

伟福®

**伟福 COP2000 型
计算机组成原理实验仪**

南京伟福实业有限公司

目录

第一章 性能特点	
1.1 硬件先进特点	1
1.2 软件先进特点	3
1.3 实验系统组成	4
第二章 分部实验项目	
2.1 寄存器实验	5
实验 1: A、W 寄存器实验	6
实验 2: R0、R1、R2、R3 寄存器实验	8
实验 3: MAR 地址寄存器、ST 堆栈寄存器、OUT 输出寄存器实验	12
2.2 运算器实验	14
2.3 数据输出实验/移位门实验	16
实验 1: 数据输出实验	17
实验 2: 移位实验	17
2.4 uPC 实验	20
实验 1: uPC 加 1 实验	21
实验 2: uPC 打入实验	21
2.5 PC 实验	22
实验 1: PC 加 1 实验	24
实验 2: PC 打入实验	24
2.6 存储器 EM 实验	25
实验 1: PC/MAR 输出地址选择	26
实验 2: 存储器 EM 写实验	26
实验 3: 存储器 EM 读实验	27
实验 4: 存储器打入 IR 指令寄存器/uPC 实验	28
实验 5: 使用实验仪小键盘输入 EM	29
2.7 微程序存储器 uM 实验	30
实验 1: 微程序存储器 uM 读出	31
实验 2: 使用实验仪小键盘输入 uM	31
2.8 中断实验	32
第三章 COP2000 模型机	
3.1 模型机总体结构	33
3.2 模型机寻址方式	34
3.3 模型机指令集	34
3.4 模型机微指令集	36

第四章 模型机综合实验(微程序控制器)	
实验 1: 数据传送实验/输入输出实验.....	45
实验 2: 数据运算实验(加/减/与/或).....	48
实验 3: 移位/取反实验.....	48
实验 4: 转移实验.....	51
实验 5: 调用实验.....	51
实验 6: 中断实验.....	55
实验 7: 指令流水实验.....	55
实验 8: RISC 模型机.....	55
第五章 组合逻辑控制器.....	58
组合逻辑控制器.....	58
用 EPLD 实现运算器功能.....	76
用 EPLD 实现堆栈功能.....	79
用 EPLD 实现 R0..R3 功能.....	79
第六章 设计指令/微指令系统.....	81
第七章 扩展实验	
实验 1: 用 8255 扩展 I/O 端口实验.....	85
实验 2: 用 8253 扩展定时器实验.....	86
第八章 实验仪键盘使用.....	87
观察内部寄存器.....	88
观察、修改程序存储器内容.....	88
观察、修改微程序存储器内容.....	89
用小键盘调试实验一.....	90
COP2000 实验仪自动检测.....	92
第九章 COP2000 集成开发环境使用.....	93
主菜单.....	94
快捷键图标.....	95
源程序/机器码窗口.....	95
结构图/逻辑分析窗口.....	96
指令/微程序/跟踪窗口.....	97
寄存器状态.....	98

第一章 性能特点

COP2000 计算机组成原理实验系统主要是为配合讲授与学习《计算机组成原理》课程而研制的。与其它产品相比，具有以下特点：

1.1 硬件先进特点：

实时监视器

各单元部件都以计算机结构模型布局，清晰明了，各寄存器、部件均有 8 位数据指示灯显示其二进制值，两个 8 段码 LED 显示其十六进制值，清楚明了，两个数据流方向指示灯，以直观反映当前数据值及该数据从何处输出，而又是被何单元接收的。这是该产品独创的“实时监视器”，使得系统在实验时即使不借助 PC 机，也可实时监控数据流状态及正确与否，彻底改变了其它实验设备为监控状态必须加入读操作的不真实实验方法，使得学生十分容易认识和理解计算机组成结构。实验系统各部件可以通过 J1、J2、J3 座之间不同的连线组合，可进行各部件独立的实验，也可进行各部件组合实验，再通过与控制线的组合，就可构造出不同结构及复杂程度的原理性计算机。

开放式设计

实验系统的软硬件对用户的实验设计具有完全的开放特性。与众不同的是：COP2000 各实验模块的数据线、地址线与系统之间的挂接是通过三态门，而不是其它实验设备所采用的扁平连线方法，而数据线、地址线是否要与系统连通，则由用户连线控制，这样，就真实的再现了计算机工作步骤。需要强调指出的是：用“连线跨接”并不能说明其开放特性，而所谓的开放性应指的是运算器、控制器及微程序指定的格式及定义能否进行修改和重新设计。COP2000 系统的运算器采用了代表现代科技的 EDA 技术设计，随机出厂时，已提供一套已装载的方案，能进行加、减、与、或、带进位加、带进位减、取反、直通八种运算方式，若用户不满意该套方案，也可自行重新设计并通过 JTAG 口下载。控制器微指定格式及定义可通过键盘和 PC 机进行重新设计，从而产生与众不同的指令系统。

系统的数据线、地址线、控制线均在总线插孔区引出，并设计了 40 芯锁进插座，供用户进行 RAM、8251、8255、8253、8259 等接口器件的扩展实验。

系统提供的两种控制器之一的组合逻辑控制器已下载有一套完整的实验方案，用户也可使用 CPLD 工具在 PC 机上进行自动化设计。对于不熟悉 EDA 语言的用户，可利用 COP2000 调试环境中的图形表格组合自动产生 EDA 语言，然后在 CPLD 工具下载入大规模逻辑器件中，对于熟悉 EDA 的语言的用户，则还可直接利用 ABEL 或 VHDL 进行重新设计。其开放程度非一般设备所及。

开放式设计的特点还在于，用户可以设计自己的指令/微指令系统。系统中已带三套指令/微程序系统，用户可参照来设计新的指令/微程序系统。

万用汇编器

用户可以自定义指令/微指令系统，COP2000 软件可以对用户自己定义的汇编助记符进

行编译，自动生成代码/微代码。实验系统出厂时提供了完善的指令系统：

算术运算：ADD、ADDC、SUB、SUBC

逻辑运算：AND、OR、CPL

赋值运算：MOV

转移指令：JMP、JC、JZ

调子程序：CALL、RET

中断指令：INT、RETI

端口输入输出：IN、OUT

外部设备输入输出：READ、WRITE

其中的输入输出指令：IN、OUT，模仿 CPU 的端口的输入输出，外部设备输入输出指令：READ、WRITE，可用来访问外设，这两条指令有否直接决定其能否进行接口器件的实验。若没有则表明其无法进行完整模型机实验。

很多实验机还不支持子程序调用、返回指令 CALL、RET。我们知道在调用子程序时涉及到压栈、退栈的概念，这在 COP2000 实验仪中可从微程序层面上十分形象地观察整个执行过程。

完善的寻址方式

累加器寻址：如 CPL A

寄存器寻址：如 MOV A, R0

寄存器间接寻址：如 MOV A, @R0

立即数寻址：如 MOV A, #12H

存储器寻址：如 MOV 34H, A

支持中断实验

采用最底层的器件设计而非有些产品所采用的集成器件 8259，从而让学生可以从微程序层面上学习中断请求、中断响应、中断处理、中断入口地址的产生、中断服务程序及中断返回 (RETI) 整个过程。

专家指出：“中断”是单片机、微机、DSP 等学科中不可或缺的功能，故应在计算机原组成原理这门基础学科中对其进行充分的学习和实验。

两种控制器方式

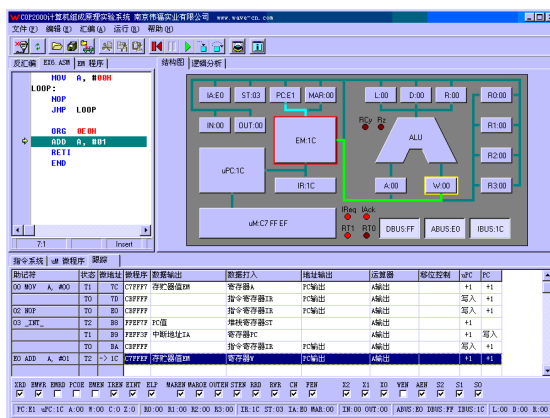
系统提供两种控制器方式，即微程序控制器和组合逻辑控制器。在微程序控制器中，系统能提供在线编程，实时修改程序，显示程序并调试进行的操作环境。组合逻辑控制器，已下载有一套完整的实验方案，用户也可使用 CPLD 工具在 PC 机上进行自动化设计。对于不熟悉 EDA 语言的用户，可利用 COP2000 调试环境中的图形表格组合自动产生 EDA 语言，然后在 CPLD 工具下载入大规模逻辑器件中，对于熟悉 EDA 的语言的用户，则还可直接利用 ABEL 或 VHDL 进行重新设计。其开放程序非一般设备所及。

微程序控制器和组合逻辑控制器两种类型都有流水和非流水两种方案。

三种工作方式

1、“手动方式”——不连 PC 机，通过 COP2000 实验仪的键盘输入程序、微程序，用 LCD 及各部件的 8 个状态 LED，两个方向 LED 观察运行状态和结果，手动进行实验；

2、“联机方式”——连 PC 机，通过 WINDOWS 调试环境及图形方式进行更为直观的实验。



在 WINDOWS 调试环境中提供了功能强大的逻辑分析和跟踪功能,既可以以波形的方式显示各逻辑关系,也可在跟踪器中,观察到当前状态的说明及提示;

3、“模拟方式”不需实验仪,仅需计算机即可进行实验。

强大自检功能

系统设计强有力的自检功能,能自动检测各部件的工作正确与否,并可定位、提示存在问题的部分,并在 LCD 上精确提示。

适当的集成度

计算机组成原理如何解决集成度的问题是各厂家所深感矛盾的难题。伟福公司利用“软件硬化,硬件软化”技术对其进行了适度的分配:运算器、组合逻辑控制器利用大规模可编程逻辑器件实现,其它部件则采用通过逻辑器件实现。这就既可让一般学生利用现有的逻辑知识去认识计算机原理,也可让熟练的学生进行更高层次的开发实践。

完善的保护措施

随机提供了高性能的开关稳压电源,系统中多次采用了抗短路,防过流的设计方法,使其具备良好的稳定性。深入考虑了学生实验的一般特点。

1.2 软件先进特点:

COP2000 软件支持 WINDOWS95/98/2000/XP/ME,集成编辑器、汇编器、调试器。独一无二的“模拟调试”能力,可完全模拟实验机的所有功能。强大的功能、友好的界面定会成为计算机组成原理实验系统的行业标准。

* 多媒体教学:

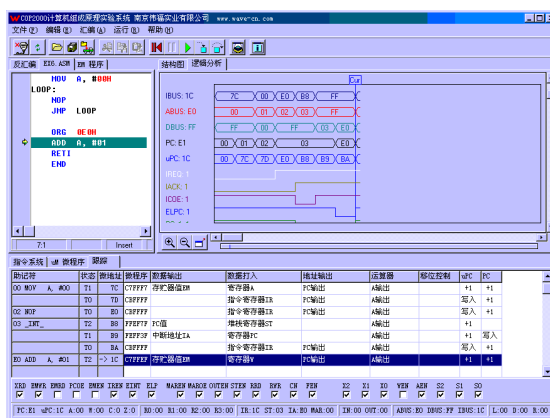
凭借伟福在软件设计上的精湛技术,COP2000 计算机组成原理所配备的 PC 机 WINDOWS 调试软件不仅能进行编辑、编译,并向系统装载实验程序,还提供了单步、断点、实时运行的调试手段,同时,还提供了实验各部件的结构图、时序图、电路原理图。结构图中实时反映程序执行过程中的数据流向及相关的部件;原理图中再现了各部件的详细的组成原理;时序图中则实时反映当前的逻辑关系。所有这些均可通过投影仪可把当前的信息、状态和对应关系进行多媒体教学实践。

* 强大的模拟调试功能

“模拟调试”是指无硬件的情况下,只利用计算机即可进行编辑、编译、改错、调试。计算机组成原理是一门实践性很强的学科,长期以来学时的紧缺成为该科目的主要矛盾。学校即使能做到“人手一机”,也不可能让学生把实验设备带出实验室,也不可能二十四小时开放。“模拟调试”为让实验室向学生寝室、实验课时向业余时间延伸提供了条件,同时也确保了实验室的有效管理,因为“模拟调试”只需给学生一张光盘既可,并可任意复制。这已成为很多学校采用的实验方式。

* 逻辑分析仪

对于教师而言,不难体会要讲清时序关系是不容易的。而学生理解并利用时序关系则难上加难。而由于现代集成技术的迅猛发展,在实际工作中需要更多的利用逻辑分析工具进行时序分析。伟福 COP2000 计算机组成原理与系统结构教学实验系统具备高性能逻辑分析功能,老师可通过电化教学设备向学生现场展示指令与时序的关系,可让学生在实验时直观地观测到指令与时序的关系,可有效的提高教学效果。



*** 模型机结构图:**

该窗口中完全模拟了模型机结构框图,能实时反应程序执行过程中各单元状态变化,总线的数据流向。点击各模块即弹出电路原理图。

*** 微程序及跟踪器:**

跟踪器窗口跟踪程序的执行过程,包括:助记符号、状态、微地址、微程序、数据输出、数据输入、地址输出、运算器、移位控制、uPC、PC。

1.3 实验系统组成:

COP2000 计算机组成原理实验系统由实验平台、开关电源、软件三大部分组成。

实验平台上有寄存器组 R0-R3、运算单元、累加器 A、暂存器 W、直通/左移/右移单元、地址寄存器、程序计数器、堆栈、中断源、输入/输出单元、存储器单元、微地址寄存器、指令寄存器、微程序控制器、组合逻辑控制器、扩展座、总线插孔区、微动开关/指示灯、逻辑笔、脉冲源、20 个按键、字符式 LCD、RS232 口。

第二章 分部实验项目

对于硬件的描述可以有多种方法：如原理图，真值表，高级语言（本手册使用 ABEL），时序图等等，在本手册中使用以上的四种方式来综合描述硬件。

2.1 寄存器实验

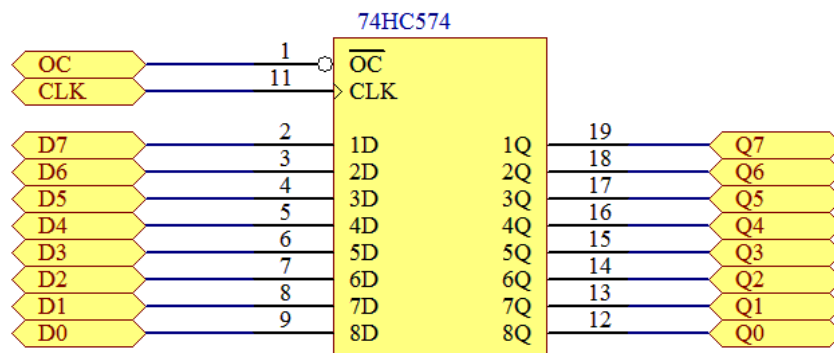
实验要求：利用 COP2000 实验仪上的 K16..K23 开关做为 DBUS 的数据，其它开关做为控制信号，将数据写入寄存器，这些寄存器包括累加器 A，工作寄存器 W，数据寄存器组 R0..R3，地址寄存器 MAR，堆栈寄存器 ST，输出寄存器 OUT。

