《计算机组成与设计》课程设计报告

课程设计题目：简单模型机的设计

姓名：郑凯饶

学号：202000130143

班级：计科 20.1

日期：2022.11

目录

一、拟定指令系统

1. 指令集设计

模型机**基本字长为 8 位**，**寻址空间为 256×8 位**。

设计指令类别有访存类、运算类（逻辑、算术）、跳转类、寄存器转移类、递归功能类、循环功能类以及其他类别。

其中由于总体架构的局限性，仅设置了 4 个通用寄存器，绝大多数运算结果需存回主存。为了方便应用程序的编写，**简化汇编代码，将运算类拓展为 16 位**，运算结果直接存回内存的指定单元。

递归功能类和循环功能类指令是专门为提升递归和循环类应用程序编写效率而封装的复杂指令集合，可分解其他基本指令。

指令类别	助记符	指令长度	功能
访存	Ld, R0, add	16	读取内存单元 add 至 R0
	Ld, R1, add	16	读取内存单元 add 至 R1
	St, R0, add	16	将 R0 内容存入内存单元 add
	St, R1, add	16	将 R1 内容存入内存单元 add
算术运算	Add, add	16	R0 与 R1 内容**相加**，结果存入 R2，并存入内存单元 add
	Sub, add	16	R0 与 R1 内容**相减**，结果存入 R2，并存入内存单元 add
	Mul, add	16	R0 与 R1 内容**相乘**，结果存入 R2，并存入内存单元 add
	Div, add	16	R0 与 R1 内容**相除**，结果存入 R2，并存入内存单元 add
	Mod, add	16	R0 内容对 R1 **取模**，结果存入 R2，并存入内存单元 add
	Rs, add	16	R0 内容**算术右移**，结果存入 R2，并存入内存单元 add
	Ls, add	16	R0 内容**逻辑左移**，结果存入 R2，并存入内存单元 add
逻辑运算	Xor, add	16	R0 与 R1 内容**相异或**，结果存入 R2，并存入内存单元 add
	And, add	16	R0 与 R1 内容**相与**，结果存入 R2，并存入内存单元 add
	Or, add	16	R0 与 R1 内容**相或**，结果存入 R2，并存入内存单元 add
	Not, add	16	R0 内容**取非**，结果存入 R2，并存入内存单元 add
	Jmp, add	16	**无条件跳转**到指定地址
	Jz, add	16	R0 内容**等于** R1 则跳转到指定地址
	Jnz, add	16	R0 内容**不等于** R1 则跳转到指定地

跳转			址
	Jl, add	16	（有符号数）R0 内容小于 R1 则跳转到指定地址
	Jg, add	16	（有符号数）R0 内容大于 R1 则跳转到指定地址
	Ja, add	16	（无符号数）R0 内容小于 R1 则跳转到指定地址
	Jc, add	16	（无符号数）R0 内容大于 R1 则跳转到指定地址
寄存器间转移	Mov, R2, R0	8	将 R2 内容存入 R0
	Mov, R2, R1	8	将 R2 内容存入 R1
递归功能	Ld, SP, add	16	读取内存单元 add 至栈指针寄存器 SP
	Push	8	入栈（将 R0 内容存入 RAM(*SP)）
	Pop	8	出栈（将 add(SP)存入 R0）
	Exec, func	16	执行函数 func
	Ret	8	退出函数
循环功能	Ld, CX, add	16	读取内存单元 add 至循环变量寄存器 CX
	Exec, loop	8	执行循环 loop
其他	Inc, R0	16	R0 内容自增（R0++），并存入内存单元 add
	Hlt	8	停机指令

2. 指令格式

采用简单架构，没有实现对操作数的动态解析，操作码给定后操作数也确定。

指令类型分为 2 种：单地址码和无地址码指令。操作码长度为 6 位，留有 2 位供进一步拓展，目前支持 64 条指令。

	操作码（6 位）

	操作码（6 位）	地址码（8 位）

3. 操作码

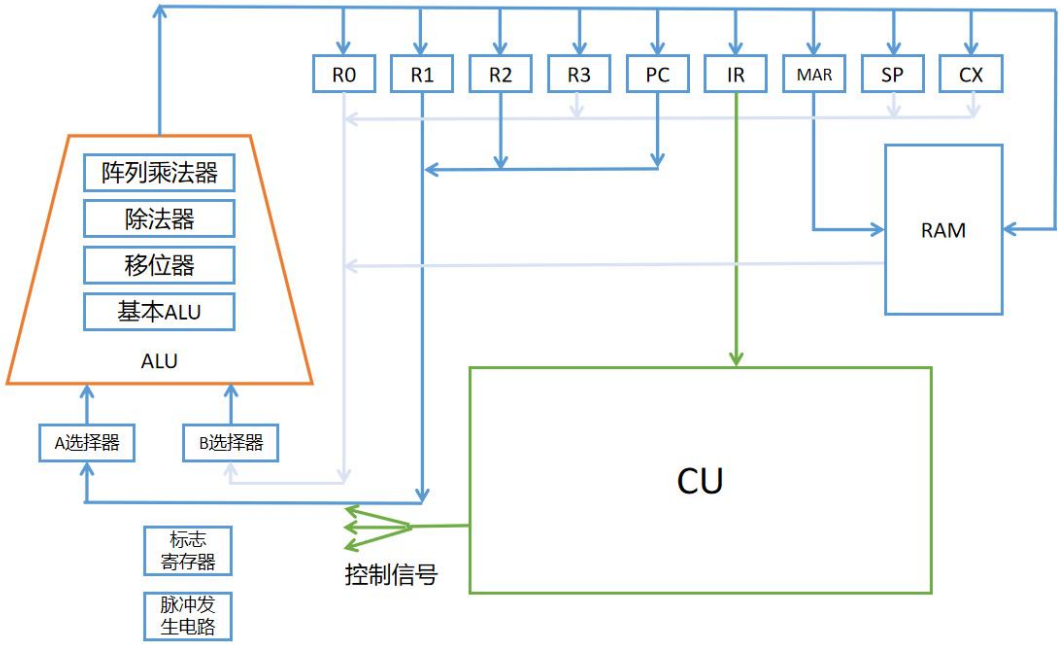
为方便操作码解析，指令操作码的编码为对应微程序入口地址的前 6 位。设编码为 IR5IR4IR3IR2IR1IR0，微程序入口地址解析为 uPC=IR5IR4IR3IR2IR1IR0000000.

目前支持寻址方式：直接寻址、寄存器寻址和间接寻址。双字长（16 位）的指令第二个字为操作数所在地址，通过直接寻址获取；寄存器间转移指令，可理解为采用了寄存器寻址，实际上操作数隐含于操作码当中；递归功能 Push 指令，操作数位于 RAM(*SP)，通过间接寻址读取；其他指令不含操作数。

微程序	操作码	微程序	操作码	微程序	操作码
取指	0 0	Xor, add	0 C	Mov, R2, R0	1 7
Ld, R0, add	0 1	And, add	0 D	Mov, R2, R1	1 8
Ld, R1, add	0 2	Or, add	0 E	Ld, SP, add	1 9
St, R0, add	0 3	Not, add	0 F	Push	1 A
St, R1, add	0 4	Jmp, add	1 0	Pop	1 B
Add, add	0 5	Jz, add	1 1	Exec, func	1 C
Sub, add	0 6	Jnz, add	1 2	Ret	1 D
Mul, add	0 7	Jl, add	1 3	Ld, CX, add	1 E
Div, add	0 8	Jg, add	1 4	Exec, loop	1 F
Mod, add	0 9	Ja, add	1 5	Inc, R0	2 0
Ls, add	0 A	Jc, add	1 6	Hlt	2 1
Rs, add	0 B				

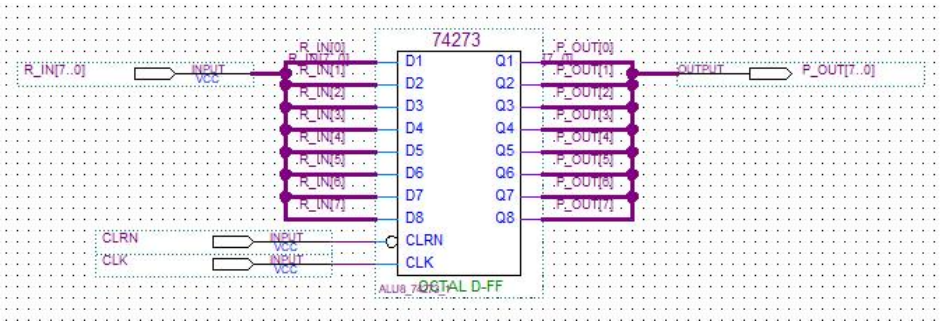
二、设计总体结构与数据通路框图

1. 总体结构和数据通路框图



（1）寄存器堆

寄存器均为 8 位，R0-3 为通用寄存器，IR 为指令寄存器，PC 为程序计数器，MAR 为地址寄存器，CX 为循环计数寄存器，SP 为堆栈寄存器。采用 74273 芯片实现。

