一．模型机实现的指令

1. 指令组

以下指令中的寄存器可以是任意一个通用寄存器。

MOV1 R1 R2	将 R2 中数据移到 R1 中
MOV2 # R1	将 R1 中数据移到#指向的地址
MOV3 R1 立即数	将立即数移到 R1 中
MOV4 S R1	将 R1 中的数据移到 S 指向的地址所指向的地址
MOV5 R1 #	将#指向地址中的内容移动到 R1 中
MOV6 R1 S	将 S 指向地址所指向的地址中的内容移动到 R1 中
JUMP1 立即数	无条件转移到立即数所表示的地址处
JUMP2 #	无条件转移到#指向的地址中的数据所表示的地址处
JUMP3 R1	无条件转移到 R1 中数据所表示的地址处
JUMP1= 立即数	相等条件转移到立即数表示的地址处
JUMP2= #	相等条件转移到#指向的地址中数据表示的地址处
JUMP3= R1	相等条件转移到 R1 中数据所表示的地址处
JUMP1> 立即数	大于条件转移到立即数表示的地址处
JUMP2> #	大于条件转移到#指向的地址中数据表示的地址处
JUMP3> R1	大于条件转移到 R1 中数据所表示的地址处
JUMP1< 立即数	小于条件转移到立即数表示的地址处
JUMP2< #	小于条件转移到#指向的地址中数据表示的地址处
JUMP3< R1	小于条件转移到 R1 中数据所表示的地址处
CLEAR	清空 PC 中的内容
ADD	加法
LOADD	逻辑加
SUB	减法
INTERSECT	逻辑交
XOR	异或
LL	A 左移
RR	A 右移
NOT	A 非
ADD1	A 加 1
SUB1	A 减 1
ADD2	A 加 B 加 1
MULTI	八位整数乘法
HALT	终止

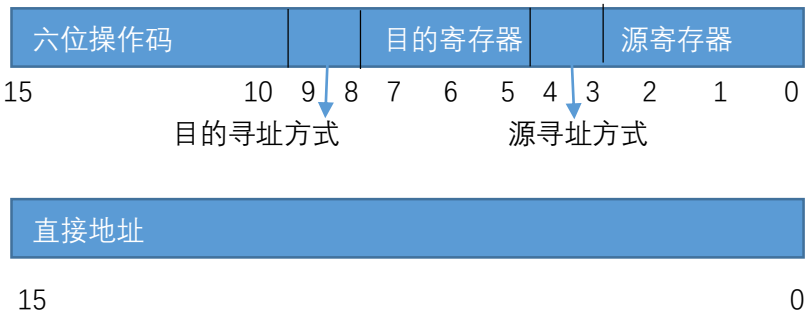
运算指令的计算结果默认存储在 R6 中。

2. 指令格式

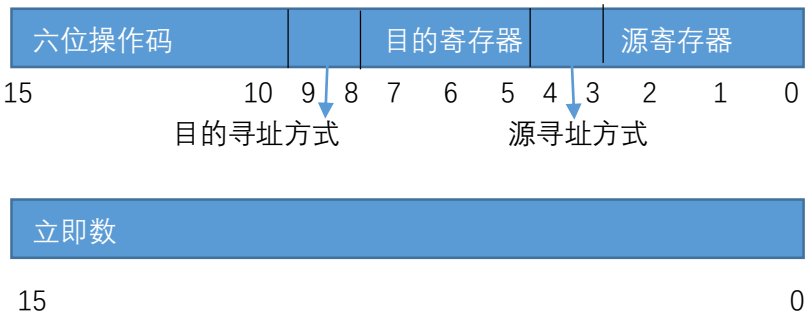
六位操作码		目的寄存器		源寄存器

高六位为操作码，共表示 32 个指令。对于 MOV 系列指令目的寻址方式和源寻址方式各三种：立即数，直接寻址，间接寻址，寄存器寻址；目的寻址方式和源寻址方式各用两位来表示可用的寻址方式。若为寄存器寻址，则用寻址方式的后三位表示选择哪个寄存器，共六个寄存器可供选择。为方便设计，不存在源寻址方式和目的寻址方式都需要访问内存的情况。若两个寻址方式都不访问内存，则上述指令即为完整指令。

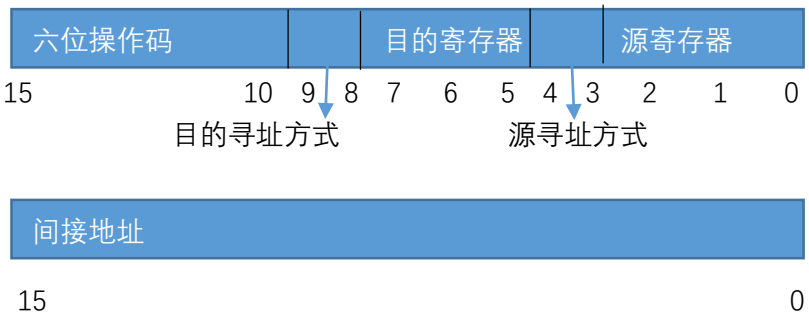
若为两个寻址方式中存在直接寻址，则地址放在后十六位中，完整指令格式如下：



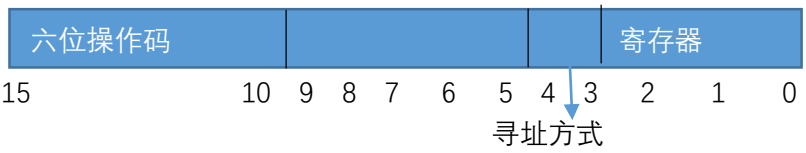
若两个寻址方式中存在立即数寻址，则地址放在第二个十六位中，完整指令格式如下：



若两个寻址方式中存在间接寻址，则间接地址放在第二个十六位中，完整指令格式如下：



对 JUMP 系列指令，指令格式为：



若采用非寄存器寻址则所需要的数据在之后 16 位上。

对运算系列指令和 CLEAR.HALT 指令，指令格式为：

运算指令默认数据已经放入寄存器 R1,R2 中，故而不需要寻址，只需要表示出操作码。

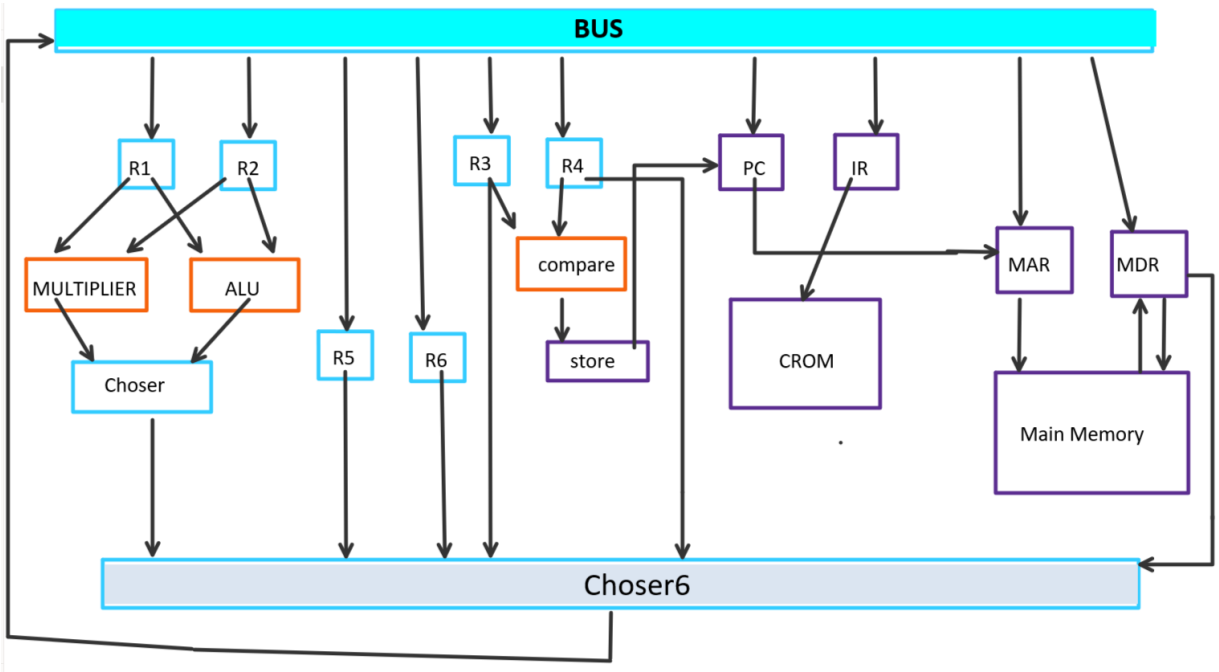
3. 每条指令对应的编码

指令	操作码	寻址方式
MOV1 R1 R2	000001	寄存器寻址
MOV2 # R1	000010	直接寻址，寄存器寻址
MOV3 R1 立即数	000011	寄存器寻址，立即数寻址
MOV4 S R1	000100	间接寻址，寄存器寻址
MOV5 R1 #	000101	寄存器寻址，直接寻址
MOV6 R1 S	000110	寄存器寻址，间接寻址
JUMP1 立即数	000111	立即数寻址
JUMP2 #	001000	直接寻址
JUMP3 R1	001001	寄存器寻址
JUMP1= 立即数	001010	立即数寻址
JUMP2= #	001011	直接寻址
JUMP3= R1	001100	寄存器寻址
JUMP1> 立即数	001101	立即数寻址
JUMP2> #	001110	直接寻址
JUMP3> R1	001111	寄存器寻址
JUMP1< 立即数	010000	立即数寻址
JUMP2< #	010001	直接寻址
JUMP3< R1	010010	寄存器寻址
CLEAR	010011	
ADD	010100	
LOADD	010101	
SUB	010110	
INTERSECT	010111	
XOR	011000	
LL	011001	