实验目的：了解模型机中各种寄存器结构、工作原理及其控制方法。

实验说明：

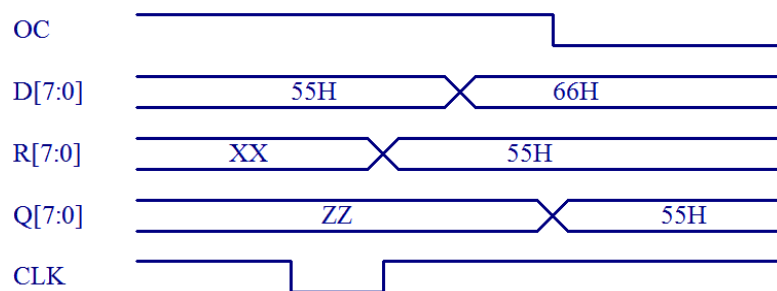
寄存器的作用是用于保存数据的，因为我们的模型机是 8 位的，因此在本模型机中大部寄存器是 8 位的，标志位寄存器(Cy, Z)是二位的。

COP2000 用 74HC574 来构成寄存器。74HC574 的功能如下：



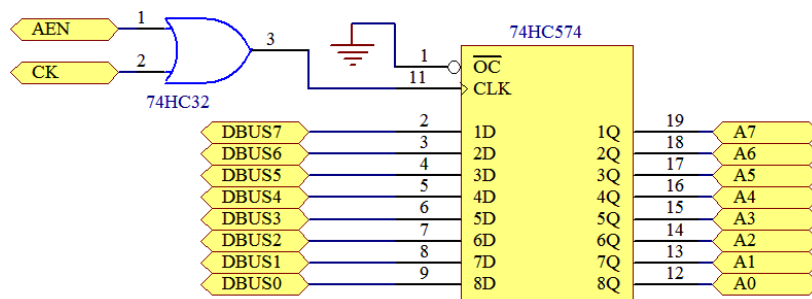
1. 在 CLK 的上升沿将输入端的数据打入到 8 个触发器中
2. 当 OC = 1 时触发器的输出被关闭，当 OC=0 时触发器的输出数据

OC	CLK	Q7..Q0	注释
1	X	ZZZZZZZZ	OC 为 1 时触发器的输出被关闭
0	0	Q7..Q0	当 OC=0 时触发器的输出数据
0	1	Q7..Q0	当时钟为高时，触发器保持数据不变
X	↑	D7..D0	在 CLK 的上升沿将输入端的数据打入到触发器中

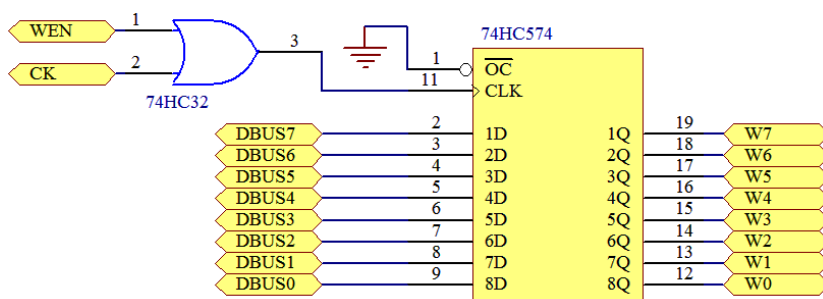


74HC574 工作波形图

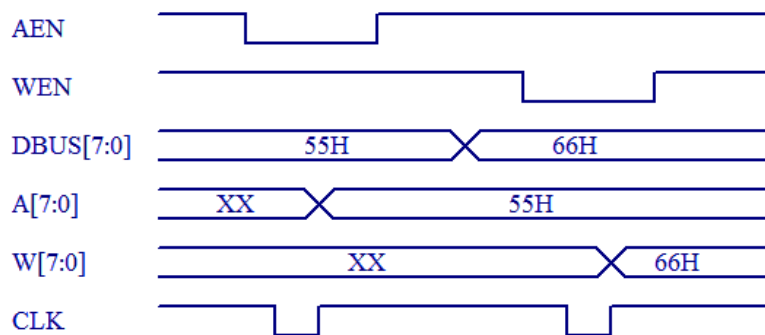
实验 1: A, W 寄存器实验



寄存器 A 原理图



寄存器 W 原理图



寄存器 A, W 写工作波形图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J1 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	AEN	K3	选通 A	低电平有效
3	WEN	K4	选通 W	低电平有效
4	ALUCK	CLOCK	ALU 工作脉冲	上升沿打入

将 55H 写入 A 寄存器

二进制开关 K23-K16 用于 DBUS[7:0] 的数据输入，置数据 55H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	0	1	0	1	0	1

置控制信号为：

K4(WEN)	K3(AEN)
1	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 A 的黄色选择指示灯亮，表明选择 A 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 55H 被写入 A 寄存器。

将 66H 写入 W 寄存器

二进制开关 K23-K16 用于 DBUS[7:0] 的数据输入，置数据 66H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	1	0	0	1	1	0

置控制信号为：

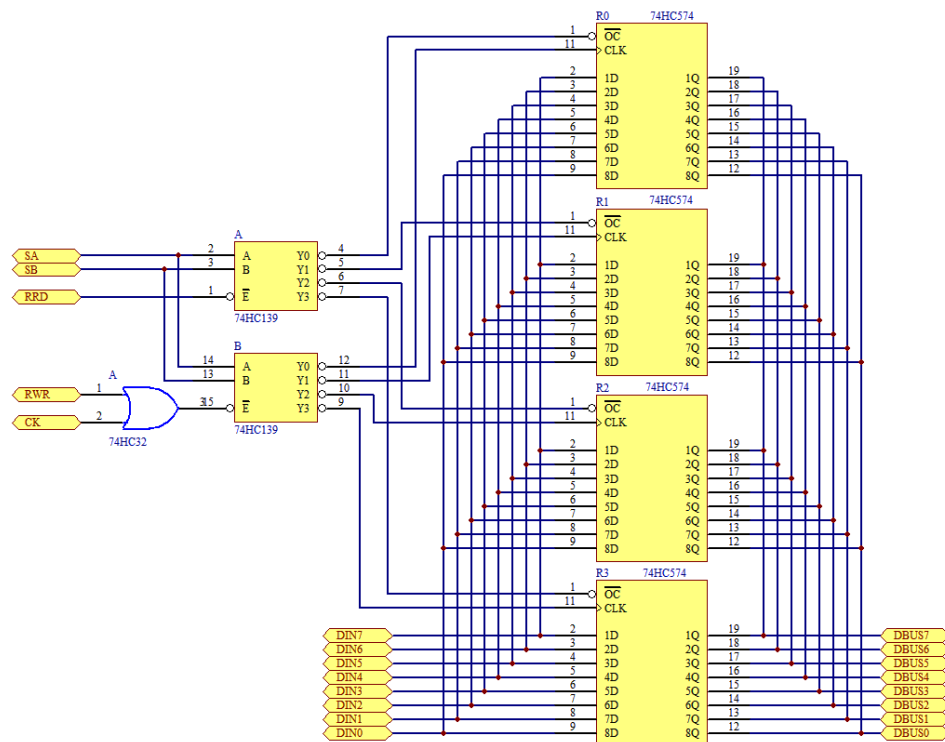
K4(WEN)	K3(AEN)
0	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 W 的黄色选择指示灯亮，表明选择 W 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 66H 被写入 W 寄存器。

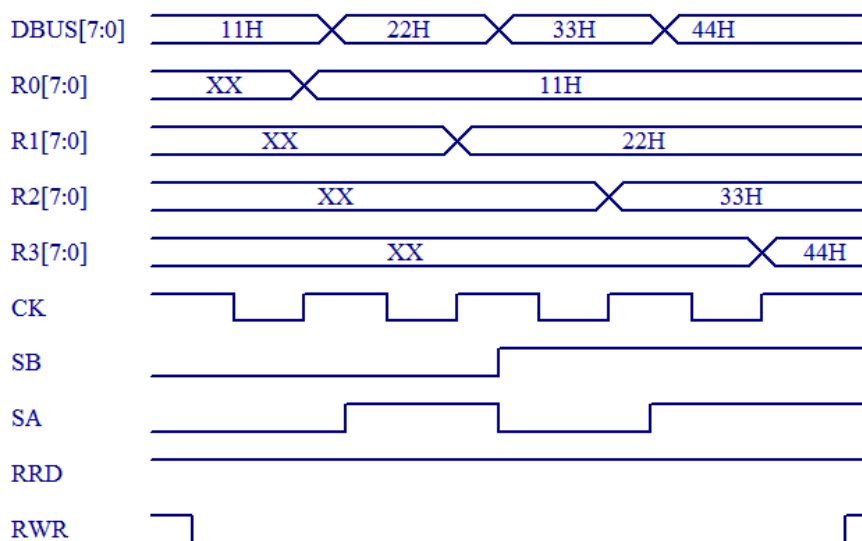
注意观察：

1. 数据是在放开 CLOCK 键后改变的，也就是 CLOCK 的上升沿数据被打入。
2. WEN, AEN 为高时，即使 CLOCK 有上升沿，寄存器的数据也不会改变。

实验 2: R0, R1, R2, R3 寄存器实验



寄存器 R 原理图



寄存器 R 写工作波形图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J1 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	RRD	K11	寄存器组读使能	低电平有效
3	RWR	K10	寄存器组写使能	低电平有效
4	SB	K1	寄存器选择 B	
5	SA	K0	寄存器选择 A	
6	RCK	CLOCK	寄存器工作脉冲	上升沿打入

将 11H 写入 R0 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 11H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	0	1

置控制信号为：

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
1	0	0	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 R0 的黄色选择指示灯亮，表明选择 R0 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 11H 被写入 R0 寄存器。

将 22H 写入 R1 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 22H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	0	0	0	1	0

置控制信号为：

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
1	0	0	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 R1 的黄色选择指示灯亮，表明选择 R1 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 22H 被写入 R1 寄存器。

将 33H 写入 R2 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 33H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	1	0	0	1	1

置控制信号为：

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
1	0	1	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 R2 的黄色选择指示灯亮，表明选择 R2 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 33H 被写入 R2 寄存器。

将 44H 写入 R3 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 44H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	0	0	0	1	0	0

置控制信号为：

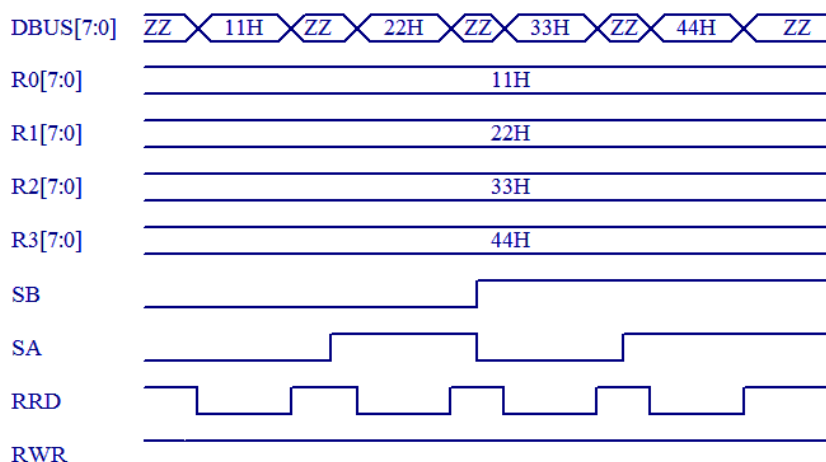
K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
1	0	1	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 R3 的黄色选择指示灯亮，表明选择 R3 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 44H 被写入 R3 寄存器。

注意观察：

1. 数据是在放开 CLOCK 键后改变的，也就是 CLOCK 的上升沿数据被打入。
2. K1(SB), K0(SA) 用于选择寄存器。

K1(SB)	K0(SA)	选择
0	0	R0
0	1	R1
1	0	R2
1	1	R3



寄存器 R 读工作波形图

读 R0 寄存器

置控制信号为:

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
0	1	0	0

这时寄存器 R0 的红色输出指示灯亮, R0 寄存器的数据送上数据总线。此时液晶显示 DBUS: 11 00010001. 将 K11(RRD)置为 1, 关闭 R0 寄存器输出。

读 R1 寄存器

置控制信号为:

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
0	1	0	1

这时寄存器 R1 的红色输出指示灯亮, R1 寄存器的数据送上数据总线。此时液晶显示 DBUS: 22 00100010. 将 K11(RRD)置为 1, 关闭 R1 寄存器输出。

读 R2 寄存器

置控制信号为:

K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
0	1	1	0

这时寄存器 R2 的红色输出指示灯亮, R2 寄存器的数据送上数据总线。此时液晶显示 DBUS: 33 00110011. 将 K11(RRD)置为 1, 关闭 R2 寄存器输出。

读 R3 寄存器

置控制信号为:

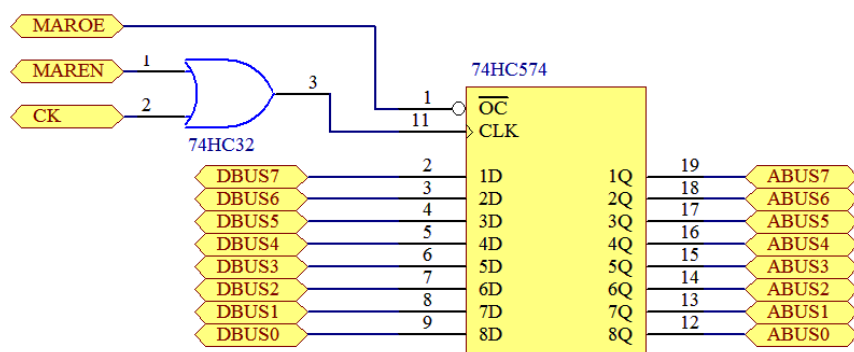
K11(RRD)	K10(RWR)	K1(SB)	K0(SA)
0	1	1	1

这时寄存器 R3 的红色输出指示灯亮, R3 寄存器的数据送上数据总线。此时液晶显示 DBUS: 44 01000100. 将 K11(RRD)置为 1, 关闭 R3 寄存器输出。

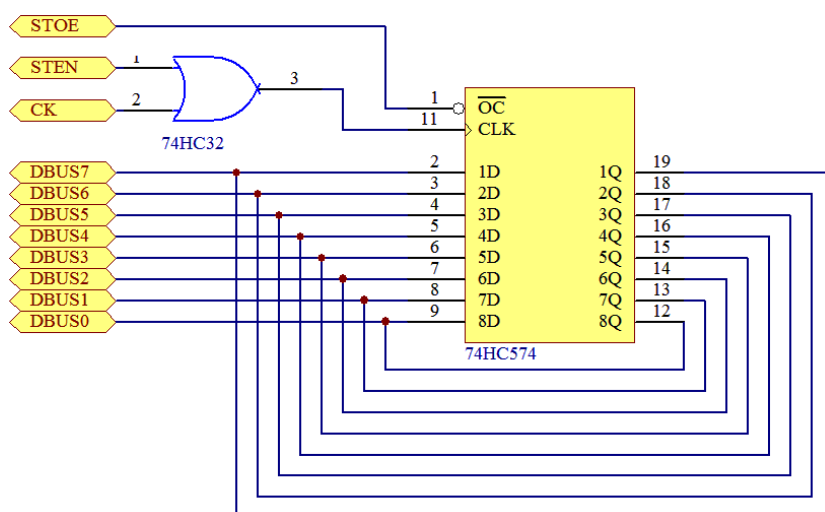
注意观察:

1. 数据在 K11(RRD)为 0 时输出, 不是沿触发, 与数据打入不同。

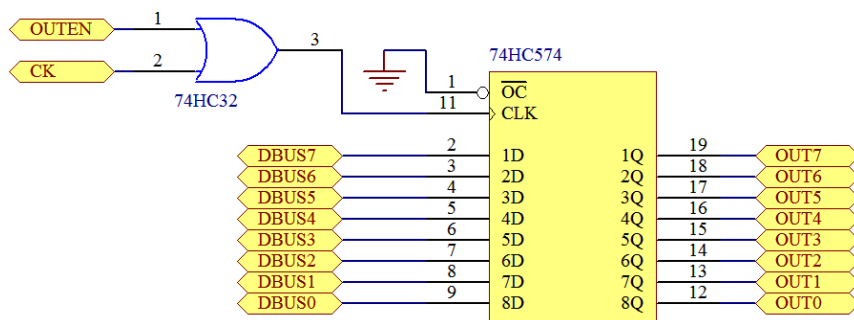
实验 3: MAR 地址寄存器, ST 堆栈寄存器, OUT 输出寄存器



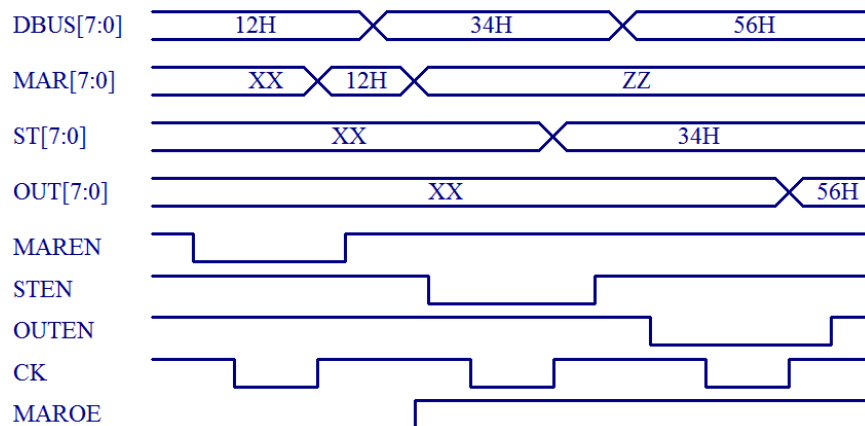
寄存器 MAR 原理图



寄存器 ST 原理图



寄存器 OUT 原理图



寄存器 MAR, ST, OUT 写工作波形图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J2 座	J3 座	将K23-K16接入DBUS[7:0]	
2	MAROE	K14	MAR 地址输出使能	低电平有效
3	MAREN	K15	MAR 寄存器写使能	低电平有效
4	STEN	K12	ST 寄存器写使能	低电平有效
5	OUTEN	K13	OUT 寄存器写使能	低电平有效
6	CK	CLOCK	寄存器工作脉冲	上升沿打入

将 12H 写入 MAR 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 12H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	1	0

置控制信号为：

K14(MAROE)	K15(MAREN)	K12(STEN)	K13(OUTEN)
0	0	1	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 MAR 的黄色选择指示灯亮，表明选择 MAR 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 12H 被写入 MAR 寄存器。

K14(MAROE)为 0, MAR 寄存器中的地址输出. MAR 红色输出指示灯亮.
将 K14(MAROE)置为 1. 关闭 MAR 输出.

将 34H 写入 ST 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 34H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	1	0	1	0	0

置控制信号为：

K14(MAROE)	K15(MAREN)	K12(STEN)	K13(OUTEN)
1	1	0	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 ST 的黄色选择指示灯亮，表明选择 ST 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 34H 被写入 ST 寄存器。

将 56H 写入 OUT 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 56H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	0	1	0	1	1	0

置控制信号为：

K14(MAROE)	K15(MAREN)	K12(STEN)	K13(OUTEN)
1	1	1	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 OUT 的黄色选择指示灯亮，表明选择 OUT 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 56H 被写入 OUT 寄存器。

2.2 运算器实验

实验要求：利用 COP2000 实验仪的 K16..K23 开关做为 DBUS 数据，其它开关做为控制信号，将数据写累加器 A 和工作寄存器 W，并用开关控制 ALU 的运算方式，实现运算器的功能。

实验目的：了解模型机中算术、逻辑运算单元的控制方法。

实验说明：

COP2000 中的运算器由一片可编程芯片 EPLD 实现。有 8 种运算，通过 S2,S1,S0 来选择。运算数据由寄存器 A 及寄存器 W 给出，运算结果输出到直通门 D。有兴趣的同学可以参考第 76 页实现本 ALU 功能的 ABLE 语言。了解 ALU 的实现方法。

S2 S1 S0	功能
0 0 0	A+W 加
0 0 1	A-W 减
0 1 0	A W 或
0 1 1	A&W 与
1 0 0	A+W+C 带进位加
1 0 1	A-W-C 带进位减
1 1 0	~A A 取反
1 1 1	A 输出 A

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J1 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	S0	K0	运算器功能选择	
3	S1	K1	运算器功能选择	
4	S2	K2	运算器功能选择	
5	AEN	K3	选通 A	低电平有效
6	WEN	K4	选通 W	低电平有效
7	Cy IN	K5	运算器进位输入	
8	ALUCK	CLOCK	ALU 工作脉冲	上升沿打入

将 55H 写入 A 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 55H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	0	1	0	1	0	1

置控制信号为：

K5(Cy IN)	K4(WEN)	K3(AEN)	K2(S2)	K1(S1)	K0(S0)
0	1	0	0	0	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 A 的黄色选择指示灯亮，表明选择 A 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 55H 被写入 A 寄存器。

将 33H 写入 W 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 33H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	1	0	0	1	1

置控制信号为：

K5(Cy IN)	K4(WEN)	K3(AEN)	K2(S2)	K1(S1)	K0(S0)
0	0	1	0	0	0

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 W 的黄色选择指示灯亮，表明选择 W 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 33H 被写入 W 寄存器。

置下表的控制信号，检验运算器的运算结果

K5(Cy IN)	K2(S2)	K1(S1)	K0(S0)	结果(直通门 D)	注释
X	0	0	0	88H	加运算
X	0	0	1	22H	减运算
X	0	1	0	77H	或运算
X	0	1	1	11H	与运算
0	1	0	0	88H	带进位加运算
1	1	0	0	89H	带进位加运算
0	1	0	1	22H	带进位减运算
1	1	0	1	21H	带进位减运算
X	1	1	0	AAH	取反运算
X	1	1	1	55H	输出 A

注意观察：

运算器在加上控制信号及数据(A,W)后，立刻给出结果，不须时钟。

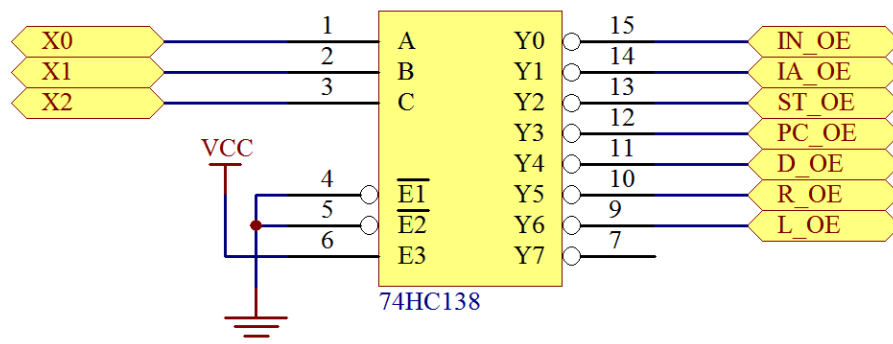
2.3 数据输出实验/移位门实验

实验要求：利用 COP2000 实验仪的开关做为控制信号，将指定寄存器的内容读到数据总线 DBUS 上。

实验目的：1。了解模型机中多寄存器接数据总线的实现原理。2。了解运算器中移位功能的实现方法。

实验说明：

COP2000 中有 7 个寄存器可以向数据总线输出数据，但在某一特定时刻只能有一个寄存器输出数据。由 X2,X1,X0 决定那一个寄存器输出数据。



数据输出选择器原理图

X2 X1 X0	输出寄存器
0 0 0	IN_OE 外部输入门
0 0 1	IA_OE 中断向量
0 1 0	ST_OE 堆栈寄存器
0 1 1	PC_OE PC 寄存器
1 0 0	D_OE 直通门
1 0 1	R_OE 右移门
1 1 0	L_OE 左移门
1 1 1	没有输出

连接线表

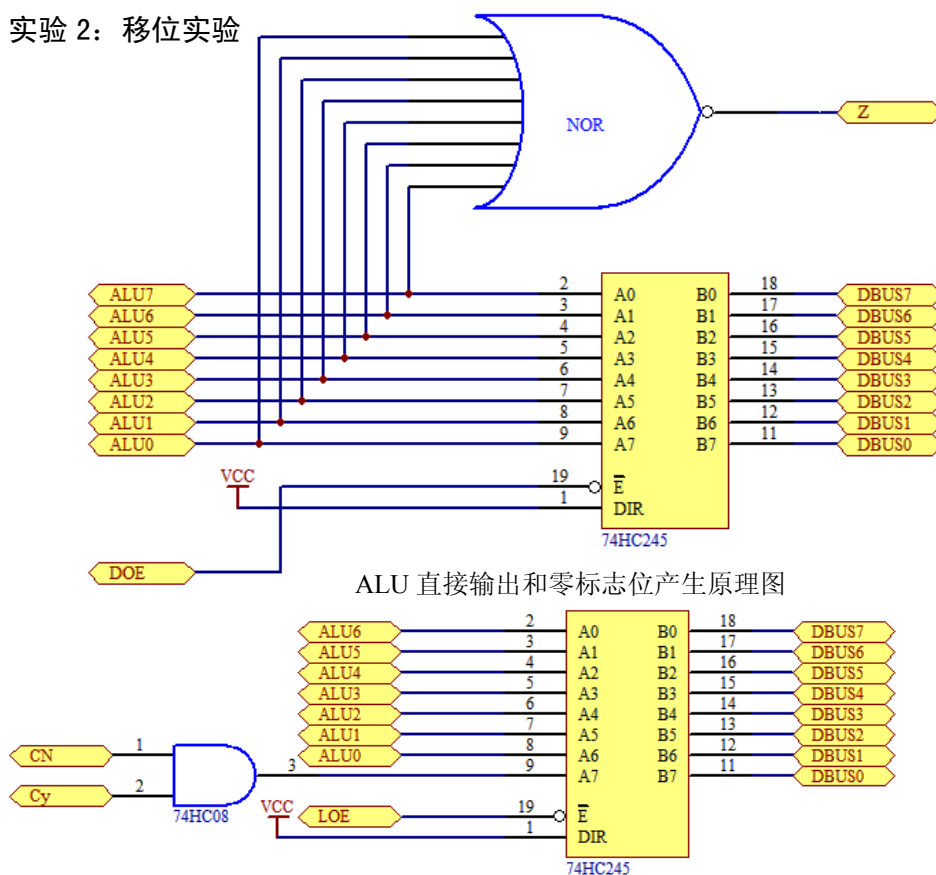
连接	信号孔	接入孔	作用	有效电平
1	J1 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	X0	K5	寄存器输出选择	
3	X1	K6	寄存器输出选择	
4	X2	K7	寄存器输出选择	
5	AEN	K3	选通 A	低电平有效
6	CN	K9	移位是否带进位	0:不带进位 1:带进位
7	Cy IN	K8	移位进位输入	
8	S2	K2	运算器功能选择	
9	S1	K1	运算器功能选择	
10	S0	K0	运算器功能选择	
11	ALUCK	CLOCK	ALU 工作脉冲	上升沿打入

实验 1：数据输出实验

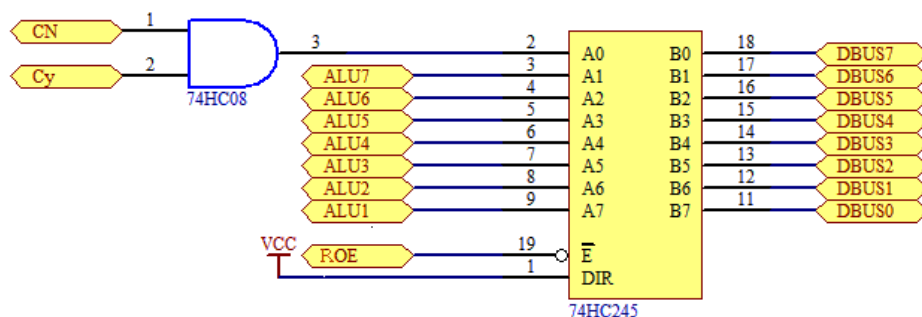
置下表的控制信号，检验输出结果

X2	X1	X0	指示灯（红色）	液晶显示（数据总线值）
0	0	0	IN 指示	输入门（K23-K16）
0	0	1	IA 指示	中断向量（由拨动开关给出）
0	1	0	ST 指示	堆栈寄存器
0	1	1	PC 指示	PC 寄存器
1	0	0	D 直通门指示	D 直通门
1	0	1	R 右移门指示	R 右移门
1	1	0	L 左移门指示	L 左移门
1	1	1		没有输出

实验 2：移位实验



ALU 左移输出原理图



ALU 右移输出原理图

直通门将运算器的结果不移位送总线。当 $X_2X_1X_0=100$ 时运算器结果通过直通门送到数据总线。同时，直通门上还有判 0 电路，当运算器的结果为全 0 时， $Z=1$

右移门将运算器的结果右移一位送总线。当 $X_2X_1X_0=101$ 时运算器结果通过右通门送到数据总线。具体连线是：

Cy 与 CN \rightarrow DBUS7

ALU7 \rightarrow DBUS6

ALU6 \rightarrow DBUS5

ALU5 \rightarrow DBUS4

ALU4 \rightarrow DBUS3

ALU3 \rightarrow DBUS2

ALU2 \rightarrow DBUS1

ALU1 \rightarrow DBUS0

Cy 与 CN \rightarrow DBUS7

当不带进位移位时(CN=0):

0 \rightarrow DBUS7

当带进位移位时(CN=1):

Cy \rightarrow DBUS7

左移门将运算器的结果左移一位送总线。当 $X_2X_1X_0=110$ 时运算器结果通过左通门送到数据总线。具体连线是：

ALU6 \rightarrow DBUS7

ALU5 \rightarrow DBUS6

ALU4 \rightarrow DBUS5

ALU3 \rightarrow DBUS4

ALU2 \rightarrow DBUS3

ALU1 \rightarrow DBUS2

ALU0 \rightarrow DBUS1

当不带进位移位时(CN=0):

0 \rightarrow DBUS0

当带进位移位时(CN=1):