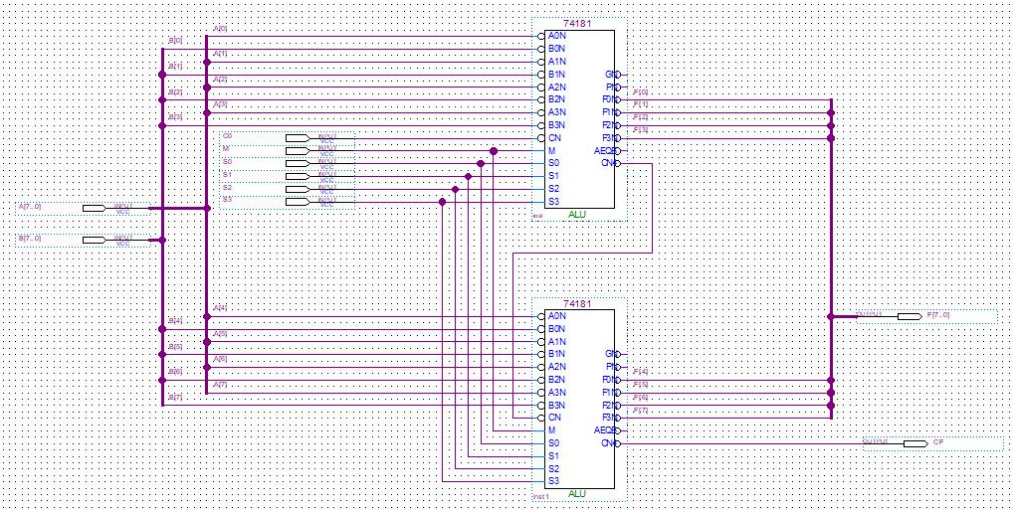


寄存器设计原理图

（2）运算器 ALU

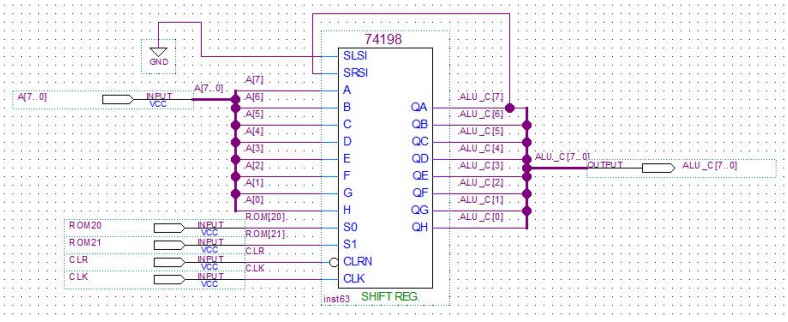
由基本 ALU、移位器、除法器以及阵列乘法器组成，通过四路选择器控制输出至总线上。

基本 ALU 由 2 片 4 位 74181 芯片级联而来，采用串行进行链，可实现数据直传、算术逻辑运算等功能。



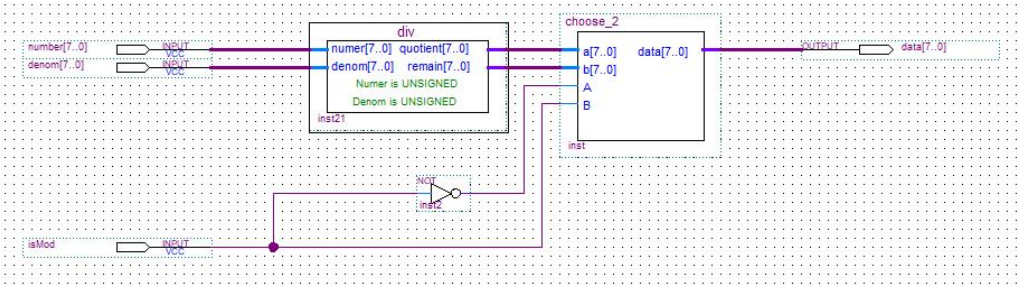
基本 ALU 设计原理图

移位器使用 74198 芯片，是运算器中唯一的时序电路，因此移位运算往往需要 2 个时钟周期实现，先同步置数，再进行左移/右移。



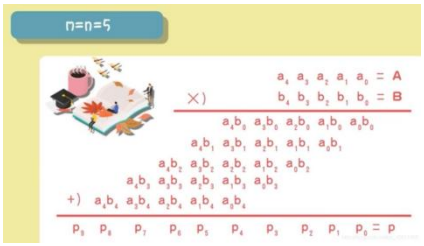
移位器设计原理图

除法器调用 PlugMegaWizard Plug-In Manager MegaWizard 中的 LPM_DIVIDE 器件，通过二路选择结果或者余数输出，是除法和取模指令的硬件基础。

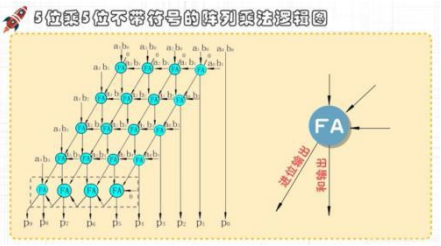


除法器设计原理图

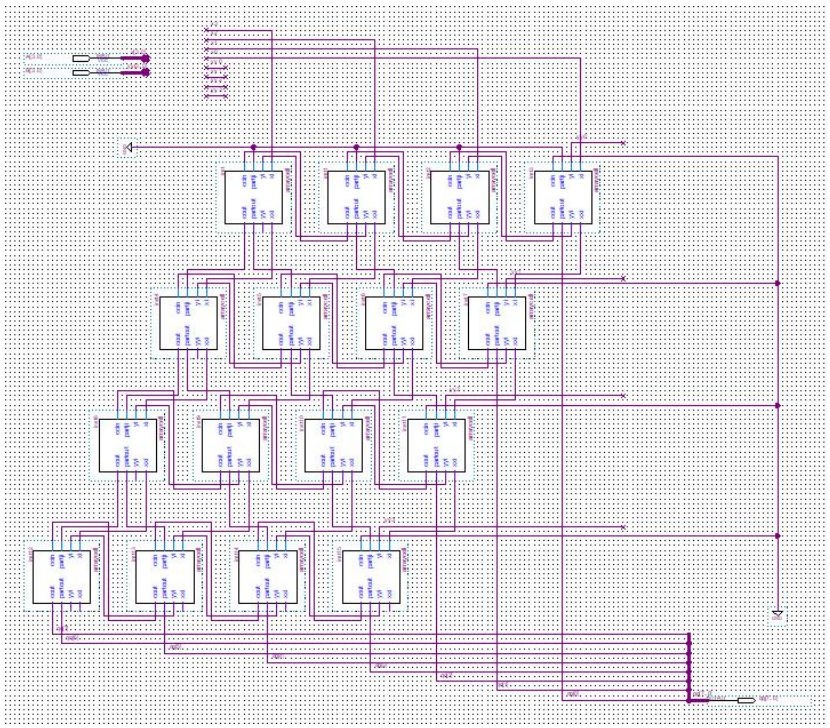
阵列乘法器是**由全加器构成的阵列**，手工计算乘法时实际上将乘法分为多次加法，全加器阵列模拟了该过程，下右图为逻辑图，表示全加器之间的连接关系。



乘法竖式计算图



阵列乘法逻辑图（引用他人博客）

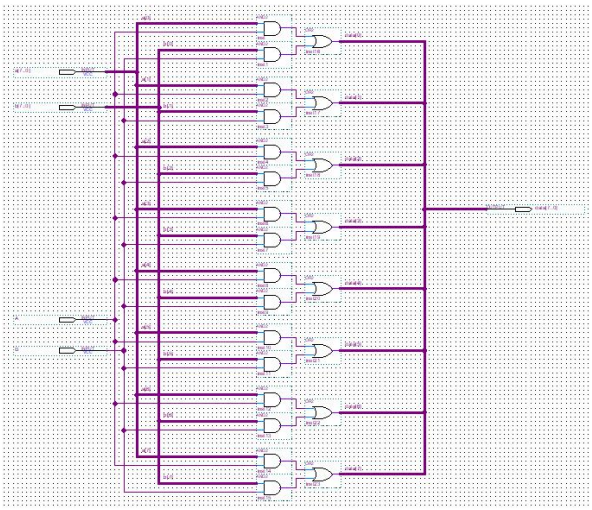


阵列乘法器设计原理图

注意到对于无符号数，该实现仅支持 4 位×4 位的乘法运算，因此在后面 RAM 应用程序采用**乘法后取模**的做法，做**模 k 系统下的乘法**，避免溢出。

（3）选择器

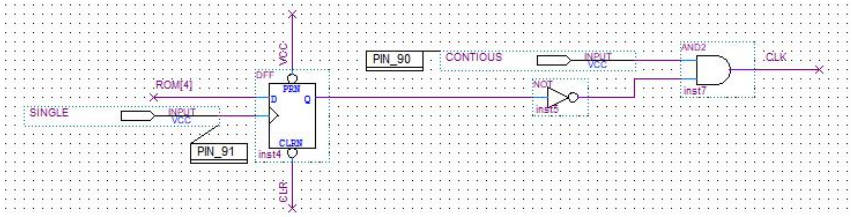
本设计中所有选择器均通过与门和或门组合实现，每个输入对应一个控制信号，理论上增加输入后可通过译码器减少控制信号数，但设计中用到的选择器最多路数为 4，数目较少，因此采用简单的输入与信号一对一的方式。



2 路选择器设计原理图

（4）脉冲发生电路

CONTINUOUS 与 SINGLE 信号均设置为连续脉冲，前者提供模型机运行的时钟信号，后者提供停机信号，仅仅发挥一次作用，将 G=1 信号打入 D 触发器，阻断时钟，也可视为单脉冲。



