《计算机组成与设计》课程

设计报告

课程设计题目 基于微程序控制的模型机设计

姓　　名 _____

学　　号 _____

专　　业 _____

年　　级 _____

2019.6

目录

一. 指令组设计与介绍 返回目录

1.1 指令组总体设计与创新 返回目录

本模型机设计指令字长为 16 位，采用定长指令码格式。实现直接寻址、寄存器寻址、寄存器间接寻址和 PC 相对寻址四种寻址方式。指令功能包括加减乘除，逻辑运算，移位运算和跳转指令，load/store，寄存器之间 mov 等，基本实现一个简易计算器的功能。

在原有课程设计介绍的指令设计样例基础上，本设计对指令格式做了以下修改，改进创新如下：

a. 扩充指令字长 16 位，操作码扩增为 5 位，扩大了指令数量和直接寻址的寻址空间；同时紧邻操作码后还增加一位识别码，用于选择不同的寻址方式，节省操作码的位数

b. 增加乘法，除法，与，或，非，异或，移位，比较跳转等指令，指令功能更加丰富

c. 寻址方式增加寄存器间接寻址和 PC 相对寻址两种

d. 所有运算指令都自动将运算结果保存在 RAM 中特定位置

1.2 指令格式与编码介绍 返回目录

以 load R0 指令为例，指令格式如下：

位数 15 ... 11　10　　9　　...　　0

00001　　X　　XX XXXX　XXXX

其中第 15-11 位为操作码；对于 load 和 store 指令，第 10 位为识别码，决定寻址方式；若选择直接寻址(第 10 位是 0)，后 10 位为直接寻址的操作数;若选择寄存器间接寻址（第 10 位是 1），则默认 load 或 store 到 R0 寄存器中；对于其他指令，第 10 位识别码决定选择哪一个寄存器来作为寄存器间接寻址。具体指令详细介绍请见 1.3。

1.3 指令详细介绍 返回目录

本设计实现的指令组如下：

指令组 1：控制指令(共 6 条)

指令	操作码	寻址方式	指令功能	指令编码
Halt	00000	无	停机指令	0000 0000 0000 0000
Load R0	00001	第 10 位= 0：直接寻址 地址码为后 10 位（指令编码中“X”部分）作为	将 RAM 中对应地址单元的数据	0000 10XX XXXX XXXX

		RAM 地址	load 到 R0 寄存器中	
		第 10 位=1：寄存器间接寻址 以 R0 寄存器为寻址寄存器		0000 1100 0000 0000
Store R0	00010	第 10 位= 0：直接寻址 地址码为后 10 位（指令编码中“X”部分）作为 RAM 地址	将 R0 寄存器上的数据 Store 到 RAM 对应地址单元中	0001 00XX XXXX XXXX
		第 10 位=1：寄存器间接寻址 以 R1 寄存器为寻址寄存器		0001 0100 0000 0000
Mov	00011	采用寄存器寻址，第 10 位= 0 表示 Mov 指令的第一个操作数在 R0	把 R0 寄存器的值 Move 到 R1	0001 1000 0000 0000
		采用寄存器寻址，第 10 位= 1 表示 Mov 指令的第一个操作数保在 R1	把 R1 寄存器的值 Move 到 R0	0001 1100 0000 0000
Jump	00100	采用寄存器寻址，第 10 位= 0 表示 Jump 指令跳转的地址保存在 R0	无条件跳转指令：程序跳转到指定寄存器保存的地址	0010 0000 0000 0000
		采用寄存器寻址，第 10 位= 1 表示 Jump 指令跳转的地址保存在 R1		0010 0100 0000 0000
If R0 EQL R1 then Jump	00101	采用 **PC 相对寻址**，地址偏移量为 10 位地址码（指令编码中“X”部分），偏移量+PC 值-> PC	条件跳转指令，如果 R0=R1，程序跳转到“PC+地址偏移量”的地址上；否则仍顺序执行下一条指令	0010 10XX XXXX XXXX

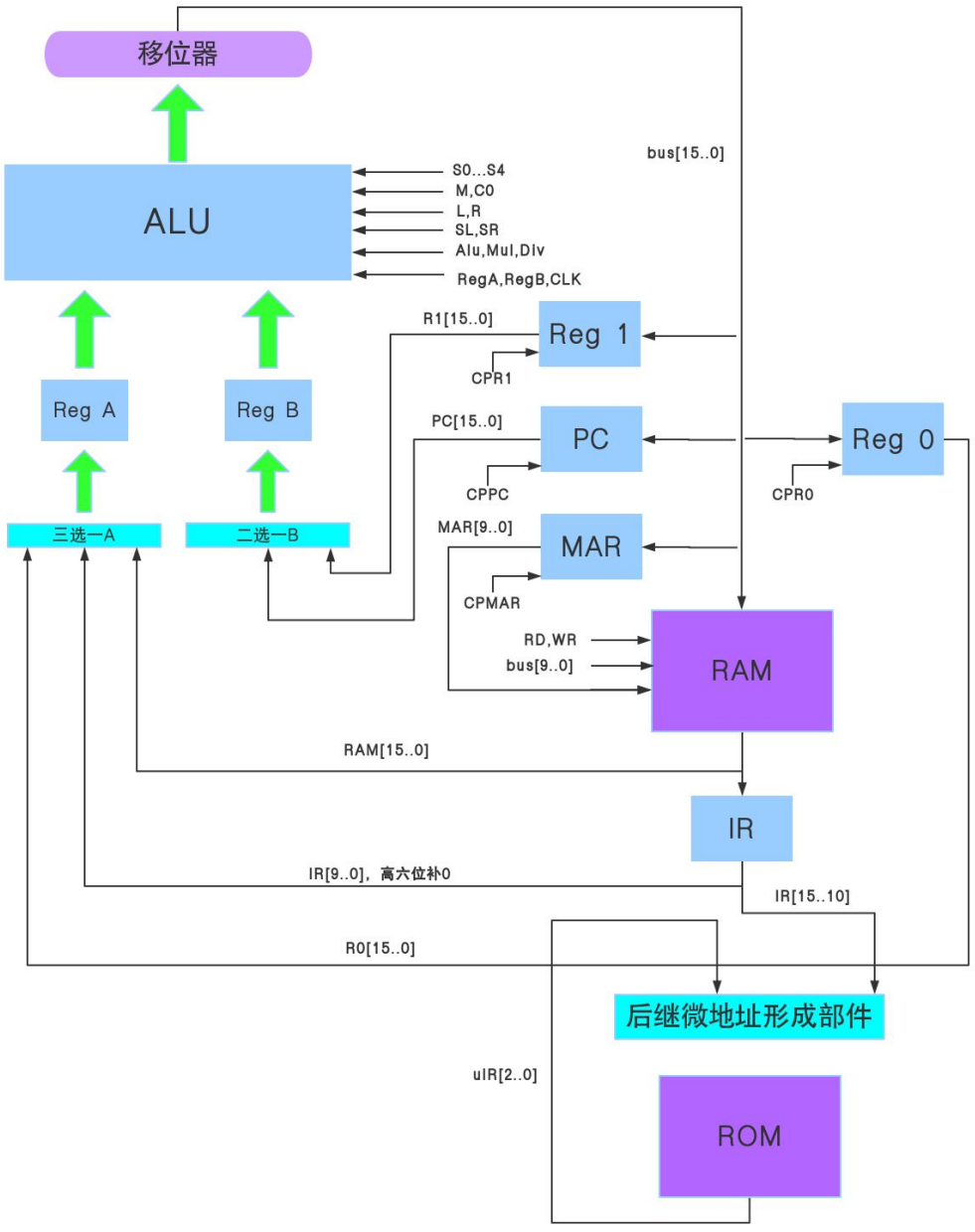
指令组 2: 运算指令（共 9 条）

注： 以下所有指令还要自动调用 Store R0 XX XXXX XXXX 微程序，把每一步的运算结果保存在相应内存单元中，寻址方式采用直接寻址，地址码 10 位。