Cy → DBUS0

将 55H 写入 A 寄存器

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 55H

K23	K22	K21	K20	K19	K18	K17	K16
0	1	0	1	0	1	0	1

置控制信号为：

K3(AEN)	K2(S2)	K1(S1)	K0(S0)
0	1	1	1

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 A 的黄色选择指示灯亮，表明选择 A 寄存器。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 55H 被写入 A 寄存器。

S2S1S0=111 时运算器结果为寄存器 A 内容

CN	Cy IN	L	D	R
0	X	AA 1010 1010	55 0101 0101	2A 0010 1010
1	0	AA 1010 1010	55 0101 0101	2A 0010 1010
1	1	AB 1010 1011	55 0101 0101	AA 1010 1010

注意观察：

移位与输出门是否打开无关，无论运算器结果如何，移位门都会给出移位结果。但究竟把那一个结果送数据总线由 X2X1X0 输出选择决定。

2.4 微程序计数器 uPC 实验

实验要求：利用 COP2000 实验仪上的 K16..K23 开关做为 DBUS 的数据，其它开关做为控制信号，实现微程序计数器 uPC 的写入和加 1 功能。

实验目的：1。了解模型机中微程序的基本概念。2。了解 uPC 的结构、工作原理及其控制方法。

实验说明：

74HC161 是一片带预置的 4 位二进制计数器。功能如下：

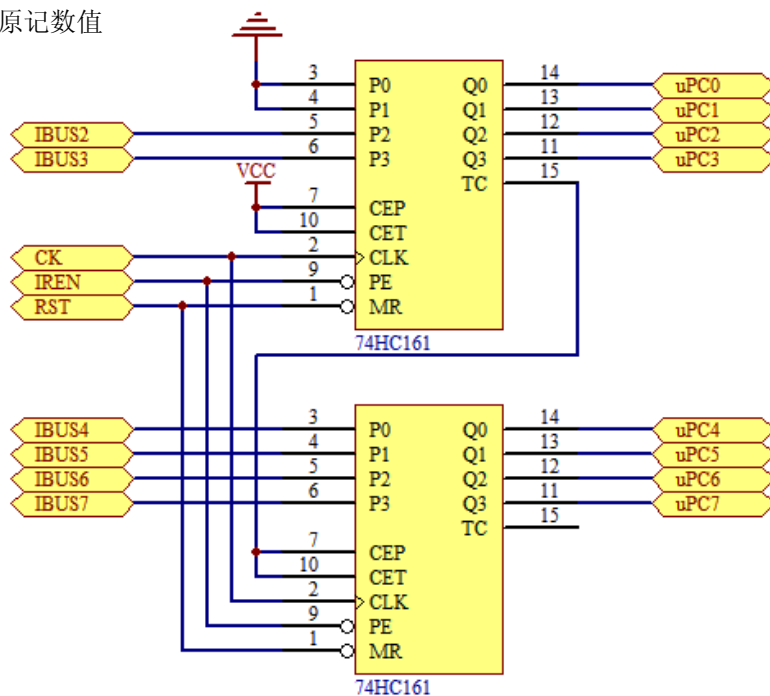
当 RST = 0 时，计数器被清 0

当 IREN = 0 时，在 CK 的上升沿，预置数据被打入计数器

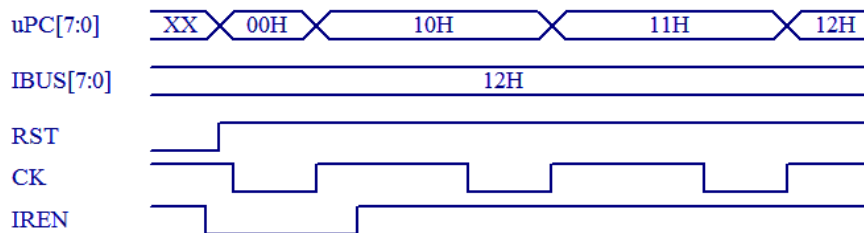
当 IREN = 1 时，在 CK 的上升沿，计数器加一

TC 为进位，当记数到 F (1111) 时，TC=1

CEP, CET 为记数使能，当 CEP, CET=1 时，计数器工作，CEP, CET=0 时，计数器保持原记数值



uPC 原理图



uPC 工作波形图

在 COP2000 中，指令 IBUS[7:0]的高六位被接到 uPC 预置的高六位，uPC 预置的低两位被置为 0。一条指令最多可有四条微指令。

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J2 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	IREN	K0	预置 uPC	低电平有效
3	EMEN	K1	EM 存储器工作使能	低电平有效
4	EMWR	K2	EM 存储器写使能	低电平有效
5	EMRD	K3	EM 存储器读使能	低电平有效
6	IRCK	CLOCK	uPC 工作脉冲	上升沿打入

实验 1：uPC 加一实验

置控制信号为：

K3(EMRD)	K2(EMWR)	K1(EMEN)	K0(IREN)
1	1	1	1

按一次 CLOCK 脉冲键，CLOCK 产生一个上升沿，数据 uPC 被加一。

实验 2：uPC 打入实验

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 12H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	1	0

置控制信号为：

K3(EMRD)	K2(EMWR)	K1(EMEN)	K0(IREN)
1	0	0	0

当 EMWR，EMEN=0 时，数据总线（DBUS）上的数据被送到指令总线（IBUS）上。

按住 CLOCK 脉冲键，CLOCK 由高变低，这时寄存器 uPC 的黄色预置指示灯亮，表明 uPC 被预置。放开 CLOCK 键，CLOCK 由低变高，产生一个上升沿，数据 10H 被写入 uPC 寄存器。

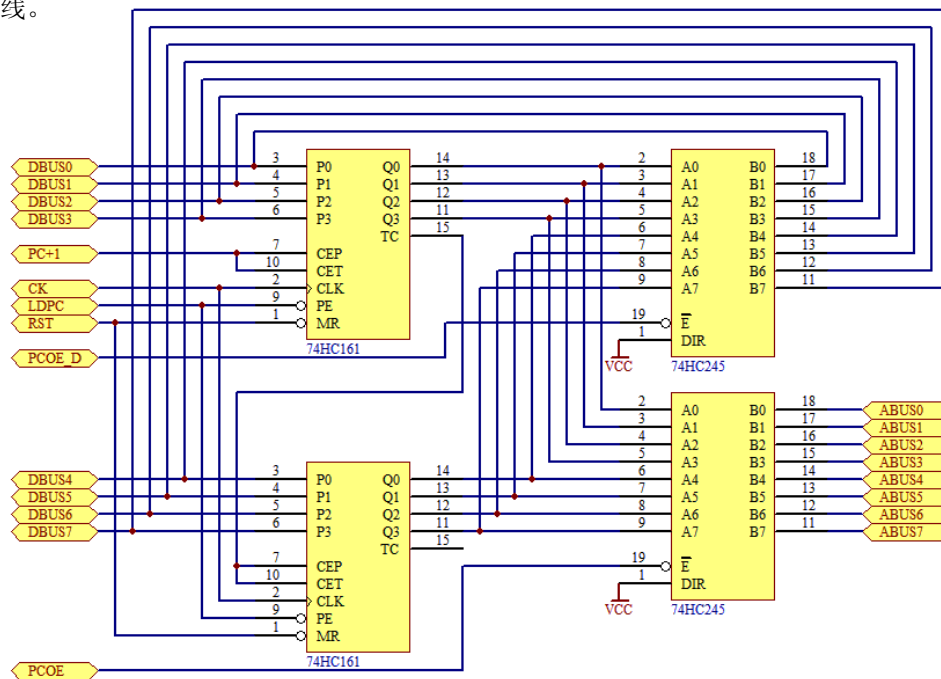
2.5 PC 实验

实验要求：利用 COP2000 实验仪上的 K16.K23 开关做为 DBUS 的数据，其它开关做为控制信号，实现程序计数器 PC 的写入及加 1 功能。

实验目的：1. 了解模型机中程序计数器 PC 的工作原理及其控制方法。2. 了解程序执行过程中跳转指令的实现方法。

实验说明：

PC 是由两片 74HC161 构成的八位带预置计数器，预置数据来自数据总线。计数器的输出通过 74HC245 (PCOE) 送到地址总线。PC 值还可以通过 74HC245 (PCOE_D) 送回数据总线。



PC 原理图

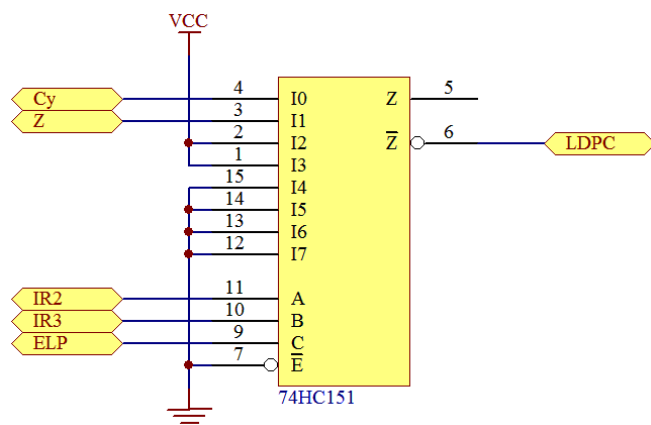
在 COP2000 中，PC+1 由 PCOE 取反产生。

当 RST = 0 时，PC 计数器被清 0

当 LDPC = 0 时，在 CK 的上升沿，预置数据被打入 PC 计数器

当 PC+1 = 1 时，在 CK 的上升沿，PC 计数器加一

当 PCOE = 0 时，PC 值送数据总线



PC 打入控制原理图

PC 打入控制电路由一片 74HC151 八选一构成。

ELP	IR3	IR2	Cy	Z	LDPC
1	X	X	X	X	1
0	0	0	1	X	0
0	0	0	0	X	1
0	0	1	X	1	0
0	0	1	X	0	1
0	1	X	X	X	0

当 ELP=1 时，LDPC=1，不允许 PC 被预置

当 ELP=0 时，LDPC 由 IR3，IR2，Cy，Z 确定

当 IR3 IR2 = 1 X 时，LDPC=0，PC 被预置

当 IR3 IR2 = 0 0 时，LDPC=非 Cy，当 Cy=1 时，PC 被预置

当 IR3 IR2 = 0 1 时，LDPC=非 Z，当 Z=1 时，PC 被预置

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J2 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	PCOE	K5	PC 输出到地址总线	低电平有效
3	JIR3	K4	预置选择 1	
4	JIR2	K3	预置选择 0	
5	JRZ	K2	Z 标志输入	
6	JRC	K1	C 标志输入	
7	ELP	K0	预置允许	低电平有效
8	PCCK	CLOCK	PC 工作脉冲	上升沿打入

实验 1：PC 加一实验

置控制信号为：

K5 (PCOE)	K0 (ELP)
0	1

按一次 CLOCK 脉冲键，CLOCK 产生一个上升沿，数据 PC 被加一。

实验 2：PC 打入实验

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 12H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	1	0

置控制信号为：

IR3 (K4)	IR2 (K3)	JRZ (K2)	JRC (K1)	ELP (K0)	LDPC	黄色 PC 预置指示灯
X	X	X	X	1	1	灭
0	0	X	1	0	0	亮
0	0	X	0	0	1	灭
0	1	1	X	0	0	亮
0	1	0	X	0	1	灭
1	X	X	X	0	0	亮

每置控制信号后，按一下 CLOCK 键，观察 PC 的变化。

2.6 存储器 EM 实验

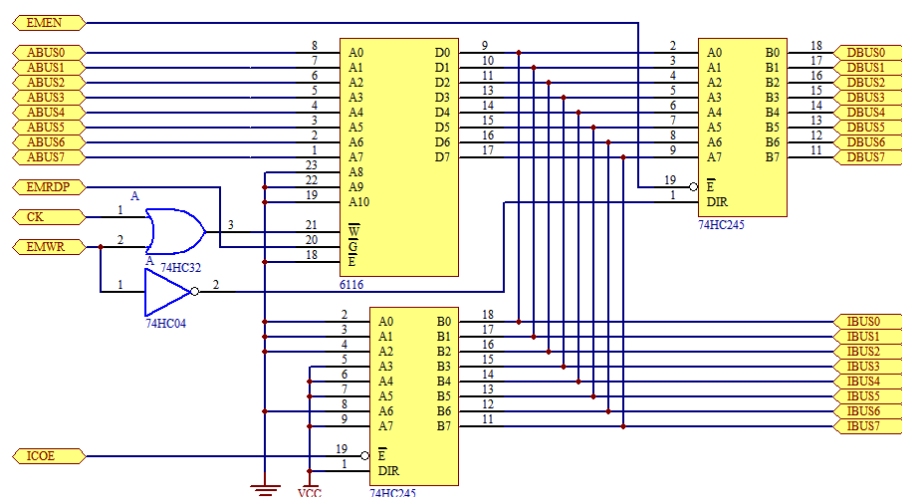
实验要求：利用 COP2000 实验仪上的 K16..K23 开关做为 DBUS 的数据，其它开关做为控制信号，实现程序存储器 EM 的读写操作。

实验目的：了解模型机中程序存储器 EM 的工作原理及控制方法。

实验说明：

存储器 EM 由一片 6116RAM 构成，通过一片 74HC245 与数据总线相连。存储器 EM 的地址可选择由 PC 或 MAR 提供。

存储器 EM 的数据输出直接接到指令总线 IBUS，指令总线 IBUS 的数据还可以来自一片 74HC245。当 ICOE 为 0 时，这片 74HC245 输出中断指令 B8。



EM 原理图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J2 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	IREN	K6	IR, uPC 写允许	低电平有效
3	PCOE	K5	PC 输出地址	低电平有效
4	MAROE	K4	MAR 输出地址	低电平有效
5	MAREN	K3	MAR 写允许	低电平有效
6	EMEN	K2	存储器与数据总线相连	低电平有效
7	EMRD	K1	存储器读允许	低电平有效
8	EMWR	K0	存储器写允许	低电平有效
9	PCCK	CLOCK	PC 工作脉冲	上升沿打入
10	MARCK	CLOCK	MAR 工作脉冲	上升沿打入
11	EMCK	CLOCK	写脉冲	上升沿打入
12	IRCK	CLOCK	IR, uPC 工作脉冲	上升沿打入

实验 1: PC/MAR 输出地址选择

置控制信号为:

K5 (PCOE)	K4 (MAROE)	地址总线	红色地址输出指示灯
0	1	PC 输出地址	PC 地址输出指示灯亮
1	0	MAR 输出地址	MAR 地址输出指示灯亮
1	1	地址总线浮空	
0	0	错误, PC 及 MAR 同时输出	PC 及 MAR 地址输出指示灯亮

以下存储器 EM 实验均由 MAR 提供地址

实验 2: 存储器 EM 写实验

将地址 0 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入, 置数据 00H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	0

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键, 将地址 0 写入 MAR

将地址 11H 写入 EM[0]

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入, 置数据 11H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	0	1

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	0	1	0	1	0

按 CLOCK 键, 将地址 11H 写入 EM[0]

将地址 1 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入, 置数据 01H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	1

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键, 将地址 1 写入 MAR

将地址 22H 写入 EM[1]

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入, 置数据 22H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	0	0	0	1	0

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	0	1	0	1	0

按 CLOCK 键, 将地址 22H 写入 EM[1]

实验 3: 存储器 EM 读实验

将地址 0 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入, 置数据 00H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	0

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键, 将地址 0 写入 MAR

读 EM[0]