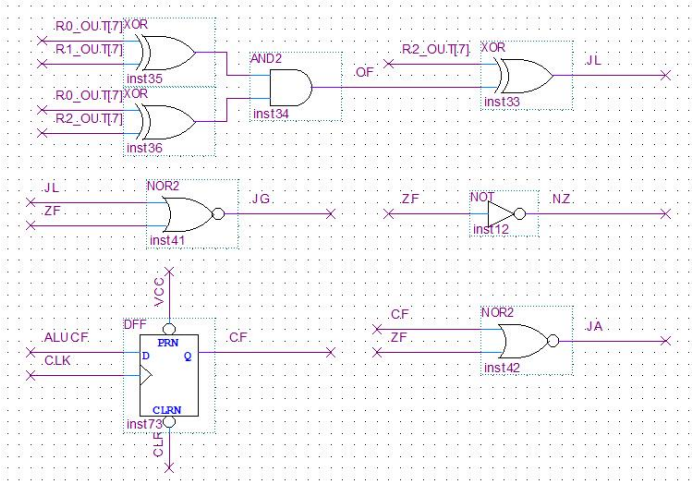
脉冲发生电路设计原理图

（5）标志寄存器

OF 为有符号数运算溢出标志位，通过和结果符号位或非可进行有符号数之间的大小比较，进一步实现 JL 和 JG。

ZF 为零标志位，表示结果是否为 0，通过取非得到 NZ。

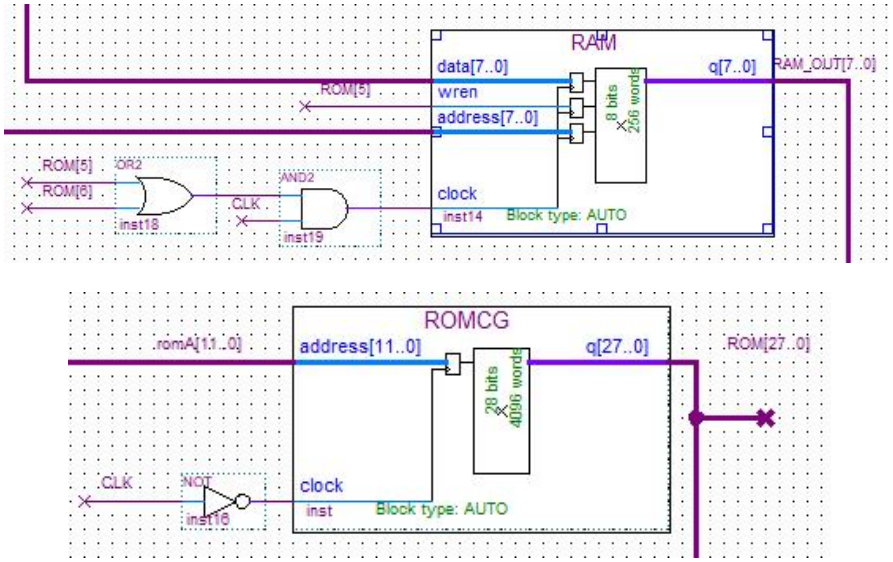
CF 为基本 ALU 的进位，通过和 ZF 或非实现 JA 和 JC。



标志寄存器原理设计图

（6）存储器

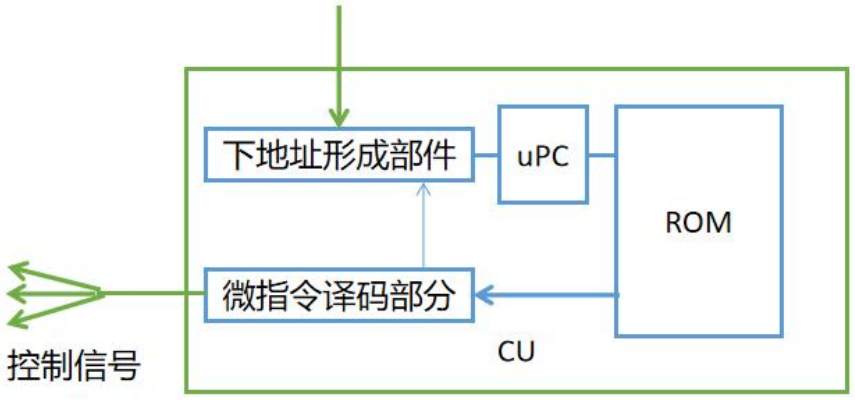
主存为 256×8 的 RAM，由读写信号 W/R 以及时钟信号 CLK 使能，微程序中控存为 4096×28 位的 ROM，由反时钟信号使能，取非是为了相对于 uPC 延后半个时钟周期，**确保在上机测试时微地址有充足的时间形成**。调用 PlugMegaWizard 中的实现。



ROM 和 RAM 设计原理图

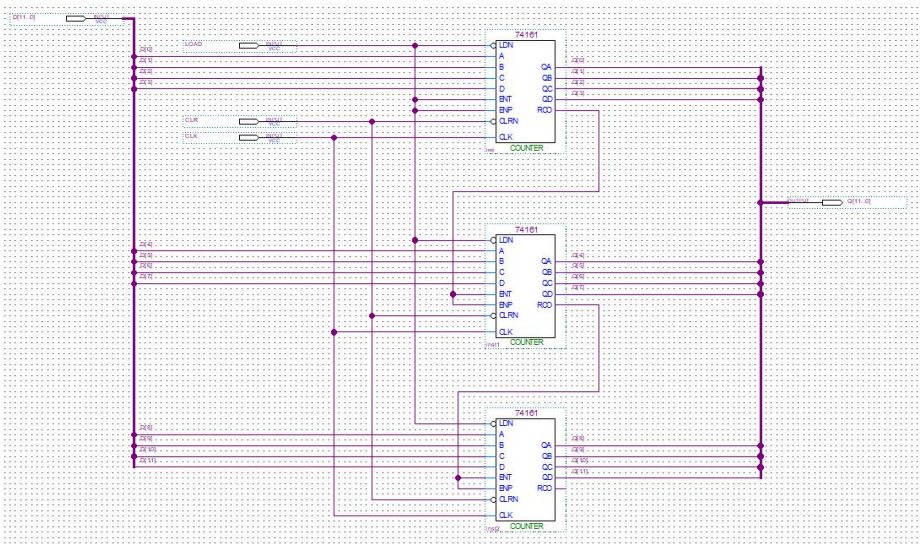
三、微程序实现的控制部件 CU

1. 微程序实现的 CU 框图



（1） uPC

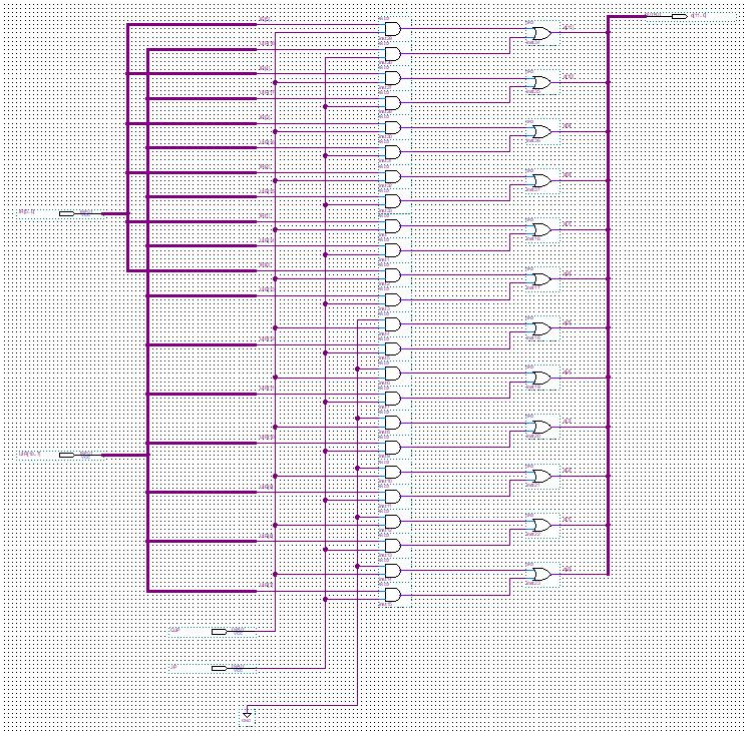
又称微程序计数器，PC 是 RAM 程序当前执行位置的指针，而 uPC 是 ROM 中微程序当前位置的指针，实现 load 和自增功能，前者和 JP 类的下地址形成方式相关，后者用于实现 uPC+1。因为微地址共 12 位，通过 3 片 74161 拼接而成，每片输出 4 位微地址。通过将下级芯片的 RCO 输出接至上级的 ENT、ENP 输入实现进位模拟。



uPC 设计原理图

（2） 下地址形成部件

实际上为 2 路选择器，**QJP 和 JP 作为控制信号，选择 IR 或者 uIR 输出**。当 QJP=1 时，q=IR5IR4IR3IR2IR1IR0000000，根据指令进行跳转，找到对应微程序入口地址；当 JP=1 时，q=uIR18-uIR7，跳转至取指指令，因此微指令中 uIR18-uIR7 区间设置为 0。



