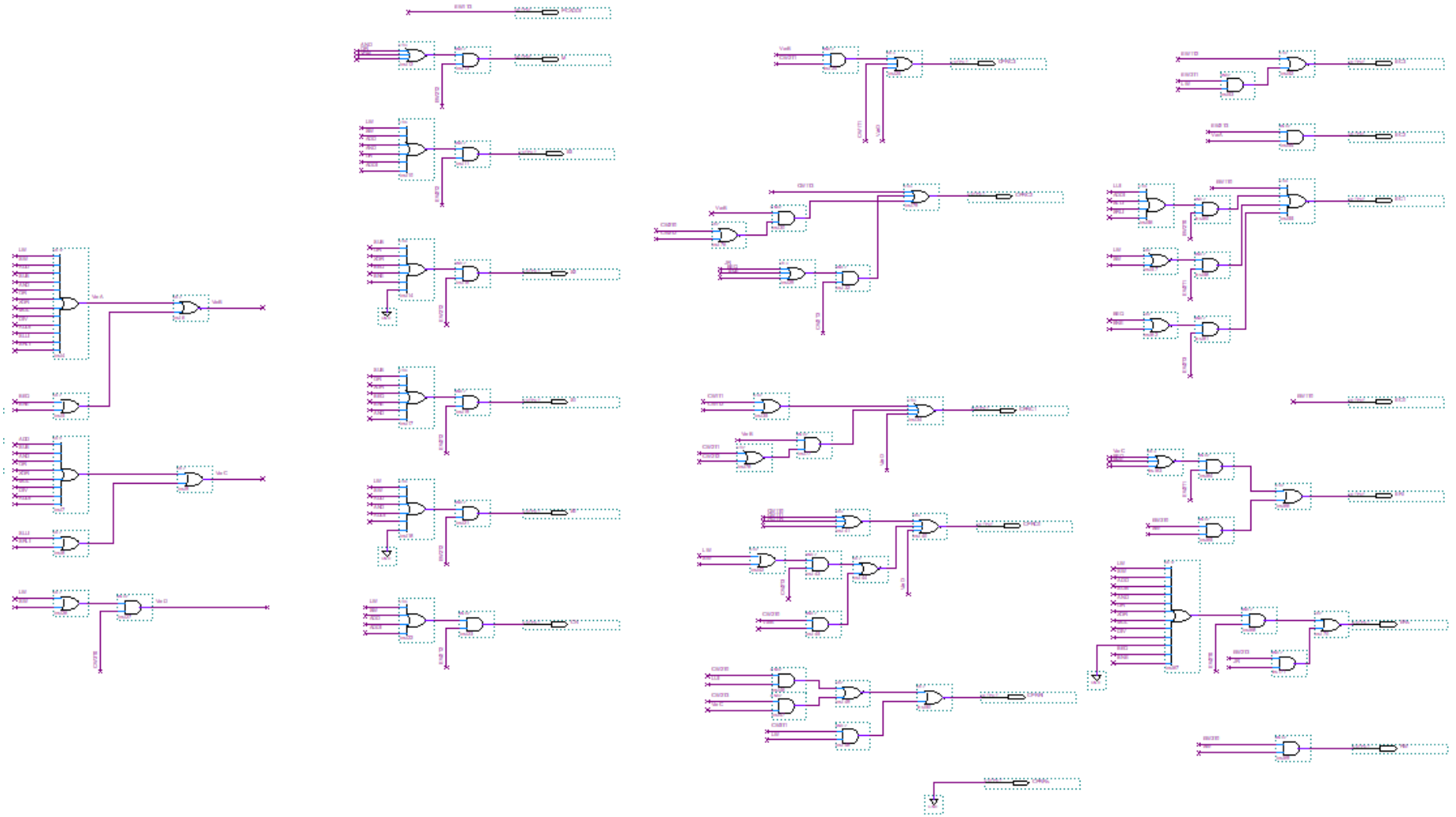
$$ERt = W2 * T1 * (ADD + SUB + AND + OR + XOR + MUL + DIV + ADDI + SLLI + SRLI + BEQ + BNE) + W3 * T0 * SW$$