置控制信号为:

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	0	1	1	0	1

EM[0]被读出: 11H

将地址 1 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 01H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	1

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键，将地址 0 写入 MAR

读 EM[1]

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	0	1	1	0	1

EM[1]被读出: 22H

实验 4：存储器打入 IR 指令寄存器/uPC 实验

将地址 0 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 00H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	0

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键，将地址 0 写入 MAR

读 EM[0]，写入 IR 及 uPC

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
0	1	0	1	1	0	1

EM[0]被读出: 11H

按 CLOCK 键，将 EM[0]写入 IR 及 uPC, IR = 11H, uPC=10H

将地址 1 写入 MAR

二进制开关 K23-K16 用于 DBUS[7:0]的数据输入，置数据 01H

K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	1

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
1	1	1	0	1	1	1

按 CLOCK 键，将地址 0 写入 MAR

读 EM[1]，写入 IR 及 uPC

置控制信号为：

K6 (IREN)	K5 (PCOE)	K4 (MAROE)	K3 (MAREN)	K2 (EMEN)	K1 (EMRD)	K0 (EMWR)
0	1	0	1	1	0	1

EM[1]被读出: 22H

按 CLOCK 键，将 EM[1]写入 IR 及 uPC, IR = 22H, uPC=20H

实验 5：使用实验仪小键盘输入 EM

1. 连接 J1, J2
2. 打开电源
3. 按 TAB 键，选择 EM
4. 输入两位地址, 00
5. 按 NEXT, 进入程序修改
6. 按两位程序数据
7. 按 NEXT 选择下个地址/按 LAST 选择上个地址
8. 重复 6,7 步输入程序
9. 按 RESET 结束

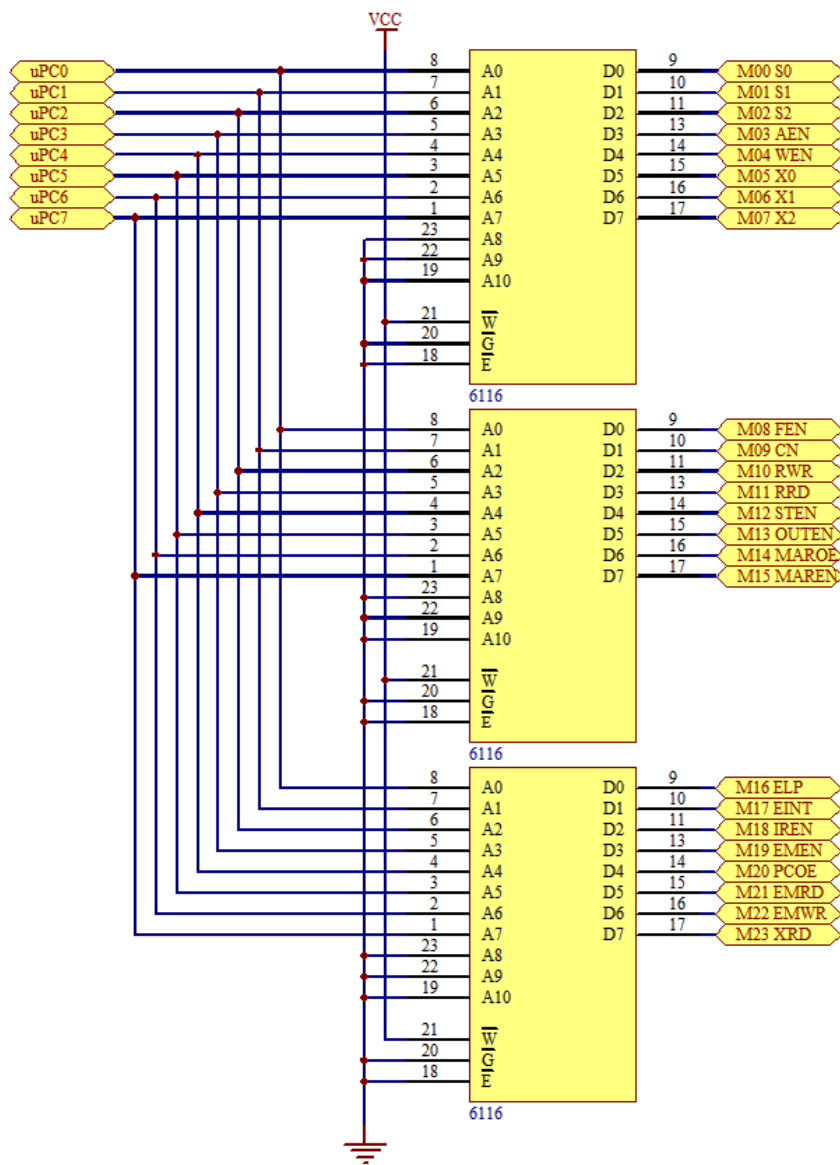
2.7 微程序存储器 uM 实验

实验要求：利用 COP2000 实验仪上的开关做为控制信号，实现微程序存储器 uM 的输出功能。

实验目的：1. 了解微程序控制方式模型机的基本工作原理。2. 了解微程序存储器 uM 的控制方法。

实验说明：

存储器 uM 由三片 6116RAM 构成，共 24 位微指令。存储器的地址由 uPC 提供，片选及读信号恒为低，写信号恒为高。存储器 uM 始终输出 uPC 指定地址单元的数据。



uM 原理图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	J2 座	J3 座	将 K23-K16 接入 DBUS[7:0]	
2	IREN	K0	IR, uPC 写使能	低电平有效
3	IRCK	CLOCK	uPC 工作脉冲	上升沿打入

实验 1：微程序存储器 uM 读出

置控制信号为：K0 为 1

uM 输出 uM[0]的数据

按一次 CLOCK 脉冲键，CLOCK 产生一个上升沿，数据 uPC 被加一。

uM 输出 uM[1]的数据

按一次 CLOCK 脉冲键，CLOCK 产生一个上升沿，数据 uPC 被加一。

uM 输出 uM[2]的数据

实验 2：使用实验仪小键盘输入 uM

1. 连接 J1, J2
2. 打开电源
3. 按 TAB 键, 选择 uM
4. 输入两位地址, 00
5. 按 NEXT, 进入微程序修改
6. 按六位微程序数据
7. 按 NEXT 选择下个地址/按 LAST 选择上个地址
8. 重复 6,7 步输入微程序
9. 按 RESET 结束

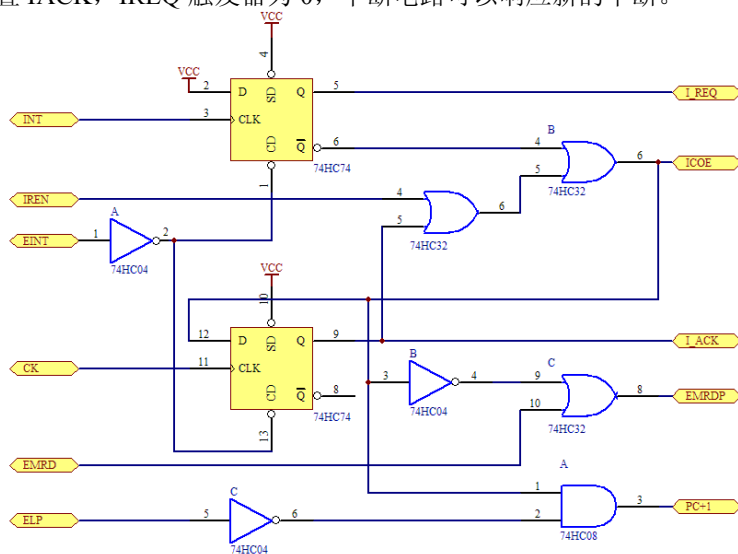
2.8 中断实验

实验要求：利用 COP2000 实验仪上的开关做控制信号，实现中断功能。

实验目的：1. 了解模型机的中断功能的工作原理及中断过程中，申请、响应、处理、返回各阶段时序。

实验说明：

中断电路有两个 D 触发器，分别用于保存中断请求信号(IREQ)及中断响应信号(IACK)。INT 有上升沿时，IREQ 触发器被置为 1。当下一条指令取指时(IREN=0)，存储器 EM 的读信号(EMRDP)被关闭，同时产生读中断指令(ICEN)信号，程序的执行被打断转而去执行 B8 指令响应中断。在取 B8 的同时置 IACK 触发器被置为 1，禁止新的中断响应。EINT 信号置 IACK，IREQ 触发器为 0，中断电路可以响应新的中断。



中断控制器原理图

连接线表

连接	信号孔	接入孔	作用	有效电平
1	IREN	K0	IR, uPC 写允许	低电平有效
2	EINT	K1	清中断寄存器	低电平有效
3	INT	INT 脉冲	中断输入	上升沿有效
4	CLOCK	CLOCK 脉冲	时钟输入	上升沿有效

按 INT 脉冲键，产生中断请求，此时黄色 IREQ 指示灯亮。

置控制信号为：

K0(IREN)	K1(EINT)
0	1

按 CLOCK 脉冲键，产生取指脉冲，黄色 IACK 指示灯亮，同时 B8 输出红色指示灯。

置控制信号为：

K0(IREN)	K1(EINT)
1	0

IREQ, IACK 灯灭。

第三章 COP2000 模型机

3.1 模型机总体结构

COP2000 模型机包括了一个标准 CPU 所具备所有部件, 这些部件包括: 运算器 ALU、累加器 A、工作寄存器 W、左移门 L、直通门 D、右移门 R、寄存器组 R0-R3、程序计数器 PC、地址寄存器 MAR、堆栈寄存器 ST、中断向量寄存器 IA、输入端口 IN、输出端口寄存器 OUT、程序存储器 EM、指令寄存器 IR、微程序计数器 uPC、微程序存储器 uM, 以及中断控制电路、跳转控制电路。其中运算器和中断控制电路以及跳转控制电路用 CPLD 来实现, 其它电路都是用离散的数字电路组成。微程序控制部分也可以用组合逻辑控制来代替。

模型机为 8 位机, 数据总线、地址总线都为 8 位, 但其工作原理与 16 位机相同。相比而言 8 位机实验减少了烦琐的连线, 但其原理却更容易被学生理解、吸收。

模型机的指令码为 8 位, 根据指令类型的不同, 可以有 0 到 2 个操作数。指令码的最低两位用来选择 R0-R3 寄存器, 在微程序控制方式中, 用指令码做为微地址来寻址微程序存储器, 找到执行该指令的微程序。而在组合逻辑控制方式中, 按时序用指令码产生相应的控制位。在本模型机中, 一条指令最多分四个状态周期, 一个状态周期为一个时钟脉冲, 每个状态周期产生不同的控制逻辑, 实现模型机的各种功能。模型机有 24 位控制位以控制寄存器的输入、输出, 选择运算器的运算功能, 存储器的读写。24 位控制位分别介绍如下:

- XRD : 外部设备读信号, 当给出了外设的地址后, 输出此信号, 从指定外设读数据。
- EMWR: 程序存储器 EM 写信号。
- EMRD: 程序存储器 EM 读信号。
- PCOE: 将程序计数器 PC 的值送到地址总线 ABUS 上。
- EMEN: 将程序存储器 EM 与数据总线 DBUS 接通, 由 EMWR 和 EMRD 决定是将 DBUS 数据写到 EM 中, 还是从 EM 读出数据送到 DBUS。
- IREN: 将程序存储器 EM 读出的数据打入指令寄存器 IR 和微指令计数器 uPC。
- EINT: 中断返回时清除中断响应和中断请求标志, 便于下次中断。
- ELP: PC 打入允许, 与指令寄存器的 IR3、IR2 位结合, 控制程序跳转。

- MAREN: 将数据总线 DBUS 上数据打入地址寄存器 MAR。
- MAROE: 将地址寄存器 MAR 的值送到地址总线 ABUS 上。
- OUTEN: 将数据总线 DBUS 上数据送到输出端口寄存器 OUT 里。
- STEN: 将数据总线 DBUS 上数据存入堆栈寄存器 ST 中。
- RRD: 读寄存器组 R0-R3, 寄存器 R? 的选择由指令的最低两位决定。
- RWR: 写寄存器组 R0-R3, 寄存器 R? 的选择由指令的最低两位决定。
- CN: 决定运算器是否带进位移位, CN=1 带进位, CN=0 不带进位。
- FEN: 将标志位存入 ALU 内部的标志寄存器。

X2: X2、X1、X0 三位组合来译码选择将数据送到 DBUS 上的寄存器。
X1: 见 16 页表。
X0:
WEN: 将数据总线 DBUS 的值打入工作寄存器 W 中。
AEN: 将数据总线 DBUS 的值打入累加器 A 中。
S2: S2、S1、S0 三位组合决定 ALU 做何种运算。
S1: 见 14 页表。
S0:

3.2 模型机寻址方式

模型机的寻址方式分五种:

累加器寻址: 操作数为累加器 A, 例如 “CPL A” 是将累加器 A 值取反, 还有些指令是隐含寻址累加器 A, 例如 “OUT” 是将累加器 A 的值输出到输出端口寄存器 OUT。
寄存器寻址: 参与运算的数据在 R0-R3 的寄存器中, 例如 “ADD A, R0” 指令是将寄存器 R0 的值加上累加器 A 的值, 再存入累加器 A 中。
寄存器间接寻址: 参与运算的数据在存储器 EM 中, 数据的地址在寄存器 R0-R3 中, 例如 “MOV A, @R1” 指令是将寄存器 R1 的值做为地址, 把存储器 EM 中该地址的内容送入累加器 A 中。
存储器直接寻址: 参与运算的数据在存储器 EM 中, 数据的地址为指令的操作数。例如 “AND A, 40H” 指令是将存储器 EM 中 40H 单元的数据与累加器 A 的值做逻辑与运算, 结果存入累加器 A。
立即数寻址: 参与运算的数据为指令的操作数。例如 “SUB A, #10H” 是从累加器 A 中减去立即数 10H, 结果存入累加器 A。

3.3 模型机指令集

模型机的缺省的指令集分几大类: 算术运算指令、逻辑运算指令、移位指令、数据传输指令、跳转指令、中断返回指令、输入/输出指令。用户可以通过 COP2000 计算机组成原理实验软件或组成原理实验仪来设计自己的指令集, 有关如何设计指令/微指令的介绍将在后面第六章说明。