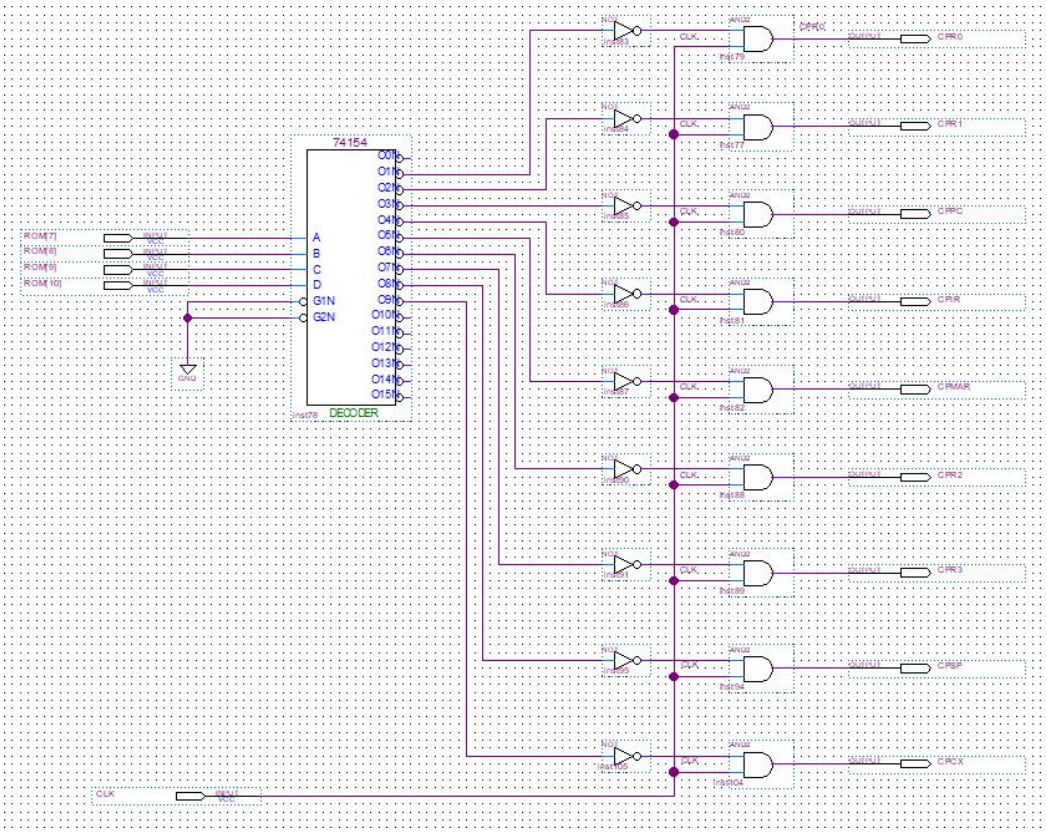
下地址形成部件设计原理图

（3） 控存 ROM

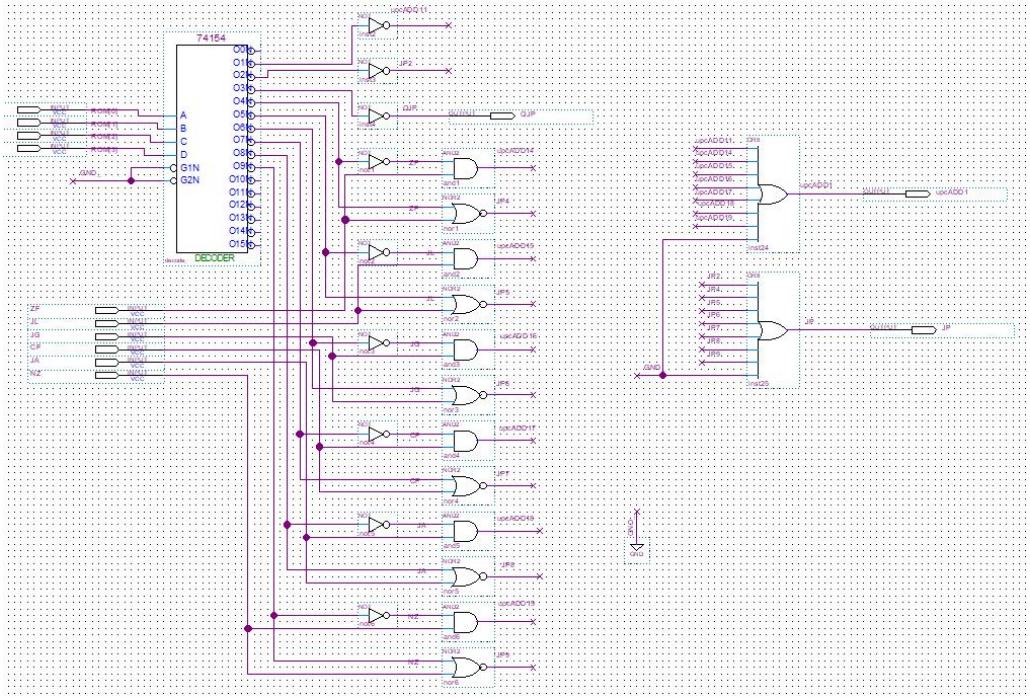
在总体结构部分**存储器**中已说明。

（4） 微地址译码部分

在保证微指令可读性和指令长度适中下，**将微指令按语义划分为多个字段，分别进行译码**。主要采用 3-8 译码器 74138、74139 和 4-16 译码器 74154，将微指令转为为控制信号。



译码寄存器选择字段 uIR10-uIR7



译码下地址形成字段 uIR3-uIR0

2. 指令执行流程

（1）所有指令最后一步微操作均为 JP，跳转到取指微程序，取下一条指令，但对于跳转类指令而言，**微程序可能不会完整执行**，例如 Jz，add 在 Z 操作可以跳转，跳过剩下部分；

（2）一类指令一般只有一个微操作不同。因此下表以及附录 3 中微程序流程图仅选取代表指令，不同部分粗体表示。

（3）有些微操作可能**无法在 1 个时钟周期完成**。例如 RAM->R，将 RAM 指定内存单元读至寄存器 R，RAM 和 R 均需 CLK，由于在 FPGA 硬件延迟，CLK 必须先后触发 RAM，R，因此必须用 2 个时钟周期完成该微操作。

微程序	微操作	微程序	微操作
取指	RAM->IR	Jz, add	PC->MAR
			PC+1->PC
	PC+1->PC		RAM->R3
	QJP		
			PC->MAR
Ld, R0, add	PC->MAR		**Z**
	PC+1->PC		R3->PC
	RAM->MAR		PC->MAR
			JP
	RAM->R0		
		Exec, func	PC->MAR
	PC->MAR		PC+1->PC
	JP		RAM->R3
St, R0, add	PC->MAR		**SP-1->SP**
	PC+1->PC		**SP->MAR**
	RAM->MAR		PC->RAM
			R3->PC
	R0->RAM		PC->MAR
	PC->MAR		JP
	JP		
		Ret	**SP->MAR**
Add, add	**R0+R1->R2**		RAM->PC
	PC->MAR		
	PC+1->PC		**SP+1->SP**
	RAM->MAR		PC->MAR
			JP
	R2->RAM		
	PC->MAR	Exec, loop	PC->MAR
	JP		PC+1->PC
			RAM->R3

Ls, add	R0<<1		
			PC->MAR
	R0<<1->R2		CX-1->CX
	PC->MAR		CX->R2
	PC+1->PC		NZ
	RAM->MAR		R3->PC
			PC->MAR
	R2->RAM		JP
	PC->MAR		
	JP	Hlt	G=1

3. 微指令

（1）微指令格式：

微指令字长为 28 位，即 uIR27-uIR0，根据控制语义划分为多个字段，ALU 选择，ALU 控制和后继地址形成等等。

27	26	25	24	23	22	21	20	19
ALU 选择			M	S3	S2	S1	S0	CN

18	17	16	15	14	13	12	11	10	9	8	7
	A 选择器			B 选择器				寄存器选择			
后继地址（12 位）											

6	5	4	3	2	1	0
RD	WR	G	后继地址形成			

（2）微指令字段定义：

1. ALU 选择：uIR27，uIR26，uIR25

将基本 ALU(ALUA)、阵列乘法器(ALUB)、移位器(ALUC)、除法器(ALUD)的结果以及余数的输出之一传输至总线上。

uIR27	uIR26	uIR25	
0	0	0	ALUA
0	0	1	ALUB
0	1	0	ALUC
0	1	1	ALUD(结果)
1	1	1	ALUD(余数)

2. ALU 控制：uIR24，uIR23，uIR22，uIR21，uIR20，uIR19

控制 ALUA 进行算术运算、逻辑运算、数据直传等，控制 ALUC 进行移位运算。

3. A 选择器控制：uIR17，uIR16，uIR15

控制 ALU 的 A 端输入，选择范围为{R0, RAM, SP, R3, CX}.

uIR17	uIR16	uIR15	
0	0	0	
0	0	1	R0
0	1	0	RAM
0	1	1	SP
1	0	0	R3
1	0	1	CX

4. B 选择器控制：uIR14，uIR13，uIR12
   控制 ALU 的 B 端输入，选择范围为{PC, R1, R2}.

uIR14	uIR13	uIR12	
0	0	0	
0	0	1	PC
0	1	0	R1
0	1	1	R2

5. 寄存器选择：uIR10，uIR9，uIR8，uIR7
   将运算结果存入指定的寄存器。

uIR10	uIR9	uIR8	uIR7	
0	0	0	0	
0	0	0	1	R0
0	0	1	0	R1
0	0	1	1	PC
0	1	0	0	IR
0	1	0	1	MAR
0	1	1	0	R2
0	1	1	1	R3
1	0	0	0	SP
1	0	0	1	CX