RR	011010
NOT	011011
ADD1	011100
SUB1	011101
ADD2	011110
MULTI	011111
HALT	100000

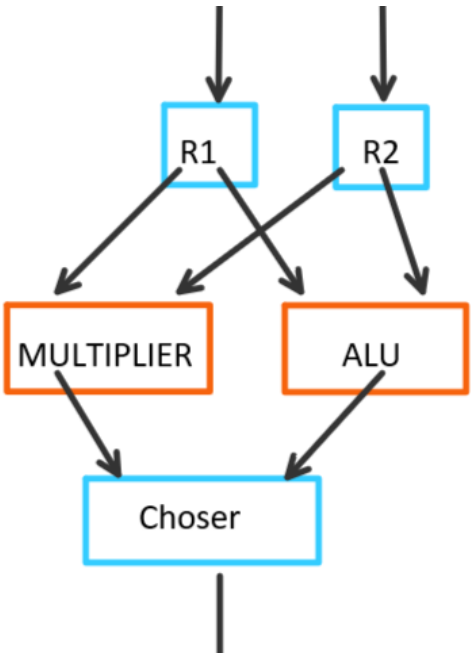
二．模型机总体结构

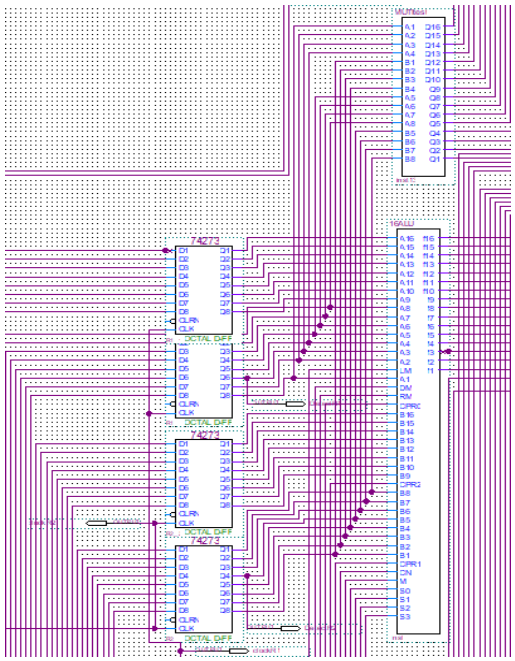
1. 模型机总体框图



该模型机采用总线结构，各个寄存器都从总线得到数据，寄存器的值和计算结果通过选择器“Choser6”返回总线。R1 与 R2 为 16 位寄存器用于存储计算相关的数据，“ALU”可进行 16 位 ALU 中相关计算，"MULTIPLIER"进行 8 位乘法运算。R3，R4，R5，R6 为 16 位通用寄存器，都可从总线得到数据，和把数据写回总线。其中 R3,R4 可以负责存储比较器“compare”中要比较的数据。"compare"比较的结果存入"store"中，"store"直接控制 PC 的置位端，用于支持跳转指令。PC 为 8 位寄存器，PC 的值可以由总线的低 8 位得到或通过累加得到。MAR 位 8 位寄存器，其值来源于总线低 8 位或者 PC，MAR 用作主存的地址寄存器。MDR 为 16 位数据寄存器，数据来源于总线或者主存，可将内容写入总线或者主存。IR 为 16 位指令寄存器，IR 中的数据传递给 CROM。

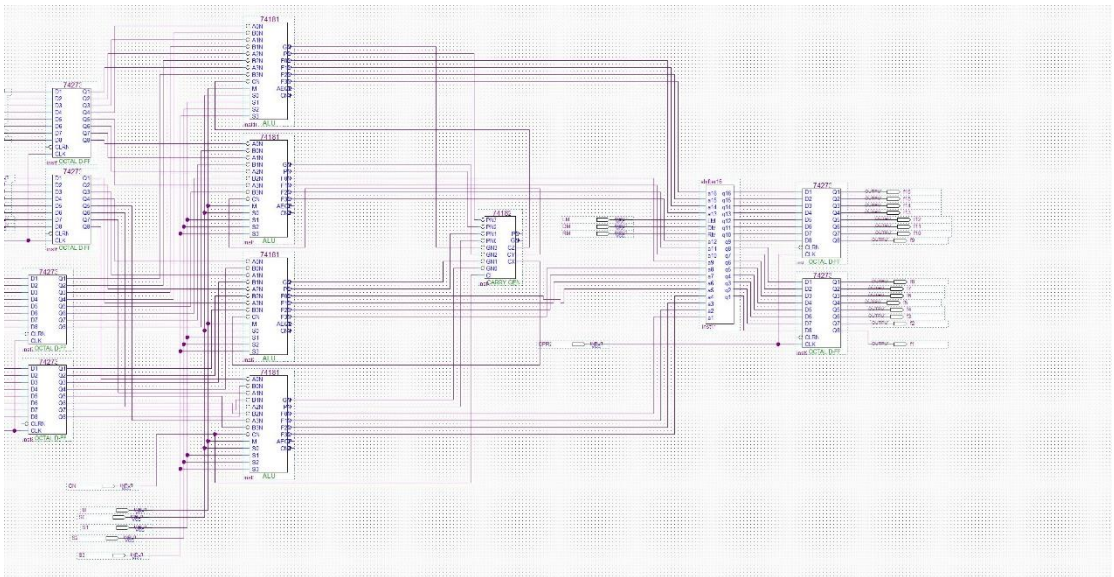
2. 负责计算的部分





左图为总体框图中负责进行计算的部分，右图为 R1，R2 与 ALU,MULTIPLIER。4 个 74273 作为两个 16 位寄存器 R1 与 R2。R1 与 R2 的输出会输入 16ALU，R1 与 R2 的低 8 位输出会输入 MUTItest(MULTIPLIER)。ALU 的输入为两个 16 位数据，LM,DM,RM 的移位选择，CPR0,CPR1,CPR2 的脉冲，CN,M,S0,S1,S2,S3 的控制端。输出为计算后的 16 位数据。MULTIPLIER 输入为两个 8 位数据，输出为一个 16 位数据。ALU 和 MUTItest(MULTIPLIER)的输出会进入 16choser(16 位选择器)决定二者中哪个输入总线。