$$= W2 * T1 * (VarC + BEQ + BNE) + W3 * T0 * SW$$

$$ERs = W2 * T0 * (LW + SW + ADD + SUB + AND + OR + XOR + MUL + DIV + BEQ + BNE) + W2 * T3 * JR$$

$$RD/WR = W3 * T0 * SW$$

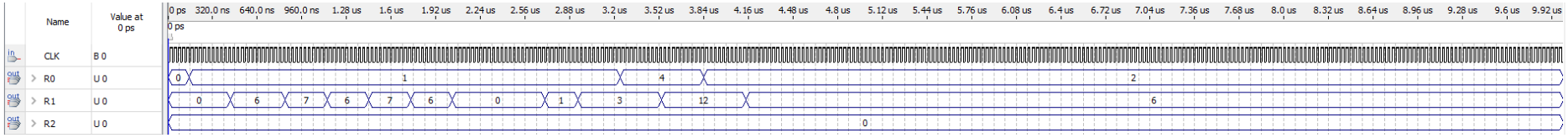
(c) 根据逻辑表达式，利用基本的与门、或门、非门实现控制信号电路



4.4 汇编程序测试

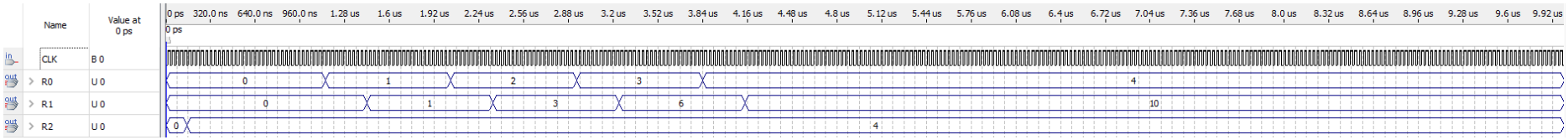
（1）测试运算指令与存取指令

LUI R0,1	R0 = 1	0A01H
LUI R1,6	R1 = 6	0C06H
ADD R1,R0	R1 = 7	2480H
SUB R1,R0	R1 = 6	2C80H
OR R1,R0	R1 = 7	3C80H
XOR R1,R0	R1 = 6	4480H
AND R1,R0	R1 = 0	3480H
SW R0,R1,50	M[50]=1	1B32H
LW R1,R0,49	R1=M[50]=1	14B1H
ADDI R1,2	R1=3	5C02H
SLLI R0,2	R0=4	6202H
MUL R1,R0	R1=12	4C80H
SRLI R0,1	R0=2	6A01H
DIV R1,R0	R1=6	5480H



（2）测试转移指令

LUI R2,4	0E04H
JR R2	7180H
LUI R0,99	0A63H
LUI R1,99	0C63H
BEQ R1,R0,7	7C87H
LUI R0,100	0A64H
LUI R1,100	0C64H
ADDI R0,1	5A01H
ADD R1,R0	2480H
BNE R0,R2,7	8387H



5. 课程设计总结

5.1 遇到的问题和解决的方法

总得来说，在实验中遇到的问题比较零碎，大多精确到每一个设计步骤中的某一细节，这里选取了几个比较重要的问题，简单总结如下：

1. 基本字长的确定

在拟定指令系统时，由于制定的设计目标为做到 32 条指令，若不采用扩展操作码技术的话，那么操作码字段至少保证 5 位，为了设计与实现方便以及保证寻址范围不至于太小，原打算确定基本字长为 32 位，但在实际使用 Quartus 仿真实现时出现了问题，器件库中的 RAM 芯片的存储容量有限，无法满足设计的需要，所以只好缩短基本字长为 16 位，5 位为操作码，剩下的 11 位根据不同的指令类型划分为不同的字段（操作数字段、寄存器寻址字段、内存寻址字段）

2. 微指令格式的设计

微指令的基本字长定为 24 位，但需要实际分配给不同器件的不同控制信号其实是远多于 24 个的，这时只好再次划分微指令中不同的位数作为不同的控制字段，根据控制信号类别分别使用译码器来扩充控制信号的个数。

3. 节拍安排

节拍的安排对整个模型机的运行过程至关重要，实验四中由 CU 中的节拍发生器来产生不同周期（取指、执行、存取）的不同脉冲，对于每一个与总线相连的寄存器，其在每一个脉冲周期都对应着两个控制信号：时钟控制信号与三态门控制信号，而且总线数据传送的流程均为源寄存器打开三态门将数据送入总线，然后目的寄存器经一个上升沿触发将数据打入寄存器内部，若两个信号分别由同一脉冲周期上升沿、下降沿来触发，则会出现冒险现象。本次实验中的解决方案为，将时钟控制信号的脉冲相比三态门控制信号的脉冲相位后移四分之一个时钟周期并均采用上升沿触发，从而在保证控制顺序的同时，避免了冒险。

4. 逻辑表达式的化简

为了实现方便，实验四只选取了实验三中的 16 条指令用来测试，但最终根据控制信号列表列出来的逻辑表达式依然非常繁杂，这时通过观察指令的操作时间表发现了规律，相同类型的指令在相同的时间存在着许多相同的

微操作，如运算指令的操作数送运算器、跳转指令的地址送程序计数器等，根据这一特性对逻辑表达式组中相同类型的指令进行聚类归并，即可大大降低逻辑表达式的复杂程度，从而更有利于电路的实现。

5.2 收获与体会

本次课程设计实验的工程量还是比较大的，项目文件前前后后加起来大约 300 多个，所以设计层次化是不可或缺的，严格遵循模型机的设计流程，自顶向下，首先要完成顶层结构的设计，然后再落实到底层细节的实现。当然，顶层的设计与底层的实现也是相互影响的，前期的设计多多少少会存在一些问题，只有通过后期大量的指令测试才能不断发现问题并去修正前期的设计方案，如此不断迭代，才能真正实现一个能够正确运行的模型机。

本次实验，从指令系统的设计到最后的运行调试，我深刻体会并学习了一台模型机的设计步骤，并在对每一步骤的思考与实现的过程中，巩固了课堂知识，加深了对计算机组成原理的理解。