6. 存储器的读写控制：uIR6，uIR5

uIR6	uIR5	
0	0	
0	1	WR
1	1	RD

7. 停机控制：uIR4
   uIR4 为 0 时，G=0，机器运行；uIR4 为 1 时，G=1，机器停机。
8. 后继微地址形成控制：uIR3，uIR2，uIR1，uIR0
   一共设计了 9 种后继微地址形成方式，其中**基本跳转**为 uPC+1，JP，QJP，

分别实现微地址自增使微程序顺序执行，执行取指周期以及根据指令寄存器 IR 内容进行跳转以寻找指令对应的为程序入口地址。**条件跳转**基于前者实现，通过判断标志寄存器的值决定执行 uPC+1 或者 JP.

uIR3	uIR2	uIR1	uIR0	
0	0	0	0	
0	0	0	1	uPC+1
0	0	1	0	JP
0	0	1	1	QJP
0	1	0	0	JZ
0	1	0	1	JL
0	1	1	0	JG
0	1	1	1	JC
1	0	0	0	JA
1	0	0	1	JNZ

4. RAM 应用程序

一开始的想法就是在自己设计的模型机上实现快速幂，也是这个目标指引小组自顶向下地进行指令设计，现在我们做到了！

（1）快速幂

约定 RAM 地址 A0 开始为数据存储区，变量定义在该区域；RAM 地址 9F 为无意义单元 null，由于本设计运算指令附带存储的特性，不需要的结果可以存于该单元，视为丢弃。

快速幂综合运用了多种类型的指令，如条件跳转指令 Jnz 和 Jz，运算指令 Mul 和 Mod，寄存器间转移指令 Mov 等等，体现模型机完备而丰富的指令系统。

功能：快速幂计算：x^{y} mod z			
RAM 地址	汇编指令	机器语言	C/C++
0 0	Ld, R0, x	0 1	
0 1		A 1	
0 2	Ld, R1, mod	0 2	
0 3		A 3	
0 4	Mod, x	0 9	x %= mod;
0 5		A 1	
0 6	Ld, R0, y	0 1	
0 7		A 2	
0 8	Ld, R1, zero	0 2	
0 9		A 0	

0 A	Add, y	0 5	
0 B		A 2	y -> R2
0 C	Jz, (ad1)	1 1	
0 D		3 2	
0 E	(ad3)Ld, R0, y	0 1	while (y)
0 F		A 2	
1 0	Ld, R1, 2	0 2	
1 1		A 5	
1 2	Mod, null	0 9	
1 3		9 F	
1 4	Jz, (ad2)	1 1	if (y & 1)
1 5		2 1	
1 6	Ld, R0, res	0 1	
1 7		A 4	
1 8	Ld, R1, x	0 2	
1 9		A 1	
1 A	Mul, res	0 7	
1 B		A 4	
1 C	Mov, R2, R0	1 7	
1 D	Ld, R1, mod	0 2	
1 E		A 3	
1 F	Mod, res	0 9	res = res * x % mod;
2 0		A 4	
2 1	(ad2)Ld, R0, x	0 1	
2 2		A 1	
2 3	Ld, R1, x	0 2	
2 4		A 1	
2 5	Mul, x	0 7	
2 6		A 1	
2 7	Mov, R2, R0	1 7	
2 8	Ld, R1, mod	0 2	
2 9		A 3	
2 A	Mod, x	0 9	x = x * x % mod;
2 B		A 1	
2 C	Ld, R0, y	0 1	
2 D		A 2	
2 E	Rs, y	0 B	y >>= 1;
2 F		A 2	
3 0	Jnz, (ad3)	1 2	while (y)
3 1		0 E	
3 2	(ad1)Ld, R0, res	0 1	
3 3		A 4	
3 4	Hlt	2 1	

3 5			
3 6			
3 7			
3 8			
3 9			
9 D			
9 E			
9 F	null		
A 0	zero	0 0	
A 1	x		
A 2	y		
A 3	mod	0 7	
A 4	res	0 1	
A 5	2	0 2	
A 6			
A 7			
A 8			
A 9			

（2）循环功能

相比通过跳转类指令实现循环，使用封装的循环功能指令更加高效，同时程序具有更高的可读性。

功能：循环累加：1+2+...+n			
RAM 地址	汇编指令	机器语言	C/C++
0 0	Ld, CX, n	1 E	while (i <= n)
0 1		A 1	
0 2	Inc, R0	2 0	i++;
0 3		A 3	
0 4	Ld, R1, add	0 2	
0 5		A 2	
0 6	Add, add	0 5	sum += i;
0 7		A 2	
0 8	Exec, loop	1 F	while (i <= n)
0 9		0 2	
0 A	Ld, R0, sum	0 1	
0 B		A 2	
0 C	Hlt	2 1	
0 D			
0 E			
0 F			

9 C			
9 D			
9 E			
9 F	null		
A 0	zero	0 0	
A 1	n	0 A	
A 2	sum		
A 3	i		
A 4			
A 5			
A 6			
A 7			
A 8			
A 9			

（3）递归功能

通过递归功能指令调用函数 func，可通过指定函数临时变量区域，即函数调用栈，进一步拓展实现多级递归调用。

功能：调用函数，循环累乘：（1*2*...*n）mod k			
RAM 地址	汇编指令	机器语言	C/C++
0 0	Exec, func	1 C	f();
0 1		1 0	
0 2	Ld, R0, res	0 1	cout << res;
0 3		A 2	
0 4	Hlt	2 1	
0 5			
0 6			
0 7			
0 8			
0 9			
1 0	Ld, CX, n	1 E	while (i <= n)
1 1		A 1	
1 2	Ld, R0, i	0 1	
1 3		A 3	
1 4	Inc, R0	2 0	i++;
1 5		A 3	
1 6	Ld, R1, res	0 2	
1 7		A 2	
1 8	Mul, res	0 7	res *= i;
1 9		A 2	

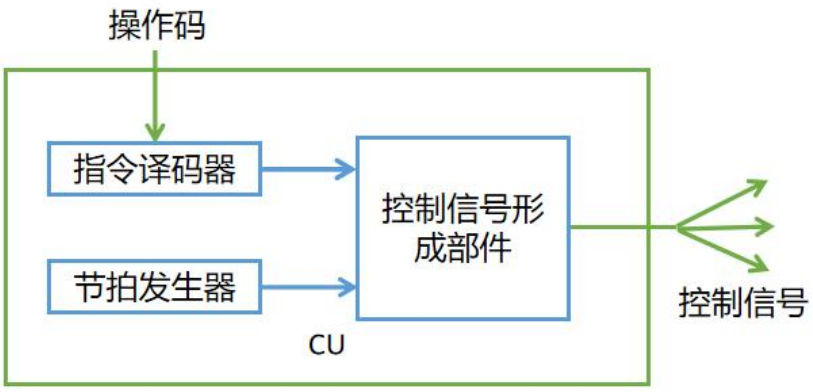
1 A	Ld, R1, mod	0 2	
1 B		A 4	
1 C	Mov, R2, R0	1 7	
1 D	Mod, res	0 9	res %= mod;
1 E		A 2	
1 F	Exec, loop	1 F	while (i <= n)
2 0		1 2	
2 1	Ret	1 D	
2 2			
2 3			
9 C			
9 D			
9 E			
9 F	null		
A 0	zero	0 0	
A 1	n	0 A	
A 2	res	0 1	
A 3	i		
A 4	mod	0 B	
A 5			
A 6			
A 7			
A 8			
A 9			

四、硬布线实现的控制部件 CU

由于时间关系，按照 PPT 仅实现部分指令，**不同于微指令实现的指令系统**，如下表：

指令	功能
Mov, R0, imm	将立即数 imm 存入 R0
Mov, R1, imm	将立即数 imm 存入 R1
R0+R1->R1	将 R0 内容和 R1 **相加**，存入 R1
Mov, add, R1	将 R1 内容存入内存单元 add
Hlt	停机
R0*R1->R1	将 R0 内容和 R1 **相乘**，存入 R1
R0/R1->R1	将 R0 内容和 R1 **相除**，存入 R1

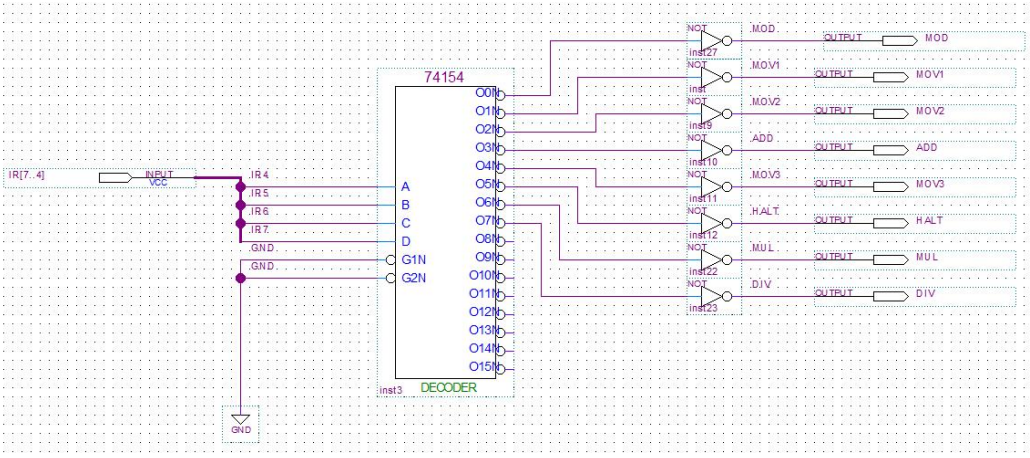
1. 硬布线实现的 CU 框图



（1）指令译码器

编排指令操作码如下表，通过指令译码器将操作码“翻译”为 MOV,ADD,MUL 等等指令信号。由于仅有 7 条指令，基于 4-16 译码器 74154 实现即可。