指令	操作码	寻址方式	指令功能	指令编码
Add R0 R1	00110	寄存器寻址 直接寻址	16 位补码加法： R0 <- R0+R1	0011 00XX XXXX XXXX
Sub	00111	寄存器寻址 直接寻址	16 位补码减法： R0 <- R0-R1	0011 10XX XXXX XXXX
Mul	01000	寄存器寻址 直接寻址	8 位补码乘法：（乘法阵列器实现）： R0 <- R0*R1	0100 00XX XXXX XXXX
Div	01001	寄存器寻址 直接寻址	8 位补码除法：（除法阵列器实现）： R0 <- R0/R1	0100 10XX XXXX XXXX
And	01010	寄存器寻址 直接寻址	按位运算： R0 <- R0 and R1	0101 00XX XXXX XXXX
Or	01011	寄存器寻址 直接寻址	按位运算： R0 <- R0 or R1	0101 10XX XXXX XXXX
Not	01100	寄存器寻址:第 10 位= 0 表示操作数存在 R0 寄存器； 直接寻址	按位运算： R0 <- not R0	0110 00XX XXXX XXXX
		寄存器寻址:第 10 位= 1 表示操作数存在 R1 寄存器； 直接寻址	按位运算： R1 <- not R1	0110 01XX XXXX XXXX
Xor	01101	寄存器寻址 直接寻址	按位运算： R0 <- R0 xor R1	0110 10XX XXXX XXXX
SAL/SAR	01110	寄存器寻址 直接寻址	逻辑右移：将 R0 右移 1 位，高位根据正负对应补 0/1	0111 01XX XXXX XXXX
		寄存器寻址 直接寻址	逻辑左移：将 R0 左移 1 位，低位补 0	0111 00XX XXXX XXXX

二．总体结构与数据通路框图 返回目录

总体结构与数据通路框图见下图：



数据通路框图

其中重要组件及周边部件构成的模块如下：

2.1 ALU 模块 返回目录

功能：能够对数据进行算术运算（加减乘除），逻辑运算（与或非异或判断相等），移位运算

特殊作用：1.能够对数据直传，实现各个寄存器之间数据的流动（例如 PC->MAR，R0 -> R1 的移动等）

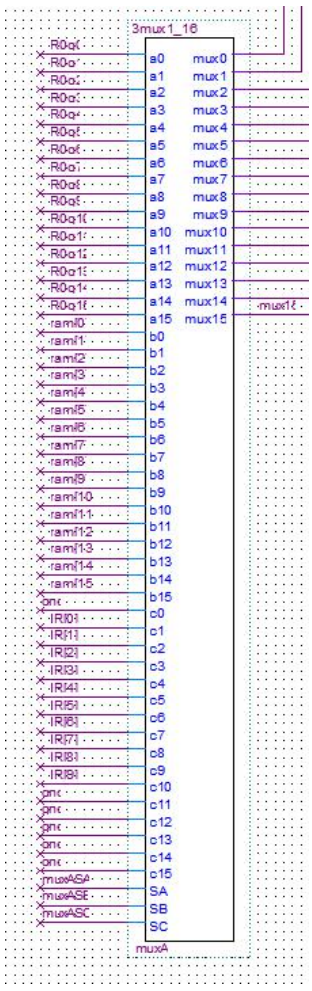
2. 能够进行地址运算，包括：“PC+1”，条件跳转时的地址偏移运算，直接寻址时的低位补零等

3. ALU 的两个寄存器 A,B 可以实现对运算数据暂存的作用

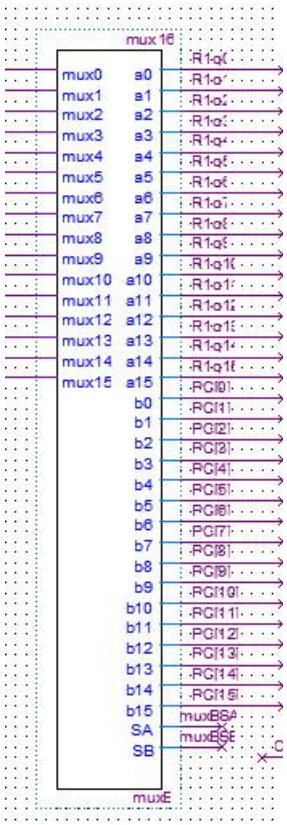
4. ALU 中分为三大块：算术逻辑运算，阵列乘法器，阵列除法器，最后通过移位和一个三选一门决定输出哪一个

实际设计原理图：

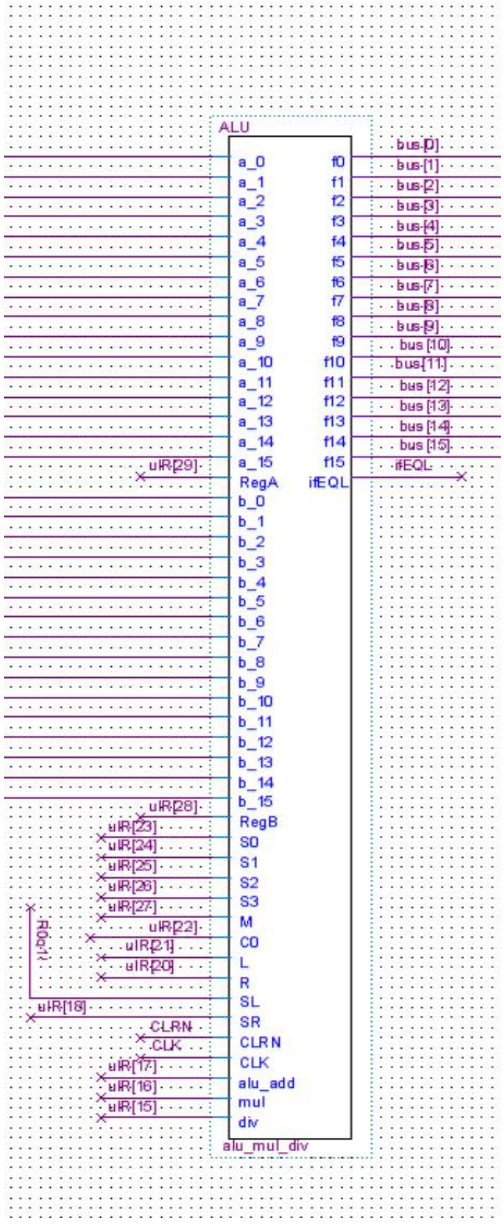
选择器 A:



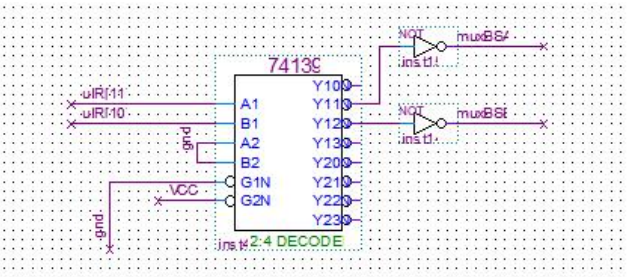
选择器 B：



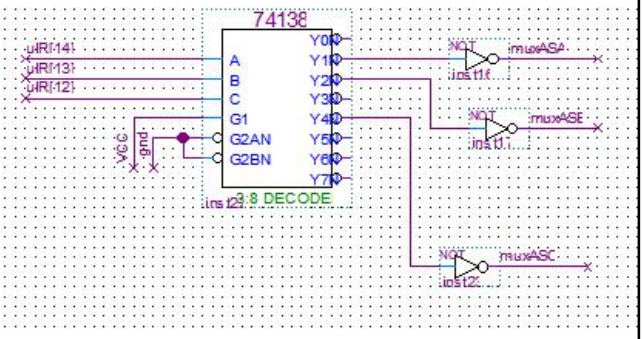
ALU 输入输出管脚图（在实际设计中此图向左旋转 90 度）



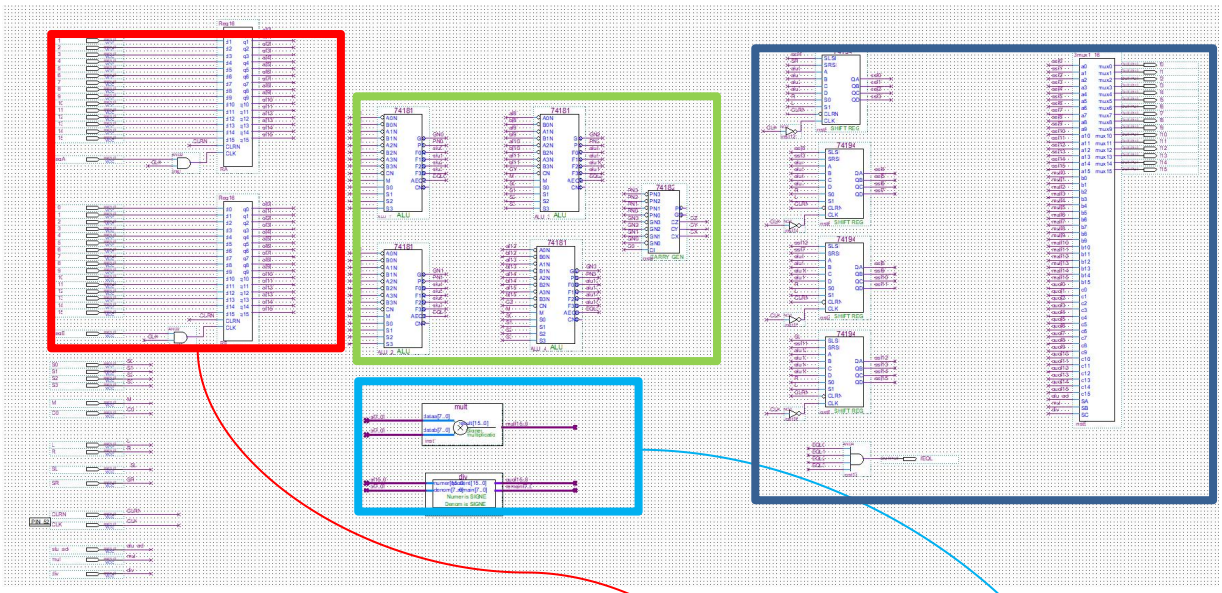
2-4 译码器控制 B 选择器

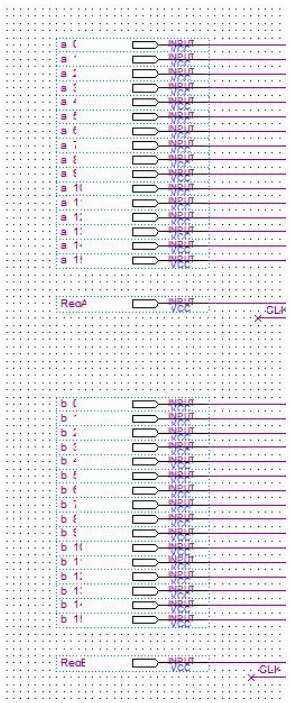


3-8 译码器控制 A 选择器



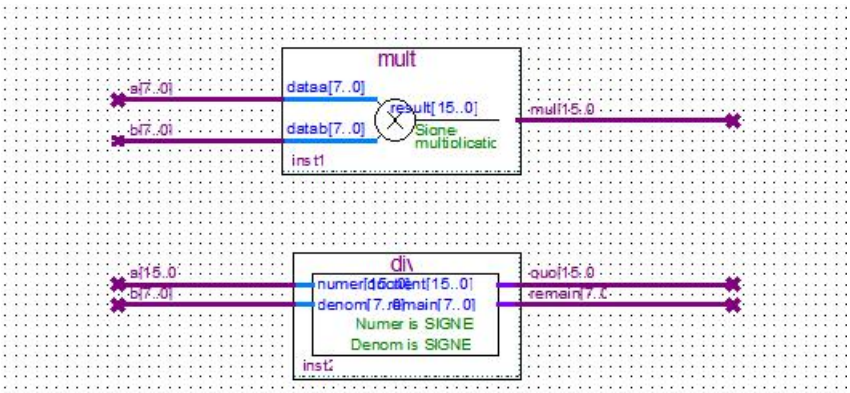
ALU 内部器件图



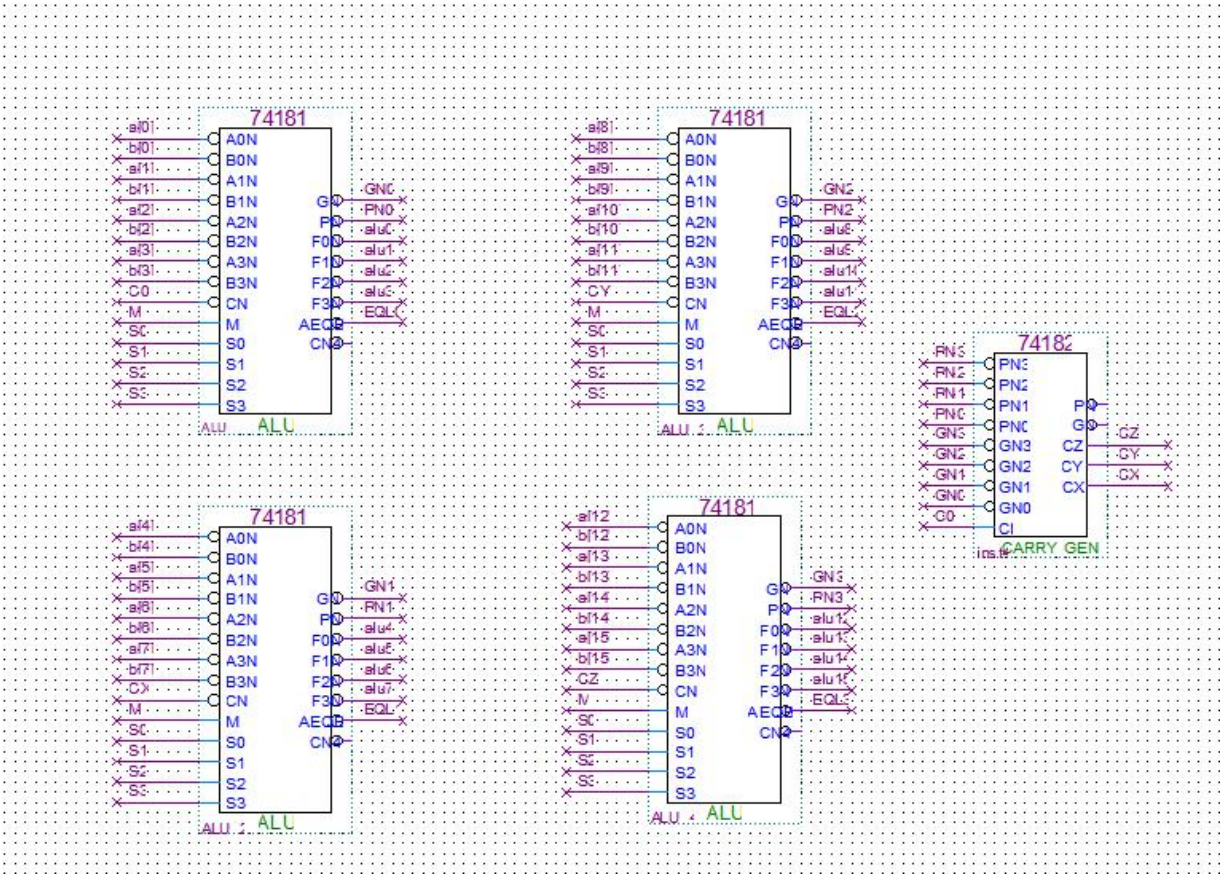


RegA, RegB 寄存器

乘法除法阵列器



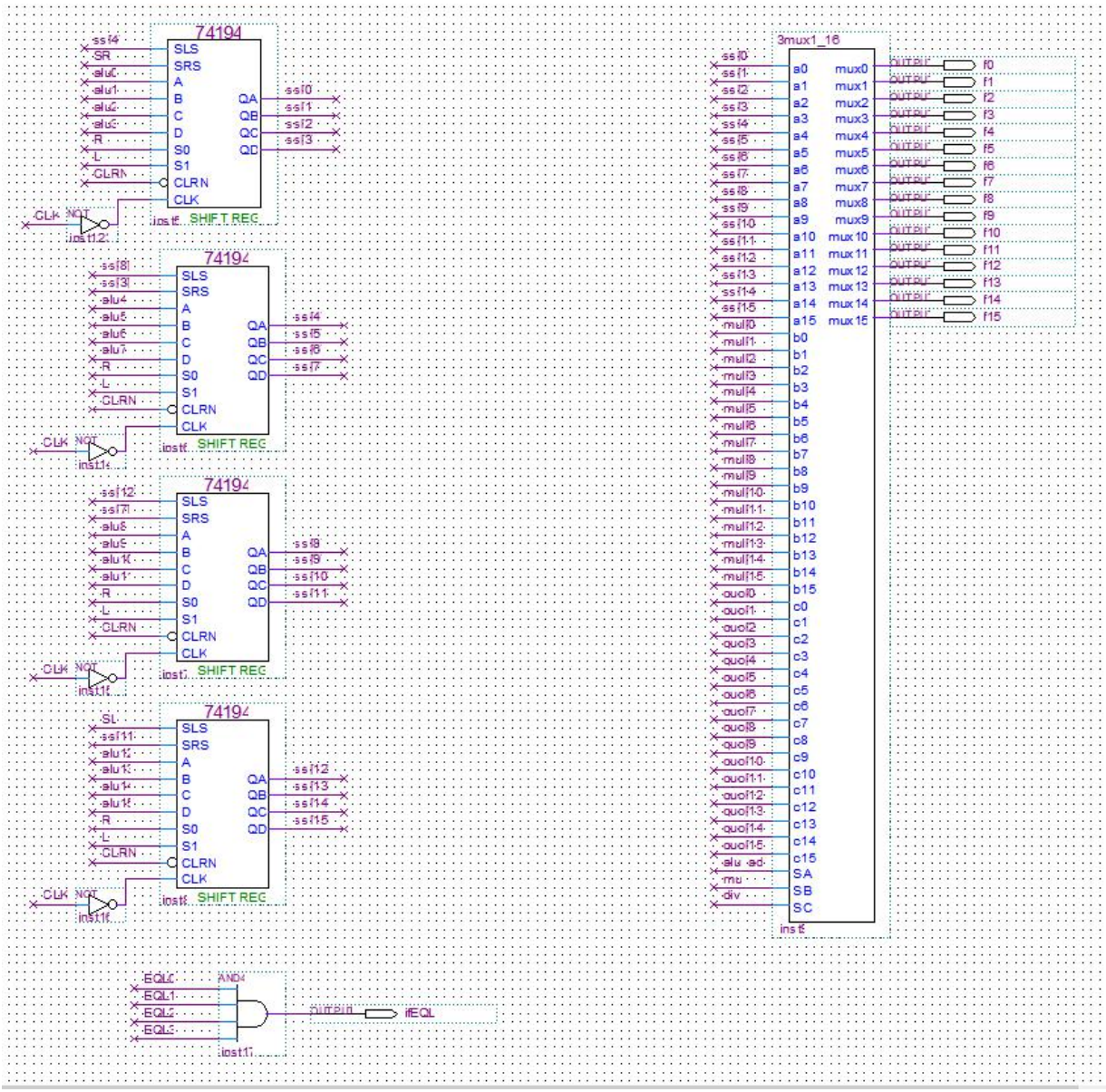
74181 超前加法进位器



注：由四片 74181 和 1 片 74182 构成的 16 为超前加法进位器（可以做加减和各种逻辑运算）

首先根据微指令，在时钟周期上升沿将数据传输并锁存到 RegA 或 RegB 中，同时把微指令中 ALU 的控制信号传输到 ALU 的管脚上。控制信号中包含一个三选一的选择器：选择 74181 芯片，选择乘法阵列器，选择除法阵列器。**若选择 74181**，则寄存器 A,B 里的数据在 74181 的控制信号下完成相应运算，其结果在时钟周期下降沿锁存到移位寄存器中（移位器实际设计图见下一页）。再根据控制信号判断是左移一位，右移一位还是直传。最后，移位寄存器中的数据通过三选一选择器被加载到总线上。**若选择乘法或除法阵列器**，寄存器 A,B 里的数据通过阵列器运算后，直接通过三选一选择器被加载到总线上。

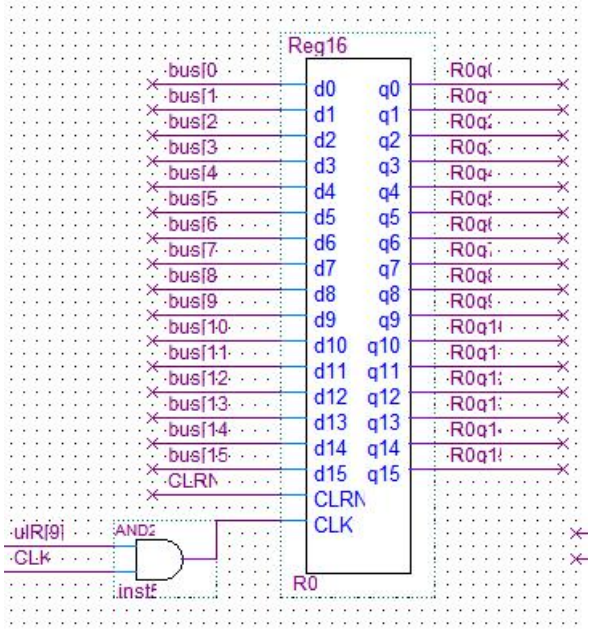
74194 移位器


74194