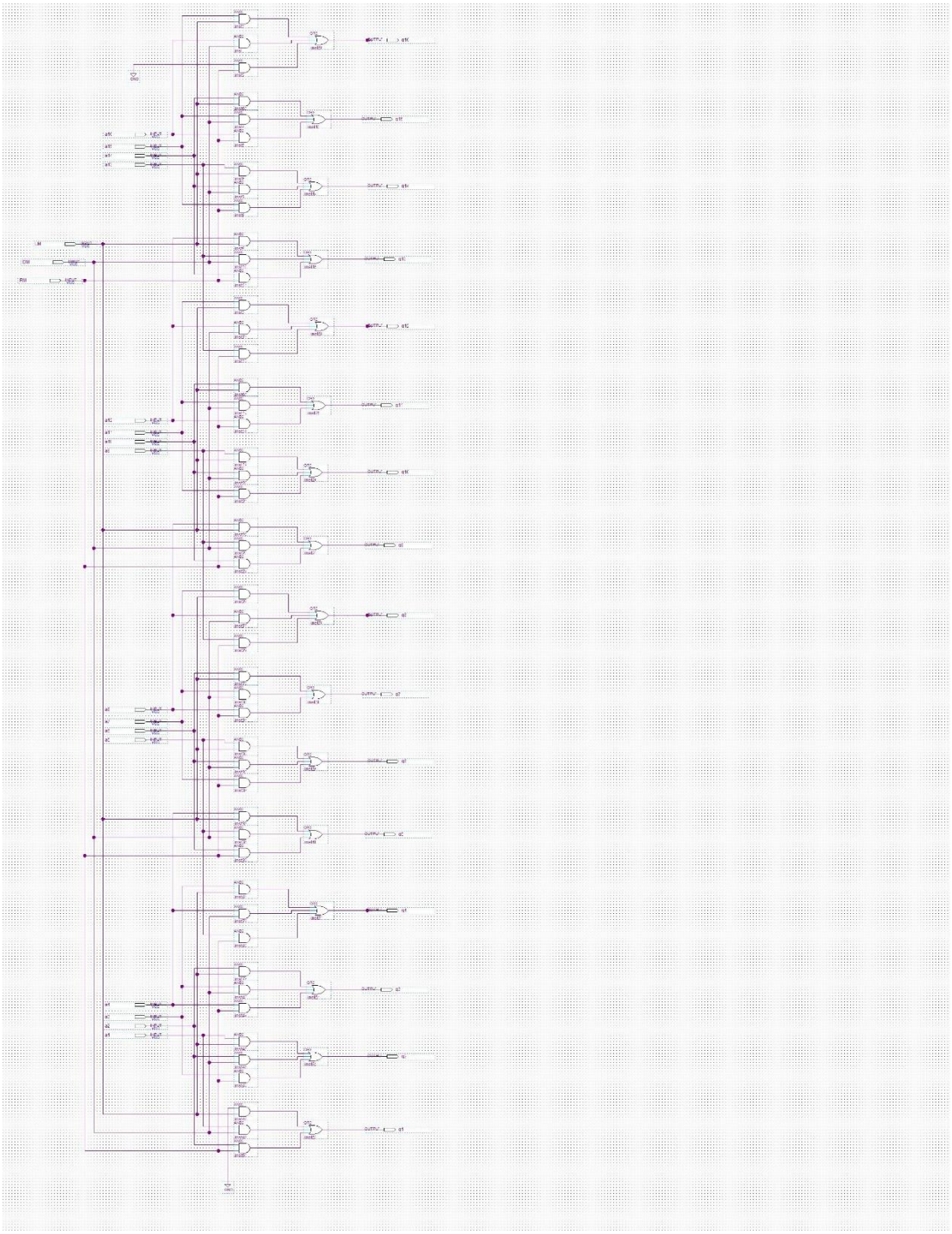
2.1 ALU 原理图



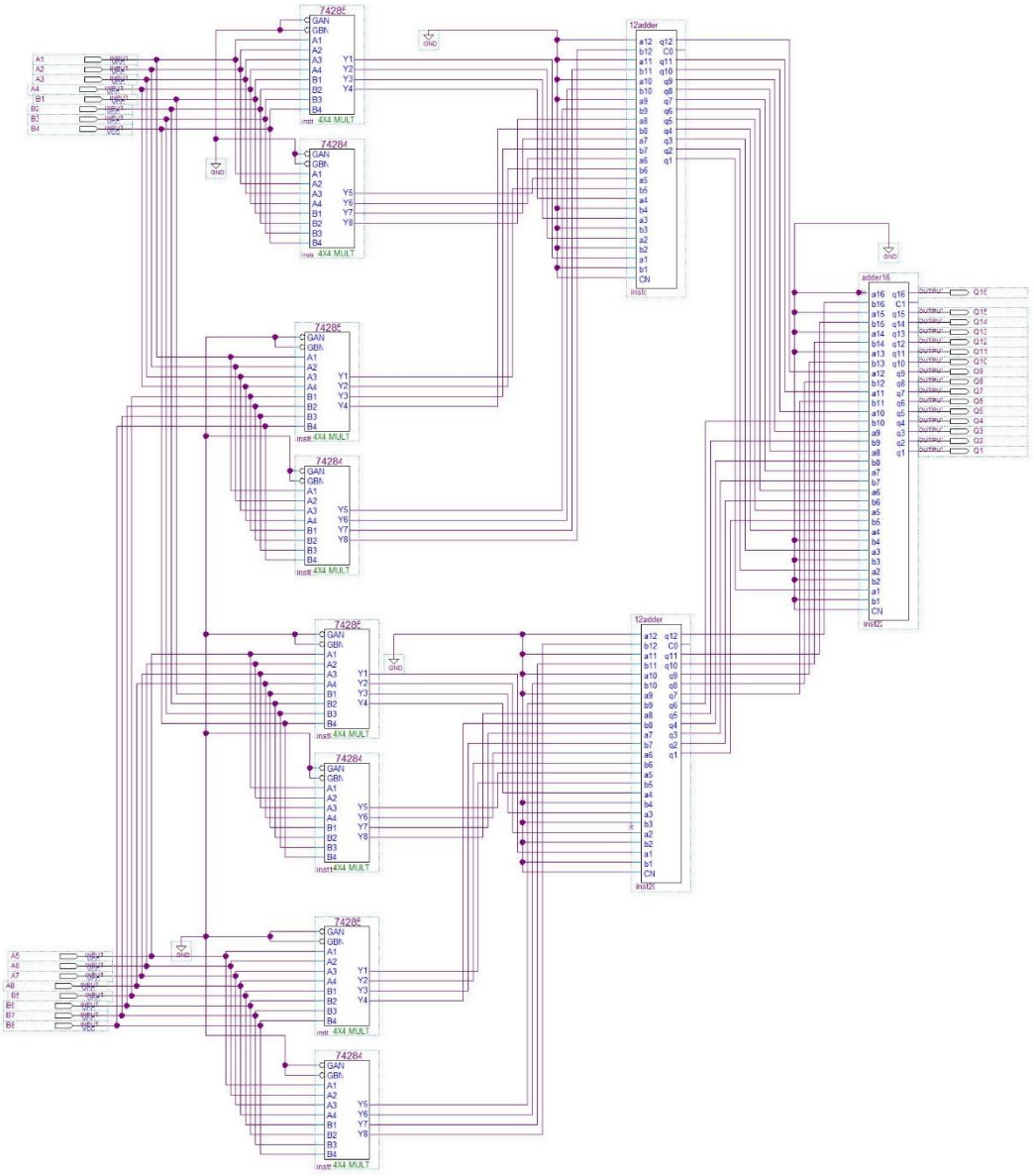
最左侧四个寄存器分别存放输入的 16 位数据 A 和 B，由 CPR0 和 CPR1 控制两个寄存器的

写入。采用 4 片 74181 和 1 片 74182 构造进位链，CN,M,S0,S1,S2,S3 控制进行的计算的类型。进位链输出的 16 位结果输入移位器“shifter16”中，由 DM 直传，LM 左移，RM 右移控制移位器的操作。移位器输出的结果在打入脉冲 CPR2 下存入寄存器。

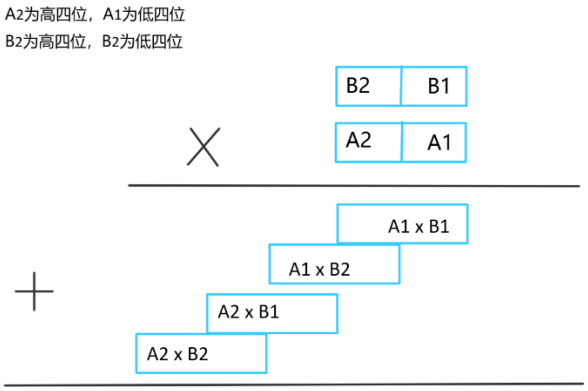
其中 shifter16 以 16 位数据和 DM,LM,RM 为输入，输出移位之后的数据，shifter16 原理图如下：



2.2 MULTIPLIER 原理图



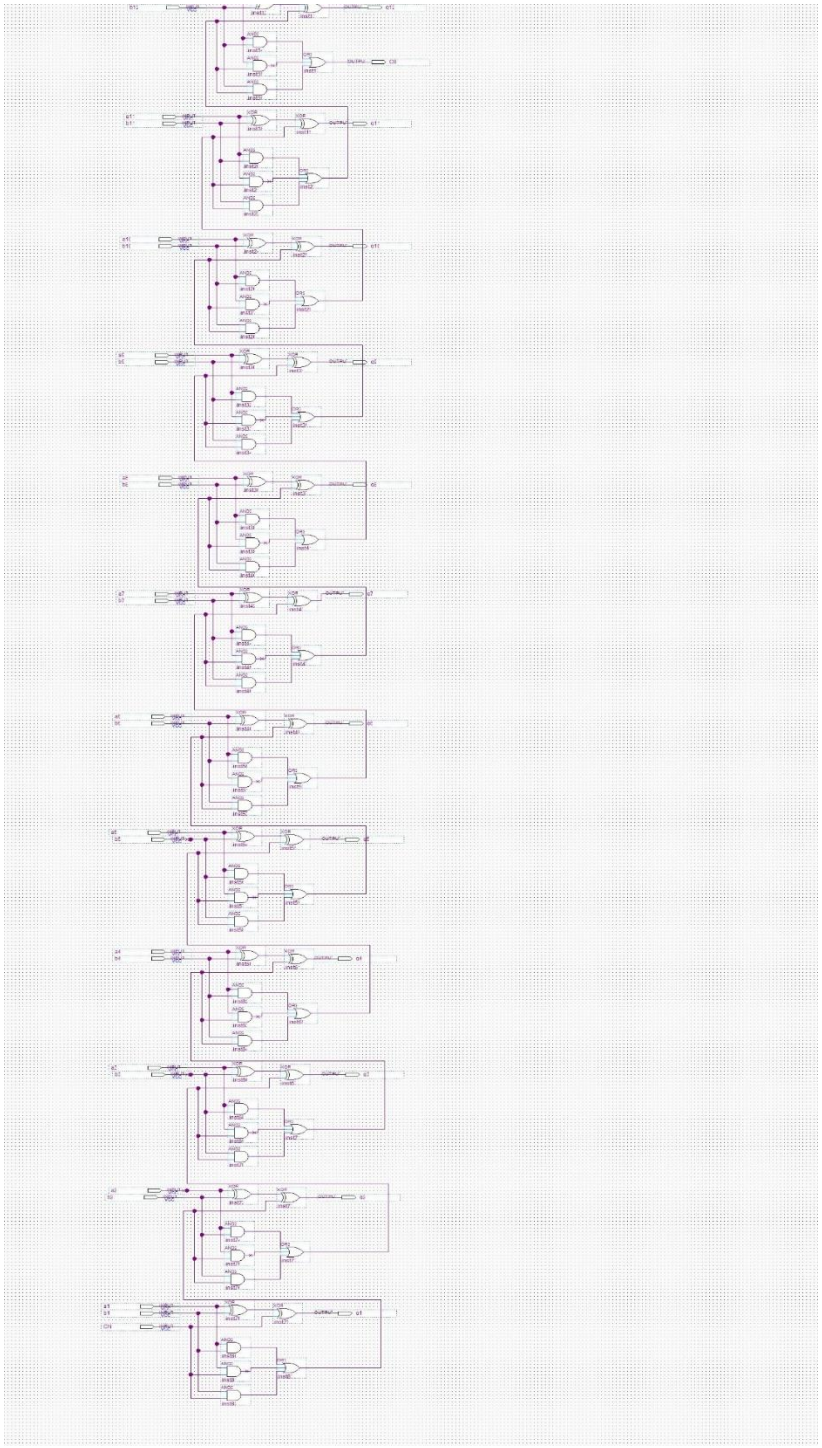
74285 和 74284 可用于四位乘法，其中 74285 输出低四位结果，74284 输出高四位结果。本 MULTIPLIER 将 8 位乘法转化为四位乘法:



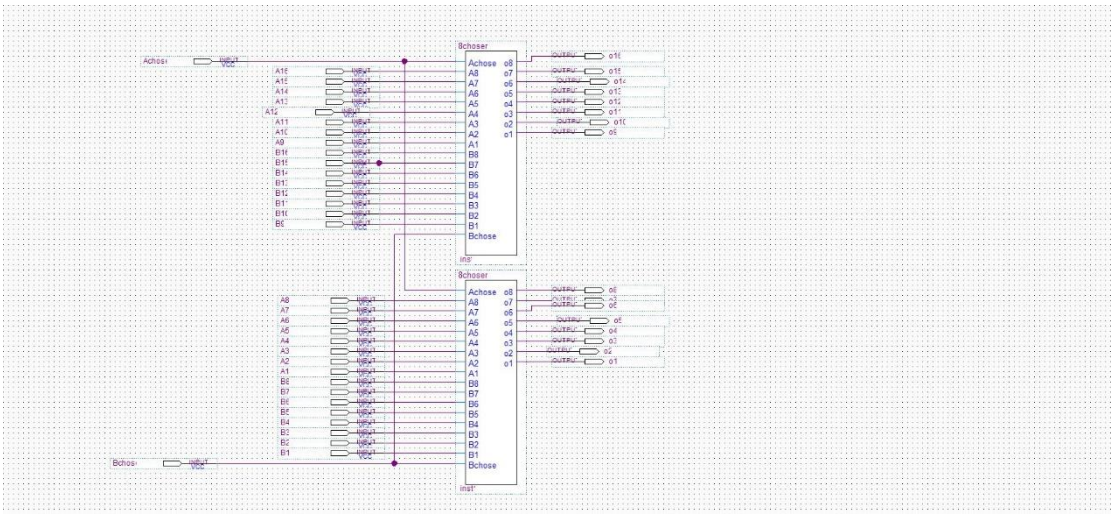
两个 12adder 为 12 位加法器，分别负责 A1xB1 与 A1xB2 相加和 A2xB1 与 A2xB2 相加。相加时，A1xB1 应为低 8 位，高位补 0；A1xB2 为高 8 位，低位补 0；记相加结果位 M。A2xB1 应为低 8 位，高位补 0；A2xB2 为高 8 位，低位补 0,记相加结果为 N。用 adder16 完成 M,N 化为 16 位数后的相加，其中 M 为低 12 位，高位补 0；N 为高 12 位，低位补 0。adder16 计算的结果即为 8 位乘法所得的结果。

其中为了保证乘法过程中不需要消耗时钟周期，12adder 与 adder16 都是直接相加。

12adder 与 adder16 都是由多个全加器组成，组织形式如下：

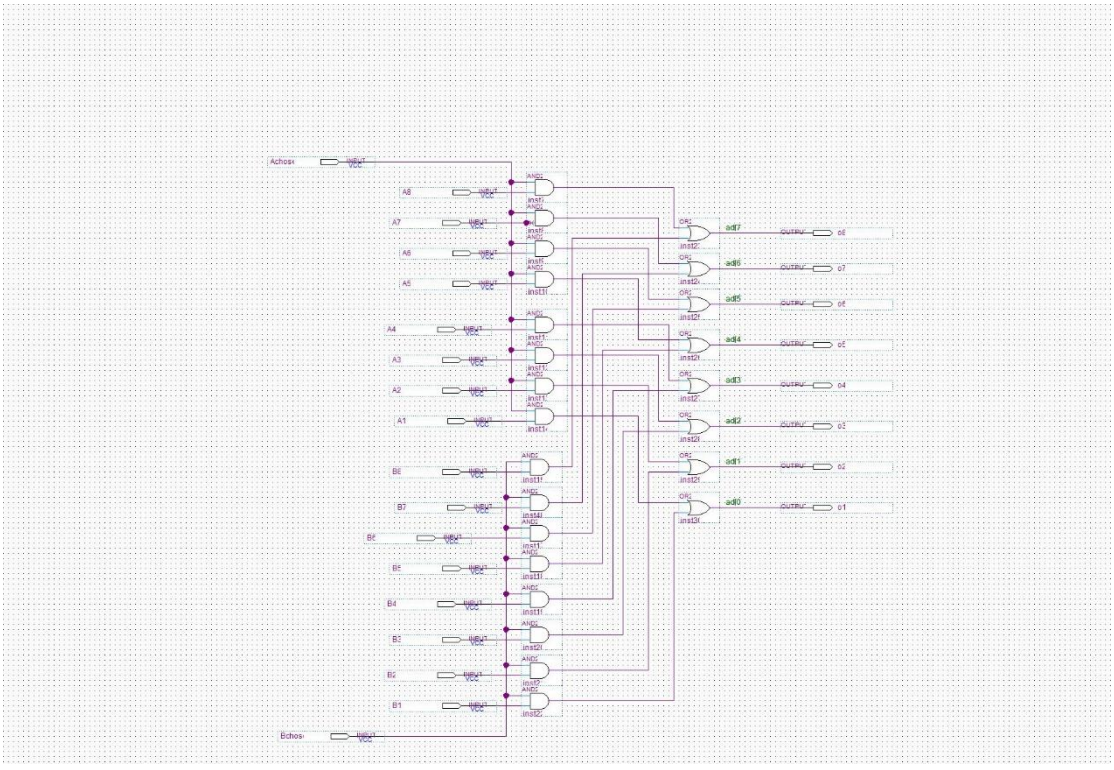


2.3 choser(选择器)原理图

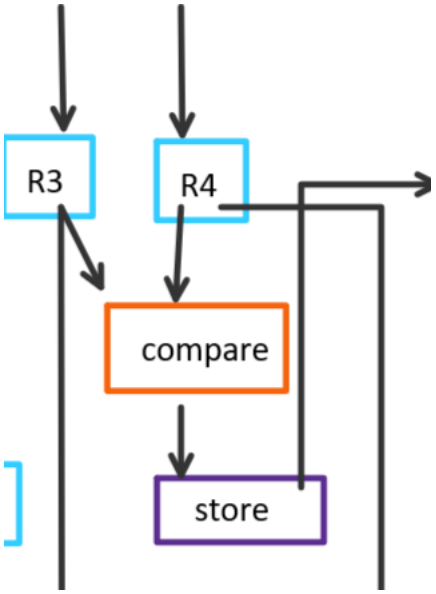


通过两个 8 位选择器组成 16 位选择器。

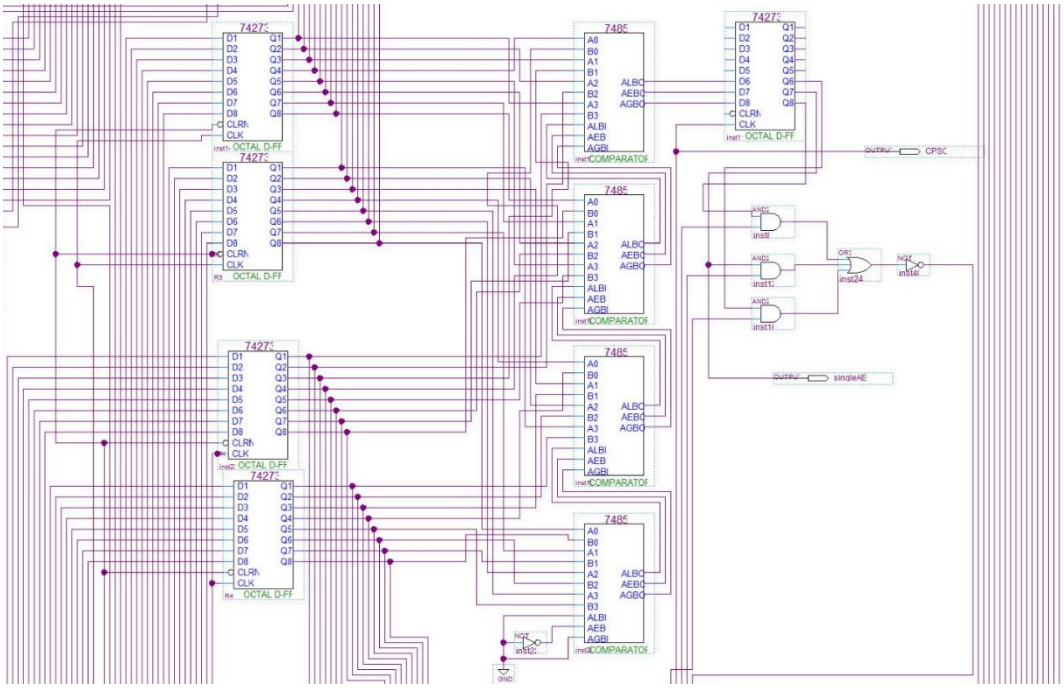
8 位选择器原理图:



3. 负责进行比较的部分

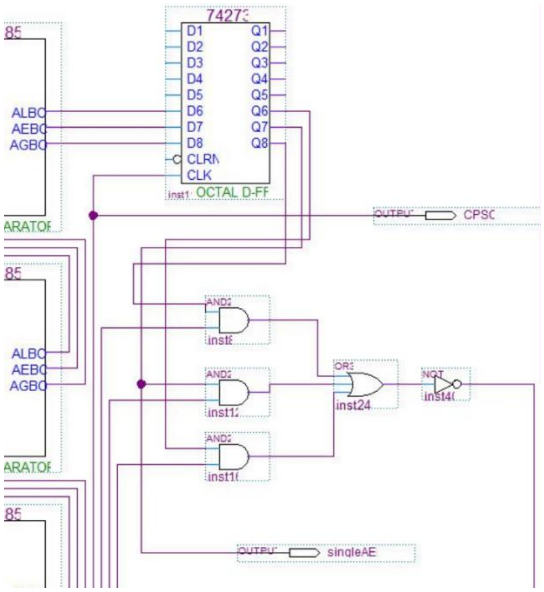


上图为总体框图中和比较相关的部分。



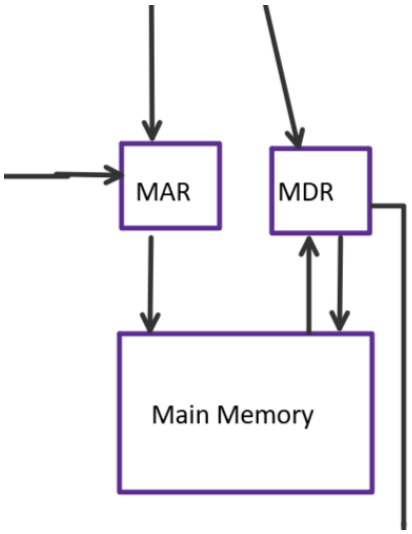
上图为比较部分相关的原理图。左边四个 74273 为两个 16 位寄存器 R3 和 R4。四个 7485 构成了 16 位比较器。16 位比较器的输入为两个 16 位数，输出为 ALB,AEB,AGB。若 ALB 输出 0，则 A 小于 B；若 AEB 输出 0，则 A=B；若 AGB 输出 0，则 A 大于 B。7485 右侧用 74273 的高三位作为 store 寄存器。Store 寄存器控制 PC 的置位端。Store 的打入脉冲为 CPSC。

通过 store 控制 PC 置位，实现跳转指令：

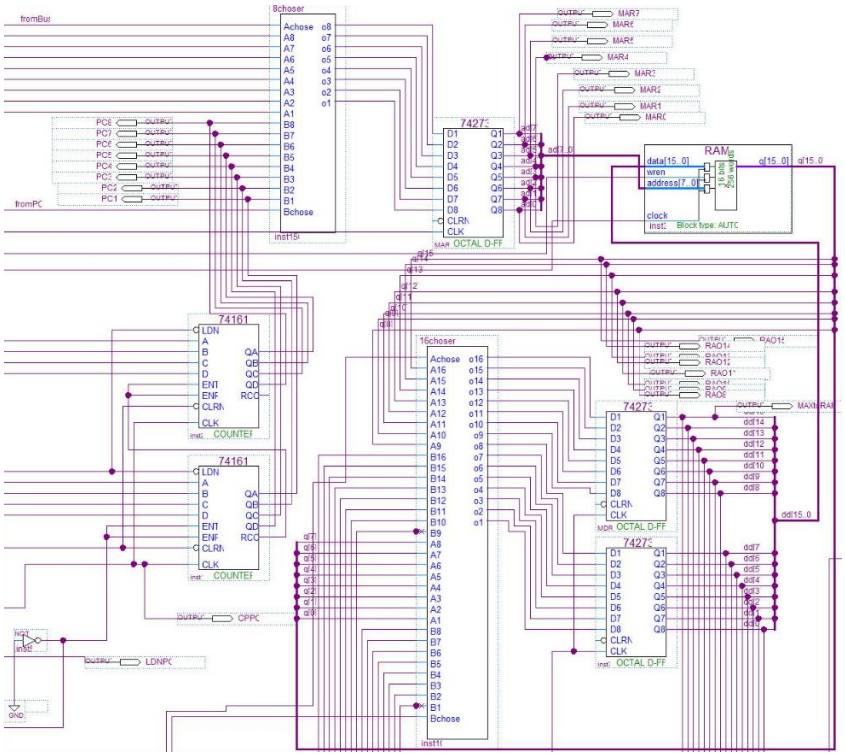


图中最上面的 and2 的输入为 AGB 的结果和跳转指令中的判定条件 A 大于 B 的信号。若二者都为 1 则说明满足跳转条件，则 PC 置位。第二个 and2 的输入为 AEB 的结果和跳转指令中判定 A 等于 B 的信号，若二者都为 1 则满足跳转条件，PC 置位。第三个 and2 的输入为 ALB 的结果和跳转指令中的判定条件 A 小于 B 的信号，若二者都为 1 则满足跳转条件，PC 置位。JUMP 和 JUMP=指令的跳转条件是 AEB,JUMP<是 ALB,JUMP>是 AGB。在判定跳转条件之前，要转移到的地址已经提前放在总线上；一旦置位，则 PC 的值变成要转移到的地址。

4. 内存读写部分

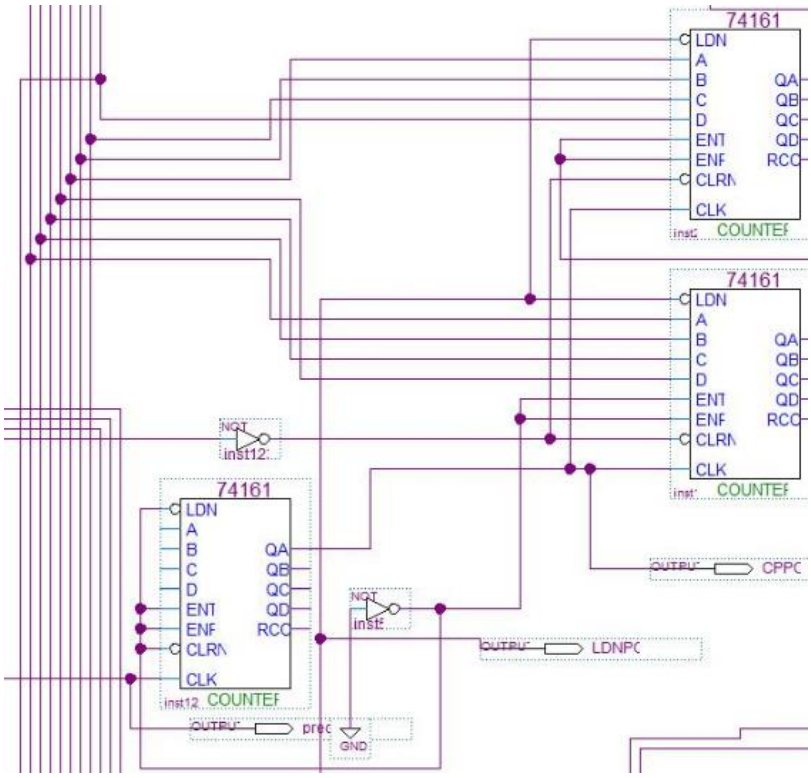


上图为与内存读写相关的部分。MAR 和 MDR 都有两个输入来源，输入来源的选择用选择器实现。MAR 输出到 RAM 中，而 MDR 可以输出到 RAM 也可以输出到总线。本模型机使用的 RAM 为 16 x 256，8 为地址，16 位数据。

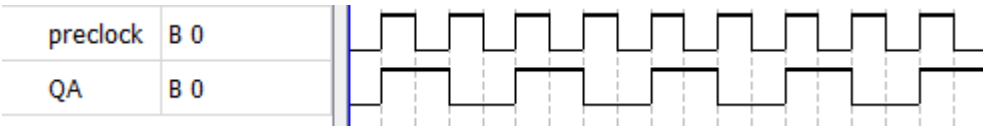


上图为内存读写部分的原理图。MAR 通过 8choser 完成对 PC 和总线的选择，输出连接 RAM 的 address 端。MDR 用 16choser 选择 RAM 或总线，输出连接 RAM 的 data 端和写入总线选择器 Choser6。