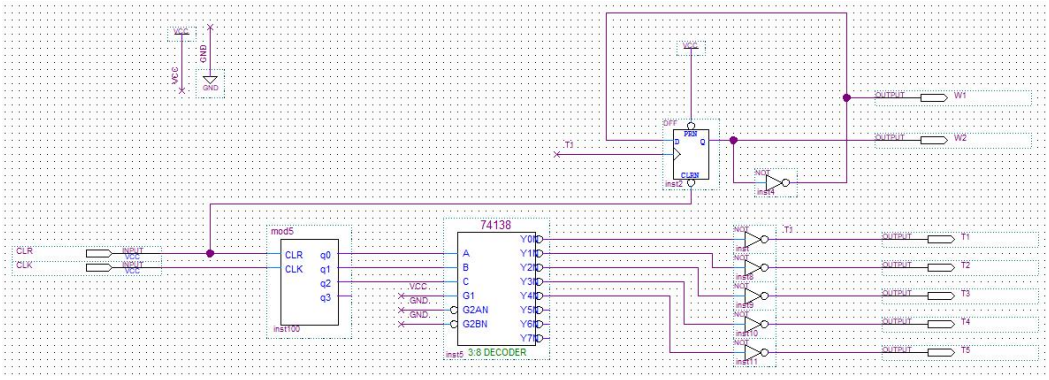
指令	操作码	指令	操作码
Mov, R0, imm	0001	Hlt	0101
Mov, R1, imm	0010	R0*R1->R1	0110
R0+R1->R1	0011	R0/R1->R1	0111
Mov, add, R1	0100		



指令译码器原理设计图

（2）节拍发生器

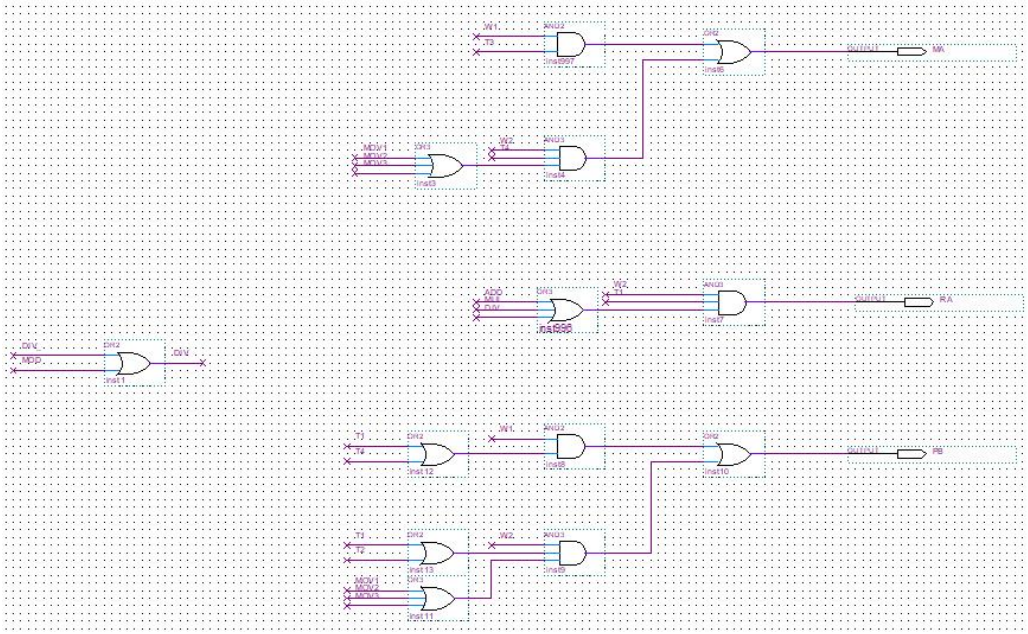
根据指令执行流程，设计时序为：分为取指周期和执行周期，每个周期为 5 节拍，W1 高电平时是取指周期的 5 节拍，W2 高电平时是执行周期的 5 节拍。将模 4 增 1 计数器的输出接 3-8 译码器 74138 生成节拍 T1-5，**将 T1 设置为触发脉冲**，每次 T1 有效反转 W 信号，进入下一个时钟周期。



节拍发生器原理设计图

（3）控制信号形成部件

实质为控制信号逻辑表达式的实现，**均通过与门和或门组合实现**。输入为指令信号以及时序信号（W1,W2,T1-5），输出为控制信号。同一时间可能有多个控制信号输出。



（部分）控制信号形成部件原理设计图

2. 指令执行流程

所有指令取指周期 W1 相同，微操作均为：

W1	T1	PC->MAR
	T2	RAM(R)
	T3	RAM->IR
	T4	PC+1->PC
	T5	

指令的执行周期，对应指令的功能实现，各自微操作为：

		MOV1	MOV2
W2	T1	PC->MAR	PC->MAR
	T2	PC+1->PC	PC+1->PC
	T3	RAM(R)	RAM(R)
	T4	RAM->R0	RAM->R1
	T5		
		ADD	MOV3
W2	T1	R0+R1->R1	PC->MAR
	T2		PC+1->PC
	T3		RAM(R)
	T4		RAM->MAR
	T5		R1->RAM

		HALT	MUL	DIV
W2	T1	G=1	R0*R1->R1	R0/R1->R1
	T2			
	T3			
	T4			
	T5			

3. 控制信号

将微操作“翻译”为对应的控制信号，以下为取值周期 W1 的控制信号列表：

W1	T1	PB M S3 S1 CPMAR
	T2	RD
	T3	MA M S3 S2 S1 S0 CPIR
	T4	PB S3 S0 CPPC
	T5	

各指令执行周期 W2 的控制信号列表：

		MOV1	MOV2
W2	T1	PB M S3 S1 CPMAR	PB M S3 S1 CPMAR
	T2	PB S3 S0 CPPC	PB S3 S0 CPPC
	T3	RD	RD
	T4	MA M S3 S2 S1 S0 CPR0	MA M S3 S2 S1 S0 CPR1
	T5		
		ADD	MOV3
W2	T1	RA RB S3 S0 CN CPR1	PB M S3 S1 CPMAR
	T2		PB S3 S0 CPPC
	T3		RD
	T4		MA M S3 S2 S1 S0 CPMAR

	T5		RB M S3 S1 WR	
		HALT	MUL	DIV
W2	T1	G	RA RB DYC0 CPR1	RA RB DYC1 CPR1
	T2			
	T3			
	T4			
	T5			

从控制信号的角度，通过逻辑表达式表征该控制信号的作用时期，比如从控制列表中看到控制信号 PB 在所有指令的取指周期 W1 的 T1 节拍中作用，因此 PB 信号的表达式中包含子项 W1*T1*(MOV1+MOV2+ADD+MOV3+HALT+MUL+DIV)，因为 MOV1+MOV2+ADD+MOV3+HALT+MUL+DIV=1，式子可简写为 W1*T1.所有控制信号逻辑表达式如下表：

MA	W1*T3+W2*T4(MOV1+MOV2+MOV3)
RA	W2*T1*(ADD+MUL+DIV)
PB	W1*(T1+T4)+W2*(T1+T2)*(MOV1+MOV2+MOV3)
RB	W2*T1*(ADD+MUL+DIV)+W2*T5*MOV3
CPR0	W2*T4*MOV1
CPR1	W2*T1*(ADD+MUL+DIV)+W2*T4*MOV2
CPPC	W1*T4+W2*T2*(MOV1+MOV2+MOV3)
CPIR	W1*T3
CPMAR	W1*T1+W2*T1*(MOV1+MOV2+MOV3)+W2*T4*MOV3
RD	W1*T2+W2*T3*(MOV1+MOV2+MOV3)
WR	W2*T5*MOV3
G	W2*T1*HALT
M	W1*(T1+T3)+W2*((T1+T4)*MOV1+(T1+T4)*MOV2+(T1+T4+T5)*MOV3))
S3	1
S2	W1*T3+W2*T4*(MOV1+MOV2+MOV3)
S1	W1*(T1+T3)+W2*(T1+T4)*(MOV1+MOV2+MOV3)+W2*T5*MOV3
S0	W1*(T3+T4)+W2*(T2+T4)*(MOV1+MOV2+MOV3)+W2*T1*ADD
CN	W2*T1*ADD
DYC0	W2*T1*MUL
DYC1	W2*T1*DIV

4. RAM 的应用程序

通过基本四则运算检验硬布线控制器的功能。

功能：基本四则运算			
RAM 地址	汇编指令	机器语言	C/C++
0 0	Mov, R0, 6	10	
0 1		6	
0 2	Mov, R1, 2	20	
0 3		2	
0 4	R0/R1->R1	70	6 ÷ 2 = 3
0 5	Mov, 20H, R1	40	
0 6		20	
0 7	R0+R1->R1	30	6 + 3 = 9
0 8	Mov, 21H, R1	20	
0 9		21	
0 A	R0*R1->R1	60	6 * 9 = 54
0 B	Mov, 22H, R1	20	
0 C		22	
0 D	Hlt	50	
0 E			
0 F			

五、总结与体会

至此，计组课设告一段落了。首先，总结一下我们设计的模型机。一开始，我们对指令系统毫无头绪，后来灵机一动：不如用我们设计的模型机实现一个算法，以此为导向，自顶向下地设计出我们的指令系统。最终，敲定了快速幂，一个可以触摸到的目标！