SLS
SRS
A
B
C
D
S0
S1
CLRN
CLK
QA
QB
QC
QD
SHIFT REG
NOT
3mux1_16
a0
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
a11
a12
a13
a14
a15
mux0
mux1
mux2
mux3
mux4
mux5
mux6
mux7
mux8
mux9
mux10
mux11
mux12
mux13
mux14
mux15
b0
b1
b2
b3
b4
b5
b6
b7
b8
b9
b10
b11
b12
b13
b14
b15
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
c11
c12
c13
c14
c15
SA
SB
SC
AND4
OUTPUT
f0
f1
f2
f3
f4
f5
f6
f7
f8
f9
f10
f11
f12
f13
f14
f15
fEQL

2.2 寄存器模块 返回目录

功能：暂存 ALU 的中间运算结果，其中 R0,R1 为通用寄存器，PC,MAR,IR,ZF 为四个专用寄存器。每个寄存器结构与下图相似，

原理设计图：以 R0 寄存器原理设计图为例——



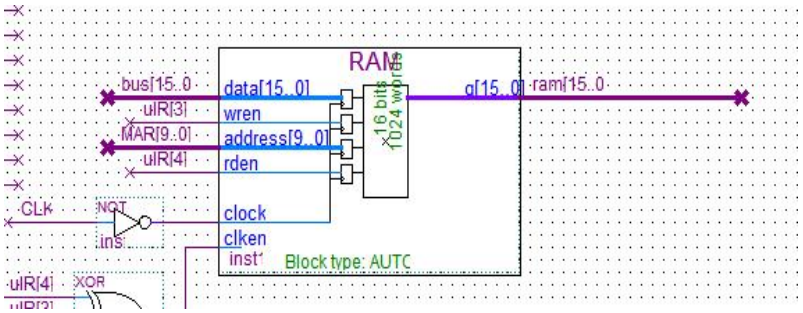
每个寄存器都是 Reg 16（16 为寄存器）。

由于 CPU 内部采用总线连接的方式，因此 R0,R1,PC 标志寄存器直接连接到片内总线上，传输的数据经过 ALU 直传后进入这三个寄存器。为简化数据传输，MAR 寄存器输入为 PC 的输出，MAR 的输出连接 ROM 的数据线，而不是把输入输出都挂在总线上。同理，IR 寄存器的输入为 RAM 的输出；IR 输出中的第 15-10 位作为后继微地址形成部件的输入，IR 输出的第 9-0 位（地址码）连接到 ALU 中三选一选择器 A 的输入。