5. 其他部分

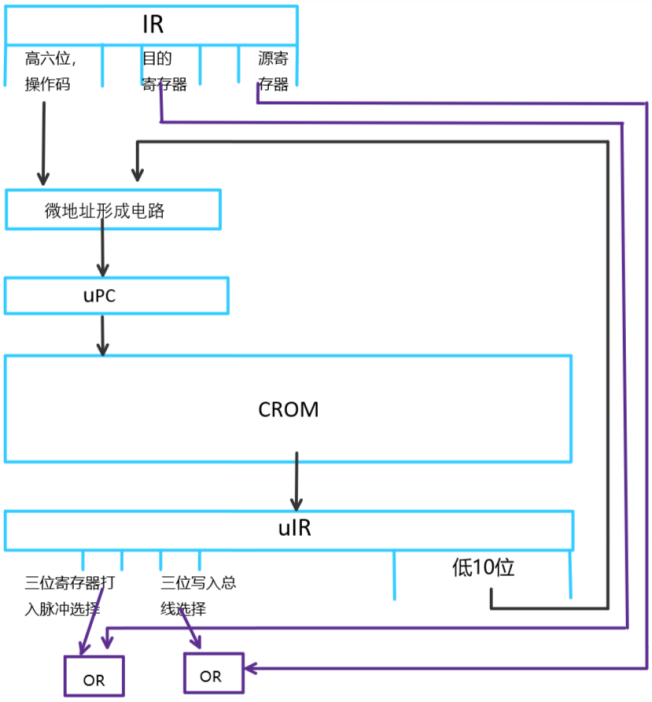


右上角的两个 74161 是 PC，两个 74161 的 ABCD 都连接到总线上，CLRN 与 clear 信号相连，LDN 置位端连接上文所述的比较部分。为了让 PC 的 CLK 足够宽，用左下角的 74161 使得该 74161 的 QA 比系统时钟更宽。



三．微程序实现的控制部件 CU

1. CU 总体框图



IR 存储当前指令，IR 的高六位操作码和 uIR 的低 10 位作为微地址形成电路的地址来源。若微地址由 IR 生成，则生成的 10 位微地址高六位为操作码，低四位为 0；该微地址是指令对应的微程序的入口地址。微地址也可以直接由 uIR 的低 10 位直接给出。uPC 存储当前微指令的地址，可自增或者由微地址形成电路置位。CROM 为控制寄存器，存储微指令。uIR 为微指令寄存器，存储 CROM 中当前的微指令。IR 的三位目的寄存器位输入 3-8 译码器译码，若译码结果指向一个寄存器，则说明该条指令需要向该寄存器写入数据。在 uIR 中设置“toR“位，”toR“有效时，根据上述指向的寄存器发出 CP 脉冲。同时 CP 脉冲也可以直接由 uIR 中对应的三位译码产生。IR 的三位源寄存器经 3-8 译码后若指向某个寄存器，说明该指令需要从该寄存器读数据，即该寄存器的数据需要写入总线。在 uIR 中设置”fromR”位，若有效，则说明要将上述译码器指向的寄存器内容写入总线，产生对应的写入总线信号。写入总线信号也可以由 uIR 对应的三位经译码后产生。

微指令格式如下：

M	S3	S2	S1	S0	CN	A1	B1	C1	A2	B2	C2	A3	B3	C3	wren	G	RESETPC	A4	B4	A5	B5
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
操作ALU的相关信号						3-8译码器的输入			3-8译码器的输入			3-8译码器的输入			RAM写	停机	置0PC	2-4译码器输入		2-4译码器输入	

MAR	CPRAM	LM	DM	RM	toR	frR	A6	B6	C6
10	9	8	7	6	5	4	3	2	1
MAR输入方式		ALU中移位器的控制					3-8译码器的输入		

A1,B1,C1 选择一个 CP 脉冲
(与 IR 对目的寄存器的编码相同)

1	0	0	CPR1
0	1	0	CPR2
1	1	0	CPR3
0	0	1	CPR4
1	0	1	CPR5
0	1	1	CPR6
1	1	1	CPMDR

A2,B2,C2 也选择脉冲

1	0	0	CPIR
0	1	0	CPPC
1	1	0	CPSC
0	0	1	CPMAR
1	0	1	ALUCPR0
0	1	1	ALUCPR1
1	1	1	ALUCPR2

A3,B3,C3 对进入总线的入口进行选择

1	0	0	MULTIPLIER
0	1	0	ALU
1	1	0	R5
0	0	1	MDR
1	0	1	R6
0	1	1	R3
1	1	1	R4

A4,B4 选择跳转条件

1	0	AGB
0	1	AEB
1	1	ALB

A5,B5 的用处

1	0	由内存输入 MDR
0	1	由总线输入 MDR
1	1	清 0 R3,R4

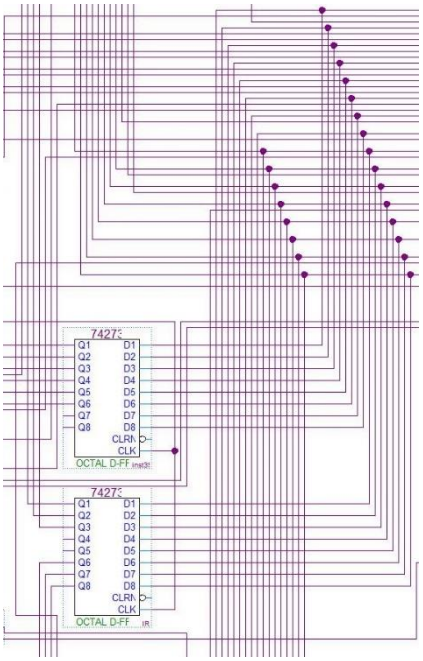
A6,B6,C6 决定下地址生成方式

0	1	0	清零
1	1	0	由 IR 生成
0	0	1	由 uIR 生成

其他情况下 uIR 自动增 1

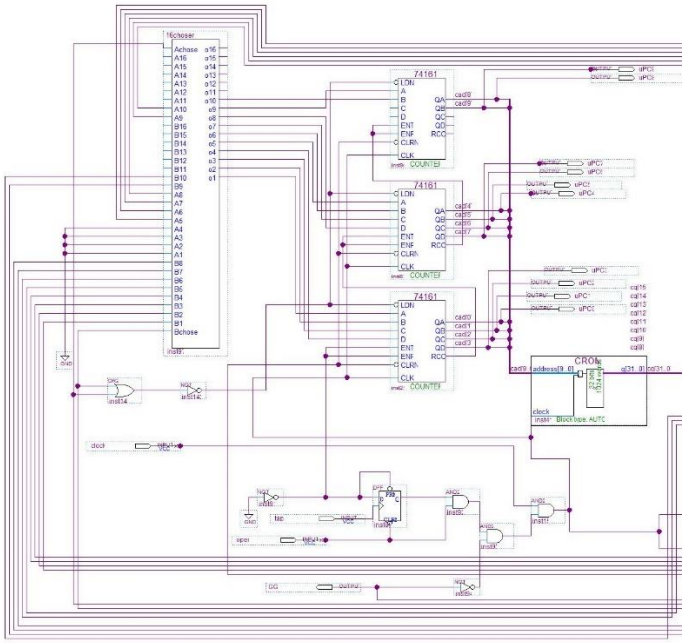
2. 每个部件的原理图

2.1 IR 原理图



IR 为两个 74273 组成的 16 位寄存器。其输入与 16 位总线相连。高六位输出连接微地址形成电路。低 8 位中和寄存器相关的六位与 uIR 中对应的六位会对信号产生共同影响。

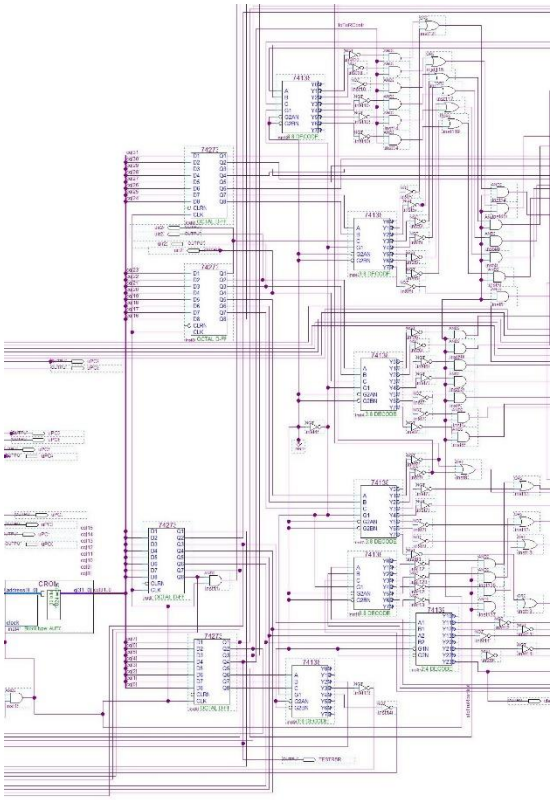
2.2 微地址形成电路,CROM,uPC 原理图



上图中 16choser 为微地址形成电路，其中 A 输入的高六位连接 IR 的高六位，低四位补 0；B 的 10 位输入连接 uIR 的低 10 位。依靠 Achoser 和 Bchoser 线对两种方式进行

选择。uPC 由 3 个 74161 组成，生成 10 位地址。其低 10 位输入为 16choser 的 10 位输出。CROM 为 32x1024 的控制寄存器。CROM 与 uPC 的 clock 直接与系统时钟相连。

2.3 uIR 原理图



uIR 由 4 片 74273 组成，uIR 的输入为 CROM 的 32 位输出端 q[31..0]。uIR 不少输出都需要经译码器译码后才能产生对应的信号。图中与 or 相关的译码器即为上述的与通用寄存器打入脉冲相关的译码器和与写入总线入口选择有关的译码器。