设计过程颇为波折。起初我们想实现 RISC 指令集的一些特点，如仅有 Load/Store 类型指令可以访存，后来发现这对操作数的解析提出了更高的要求，需要动态解析操作数，而参照 ppt 上的方法一条 Load 指令的操作数是固定的，想涵盖更多操作数，只能设计更多的指令。由于时间关系，我们采取了简单方案。在设计运算指令时，我们发现受限于寄存器的个数（增加寄存器意味着增加更多的 Move 类型指令），运算指令后会紧跟着 Store 指令，当时我突发奇想不如直接将 Store 功能加到所有运算指令之中，于是有了后来助教戏称的“变量式指令系统”。效仿现代的指令系统，我们为循环和递归功能设计了相应的指令，简化了汇编代码的编写。最终我们设计的指令系统有如下特点：

①实现访存，四则运算，取模运算，与、或、非、异或等逻辑运算以及停机等等基本指令；
②实现基于有、无符号数大小比较的多种条件跳转指令，为模型机实现复杂功能打下基础；
③实现递归功能、循环功能的复杂指令集合；
④变量式指令系统，由于运算结果直接回写，汇编指令中可用变量名替代变量地址，对应地址存储指定变量。

微程序的编写和指令设计是同步进行的，基于硬件设计对应微操作，考虑微操作对应的控制信号，这里需要极大的耐心，微程序的表格我做了整整一个星期！利用 excel 的函数工具和指令之间的相似性可以减少一些工作量。鉴于实验 2 在 FPGA 平台上遇到的种种时序问题，我们在设计一些微操作时特别注意了其控制信号的前后关系，如微操作 RAM->R1，RAM 的读信号和 R1 触发信号不能同时到达，因此，我们将该微操作分为两步，在两个时钟周期完成。

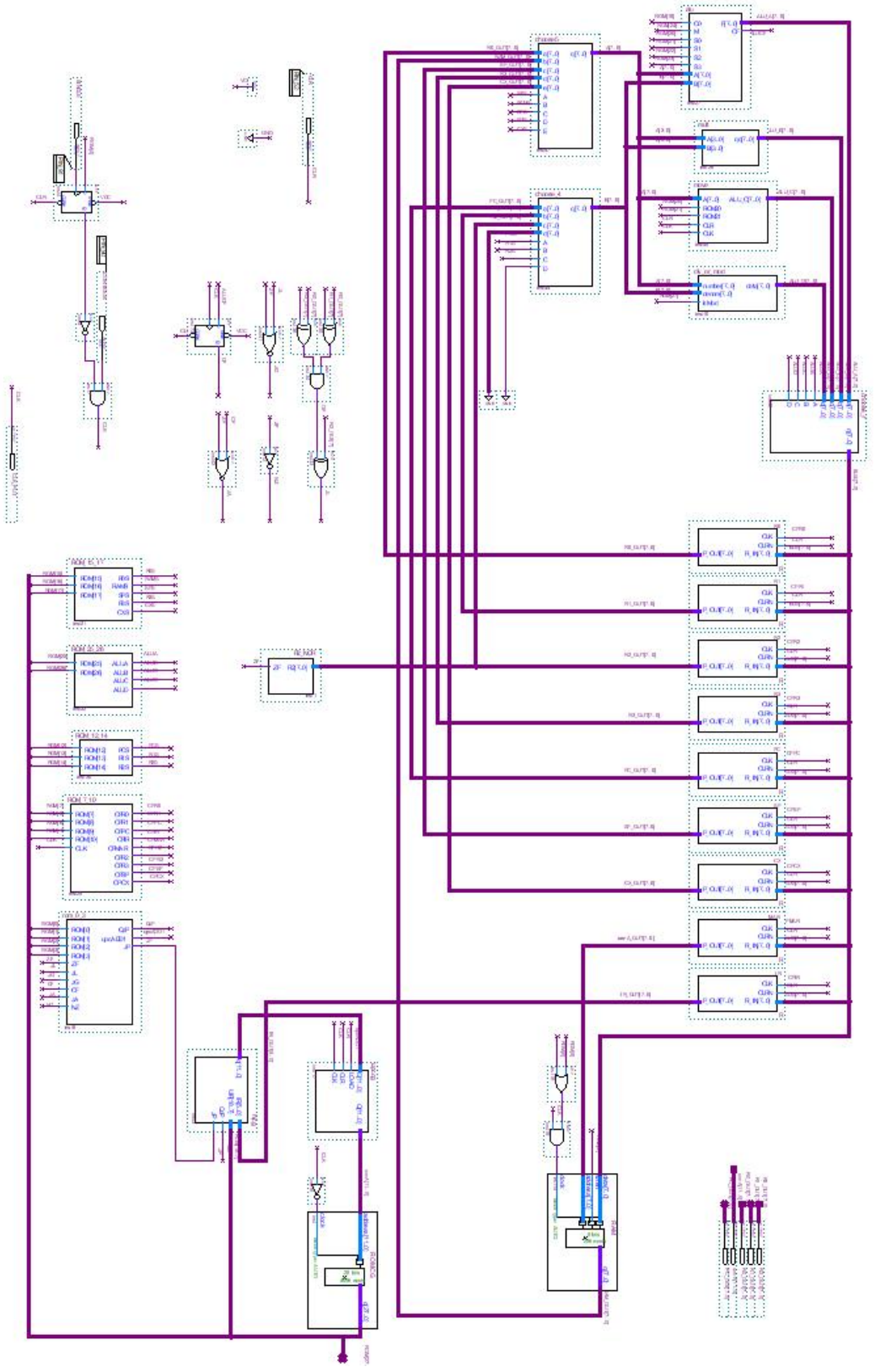
对应的测试应用程序，测试其微操作、微指令是否正确，以及 Quarters 上设计图有没有出现逻辑或者接线错误。反反复复调整 ROM 程序和设计图，在第一条指令成功执行后，我们内心是莫大的激动！之后，顺水推舟我们用测试完成的指令系统汇编了快速幂程序，通过了仿真测试。上机那天，看着 Memory Content Editor 中 x，y，res 等等变量对应内存单元的数字跳动，最后停在一个数字，而那个数字正是计算器的运算结果，那一刻，我知道我们成功了！

硬布线部分我们并没有沿用自己设计的指令而是参照 ppt 设计，硬件部分除 CU 外基本相同，数据通路相同，只是 CU 更换为硬布线实现。而硬布线实现与微程序实现有许多异曲同工之处，指令译码器和下地址形成部件“接收”来自 IR 寄存器的指令信息，控制信号形成部件和 ROM 中或显或隐地存储了指令的控制信号，前者通过逻辑门组合的方式表达了控制信号和操作码以及节拍的关系，后者通过微程序、微指令存储控制信号。设计完成指令的微操作后，对微操作进行节拍安排，主要是执行周期。之后考虑微操作中所有控制信号，从控制信号的角度出发描述它在哪个周期哪个节拍哪个指令中出现，编写并化简对应的逻辑表达式。

计算机组成原理在 CS 学科中的地位举重若轻，往往我们醉心于 CS 纷繁的应用，而忘记所有的程序、算法都是在机器上跑的，没有好的硬件设计一切都免谈，这里最是需要深耕！希望以后有机会学习一些相对成熟的组成设计、指令集项目，提升自己的学科素养。