助记符	机器码1	机器码2	注释
FATCH	000000xx		实验机占用，不可修改。复位后，所有寄存器清0，首先执行 _FATCH_ 指令取指
	000001xx		未使用
	000010xx		未使用
	000011xx		未使用
ADD A, R?	000100xx		将寄存器R?的值加入累加器A中
ADD A, @R?	000101xx		将间址存储器的值加入累加器A中
ADD A, MM	000110xx	MM	将存储器MM地址的值加入累加器A中
ADD A, #II	000111xx	II	将立即数II加入累加器A中
ADDC A, R?	001000xx		将寄存器R?的值加入累加器A中，带进位
ADDC A, @R?	001001xx		将间址存储器的值加入累加器A中，带进位
ADDC A, MM	001010xx	MM	将存储器MM地址的值加入累加器A中，带进位
ADDC A, #II	001011xx	II	将立即数II加入累加器A中，带进位
SUB A, R?	001100xx		从累加器A中减去寄存器R?的值
SUB A, @R?	001101xx		从累加器A中减去间址存储器的值
SUB A, MM	001110xx	MM	从累加器A中减去存储器MM地址的值
SUB A, #II	001111xx	II	从累加器A中减去立即数II加入累加器A中
SUBC A, R?	010000xx		从累加器A中减去寄存器R?的值，减进位
SUBC A, @R?	010001xx		从累加器A中减去间址存储器的值，减进位
SUBC A, MM	010010xx	MM	从累加器A中减去存储器MM地址的值，减进位
SUBC A, #II	010011xx	II	从累加器A中减去立即数II，减进位
AND A, R?	010100xx		累加器A“与”寄存器R?的值
AND A, @R?	010101xx		累加器A“与”间址存储器的值
AND A, MM	010110xx	MM	累加器A“与”存储器MM地址的值
AND A, #II	010111xx	II	累加器A“与”立即数II
OR A, R?	011000xx		累加器A“或”寄存器R?的值
OR A, @R?	011001xx		累加器A“或”间址存储器的值
OR A, MM	011010xx	MM	累加器A“或”存储器MM地址的值
OR A, #II	011011xx	II	累加器A“或”立即数II
MOV A, R?	011100xx		将寄存器R?的值送到累加器A中
MOV A, @R?	011101xx		将间址存储器的值送到累加器A中
MOV A, MM	011110xx	MM	将存储器MM地址的值送到累加器A中
MOV A, #II	011111xx	II	将立即数II送到累加器A中
MOV R?, A	100000xx		将累加器A的值送到寄存器R?中
MOV @R?, A	100001xx		将累加器A的值送到间址存储器中
MOV MM, A	100010xx	MM	将累加器A的值送到存储器MM地址中
MOV R?, #II	100011xx	II	将立即数II送到寄存器R?中
READ MM	100100xx	MM	从外部地址MM读入数据，存入累加器A中
WRITE MM	100101xx	MM	将累加器A中数据写到外部地址MM中
	100110xx		未使用
	100111xx		未使用
JC MM	101000xx	MM	若进位标志置1，跳转到MM地址
JZ MM	101001xx	MM	若零标志位置1，跳转到MM地址
	101010xx		未使用
JMP MM	101011xx	MM	跳转到MM地址
	101100xx		未使用
	101101xx		未使用
INT	101110xx		实验机占用，不可修改。进入中断时，实验机硬件产生 _INT_ 指令
CALL MM	101111xx	MM	调用MM地址的子程序

IN		110000xx	从输入端口读入数据到累加器A中
OUT		110001xx	将累加器A中数据输出到输出端口
		110010xx	未使用
RET		110011xx	子程序返回
RR	A	110100xx	累加器A右移
RL	A	110101xx	累加器A左移
RRC	A	110110xx	累加器A带进位右移
RLC	A	110111xx	累加器A带进位左移
NOP		111000xx	空指令
CPL	A	111001xx	累加器A取反，再存入累加器A中
		111010xx	未使用
RETI		111011xx	中断返回
		111100xx	未使用
		111101xx	未使用
		111110xx	未使用
		111111xx	未使用

3.4 模型机微指令集



Microsoft Excel
工作表

指令/微指令表

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
FATCH	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		01	FFFFFF				A输出		1	
		02	FFFFFF				A输出		1	
		03	FFFFFF				A输出		1	
UNDEF	T0	04	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		05	FFFFFF				A输出		1	
		06	FFFFFF				A输出		1	
		07	FFFFFF				A输出		1	
UNDEF	T0	08	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		09	FFFFFF				A输出		1	
		0A	FFFFFF				A输出		1	
		0B	FFFFFF				A输出		1	
UNDEF	T0	0C	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		0D	FFFFFF				A输出		1	
		0E	FFFFFF				A输出		1	
		0F	FFFFFF				A输出		1	
ADD A, R?	T2	10	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	11	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	12	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		13	FFFFFF				A输出		1	
ADD A, @R?	T3	14	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	15	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	16	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	17	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
ADD A, MM	T3	18	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	19	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	1A	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	1B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
ADD A, #II	T2	1C	C7FFE9	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	1D	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	1E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		1F	FFFFFF				A输出		1	

指令/微指令表 (续1)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
ADDC A, R?	T2	20	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	21	FFFE94	ALU直通	寄存器A 标志位C, Z		带进位加运算		1	
	T0	22	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		23	FFFFFF				A输出		1	
ADDC A, @R?	T3	24	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	25	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	26	FFFE94	ALU直通	寄存器A 标志位C, Z		带进位加运算		1	
	T0	27	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
ADDC A, MM	T3	28	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	29	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	2A	FFFE94	ALU直通	寄存器A 标志位C, Z		带进位加运算		1	
	T0	2B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
ADDC A, #II	T2	2C	C7FFE9	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	2D	FFFE94	ALU直通	寄存器A 标志位C, Z		带进位加运算		1	
	T0	2E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		2F	FFFFFF				A输出		1	
SUB A, R?	T2	30	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	31	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	32	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		33	FFFFFF				A输出		1	
SUB A, @R?	T3	34	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	35	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	36	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	37	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
SUB A, MM	T3	38	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	39	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	3A	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	3B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
SUB A, #II	T2	3C	C7FFE9	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	3D	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	3E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		3F	FFFFFF				A输出		1	

指令/微指令表 (续2)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
SUBC A, R?	T2	40	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	41	FFFE95	ALU直通	寄存器A 标志位C, Z		带进位减运算		1	
	T0	42	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		43	FFFFFF				A输出		1	
SUBC A, @R?	T3	44	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	45	D7BF EF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	46	FFFE95	ALU直通	寄存器A 标志位C, Z		带进位减运算		1	
	T0	47	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
SUBC A, MM	T3	48	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	49	D7BF EF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	4A	FFFE95	ALU直通	寄存器A 标志位C, Z		带进位减运算		1	
	T0	4B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
SUBC A, #II	T2	4C	C7FFE F	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	4D	FFFE95	ALU直通	寄存器A 标志位C, Z		带进位减运算		1	
	T0	4E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		4F	FFFFFF				A输出		1	
AND A, R?	T2	50	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	51	FFFE93	ALU直通	寄存器A 标志位C, Z		与运算		1	
	T0	52	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		53	FFFFFF				A输出		1	
AND A, @R?	T3	54	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	55	D7BF EF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	56	FFFE93	ALU直通	寄存器A 标志位C, Z		与运算		1	
	T0	57	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
AND A, MM	T3	58	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	59	D7BF EF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	5A	FFFE93	ALU直通	寄存器A 标志位C, Z		与运算		1	
	T0	5B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
AND A, #II	T2	5C	C7FFE F	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	5D	FFFE93	ALU直通	寄存器A 标志位C, Z		与运算		1	
	T0	5E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		5F	FFFFFF				A输出		1	

指令/微指令表 (续3)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
OR A, R?	T2	60	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	61	FFFE92	ALU直通	寄存器A 标志位C, Z		或运算		1	
	T0	62	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		63	FFFFFF				A输出		1	
OR A, @R?	T3	64	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	65	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	66	FFFE92	ALU直通	寄存器A 标志位C, Z		或运算		1	
	T0	67	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
OR A, MM	T3	68	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	69	D7BFEF	存贮器值EM	寄存器W	MAR输出	A输出		1	
	T1	6A	FFFE92	ALU直通	寄存器A 标志位C, Z		或运算		1	
	T0	6B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
OR A, #II	T2	6C	C77FEF	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	6D	FFFE92	ALU直通	寄存器A 标志位C, Z		或运算		1	
	T0	6E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		6F	FFFFFF				A输出		1	
MOV A, R?	T1	70	FFF7F7	寄存器值R?	寄存器A		A输出		1	
	T0	71	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		72	FFFFFF				A输出		1	
		73	FFFFFF				A输出		1	
MOV A, @R?	T2	74	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T1	75	D7BFF7	存贮器值EM	寄存器A	MAR输出	A输出		1	
	T0	76	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		77	FFFFFF				A输出		1	
MOV A, MM	T2	78	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	79	D7BFF7	存贮器值EM	寄存器A	MAR输出	A输出		1	
	T0	7A	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		7B	FFFFFF				A输出		1	
MOV A, #II	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		7E	FFFFFF				A输出		1	
		7F	FFFFFF				A输出		1	

指令/微指令表 (续4)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
MOV R?, A	T1	80	FFFB9F	ALU直通	寄存器R?		A输出		1	
	T0	81	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		82	FFFFFF				A输出		1	
		83	FFFFFF				A输出		1	
MOV @R?, A	T2	84	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T1	85	B7BF9F	ALU直通	存贮器EM	MAR输出	A输出		1	
	T0	86	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		87	FFFFFF				A输出		1	
MOV MM, A	T2	88	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	89	B7BF9F	ALU直通	存贮器EM	MAR输出	A输出		1	
	T0	8A	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		8B	FFFFFF				A输出		1	
MOV R?, #II	T1	8C	C7FBFF	存贮器值EM	寄存器R?	PC输出	A输出		1	1
	T0	8D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		8E	FFFFFF				A输出		1	
		8F	FFFFFF				A输出		1	
READ A, MM	T2	90	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	91	7FBFF7		寄存器A	MAR输出	A输出		1	
	T0	92	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		93	FFFFFF				A输出		1	
WRITE MM, A	T2	94	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	95	FF9F9F	ALU直通	用户OUT	MAR输出	A输出		1	
	T0	96	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		97	FFFFFF				A输出		1	
UNDEF	T0	98	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		99	FFFFFF				A输出		1	
		9A	FFFFFF				A输出		1	
		9B	FFFFFF				A输出		1	
UNDEF	T0	9C	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		9D	FFFFFF				A输出		1	
		9E	FFFFFF				A输出		1	
		9F	FFFFFF				A输出		1	

指令/微指令表 (续5)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
JC MM	T1	A0	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		A2	FFFFFF				A输出		1	
		A3	FFFFFF				A输出		1	
JZ MM	T1	A4	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		A6	FFFFFF				A输出		1	
		A7	FFFFFF				A输出		1	
UNDEF	T0	A8	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		A9	FFFFFF				A输出		1	
		AA	FFFFFF				A输出		1	
		AB	FFFFFF				A输出		1	
JMP MM	T1	AC	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	AD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		AE	FFFFFF				A输出		1	
		AF	FFFFFF				A输出		1	
UNDEF	T0	B0	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		B1	FFFFFF				A输出		1	
		B2	FFFFFF				A输出		1	
		B3	FFFFFF				A输出		1	
UNDEF	T0	B4	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		B5	FFFFFF				A输出		1	
		B6	FFFFFF				A输出		1	
		B7	FFFFFF				A输出		1	
INT	T2	B8	FFEF7F	PC值	堆栈寄存器ST		A输出		1	
	T1	B9	FEFF3F	中断地址IA	寄存器PC		A输出	带进位右移	1	写入
	T0	BA	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		BB	FFFFFF				A输出		1	
CALL MM	T3	BC	EF7F7F	PC值	地址寄存器MAR	PC输出	A输出		1	1
	T2	BD	FFEF7F	PC值	堆栈寄存器ST		A输出		1	
	T1	BE	D6BFFF	存贮器值EM	寄存器PC	MAR输出	A输出		1	写入
	T0	BF	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1

指令/微指令表 (续6)

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
IN	T1	C0	FFFF17	用户IN	寄存器A		A输出		1	
	T0	C1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		C2	FFFFFF				A输出		1	
		C3	FFFFFF				A输出		1	
OUT	T1	C4	FFDF9F	ALU直通	用户OUT		A输出		1	
	T0	C5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		C6	FFFFFF				A输出		1	
		C7	FFFFFF				A输出		1	
UNDEF	T0	C8	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		C9	FFFFFF				A输出		1	
		CA	FFFFFF				A输出		1	
		CB	FFFFFF				A输出		1	
RET	T1	CC	FEFF5F	堆栈寄存器ST	寄存器PC		A输出	带进位左移	1	写入
	T0	CD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		CE	FFFFFF				A输出		1	
		CF	FFFFFF				A输出		1	
RR A	T1	D0	FFFCB7	ALU右移	寄存器A 标志位C, Z		A输出	右移	1	
	T0	D1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		D2	FFFFFF				A输出		1	
		D3	FFFFFF				A输出		1	
RL A	T1	D4	FFLCD7	ALU左移	寄存器A 标志位C, Z		A输出	左移	1	
	T0	D5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		D6	FFFFFF				A输出		1	
		D7	FFFFFF				A输出		1	
RRC A	T1	D8	FFFE7B	ALU右移	寄存器A 标志位C, Z		A输出	带进位右移	1	
	T0	D9	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		DA	FFFFFF				A输出		1	
		DB	FFFFFF				A输出		1	
RLC A	T1	DC	FFFD7	ALU左移	寄存器A 标志位C, Z		A输出	带进位左移	1	
	T0	DD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
		DE	FFFFFF				A输出		1	
		DF	FFFFFF				A输出		1	

指令/微指令表 (续7)

[illegible]

第四章 模型机综合实验(微程序控制器)

在综合实验中，模型机作为一个整体来工作的，所有微程序的控制信号由微程序存储器 uM 输出，而不是由开关输出。在做综合实验之前，先用 8 芯电缆连接 J1 和 J2，这样实验仪的监控系统会自动打开 uM 的输出允许，微程序的各控制信号就会接到各寄存器、运算器的控制端口。此综合实验使用的指令是模型机的缺省指令/微指令系统。等做完本综合实验，熟悉了这套指令/微指令后，用户可以自己设计的指令/微指令系统，有关自己设计指令/微指令系统的说明在下一章介绍。

在做综合实验时，可以用 COP2000 计算机组成原理实验软件输入、修改程序，汇编成机器码并下载到实验仪上，由软件控制程序实现单指令执行、单微指令执行、全速执行，并可以在软件上观察指令或微指令执行过程中数据的走向、各控制信号的状态、各寄存器的值。COP2000 软件的使用方法见第七章“COP2000 集成开发环境使用”。也可以用实验仪自带的小键盘和显示屏来输入、修改程序，用键盘控制单指令或单微指令执行，用 LED 或用显示屏观察各寄存器的值。实验仪上的键盘使用方法见第六章“实验仪键盘使用”。

在用微程序控制方式做综合实验时，在给实验仪通电前，拔掉实验仪上所有的手工连接的接线，再用 8 芯电缆连接 J1 和 J2，控制方式开关拨到“微程序控制”方向。若想用 COP2000 软件控制组成原理实验仪，就要启动软件，并用快捷图标的“设置”功能打开设置窗口，选择实验仪连接的串行口，然后再按“连接 COP2000 实验仪”按钮接通到实验仪。

实验 1：数据传送实验/输入输出实验

1. 在 COP2000 软件中的源程序窗口输入下列程序

```
MOV    A, #12H
MOV    A, R0
MOV    A, @R0
MOV    A, 01H
IN
OUT
END
```

2. 将程序另存为 EX1.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	7C 12	MOV A, #12	立即数 12H 送到累加器 A
02	70	MOV A, R0	寄存器 R0 送到累加器 A
03	74	MOV A, @R0	R0 间址的存储器内容送到累加器 A
04	78 01	MOV A, 01	存储器 01 单元内容送到累加器 A
06	C0	IN	端口 IN 内容输入到累加器 A
07	C4	OUT	累加器 A 内容输出到端口 OUT