3. 指令执行流程

取指周期

微指令	节拍	微指令码
PC->MAR	T1MAR,C2	00100200,
M(MAR)->MDR	T2:CPRAM,fromMM(A5),CPMDR(A1,B1,C1)	03800900
MDR->IR	T3:C3,CPIR(A2)	00420000
PC->PC+1	T4:CPPC,(B2)	00200000
下地址	T5: A6,B6	00000006

各个指令的执行周期如下：(下列指令除 HALT 外都要最后加上一个周期：

	uPC->0	00000002)
MOV1 R5 R6:		
R6->R5	T1:fromR,toR	00000018
MOV2 # R3:		
PC->MAR	T1:MAR,C2	00100200
M(MAR)->MDR	T2:CPRAM,fromMM(A5),CPMDR(A1,B1,C1)	03800900
PC->PC+1	T3:CPPC,(B2)	00200000
MDR->MAR	T4:MDRo(C3),CPMAR(C2)	00120000
R3->MDR	T5:fromR, CPMDR(A1,B1,C1),busMDR(B5)	03804008
MDR->M(MAR)	T6:wren,CPRAM	00010100
MOV3 R6 立即数:		
PC->MAR	T1:MAR,C2	00100200
M(MAR)->MDR	T2:CPRAM,fromMM(A5),CPMDR(A1,B1,C1)	03800900
PC->PC+1	T3:CPPC(B2)	00200000
MDR->R6	T4:MDRo(C3),toR	00020010
MOV4 S R6:		
PC->MAR	T1:MAR,C2	00100200
M(MAR)->MDR	T2:CPRAM,A5,A1,B1,C1	03800900
PC->PC+1	T3:CPPC(B2)	00200000
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->MAR	T6:C3,C2	00120000
R6->MDR	T7:fromR,B5,CPMDR(A1,B1,C1)	03800408
MDR->M(MAR)	T8:wren,CPRAM	00010100
MOV5 R3 #		
PC->MAR	T1:MAR,C2	00100200
M(MAR)->MDR	T2:CPRAM,A5,A1,B1,C1	03800900
PC->PC+1	T3:CPPC(B2)	00200000
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->R3	T6:toR,MDRo(C3)	00020010
MOV6 R3 S		
PC->MAR	T1:MAR,C2	00100200
M(MAR)->MDR	T2:CPRAM,A5,A1,B1,C1	03800900
PC->PC+1	T3:CPPC(B2)	00200000
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,A1,B1,C1	03800900
MDR->MAR	T6:C3,C2	00120000
M(MAR)->MDR	T7:CPRAM,A5,CPMDR(A1,B1,C1)	03800900

MDR->R3	T8:toR,MDRo(C3)	00020010

JUMP1　立即数

PC->MAR	T1: MAR,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
clearR3R4(清零两个寄存器)	T4:A4,A5	00000C00
COMP	T5: CPSC(A2,B2)	00600000
COMPC(only for data in R3 R4)	T6:MDRo(C3),AEB(B4),CPPC(B2)	00221000

（COMPC 根据 COMP 比较结果设置 PC 的值）

JUMP2　#

PC->MAR	T1: MAR,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
clearR3R4	T6:A4,A5	00000C00
COMP	T7: CPSC(A2,B2)	00600000
COMPC(only for data in R3 R4)	T8:MDRo(C3),AEB(B4),CPPC(B2)	00221000

JUMP3 R6

clearR3R4	T1:A4,A5	00000C00
COMP	T2: CPSC(A2,B2)	00600000
COMPRC(only for data in R3 R4)	T3:fromR,AEB(B4),CPPC(B2)	00201008

Jump1=　　立即数

PC->MAR	T1: MAR,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T4: CPSC(A2,B2)	00600000
ECOMPC(only for data in R3 R4)	T5:MDRo(C3),AEB(B4),CPPC(B2)	00221000

Jump2=　　#

PC->MAR	T1: MAR,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T6: CPSC(A2,B2)	00600000
ECOMPC(only for data in R3 R4)	T7:MDRo(C3),AEB(B4),CPPC(B2)	00221000

Jump3=　　R6

COMP	T1: CPSC(A2,B2)	00600000
ECOMPRC(only for data in R3 R4)	T2:fromR,AEB(B4),CPPC(B2)	00201008

Jump1>　　立即数

PC->MAR	T1: MAR,C2	00100200

M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T4: CPSC(A2,B2)	00600000
GCOMPC(only for data in R3 R4)	T5:MDRo(C3),AGB(A4),CPPC(B2)	00222000

Jump2> #

PC->MAR	T1: MAR,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T6: CPSC(A2,B2)	00600000
GCOMPC(only for data in R3 R4)	T7:MDRo(C3),AGB(A4),CPPC(B2)	00222000

Jump3> R6

COMP	T1: CPSC(A2,B2)	00600000
GCOMPRC	T2:fromR,AGB(A4),CPPC(B2)	00202008

Jump1< 立即数

PC->MAR	T1: MAR1,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T4: CPSC(A2,B2)	00600000
LCOMPC(only for data in R3 R4)	T5:MDRo(C3),ALB(A4,B4),CPPC(B2)	00223000

Jump2< #

PC->MAR	T1: MAR1,C2	00100200
M(MAR)->MDR	T2: CPRAM,A5,CPMDR(A1,B1,C1)	03800900
MDR->MAR	T4:C3,C2	00120000
M(MAR)->MDR	T5:CPRAM,A5,CPMDR(A1,B1,C1)	03800900
COMP	T6: CPSC(A2,B2)	00600000
LCOMPC(only for data in R3 R4)	T7:MDRo(C3),ALB(A4,B4),CPPC(B2)	00223000

Jump3< R6

COMP	T1: CPSC(A2,B2)	00600000
LCOMPRC(only for data in R3 R4)	T2:fromR,ALB(A4,B4),CPPC(B2)	00203008

CLEAR

PC->0	T1:RESTPC	00004000

计算相关指令结果默认存储在 R6 中

ADD

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
ADD R1 R2	T3:CN,S3,S0,ALUCPR2(A2,B2,C2),DM	4C700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000

LOADD

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
LADD R1 R2	T3:M,S3,S2,S1,ALUCPR2(A2,B2,C2),DM	F0700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000

Sub

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
SUB R1 R2	T3:CN,S2,S1,ALUCPR2(A2,B2,C2),DM	34700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000

INTERSECT

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
AB R1 R2	T3:M,S3,S1,S0,ALUCPR2(A2,B2,C2),DM	D8700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000

XOR

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
XOR R1　R2	T3:M,S2,S1,ALUCPR2(A2,B2,C2),DM	B0700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000

LL

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
A 直传 LM	T2:M,S3,S2,S1,S0,LM, ALUCPR2(A2,B2,C2)	F8700080
ALU->R6	T3:ALUo(B3),CPR6(B1,C1)	01840000

RR

A RM

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
A 直传 RM	T2:M,S3,S2,S1,S0,RM, ALUCPR2(A2,B2,C2)	F8700020
ALU->R6	T3:ALUo(B3),CPR6(B1,C1)	01840000

NOT

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
A 逻辑非	T2: M,ALUCPR2(A2,B2,C2),DM	80700040
ALU->R6	T3:ALUo(B3),CPR6(B1,C1)	01840000

ADD1

R1->ALU	T1:ALUCPR0(A2,C2)	00500000
A+1	T2:ALUCPR2(A2,B2,C2)， DM	00700040
ALU->R6	T3:ALUo(B3),CPR6(B1,C1)	01840000

SUB1		
R1->ALU	T1:ALUCPR0(A2,C2)	00500000
A-1	T2:S3,S2,S1,S0,CN,ALUCPR2(A2,B2,C2),DM	7C700040
ALU->R6	T3:ALUo(B3),CPR6(B1,C1)	01840000
ADD2		
R1->ALU	T1:ALUCPR0(A2,C2)	00500000
R2->ALU	T2:ALUCPR1(B2,C2)	00300000
A+B+1	T3:S3, S0,ALUCPR2(A2,B2,C2),DM	48700040
ALU->R6	T4:ALUo(B3),CPR6(B1,C1)	01840000
MULTI		
Muti->R6	T1:Mutio(A3),CPR6(B1,C1)	01880000
HALT		
G	T1:G	00008000

4. CROM 内的微程序

CROM 为 32 x 1024 的 ROM，每个指令的执行周期对应的微程序开始处的 10 位地址中，高六位为 IR 的高六位，低四位为 0。即每个微程序占用的地址空间最多为$2^4 = 16$。微程序从对应地址处开始，每个地址存放一个 32 位微指令。每个执行周期微程序由如上指令流程中所述的执行周期的微指令码顺序组成。特别地，CROM 的 0 位开始存放实现取指周期的微程序。每个指令的执行周期对应的微程序执行完后会使用 00000002 让 uPC 清零，即开始下一个取指周期。

程序	CROM 中的位置
取指周期	000
MOV1 R1 R2	010
MOV2 # R1	020
MOV3 R1 立即数	030
MOV4 S R1	040
MOV5 R1 #	050
MOV6 R1 S	060
JUMP1 立即数	070
JUMP2 #	080
JUMP3 R1	090
JUMP= 立即数	0a0
JUMP= #	0b0
JUMP= R1	0c0
JUMP> 立即数	0d0
JUMP> #	0e0

JUMP>	R1	0f0
JUMP<	立即数	100
JUMP<	#	110
JUMP<	R1	120
CLEAR		130
ADD		140
LOADD		150
SUB		160
INTERSECT		170
XOR		180
LL		190
RR		1a0
NOT		1b0
ADD1		1c0
SUB1		1d0
ADD2		1e0
MULTI		1f0
HALT		200