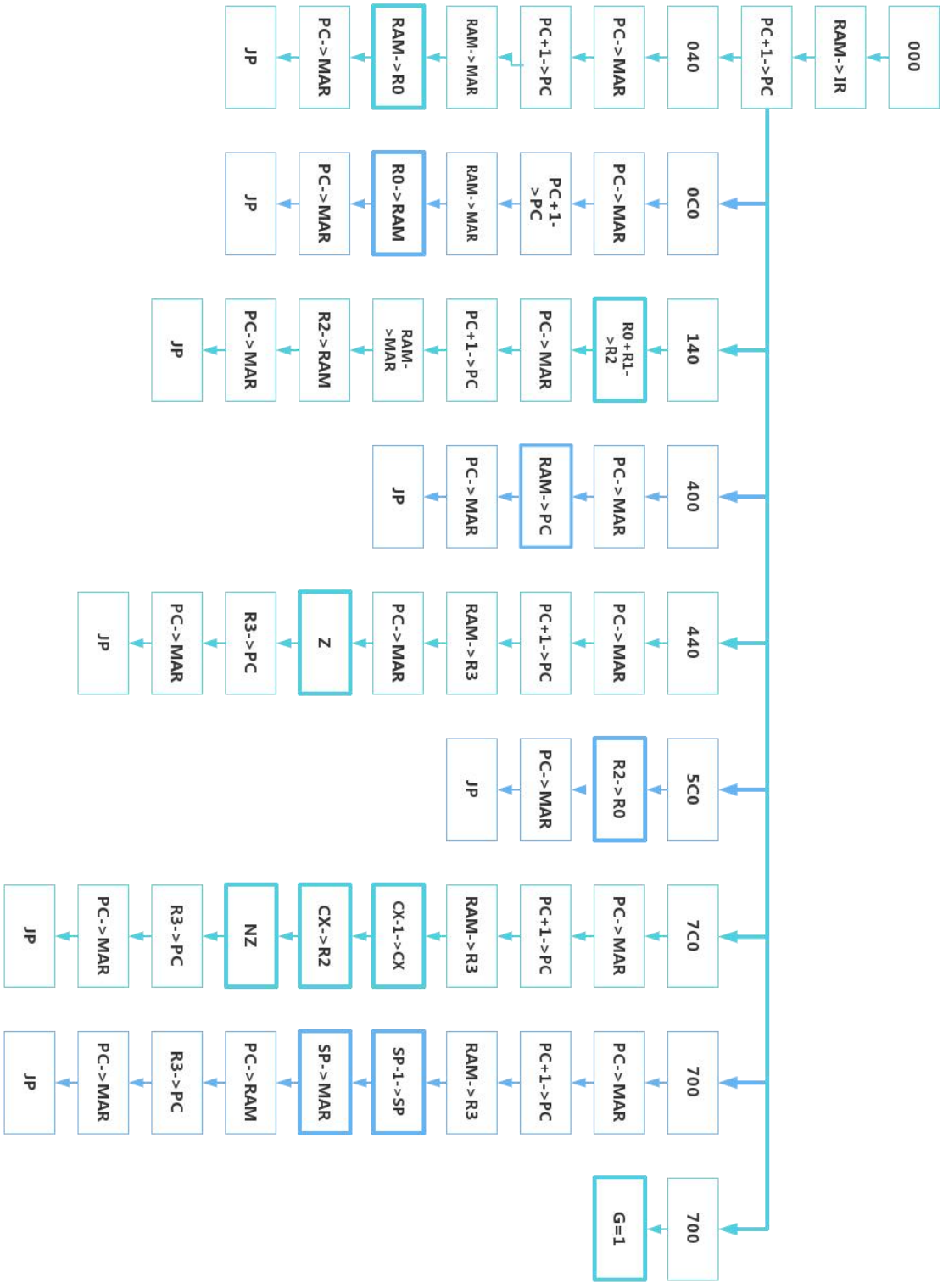
附录 1：

数据通路图



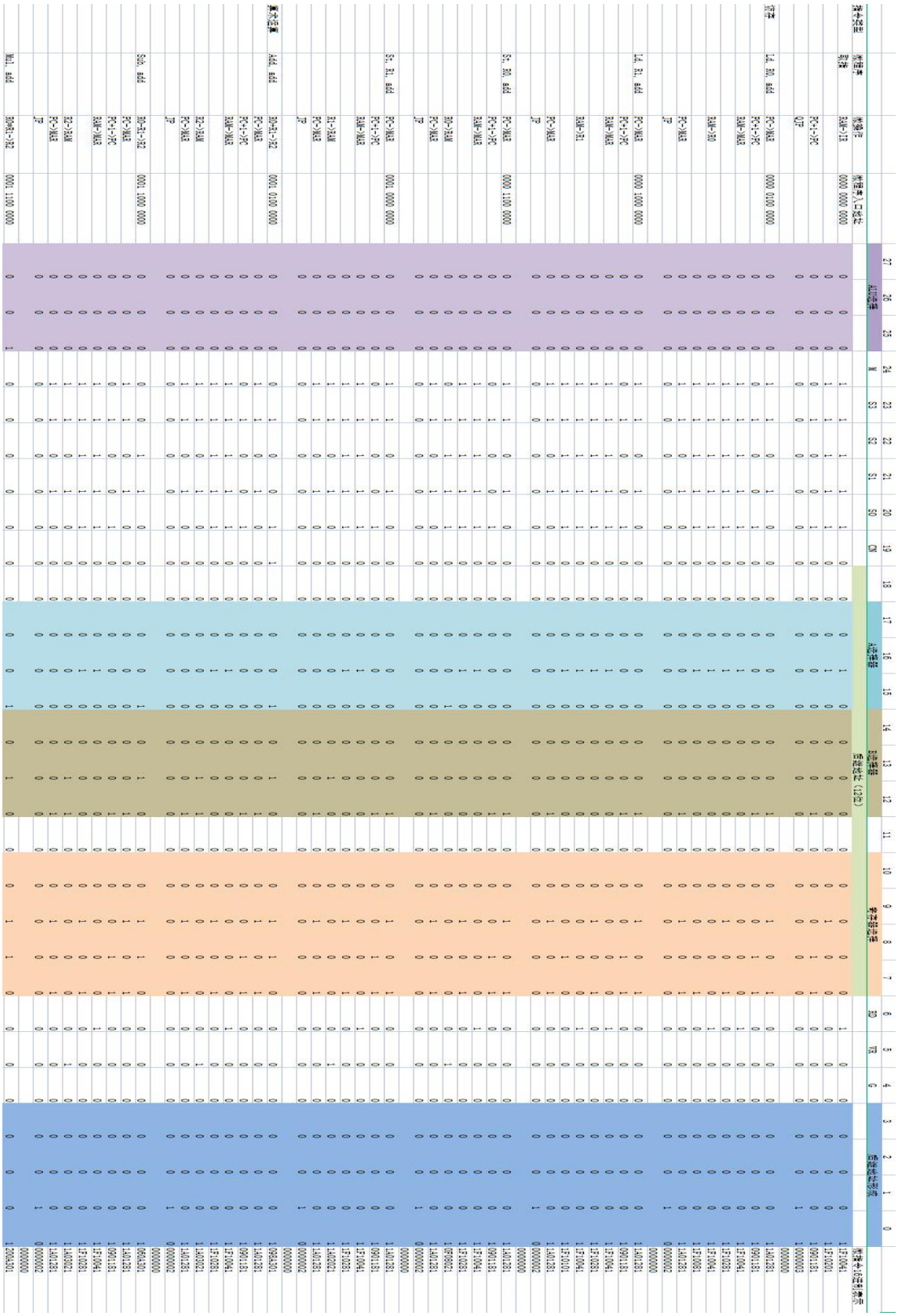
附录 2：

微程序流程图


000
RAM->IR
PC+1->PC
040
PC->MAR
RAM->R0
PC->MAR
JP
0C0
PC->MAR
RAM->R0
PC->MAR
JP
140
PC->MAR
R0+R1->R2
PC+1->PC
RAM->MAR
R2->RAM
PC->MAR
JP
400
PC->MAR
RAM->PC
PC->MAR
JP
440
PC->MAR
PC+1->PC
RAM->R3
PC->MAR
Z
R3->PC
PC->MAR
JP
5C0
PC->MAR
R2->R0
PC->MAR
JP
7C0
PC->MAR
PC+1->PC
RAM->R3
CX-1->CX
CX->R2
NZ
R3->PC
PC->MAR
JP
700
PC->MAR
PC+1->PC
RAM->R3
SP-1->SP
SP->MAR
PC->RAM
R3->PC
PC->MAR
JP
700
G=1

附录 3：

微程序



附录 4：

各控制信号的列表

指令的微操作和节拍安排

		MOV1	MOV2	ADD	MOV3	HALT	MUL	DIV
W1	T1				PC->MAR			
	T2				RAM(R)			
	T3				RAM->IR			
	T4				PC+1->PC			
	T5							
W2	T1	PC->MAR						
	T2	PC+1->PC	PC+1->PC	R0+R1->R1	PC->MAR	G=1	R0*R1->R1	R0/R1->R1
	T3	RAM(R)	RAM(R)		PC+1->PC			
	T4	RAM->R0	RAM->R1		RAM->MAR			
	T5				R1->RAM			

微操作对应的控制信号

		MOV1	MOV2	ADD	MOV3	HALT	MUL	DIV
W1	T1				PB M S3 S1 CPMAR			
	T2				RD			
	T3	MA M S3 S2 S1 S0 CPIR						
	T4	PB S3 S0 CPPC						
	T5							
W2	T1	PB M S3 S1 CPMAR	PB M S3 S1 CPMAR	RA RB S3 S0 CN CPR1	PB M S3 S1 CPMAR	G	RA RB DYCO CPR1	RA RB DYC1 CPR1
	T2	PB S3 S0 CPPC	PB S3 S0 CPPC		PB S3 S0 CPPC			
	T3	RD	RD		RD			
	T4	MA M S3 S2 S1 S0 CPR0	MA M S3 S2 S1 S0 CPR1		MA M S3 S2 S1 S0 CPMAR			
	T5				RB M S3 S1 WR			

附录 5：

各控制信号的逻辑表达式

控制信号对应的逻辑表达式
MA	W1*T3+W2*T4(MOV1+MOV2+MOV3)
RA	W2*T1*(ADD+MUL+DIV)
PB	W1*(T1+T4)+W2*(T1+T2)*(MOV1+MOV2+MOV3)
RB	W2*T1*(ADD+MUL+DIV)+W2*T5*MOV3
CPR0	W2*T4*MOV1
CPR1	W2*T1*(ADD+MUL+DIV)+W2*T4*MOV2
CPPC	W1*T4+W2*T2*(MOV1+MOV2+MOV3)
CPIR	W1*T3
CPMAR	W1*T1+W2*T1*(MOV1+MOV2+MOV3)+W2*T4*MOV3
RD	W1*T2+W2*T3*(MOV1+MOV2+MOV3)
WR	W2*T5*MOV3
G	W2*T1*HALT
M	W1*(T1+T3)+W2*((T1+T4)*MOV1+(T1+T4)*MOV2+(T1+T4+T5)*MOV3))
S3	1
S2	W1*T3+W2*T4*(MOV1+MOV2+MOV3)
S1	W1*(T1+T3)+W2*(T1+T4)*(MOV1+MOV2+MOV3)+W2*T5*MOV3
S0	W1*(T3+T4)+W2*(T2+T4)*(MOV1+MOV2+MOV3)+W2*T1*ADD
CN	W2*T1*ADD
DYC0	W2*T1*MUL
DYC1	W2*T1*DIV