RAM 与 ROM 模块：

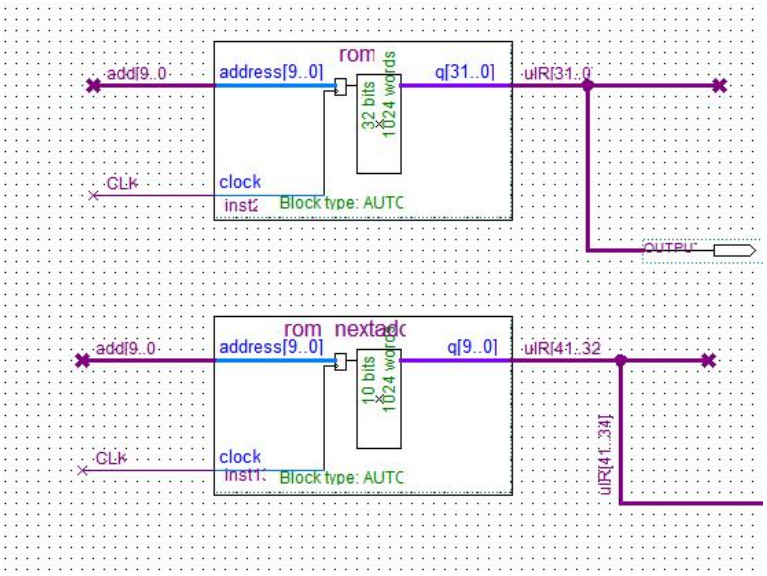
功能：RAM 存储用户编的汇编程序和数据，可读写；而 ROM 存储的是微程序控制信号的执行顺序，只读；

RAM 容量： 存储字长为 16 位, 10 位地址，容量为 1024(2^{10})* 16 = 12768 bits，



RAM

容量： 存储字长是 32 位，10 位地址，1024(2^{10})*32 = 32768bits,



ROM

ROM 在实际实现时做了一点改动，将十位后继微地址从 uIR 控制信号位中单独拿出来，存在扩展的 ROM（rom_nextadd）中。

三．微程序实现的控制部件 CU 返回目录

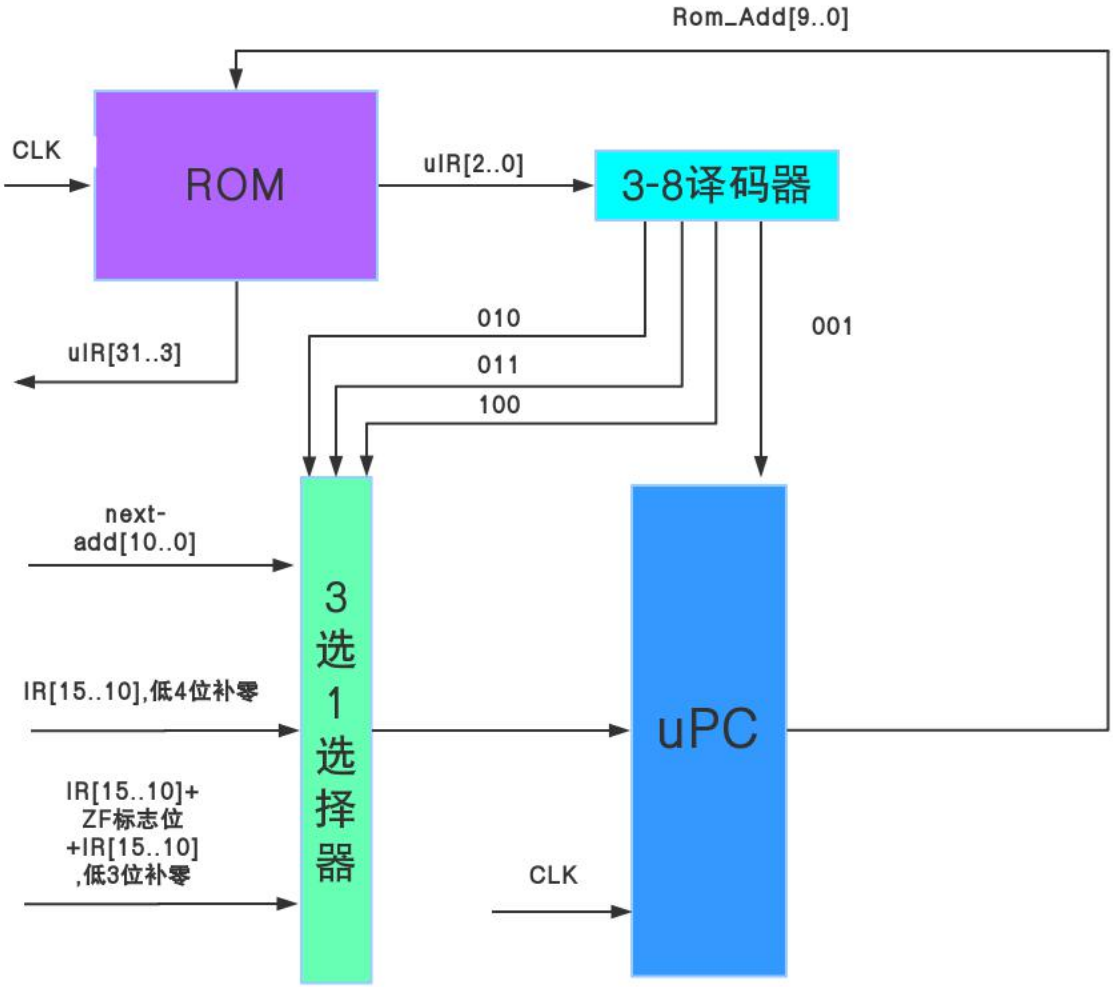
3.1 微地址形成电路各器件介绍 返回目录

译码器件（3-8 译码器）：

功能：通过将 uIR 低三位译码，确定微程序后继微地址的形成方式。低三位为“010”表示直接跳转下地址寄存器（rom_nextadd）中的地址(JP)； 低三位为“011”表示根据指令五位操作码和第六位识别码确定跳转的地址(QJP)；低三位为“100”表示根据指令五位操作码，ZF 标志寄存器确定跳转的地址（ZFQJP）。其中 ZFQJP 的跳转方式是为了实现条件跳转指令 If R0 EQL R1 then Jump 而设计的。在 ZFQJP 模式下，高五位为操作码，第六位强制为 1，第七位为 ZF 寄存器，后三位补 0 来形成后继微地址。而在 QJP 模式下，高五位为操作码，第六位为识别码，低四位补 0 来形成后继微地址。JP, QJP, ZFQJP 三个控制信号接到三选一选

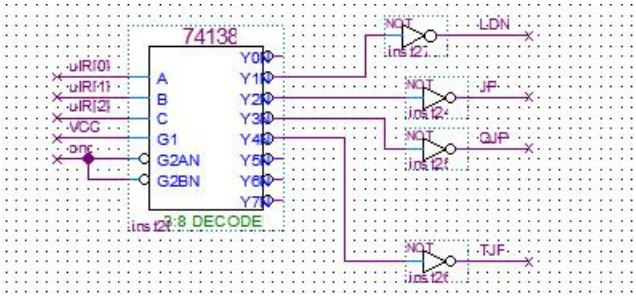
择器上，用来选择后继地址形成方式。三选一选择器的输出接到 uPC 累加器上，而 uPC 的输出管脚接到 ROM 的地址总线上。若译码电路为 JP, QJP, ZFQJP 中的任何一个时，三选一选择器根据控制信号选择一种后继微地址传到 uPC，uPC 直传到 ROM 地址总线上。