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	00100200	03800900	00420000	00200000	00000006	00000000	00000000	00000000	.....
008	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
010	00000018	00000002	00000000	00000000	00000000	00000000	00000000	00000000	.....
018	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
020	00100200	03800900	00200000	00120000	03804008	00010100	00000002	00000000	.....
028	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
030	00100200	03800900	00200000	00020010	00000002	00000000	00000000	00000000	.....
038	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
040	00100200	03800900	00200000	00120000	03800900	00120000	03800408	00010100	.....
048	00000002	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
050	00100200	03800900	00200000	00120000	03800900	00020010	00000002	00000000	.....
058	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
060	00100200	03800900	00200000	00120000	03800900	00120000	03800900	00020010	.....
068	00000002	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
070	00100200	03800900	00000C00	00600000	00221000	00000002	00000000	00000000	.....
078	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
080	00100200	03800900	00120000	03800900	00000C00	00600000	00221000	00000002	.....
088	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
090	00000C00	00600000	00201008	00000002	00000000	00000000	00000000	00000000	.....
098	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
0a0	00100200	03800900	00600000	00221000	00000002	00000000	00000000	00000000	.....
0a8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0b0	00100200	03800900	00120000	03800900	00600000	00221000	00000002	00000000	....
0b8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
0c0	00600000	00201008	00000002	00000000	00000000	00000000	00000000	00000000	....
0c8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
0d0	00100200	03800900	00600000	00222000	00000002	00000000	00000000	00000000	....
0d8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
0e0	00100200	03800900	00120000	03800900	00600000	00222000	00000002	00000000	....
0e8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
0f0	00600000	00202008	00000002	00000000	00000000	00000000	00000000	00000000	....
0f8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
100	00100200	03800900	00600000	00223000	00000002	00000000	00000000	00000000	....
108	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
110	00100200	03800900	00120000	03800900	00600000	00223000	00000002	00000000	....
118	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
120	00600000	00203008	00000002	00000000	00000000	00000000	00000000	00000000	....
128	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
130	00004000	00000002	00000000	00000000	00000000	00000000	00000000	00000000	....
138	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
140	00500000	00300000	4C700040	01840000	00000002	00000000	00000000	00000000	....
148	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
150	00500000	00300000	F0700040	01840000	00000002	00000000	00000000	00000000	....
158	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
160	00500000	00300000	34700040	01840000	00000002	00000000	00000000	00000000	.....
168	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
170	00500000	00300000	D8700040	01840000	00000002	00000000	00000000	00000000	.....
178	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
180	00500000	00300000	B0700040	01840000	00000002	00000000	00000000	00000000	.....
188	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
190	00500000	F8700080	01840000	00000002	00000000	00000000	00000000	00000000	.....
198	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1a0	00500000	F8700020	01840000	00000002	00000000	00000000	00000000	00000000	.....
1a8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1b0	00500000	80700040	01840000	00000002	00000000	00000000	00000000	00000000	.....
1b8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1c0	00500000	00700040	01840000	00000002	00000000	00000000	00000000	00000000	.....
1c8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1d0	00500000	7C700040	01840000	00000002	00000000	00000000	00000000	00000000	.....
1d8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1e0	00500000	00300000	48700040	01840000	00000002	00000000	00000000	00000000	.....
1e8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
1f0	01880000	00000002	00000000	00000000	00000000	00000000	00000000	00000000	.....
1f8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
200	00008000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
208	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

5. RAM 内编制的应用程序

汇编语言:

程序 1

```
MOV3   R2   18
MOV3   R1   4
MULTIPLIER
MOV1   R2   R6
ADD
JUMP    0x020
```

程序 2

```
MOV3   R2   18
MOV3   R1   4
MULTIPILER
MOV1   R2   R6
ADD
JUMP    0x05
```

程序 3:

```
MOV3   R2    18
MOV3   R1   4

LOADD
SUB
INTERSECT
```

020:
MOV6 R6 0x030
HALT
030:
0038

038:
AAAA

XOR
LL
RR
NOT
ADD1
SUB1
ADD2

机器语言:

程序 1
0C40
0012
0C80
0004
7C00
0445
5000
1C00
0020

020:
1860
0030
8000

030:
0038

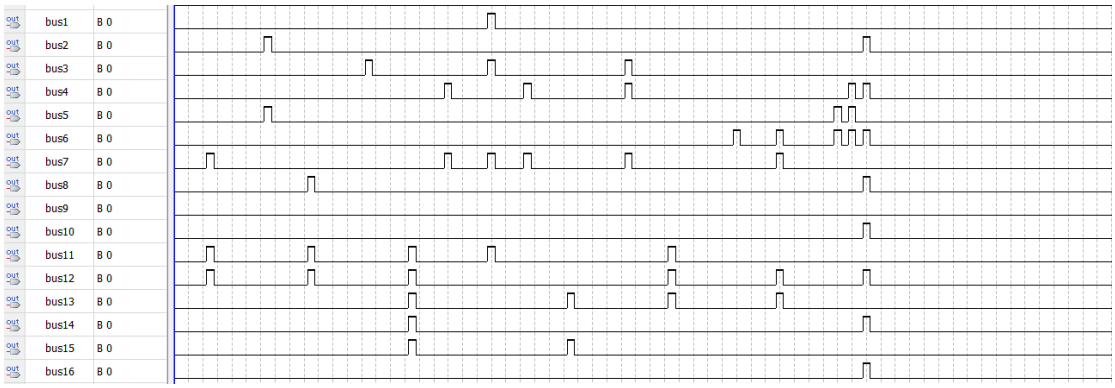
038:
AAAA

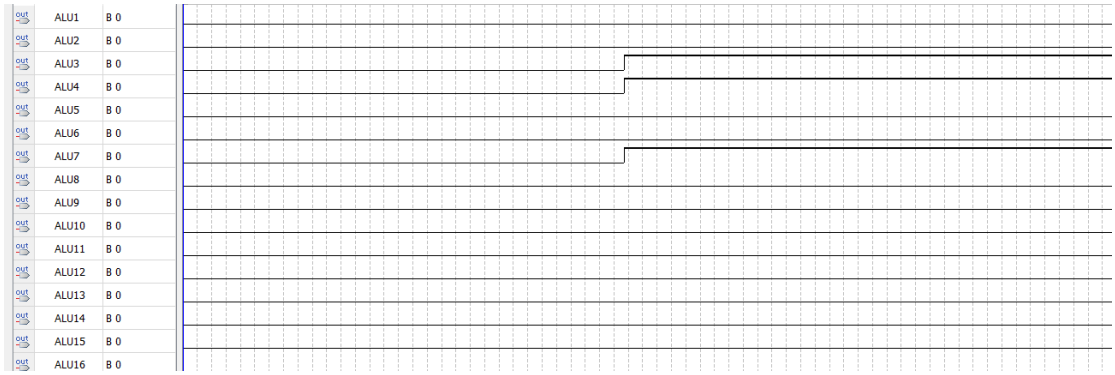
程序 2:
0C40
0012
0C80
0004
7C00
0445
5000
1C00
0005

程序 3:
0C40
0012
0C80
0004

5400
5800
5C00
6000
6400
6800
6C00
7000
7400
7800

程序 1 功能：将 18 装入 R2,4 装入 R1,算两数的乘积并将结果存入 R2,使 R1 与 R2 内的数相加。之后跳转到地址 020 处。地址 020 处的指令为将间接地址 030 指向的数据存入 R6 中。1 运行结果：呈现总线上的数据和 ALU 计算结果。

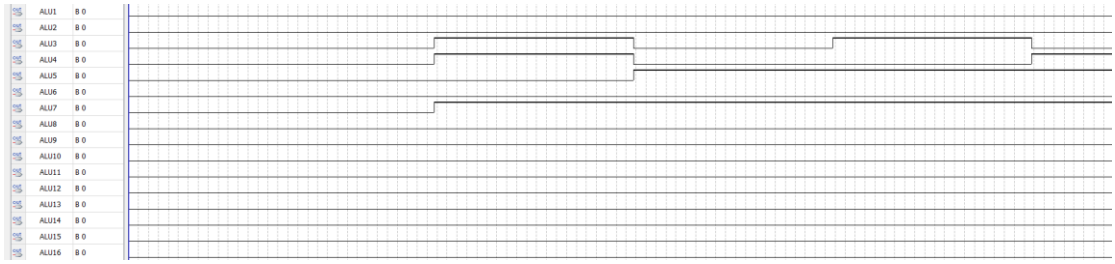




程序中只进行了一次 ALU 运算，应该为 18x4+4=76，二进制形式为 0000000001001100，和上图中 ALU 的结果相同。总线图中每个指令对应的指令码都会显示在总线上，可以依此分辨模型机的执行情况。从左向右倒数第四列为 1860，说明跳转指令执行成功，当前正在执行 MOV6。而倒数第三列为 0030,说明此时读入 MOV6 的数据。倒数第二列为 0038 说明已经得到有效地址 038.倒数第一列为 AAAA,说明 038 中的数据已经被写入 R6。

程序 2 功能：先将 18 与 4 相乘，相乘的结果不断加四

2 运行结果：



ALU 先计算出 76，加四后变为 80，之后又变为 84，再后为 88。与期望实现的功能效果相同。

程序 3 测试计算部分的功能：