3. 按快捷图标的 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。（见 EX1.ASM 程序跟踪结果）

EX1.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #12	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 MOV A, R0	T1	70	FFF7F7	寄存器值R?	寄存器A		A输出		1	
	T0	71	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
03 MOV A, @R0	T2	74	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T1	75	D7BFF7	存贮器值EM	寄存器A	MAR输出	A输出		1	
	T0	76	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 MOV A, 01	T2	78	C77FFF	存贮器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	79	D7BFF7	存贮器值EM	寄存器A	MAR输出	A输出		1	
	T0	7A	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
06 IN	T1	C0	FFFF17	用户IN	寄存器A		A输出		1	
	T0	C1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
07 OUT	T1	C4	FFDF9F	ALU直通	用户OUT		A输出		1	
	T0	C5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1

每个程序的一开始的第一条微指令一定是取指令，此微指令的值为0CBFFFFH，对应到各个控制位就是EMRD、PCOE、及IREN为低，此三位有效，其它所有位都处于无效状态。在程序第一次运行时或复位后，uPC和PC的值都为0，PCOE有效将PC值送到ABUS，做为程序存储器EM的地址，EMRD信号有效就是从程序存储器EM中读出程序指令，IREN将读出的指令送到IR寄存器和uPC，此微指令的作用就是从程序存储器EM的0地址读出程序指令机器码7CH，并存入uPC中做为微程序存储器uM的地址，从微程序存储器uM的7CH单元中读出微控制码0C7FFF7H，同时PC加1为读下一条指令或数据做准备。

MOV A, #12: 本指令为两个状态周期。在T1状态时，上次读出的指令机器码为7CH，存入uPC中做为微程序存储器uM的地址，读出微指令的值为0C7FFF7H，对应到各个控制位就是EMRD、PCOE、EMEN及AEN为低，处于有效状态，其它控制位为无效状态。由于上条微指令（取指操作）已将PC加1，此时PCOE是将加1后的PC输出到ABUS做为程序存储器EM的地址，EMRD就是从程序存储器EM中读出数据，本指令中读出的数据应为12H，EMEN将读出的数据送到DBUS总线上，AEN 是将DBUS总线上的值存入累加器A中。同时uPC加1为执行下条微指令做准备，PC加1为读取下一条指令做准备。每条指令的最后一条微指令一定是取指令操作，本指令的T0状态周期即为取指令，执行上一条微指令时uPC已经加1，按照此uPC为地址从微程序存储器uM读出的微指令的值为0CBFFFFH，参照第一步的说明，此微指令从程序存储器EM中读取指令。

MOV A, R0: 本指令为两个状态周期。在T1状态时，由上条取指操作取出的指令机器码为70H，存入uPC后做为微程序地址访问微程序存储器uM的70H单元，读出微指令的值为0FFF7F7，各控制位的状态为RRD、AEN为低电平为有效状态，RRD有效表示从寄存器组R0-R3中读数送到DBUS上，在上条取指令操作时，IREN将取出的指令机器码70H送入IR寄存器，而IR寄存器的最低两位是用来选择寄存

器R?的，此时IR寄存器最低两位为00，被读出的寄存器为R0。AEN有效表示将DBUS的数据写到累加器A中。同时uPC加1，为执行下条微指令做准备。本指令的T0状态也是取指令，完成的功能是取出下一条要执行的指令机器码，并存入uPC和IR寄存器中。

MOV A, @R0 本指令为三个状态周期。在T2状态时，由上个取指操作读出的指令机器码为74H，打入uPC后，从微程序存储器74H单元读出的微指令的值为0FF77FFH，有效的控制位为MAREN和RRD，RRD有效表示从寄存器组R0-R3中读出数据送到DBUS，MAREN有效表示将数据从DBUS总线上打入地址寄存器MAR。uPC加1取出下条微指令执行。在T1状态时，由uPC做为微程序存储器地址，从uM的75H单元中读出微指令的值为0D7BFF7H，其中有效的控制位为EMRD、EMEN、MAROE和AEN。MAROE表示程序存储器EM的地址由地址寄存器MAR输出，EMRD表示从程序存储器EM中读出数据，EMEN表示读出的数据送到地址总线DBUS上，AEN有效表示将数据总线DBUS上的值存入累加器A中。此状态下uPC要加1，为取下条微指令做准备。本指令的T0状态执行的是取指操作。取指操作详细描述见程序开始部分的取指令的说明。

MOV A, 01: 本指令为三个状态周期。在T2状态时，由上条取指操作取出的指令机器码为78H，存入uPC和IR寄存器后做为微程序存储器uM的地址，读出微指令的值为0C77FFFH，相应的有效控制位为EMRD、PCOE、EMEN和MAREN，PCOE有效表示将PC值做为程序存储器EM的地址，EMRD表示从程序存储器中读出数据，在本指令中此数据值为01H，EMEN表示将读出的数据送到DBUS总线，MAREN表示将DBUS总线上的数据打入地址寄存器MAR。uPC同时加1，取出下条微指令准备执行。在T1状态时，由uPC做为微程序存储器地址，从uM的79H单元中读出微指令的值为0D7BFF7H，可以参见上条指令的T1状态，此微指令的所完成的功能是，以MAR的值做为程序存储器的地址，读出数据并送到数据总线DBUS，同时将此数据存入累加器A中。uPC加1取出下条微指令准备执行。在T0状态，微指令执行取指令操作。

IN: 本指令分两个状态周期。在T1状态时，由上次取指操作取出的指令机器码为0C0H，以此做为微地址从uM中取出的微指令值为0FFFF17H，有效控制位为AEN、X2X1X0=000，因为X2、X1、X0为低，被选中的寄存器为输入端口IN，也就是说，输入端口IN上的数据被允许送到数据总线DBUS上，AEN有效表示将此数据打入累加器A中。同时uPC加1取出下条微指令准备执行。在T0状态，微指令执行的是取指令操作，取出下条指令准备执行。

OUT: 本指令分两个状态周期。在T1状态，由上次取出的指令机器码为0C4H，以此为微地址从微程序存储器uM中读出的微指令为0FFDF9FH，有效控制位为OUTEN、X2X1X0=100 二进制，S2S1S0=111 二进制，S2S1S0=111表示运算器做“ALU直通”运算，也就是累加器不做任何运算，直接输出结果，而X2X1X0=100表示运算器的结果不移位直接输出到数据总线DBUS，OUTEN有效表示将数据总线上的数据打入输出端口寄存器OUT内。uPC加1，取出下条微指令准备执行。在T0状态，微指令执行的是取指操作，取出下条将要执行的指令。

实验 2：数据运算实验（加/减/与/或）

1. 在 COP2000 软件中的源程序窗口输入下列程序

```

ADDC  A, R1
SUB   A, @R0
AND   A, #55H
OR    A, 02H
END

```

2. 将程序另存为 EX2.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	21	ADDC A, R1	累加器 A 的值加上寄存器 R1 加进位
01	35	SUB A, @R0	累加器 A 减去 R1 间址的存储器内容
02	5C 55	AND A, #55	累加器 A 逻辑与立即数 55H
04	68 02	OR A, 02	累加器 A 逻辑或存储器 02 单元的内容

3. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。(见“EX2.ASM 程序跟踪结果”详细介绍)

4. 在了解数据运算的原理，可以加上一些数据传输指令给累加器 A 或寄存器 R?赋值，再运算，并观察运算结果。

实验 3：移位/取反实验

1. 在 COP2000 软件中的源程序窗口输入下列程序

```

MOV   A, #55H
RR    A
RLC   A
CPL   A
END

```

2. 将程序另存为 EX3.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	7C 55	MOV A, #55	立即数 55H 存入累加器 A
02	D0	RR A	不带进位右移累加器 A
03	DC	RLC A	带进位左移累加器 A
04	E4	CPL A	累加器 A 内容取反

3. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。(见“EX3.ASM 程序跟踪结果”详细介绍)

EX2.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 ADDC A, R1	T2	20	FFF7EF	寄存器值R?	寄存器W		A输出		1	
	T1	21	FFFE94	ALU直通	寄存器A 标志位C, Z		带进位加运算		1	
	T0	22	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
01 SUB A, @R1	T3	34	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T2	35	D7BFEF	存储器值EM	寄存器W	MAR输出	A输出		1	
	T1	36	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	37	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 AND A, #55	T2	5C	C7FFE9	存储器值EM	寄存器W	PC输出	A输出		1	1
	T1	5D	FFFE93	ALU直通	寄存器A 标志位C, Z		与运算		1	
	T0	5E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 OR A, 02	T3	68	C77FFF	存储器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T2	69	D7BFEF	存储器值EM	寄存器W	MAR输出	A输出		1	
	T1	6A	FFFE92	ALU直通	寄存器A 标志位C, Z		或运算		1	
	T0	6B	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1

程序的开始执行一条取指的微指令，读入程序第一条指令。

ADDC A, R1: 本指令为三个状态周期。在T2状态，由上次取指操作取出的指令码为21H，由IREN存入指令寄存器IR，最低两位为01(二进制)，选择寄存器R1，指令码由于IREN打入uPC时，忽略掉指令的最低两位，而将uPC的最低两位置成00，uPC的值为20H，访问微程序存储器的20H单元，读出微指令值为0FFF7EFH，有效位为RRD及WEN，就是将R1内容送到工作寄存器W，uPC加1取出下条微指令。在T1状态，读出的微指令值为0FFFE94H，有效位为FEN和AEN，FEN完成的操作是将标志位存入标志寄存器F(ALU内部)，X2X1X0选择“ALU直通”到数据总线DBUS，S2S1S0选择的运算操作为“带进位的加法运算”，AEN将DBUS上的数据存入累加器A。在T0状态，取出下条将要执行的指令。

SUB A, @R1: 本指令有四个状态周期。在T3状态，上次取出的指令码为35H，最低两位用于寻址R1寄存器，uPC的最低两位置0，来访问uM的34H单元的微指令，读出值为0FF77FFH，将R1的值存入MAR。在T2状态，微指令为0D7BFEFH，表示用MAR做为地址从EM中读出数据送到DBUS再存入W中。在T1状态微指令为0FFFE91H，表示ALU做“减运算”，其结果直通到DBUS，再存入A中，同时保存标志位。T0状态为取指操作。

AND A, #55: 本指令为三个状态周期。在T2状态，微指令值为0C7FFE9H，表示以PC做为地址，从EM中读出数据送到DBUS，再将DBUS数据存入W中。在T1状态，微指令为0FFFE93H，表示A和W做“逻辑与”运算，结果直通到DBUS，再存入A中，并保存标志位。

OR A, 02: 本指令有四个状态周期。在T3状态，微指令为0C77FFFH，表示以PC做为地址，从EM中读出数据送到DBUS，并存入MAR中。在T2状态，微指令为0D7BFEFH，表示以MAR做为地址，从EM中读出数据送到DBUS，并存入W中。在T1状态微指令为0FFFE92H，表示A和W做“逻辑或”运算，结果“直通”到DBUS并存入A中。T0状态为取指操作。

EX3.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #55	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 RR A	T1	D0	FFFCB7	ALU右移	寄存器A 标志位C, Z		A输出	右移	1	
	T0	D1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
03 RLC A	T1	DC	FFFD7	ALU左移	寄存器A 标志位C, Z		A输出	带进位左移	1	
	T0	DD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 CPL A	T1	E4	FFFE96	ALU直通	寄存器A 标志位C, Z		A取反		1	
	T0	E5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1

程序的开始执行一条取指的微指令，读入程序第一条指令。

MOV A, #55 将累加器的值设为055H，以便下面观察。

RR A: 本指令为两个状态周期。在T1状态，由上次取指操作取出的指令码为D0H，访问微程序存储器的20H单元，读出微指令值为0FFFCB7H，有效位为CN、FEN及AEN，表示不带进位移位，运算器控制S2S1S0=111 二进制 表示运算不运算，输出结果就为A的值，X2X1X0=101 二进制 表示，运算器“右移”输出到总线，FEN将标志位保存，AEN将DBUS内容存入A中，uPC加1取出下条微指令。在T0状态，取出下条将要执行的指令。

RLC A: 本指令有两个状态周期。在T1状态微指令为0FFFD7H，CN=1表示带进位移位，S2S1S0=111表示ALU不做运算，直接输出A内容，X2X1X0=110 二进制 表示，运算器“左移”输出到DBUS，AEN表示DBUS内容存入A中，FEN表示保存标志位。T0状态为取指操作。取出下条将要执行的指令。

CPL A: 本指令为两个状态周期。在T1状态，微指令为0FFFE96H，S2S1S0=110表示ALU做“取反”运算，X2X1X0=100 二进制 表示，运算器结果直通到DBUS，再存入A中，并保存标志位。T0状态为取指操作。取出下条将要执行的指令。

实验 4：转移实验

1. 在 COP2000 软件中的源程序窗口输入下列程序

```

MOV    A, #01
LOOP:
SUB     A, #01
JC      LOOP
JZ      LOOP
JMP     0
CPL     A
END

```

2. 将程序另存为 EX4.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	7C 01	MOV A, #01	立即数 01H 存入累加器 A
02	3C 01	SUB A, #01	累加器 A 减 1
04	A0 02	JC 02	若有进位跳到程序 02 地址
06	A4 02	JZ 02	若 A=0 跳转到程序 02 地址
08	AC 00	JMP 00	无条件跳转到程序开始

3. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。观察在条件满足和不满足的情况下，条件跳转是否正确执行。(见“EX4.ASM 程序跟踪结果”详细介绍)

实验 5：调用实验

1. 在 COP2000 软件中的源程序窗口输入下列程序

```

MOV     A, #00H
LOOP:
CALL    INCA
JMP     LOOP
INCA:
ADD     A, #1
RET
END

```

2. 将程序另存为 EX5.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	7C 55	MOV A, #00	立即数 00H 存入累加器 A
02	BC 06	CALL 06	调用子程序
04	AC 02	JMP 02	跳转到 02 地址，循环执行
06	1C 01	ADD A, #01	累加器 A 加 1
08	CC	RET	子程序返回

3. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。观察在调用子程序和从子程序返回时，堆栈的工作情况。(见“EX5.ASM 程序跟踪结果”详细介绍)

EX4.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #01	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 SUB A, #01	T2	3C	C7FDEF	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	3D	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	3E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 JC 02	T1	A0	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
06 JZ 02	T1	A4	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 SUB A, #01	T2	3C	C7FDEF	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	3D	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	3E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 JC 02	T1	A0	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 SUB A, #01	T2	3C	C7FDEF	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	3D	FFFE91	ALU直通	寄存器A 标志位C, Z		减运算		1	
	T0	3E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 JC 02	T1	A0	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
06 JZ 02	T1	A4	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	A5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
08 JMP 00	T1	AC	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	AD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #01	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1

程序的开始执行一条取指的微指令，读入程序第一条指令。

MOV A, #01 将累加器的值设为01H，用于下面计算来产生进位标志和零标志。

SUB A, #01: A值原为1，将A值第一次减1后，应产生“零标志”位。

JC 02: 由上条取指读出的指令码为0A0H，存入IR寄存器后，IR3、IR2的值为00 二进制，表示判进位跳转功能，指令码存入uPC后，从uM读出的微指令值为0C6FFFFH，表示以PC为地址从EM中读出数据02H并送到DBUS，ELP为低成有效状态，与IR3、IR2组成进位跳转控制，此时若有进位，就会产生一个控制信号，将总线DBUS上的值02H打入PC，下条微指令取指时，就会从EM新的地址02中读指令码；此时若无进位，DBUS上的值被忽略，PC加1，下条取指操作按新PC取出指令码执行。当前无进位标志，顺序执行下条指令。