最后，若低三位为“001”时，表示 uPC+1,即顺序执行下一条微指令。此时与三选一选择器无关，只是 uPC 累加器中锁存的地址值加一后，传到 ROM 地址总线上

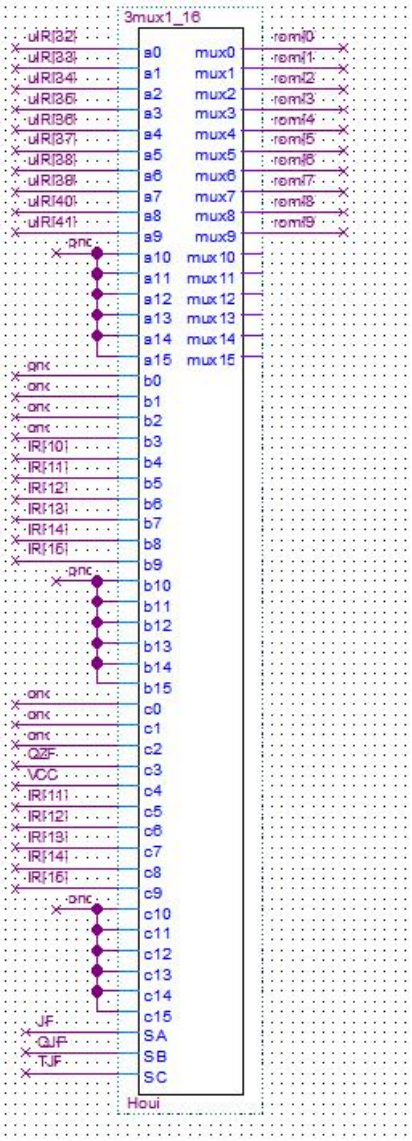


微程序实现的 CU 框图

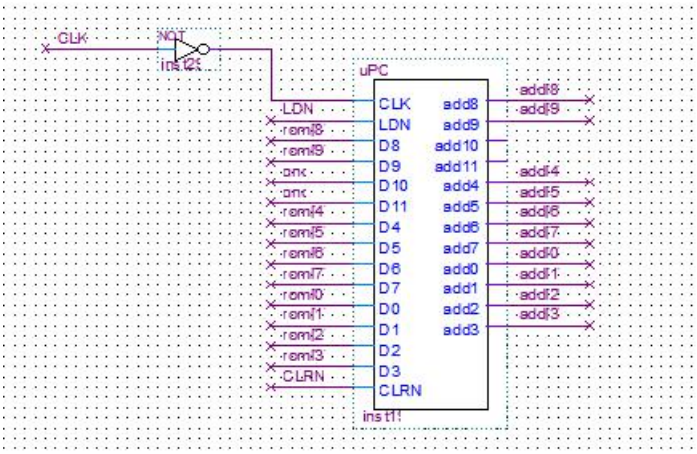
实际设计图如下：



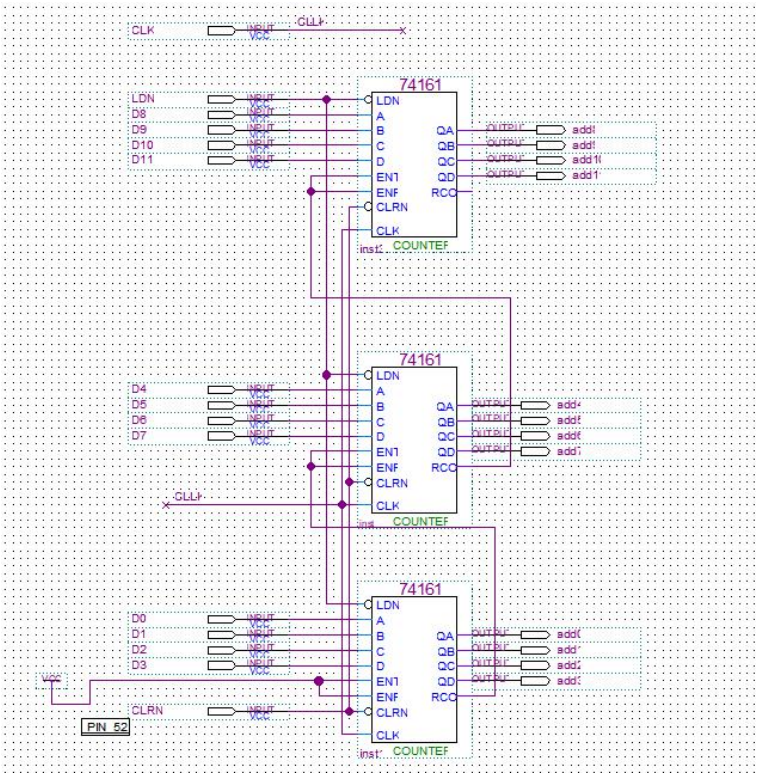
3-8 译码器



3 选 1 选择器



uPC



uPC 内部结构

uPC 器件：

功能：uPC 由三片 74161 累加器构成那个 12 位累加器，实现微程序地址不断加 1 功能

实际设计图：见上

3 选 1 选择器：

功能:从 JP,QJP,ZFQJP 三种后继微地址生成方式中选择一种锁存到 uPC

实际设计图：见上

3.2 微地址指令执行流程设计 返回目录

取址周期

PC -> ALU 传出
ALU 传出 -> MAR，RAM RD=1
RAM->IR，PC+1 -> ALU 传出
ALU 传出 -> PC
QJP

执行周期

控制指令

注： JP 00000 00001 是转移到取值周期的第一条语句

指令	操作码	寻址方式	指令功能	指令编码	微操作
Halt	00000	无	停机指令	0000 0000 0000 0000	0 -> G
Load R0	00001	第 10 位= 0：直接寻址 地址码为后 10 位（指令编码中“X”部分）作为 RAM 地址	将 RAM 中对应地址单元的数据 load 到 R0 寄存器中	0000 10XX XXXX XXXX	IR->ALU 传出 ALU 传出->MAR,RAM RD=1 RAM->ALU 传出 ALU 传出 -> R0 JP 00000 00001
		第 10 位=1：寄存器间接寻址 以 R0 寄存器为寻址寄存器		0000 1100 0000 0000	R0->ALU 传出 ALU 传出->MAR，RAM RD=1 RAM->ALU 传出 ALU 传出 -> R0 JP 00000 00001
Store R0	00010	第 10 位= 0：直接寻址 地址码为后 10 位（指令编码中“X”部分）作为 RAM 地址	将 R0 寄存器上的数据 Store 到 RAM 对应地址单元中	0001 00XX XXXX XXXX	IR->ALU 传出 ALU 传出->MAR R0->ALU 传出 RAM WR=1 空一个时钟周期 JP 00000 00001
		第 10 位=1：寄存器间接寻址 以 R1 寄存器为寻址寄存器		0001 0100 0000 0000	R1->ALU 传出 ALU 传出->MAR R0->ALU 传出 RAM WR=1 空一个时钟周期

					JP 00000 00001
Mov	00011	采用寄存器寻址，第 10 位= 0 表示 Mov 指令的第一个操作数在 R0	把 R0 寄存器的值 Move 到 R1	0001 1000 0000 0000	R0->ALU 传出 ALU 传出->R1 JP 00000 00001
		采用寄存器寻址，第 10 位= 1 表示 Mov 指令的第一个操作数在 R1	把 R1 寄存器的值 Move 到 R0	0001 1100 0000 0000	R1->ALU 传出 ALU 传出->R0 JP 00000 00001
Jump	00100	采用寄存器寻址，第 10 位= 0 表示 Jump 指令跳转的地址保存在 R0	无条件跳转指令：程序跳转到指定寄存器保存的地址	0010 0000 0000 0000	R0->ALU 传出 ALU 传出->PC JP 00000 00001
		采用寄存器寻址，第 10 位= 1 表示 Jump 指令跳转的地址保存在 R1		0010 0100 0000 0000	R1->ALU 传出 ALU 传出->PC JP 00000 00001
If R0 EQL R1 then Jump	00101	采用 **PC 相对寻址**，地址偏移量为 10 位地址码（指令编码中“X”部分），偏移量+PC 值-> PC	条件跳转指令，如果 R0=R1，程序跳转到“PC+地址偏移量”的地址上；否则仍顺序执行下一条指令	0010 10XX XXXX XXXX	R0->RegA, R1->RegB, ALU IfEQL 输出位->ZF 标志寄存器锁存 ZFQJP 转移 **（当 ZF = 0 时）** 0->RegA;0->RegB JP 00000 00001 **（当 ZF = 1 时）** IR->RegA, PC->RegB, IR+PC->ALU 传出 ALU 传出 -> PC JP 00000 00001

运算指令

注： 以下所有指令还要自动调用 Store R0 XX XXXX XXXX 微程序，把每一步的运算结果保存在相应

内存单元中，寻址方式采用直接寻址，地址码 10 位。微操作中 JP 00010 00000 便是跳转到 Store R0 XX XXXX XXXX 的第一条微指令

指令	操作码	寻址方式	指令功能	指令编码	微操组
Add R0 R1	00110	寄存器寻址 直接寻址	16 位补码加法： R0 <- R0+R1	0011 00XX XXXX XXXX	R0->RegA,R1->RegB, R0+R1->ALU 传出 ALU 传出 -> R0 JP 00010 00000
Sub	00111	寄存器寻址 直接寻址	16 位补码减法： R0 <- R0-R1	0011 10XX XXXX XXXX	R0->RegA,R1->RegB, R0-R ALU 传出 -> R0 JP 00010 00000
Mul	01000	寄存器寻址 直接寻址	8 位补码乘法：（乘法阵列器实现）： R0 <- R0*R1	0100 00XX XXXX XXXX	R0->RegA,R1->RegB, R0*R1->MUL 传出 MUL 传出 -> R0 JP 00010 00000
Div	01001	寄存器寻址 直接寻址	8 位补码除法：（除法阵列器实现）： R0 <- R0/R1	0100 10XX XXXX XXXX	R0->RegA,R1->RegB, R0/R1->DIV 传出 DIV 传出 -> R0 JP 00010 00000
And	01010	寄存器寻址 直接寻址	按位运算： R0 <- R0 and R1	0101 00XX XXXX XXXX	R0->RegA,R1->RegB, R0 & R1->ALU 传出 ALU 传出 -> R0 JP 00010 00000
Or	01011	寄存器寻址 直接寻址	按位运算： R0 <- R0 or R1	0101 10XX XXXX XXXX	R0->RegA,R1->RegB, R0 or R1->ALU 传出 ALU 传出 -> R0 JP 00010 00000
Not	01100	寄存器寻址: 第 10 位= 0 表示操作数存在 R0 寄存器； 直接寻址	按位运算： R0 <- not R0	0110 00XX XXXX XXXX	R0->RegA, Not R0->ALU 传出 ALU 传出 -> R0 JP 00010 00000
		寄存器寻址: 第 10 位= 1 表示操作数存在 R1 寄存器； 直接寻址	按位运算： R1 <- not R1	0110 01XX XXXX XXXX	R1->RegB, Not R1->ALU 传出 ALU 传出 -> R0 JP 00010 00000
Xor	01101	寄存器寻址 直接寻址	按位运算： R0 <- R0 xor R1	0110 10XX XXXX XXXX	R0->RegA,R1->RegB, R0 or R1->ALU 传出 ALU 传出 -> R0 JP 00010 00000

SAL/SAR	01110	寄存器寻址 直接寻址	逻辑右移：将 R0 右移 1 位，高位根据正负对应补 0/1	0111 01XX XXXX XXXX	R0->RegA，再锁存移位器 左移一位 ALU 传出 -> R0 JP 00010 00000
		寄存器寻址 直接寻址	逻辑左移：将 R0 左移 1 位，低位补 0	0111 00XX XXXX XXXX	R0->RegA，再锁存移位器 R0 右移一位 ALU 传出 -> R0 JP 00010 00000

3.3 微指令格式与编码 返回目录

微指令格式：

31	30	29	28	27	26	25	24	23	22	21	20
G		CP RegA	CP RegB	M	S3	S2	S1	S0	C0	L	R
G=0 停机		ALU 寄存器锁存信号		74181 控制位						控制左移，右移，直传	
19	18	17	16	15	14	13	12	11	10	9	8
SL	SR	CP ALU	CP MUL	CP DIV	R0	RAM	IR	R1	PC	CPR0	CPR1
移位补充 0/1		算逻,乘法,除法运算 3 选 1			A 选择器 3 选 1			B 选择器 2 选 1		片内寄存器	
7	6	5	4	3	2	1	0				
CPPC	CPIR	CPMAR	RD	WR	t2	t1	t0				
锁存信号			RAM 读/写		后继微地址						

微指令编码，地址与下地址：

微操作	微地址（10 位）	十六进制代码	二进制代码（u31-0）
取指周期：			
	00000 00000	8000 0001	1000 0000 0000 0000
PC->ALU 传出	00000 00001	9D32 0401	1001 1101 0011 0010
ALU 传出->MAR，RAM 读	00000 00010	8002 0031	1000 0000 0000 0010
RAM->IR，PC+1->ALU 传出	00000 00011	b4b2 0441	1011 0100 1011 0010
ALU 传出 -> PC	00000 00100	8002 0081	1000 0000 0000 0010
QJP	00000 00101	8000 0003	1000 0000 0000 0000
		后继地址	QJP
Load+寻址方式			
Load 0 XX XXXX XXXX	32 开始		
IR-> ALU 传出	00001 00000	afb2 1001	1010 1111 1011 0010
ALU 传出->MAR，RAM 读	00001 00001	8002 0031	1000 0000 0000 0010

RAM->ALU 传出	00001 00010	afb2 2001	1010 1111 1011 0010
ALU 传出 -> R0	00001 00011	8002 0201	1000 0000 0000 0010
JP	00001 00100	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Load 1 (R0->R0)	48 开始		
R0->ALU 传出	00001 10000	afb2 4001	1010 1111 1011 0010
ALU 传出 -> MAR，RAM 读	00001 10001	8002 0031	1000 0000 0000 0010
RAM->ALU 传出	00001 10010	afb2 2001	1010 1111 1011 0010
ALU 传出 -> R0	00001 10011	8002 0201	1000 0000 0000 0010
JP	00001 10100	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Store+寻址方式			
Store 0 XX XXXX XXXX	64 开始		
IR->ALU 传出	00010 00000	afb2 1001	1010 1111 1011 0010
ALU 传出->MAR	00010 00001	8002 0021	1000 0000 0000 0010
R0->ALU 传出	00010 00010	afb2 4001	1010 1111 1011 0010
Ram 写	00010 00011	8002 0009	1000 0000 0000 0010
空一个时钟周期	00010 00100	8000 0001	1000 0000 0000 0000
JP	00010 00101	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Store 1(以 R1 内容为内存地址)	80 开始		
R1->ALU 传出	00010 10000	9d32 0801	1001 1101 0011 0010
ALU 传出->MAR	00010 10001	8002 0021	1000 0000 0000 0010
R0->ALU 传出	00010 10010	afb2 4001	1010 1111 1011 0010
Ram 写	00010 10011	8002 0009	1000 0000 0000 0010
空一个时钟周期	00010 10100	8000 0001	1000 0000 0000 0000
JP	00010 10101	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Mov			
Mov R0 R1	96 开始		
R0->ALU 传出	00011 00000	afb2 4001	1010 1111 1011 0010
ALU 传出->R1	00011 00001	8002 0101	1000 0000 0000 0010
JP	00011 00010	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Mov R1 R0	112 开始		
R1->ALU 传出	00011 10000	9d32 0801	1001 1101 0011 0010

ALU 传出->R0	00011 10001	8002 0201	1000 0000 0000 0010
JP	00011 10010	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Jump R0	128 开始		
R0->ALU 传出	00100 00000	afb2 4001	1010 1111 1011 0010
ALU 传出->PC	00100 00001	8002 0081	1000 0000 0000 0010
JP	00100 00010	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
Jump R1	144 开始		
R1->ALU 传出	00100 10000	9d32 0801	1001 1101 0011 0010
ALU 传出->PC	00100 10001	8002 0081	1000 0000 0000 0010
JP	00100 10010	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
7. if R0 EQAL R1 then Jump PC+XXXXX XXXXX	160 开始		
R0->RegA,R1->RegB,IfEQL->ZF	00101 00000	f000 4804	1111 0000 0000 0000
ZF=0 (R0≠R1)			
	176		
0->RegA;0->RegB	00101 10000	b000 0001	1011 0000 0000 0000
JP	00101 10001	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
ZF=1 (R0=R1)	184		
IR->RegA,PC->RegB, IR+PC->ALU 传出	00101 11000	b4f2 1401	1011 0100 1111 0010
ALU 传出 -> PC	00101 11001	8002 0081	1000 0000 0000 0010
JP	00101 11010	8000 0002	1000 0000 0000 0000
		后继地址	00000 00001
8.ADD R0 R1 XXX	192 开始		
R0->RegA,R1->RegB, R0+R1->ALU 传出	00110 00000	b4f2 4801	1011 0100 1111 0010
ALU 传出 -> R0	00110 00001	8002 0201	1000 0000 0000 0010
JP Store R0	00110 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
9.SUB R0 R1	224 开始		
R0->RegA,R1->RegB, R0-R1->ALU 传出	00111 00000	b332 4801	1011 0011 0011 0010
ALU 传出 -> R0	00111 00001	8002 0201	1000 0000 0000 0010
JP Store R0	00111 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000

10.MUL R0 R1	256 开始		
R0->RegA,R1->RegB, R0*R1->MUL 传出	01000 00000	b001 4801	1011 0000 0000 0001
MUL 传出 -> R0	01000 00001	8001 0201	1000 0000 0000 0001
JP Store R0	01000 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
11.DIV R0 R1	288 开始		
R0->RegA,R1->RegB, R0/R1->DIV 传出	01001 00000	b000 c801	1011 0000 0000 0000
DIV 传出 -> R0	01001 00001	8000 8201	1000 0000 0000 0000
JP Store R0	01001 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
12.And R0 R1	320 开始		
R0->RegA,R1->RegB, R0 & R1->ALU 传出	01010 00000	bdb2 4801	1011 1101 1011 0010
ALU 传出 -> R0	01010 00001	8002 0201	1000 0000 0000 0010
JP Store R0	01010 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
13.Or R0 R1	352 开始		
R0->RegA,R1->RegB, R0 or R1->ALU 传出	01011 00000	bf32 4801	1011 1111 0011 0010
ALU 传出 -> R0	01011 00001	8002 0201	1000 0000 0000 0010
JP Store R0	01011 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
14.Not R0	384		
R0->RegA, Not R0->ALU 传出	01100 00000	a832 4001	1010 1000 0011 0010
ALU 传出 -> R0	01100 00001	8002 0201	1000 0000 0000 0010
JP Store R0	01100 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
14.Not R1	400		
R1->RegB, Not R1->ALU 传出	01100 10000	9ab2 0801	1001 1010 1011 0010
ALU 传出 -> R0	01100 10001	8002 0201	1000 0000 0000 0010
JP Store R0	01100 10010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
15.XOR R0 R1	416		
R0->RegA,R1->RegB, R0 or R1->ALU 传出	01101 00000	bb32 4801	1011 1011 0011 0010
ALU 传出 -> R0	01101 00001	8002 0201	1000 0000 0000 0010
JP Store R0	01101 00010	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000

16. SAR	448		
R0->RegA, R0 直传	01110 00000	afb2 4001	1010 1111 1011 0010
R0 左移一位	01110 00001	8028 0001	1000 0000 0010 1000
ALU 传出 -> R0	01110 00010	8002 0201	1000 0000 0000 0010
JP Store R0	01110 00011	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
16. SAL	464		
R0->RegA, R0 直传	01110 10000	afb2 4001	1010 1111 1011 0010
R0 右移一位	01110 10001	8010 0001	1000 0000 0001 0000
ALU 传出 -> R0	01110 10010	8002 0201	1000 0000 0000 0010
JP Store R0	01110 10011	8000 0002	1000 0000 0000 0000
		后继地址	00010 00000
17. OFF	01111 00000	0000 0000	0000 0000 0000 0000

3.4 RAM 汇编测试程序与对应机器代码 返回目录

编号	RAM 地址	汇编代码	地址码	机器代码
0.	00000	LOAD 0	00 0011 0010	0000 1000 0011 0010
1.	00001	MOV R0 ->R1		0001 1000 0000 0000
2.	00010	LOAD 0 R0	00 0011 0011	0000 1000 0011 0011
3.	00011	LOAD 1 R0		0000 1100 0000 0000
4.	00100	ADD R0 R1	00 0011 0100	0011 0000 0011 0100
5.	00101	SUB R0 R1	00 0011 0101	0011 1000 0011 0101
6.	00110	MUL R0 R1	00 0011 0110	0100 0000 0011 0110
7.	00111	DIV R0 R1	00 0011 0111	0100 1000 0011 0111
8.	01000	AND R0 R1	00 0011 1000	0101 0000 0011 1000
9.	01011	XOR R0 R1	00 0011 1011	0110 1000 0011 1011
10.	01001	OR R0 R1	00 0011 1001	0101 1000 0011 1001
11.	01010	NOT R1	00 0011 1010	0110 0100 0011 1010
12.	01100	SAL R0	00 0011 1100	0111 0100 0011 1100
13.	01101	SAR	00 0011 1101	0111 0000 0011 1101
14.	01110	LOAD 0 R0	00 0011 1110	0000 1000 0011 1110

15.	01111	JUMP R0		0010 0000 0000 0000
16.	10000	OFF		0110 1000 0000 0000
17.	10001			
18.	10010	Store 0 R0	00 0011 1111	0001 0000 0011 1111
19.	10011	Mov R0->R1		0001 1000 0000 0000
20.	10100	IF R0 EQL R1 JUMP 3	00 0000 0011	0010 1000 0000 0011
21.	10101	OFF		0110 1000 0000 0000
22.	10110			
23.	10111	Store 0 R0	00 0100 0000	0001 0000 0100 0000
24.	11000	OFF		0110 1000 0000 0000

内存空间情况：

000 0110 0100　　3
000 0110 0101
000 0110 0110　　0000 0000 0110 0111
000 0110 0111　　5
000 0110 1000　　保存 Add 结果 8
000 0110 1001
000 0110 1010　　保存 Sub 结果 5
000 0110 1011
000 0110 1100　　保存 Mul 结果 15
000 0110 1101
000 0110 1110　　保存 Div 结果 5
000 0110 1111
000 0111 0000　　保存 And 结果 1
000 0111 0001
000 0111 0010　　保存 or 结果 3
000 0111 0011
000 0111 0100　　保存 not -4（补码）
000 0111 0101
000 0111 0110　　保存 Xor 结果 -1
000 0111 0111
000 0111 1000　　保存 SAl 结果 -2
000 0111 1001
000 0111 1010　　保存 SAR 结果 -1
000 0111 1011
000 0111 1100　　0000 0000 0001 0010

000 0111 1101	
000 0111 1110	Store　结果 10010
000 0111 1111	
000 1000 0000	Store (10010)(当执行 ifEQL 跳转时)
000 1000 0001	
000 1000 0010	
000 1000 0011	