JZ 02: 由上条取指读出的指令码为0A4H，存入IR寄存器后，IR3、IR2的值为01 二进制，表示判零跳转功能，指令码存入uPC后，从uM读出的微指令值为0C6FFFFH，表示以PC为地址从EM中读出数据02H并送到DBUS，ELP为低成有效状态，与IR3、IR2组成零跳转控制，与上条指令相比，尽管微指令相同，由于指令码不同，上一个为判进位跳转，这个为判零跳转。此时若零标志位为1，即A=0时，就会产生一个控制信号，将总线DBUS上的值02H打入PC，下条微指令取指时，就会从EM新的地址02中读指令码；此时若零标志位为0，DBUS上的值被忽略，PC加1，下条取指操作按新PC取出指令码执行。由于A=0，零标志位为1，产生PC打入信号，将DBUS上的值02H打入PC。下一条取指操作，PC=02，以PC为地址从EM的02单元取出指令码执行，程序转到02地址。

SUB A, #01: A值现为0，再减1后，A=0FFH，并产生“进位标志”位。

JC 02: 此为判进位跳转指令，此时由于进位标志为1，与ELP、IR3、IR2组成的电路产生PC打入信号，将数据总线上的值存入PC，程序跳转到02H地址执行。

SUB A, #01: A值现为0FFH，再减1后，A=0FEH，无“零标志”，无“进位标志”位。

JC 02: 此为判进位跳转指令，此时无进位标志，程序顺序执行下条指令。

JZ 02: 此为判零跳转指令，此时无零标志位，程序顺序执行下条指令。

JMP 00: 由上条取指操作读出的指令码为0ACH，存入IR寄存器后，IR3、IR2的值为11 二进制，此为无条件跳转控制，指令码存入uPC后，从uM读出的微指令为0C6FFFFH，表示以PC为地址从EM中读出数据并送到数据总线DBUS上，因为ELP有效，与IR3、IR2组合产生PC的打入信号，将DBUS上的数据存入PC中，下一条取指微指令按新的PC值读出程序的指令码。

MOV A, #01 程序从开头重新执行。

EX5.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #00	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 CALL 06	T3	BC	EF7F7F	PC值	地址寄存器MAR	PC输出	A输出		1	1
	T2	BD	FFEF7F	PC值	堆栈寄存器ST		A输出		1	
	T1	BE	D6BFFF	存贮器值EM	寄存器PC	MAR输出	A输出		1	写入
	T0	BF	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
06 ADD A, #01	T2	1C	C7FFEF	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	1D	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	1E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
08 RET	T1	CC	FEFF5F	堆栈寄存器ST	寄存器PC		A输出		1	写入
	T0	CD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 JMP 02	T1	AC	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	AD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 CALL 06	T3	BC	EF7F7F	PC值	地址寄存器MAR	PC输出	A输出		1	1

程序的开始执行一条取指的微指令，读入程序第一条指令。

MOV A, #00 将累加器的值设为00H，以便下面观察A加1后的结果。

CALL 06：本指令有四个状态周期。在T3状态，根据指令码为0BCH，读出微指令值为0FF7F7FH，有效位为PCOE、MAREN，X2X1X0的值为011(二进制)，PCOE有效是将PC加1，以便在下步将PC压栈时，存入堆栈的是程序下一条指令的地址，MAREN有效及X2X1X0的值表示从PC中读出值并送到MAR中。在T2状态，读出微指令为0FFEF7FH，有效位STEN，X2X1X0=100 二进制，表示从PC中读数据并存入堆栈寄存器ST中。在T1状态，微指令值为0D6BFFFH，表示以MAR为地址从EM中读出数据，此数据就是子程序的地址，此时堆栈中保存的是调用子程序下条指令的地址。将此数据送到DBUS，再存入PC中，实现程序跳转。在T0状态，按新的PC值，取出下条将要执行的指令。

ADD A, 01：本指令将累加器加1。

RET：本指令有两个状态周期。在T1状态，上条取指操作读出的指令码为0CCH，存入IR后，IR3、IR2的值为11 二进制，取出的微指令的值为0FEFF5FH，有效位为ELP，X2X1X0=010(二进制)表示从ST中输出数据到总线，ELP有效与IR3、IR2=11表示无条件将数据总线DBUS的数据打入PC，实现子程序返回功能。在T0状态，按新PC取出指令，准备执行。

JMP 02：程序无条件跳转到02地址，执行程序。

实验 6：中断实验

1. 在 COP2000 软件中的源程序窗口输入下列程序

```

MOV    A, #00
LOOP:
JMP    LOOP

ORG    0E0H
ADD    A, #01
RETI
END

```

2. 将程序另存为 EX6.ASM，将程序汇编成机器码，反汇编窗口会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	7C 00	MOV A, #00	立即数 00H 存入累加器 A
02	AC 02	JMP 02	原地跳转等待中断
...
E0	1C 01	ADD A, #01	累加器 A 值加 1
E2	EC	RETI	中断返回

3. 按快捷图标 F7，执行“单微指令运行”功能，在跟踪程序时，按下实验仪上中断请求按钮（在软件模拟时，可以按菜单下方的中断请求快捷按钮）。中断请求灯亮，在每个指令的最后一条微指令执行完，就会响应中断，中断响应灯高。同时，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。观察程序执行时，堆栈及中断请求，中断响应位的状态。（见“EX6.ASM 程序跟踪结果”详细介绍）

实验 7：指令流水实验

指令流水操作，就是在微指令执行的过程中，在 T1 状态，如果 ABUS 和 IBUS 空闲，则可以利用这个空闲来进行预取指令，让 ABUS、IBUS 和 DBUS 并行工作，实现指令的流水工作。我们已经建立了一套可流水操作的指令/微指令系统。用户可调入这个指令/微指令系统进行实验。为了方便比较，我们仍用实验 1 的程序 EX1.ASM，其它指令用户可以自己做实验来比较、验证。

1. 在 COP2000 软件中，用菜单的[文件|调入指令系统/微程序]功能，打开 COP2000 下的“INST2.INS”，这就是流水操作的指令/微指令系统。

2. 在 COP2000 软件中，用菜单的[文件|打开文件]功能，打开 COP2000 下的“EX1.ASM”源程序。编译后产生的机器码与实验 1 相同。

3. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，寄存器的输入/输出状态，各控制信号的状态，PC 及 uPC 如何工作。特别是在每条指令的 T0 状态周期，取指操作是否和其它指令并行执行。（见“EX1.ASM 程序流水操作跟踪结果”详细介绍）

EX6.ASM程序跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #00	T1	7C	C7FFF7	存贮器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 NOP	T0	E0	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
03 JMP 02	T1	AC	C6FFFF	存贮器值EM	寄存器PC	PC输出	A输出		1	写入
	T0	AD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
INT	T2	B8	FFEF7F	PC值	堆栈寄存器ST		A输出		1	
	T1	B9	FEFF3F	中断地址IA	寄存器PC		A输出		1	写入
	T0	BA	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
E0 ADD A, #01	T2	1C	C7FFE7	存贮器值EM	寄存器W	PC输出	A输出		1	1
	T1	1D	FFFE90	ALU直通	寄存器A 标志位C, Z		加运算		1	
	T0	1E	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
E2 RETI	T1	EC	FCFF5F	堆栈寄存器ST	寄存器PC		A输出		1	写入
	T0	ED	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 NOP	T0	E0	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1

程序的开始执行一条取指的微指令，读入程序第一条指令。

MOV A, #00 将累加器的值设为00H，以便下面观察A加1后的结果。

NOP：程序空操作，等待中断请求。

JMP 02：程序无条件跳转到02地址，执行程序。在执行此指令前，按下实验仪上的中断请求钮或软件上“中断请求”快捷键，中断请求的灯会亮，表示有中断请求。在本指令的T0状态即取指状态，IREN有效将中断处理微程序地址0B8H，送到指令总线IBUS上。

INT：本指令为中断处理微程序，有三个状态周期。在T2状态，微指令的值为0FFEF7FH，其中X2X1X0=011 二进制，表示从PC中读出数据送到DBUS上，STEN有效表示将DBUS上数据存入堆栈寄存器ST中，这条微指令执行的就是将PC值(即下条将执行的指令的地址)存入堆栈。在T1状态，微指令值为0FEFF3FH，其中X2X1X0=001表示将中断地址寄存器IA的值送到DBUS上，IA的缺省值为0E0H，ELP有效，与指令寄存器IR的IR3、IR2=10(二进制)组合，将DBUS值存入PC，实现程序的跳转。在T0状态以中断地址0E0H为地址取出中断服务程序的第一条指令，准备执行。

ADD A, 01：本指令将累加器加1。

RETI：本指令有两个状态周期。在T1状态，取出的微指令为0FCFF5FH，X2X1X0=010 二进制 表示从ST读出数据送到DBUS上，EINT有效清除中断请求标志和中断响应标志，以便程序返回后，可以再次响应中断，ELP有效与IR3、IR2=11表示无条件将数据总线DBUS的值打入PC，实现中断服务程序返回功能。在T0状态，按新PC取出指令，准备执行。

NOP：上次中断是在执行完“JMP 02”指令后响应的，中断返回的地址为其下条将要执行的指令，也就是“NOP”指令。

EX1.ASM程序流水操作跟踪结果

助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
00 MOV A, #12	T1	7C	C7FFF7	存储器值EM	寄存器A	PC输出	A输出		1	1
	T0	7D	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
02 MOV A, R0	T0	70	CBF7F7	寄存器值R?	寄存器A、指令寄存器	PC输出	A输出		写入	1
03 MOV A, @R0	T2	74	FF77FF	寄存器值R?	地址寄存器MAR		A输出		1	
	T1	75	D7BFF7	存储器值EM	寄存器A	MAR输出	A输出		1	
	T0	76	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
04 MOV A, 01	T2	78	C77FFF	存储器值EM	地址寄存器MAR	PC输出	A输出		1	1
	T1	79	D7BFF7	存储器值EM	寄存器A	MAR输出	A输出		1	
	T0	7A	CBFFFF		指令寄存器IR	PC输出	A输出		写入	1
06 IN	T0	C0	CBFF17	用户IN	寄存器A、指令寄存器	PC输出	A输出		写入	1
07 OUT	T0	C4	CBDF9F	ALU直通	指令寄存器IR、用户	PC输出	A输出		写入	1

每个程序的一开始的第一条微指令一定是取指令，取出下条将要执行的指令。

MOV A, #12: 本指令为两个状态周期。在T1状态时，从程序存储器EM中读出数据送到累加器A，ABUS被占用，所以预指操作不能与数据总线DBUS上的操作并行执行。本指令的T0状态为正常的取指令操作。

MOV A, R0: 由于预指操作与数据总线可并行工作，本指令只有1个状态周期。由上条取指操作取出的指令机器码为70H，存入uPC后做为微程序地址访问微程序存储器uM的70H单元，读出微指令的值为0CBF7F7H，有效控制位为EMRD、PCOE、IREN、RRD、AEN，由于IR1、IR0的值为00，与RRD信号组合表示从R0中读出数据到DBUS总线，AEN将DBUS上的值存入累加器A，EMRD、PCOE和IREN有效表示以PC做为地址从EM中读出下条指令，并存入IR和uPC中，PC加1。

MOV A, @R0 本指令为三个状态周期。在T2状态时，将R0的值存入地址寄存器MAR。在T1状态时，以MAR为地址读出数据并送到累加器A中。在T0状态，取出下条将要执行指令。由于ABUS不空闲，所以取指操作不能并行工作。

MOV A, 01: 本指令为三个状态周期。在T2状态时，以PC为地址从EM中读出数据存到MAR中，在T1状态，以MAR为地址从EM中读出数据存入累加器A。T0为取指操作。由于ABUS不空闲，取指操作不能并行执行。

IN: 本指令为1个状态周期。取指操作和输出操作可并行执行。由上次取指操作取出的指令机器码为0C0H，以此做为微地址从uM中取出的微指令值为0CBFF17H，有效控制位为EMRD、PCOE、IREN和AEN、X2X1X0=000(二进制)表示从输入寄存器IN读数据送到DBUS，AEN表示将此数据存入A，EMRD、PCOE和IREN有效表示以PC为地址从EM中读出指令存入IR和uPC中，PC加1。

OUT: 本指令有1个状态周期。取指操作和输出操作并行完成。由上次取出微指令值为0CBDF9FH，有效控制位为EMRD、PCOE、IREN、OUTEN、X2X1X0=100(二进制，S2S1S0=111 二进制，S2S1S0=111表示运算器做“ALU直通”运算，也就是累加器不做任何运算，直接输出结果，而X2X1X0=100表示运算器的结果不移位直接输出到数据总线DBUS，OUTEN有效表示将数据总线上的数据打入输出端口寄存器OUT内。与此同时，EMRD、PCOE、IREN表示以PC为地址从EM中读出下条指令，存IR和uPC中，PC加1。

实验 8 RISC 模型机

RISC 处理器设计的一般原则:

1. 只选用使用频度高的指令, 减小指令系统, 使每一条指令能尽快的执行
2. 减少寻址方式, 并让指令具有相同的长度
3. 让大部分指令在一个时钟完成
4. 所有指令只有存 (ST)、取 (LD) 指令可访问内存, 其他指令均在寄存器间进行运算
5. 使用组合逻辑实现控制器

下面我们给出一个 RISC 的指令系统

助记符	机器码1	机器码2	注释
FATCH	000000xx		实验机占用, 不可修改。复位后, 所有寄存器清0, 首先执行 _FATCH_ 指令取指
ADDW A	000001xx		将寄存器W加入累加器A中
ADDCW A	000010xx		将寄存器W值加入累加器A中, 带进位
SUBW A	000011xx		从累加器A中减去寄存器W值
SUBCW A	000100xx		从累加器A中减去寄存器W值, 带进位
ANDW A	000101xx		累加器A “与” 寄存器W值
ORW A	000110xx		累加器A “或” 寄存器W值
LDW A	000111xx		将累加器A的值送到寄存器W中
LDW R?	001000xx		将寄存器R?的值送到寄存器W中
LDA R?	001001xx		将寄存器R?送到累加器A中
STA R?	001010xx		将累加器A的值送到寄存器R?中
RR A	001011xx		累加器A右移
RL A	001100xx		累加器A左移
RRC A	001101xx		累加器A带进位右移
RLC A	001110xx		累加器A带进位左移
CPL A	001111xx		累加器A取反, 再存入累加器A中
LD A, #II	010000xx	II	将立即数送到累加器A中
LD A, MM	010001xx	MM	将存储器中MM中的值送到累加器A中
ST A, MM	010010xx	MM	将累加器A的值送到存储器MM地址中
JMP MM	010011xx		跳转到地址MM
JC MM	010100xx		若进位标志置1, 跳转到地址MM
JZ MM	010101xx		若零标志位置1, 跳转到地址MM

可以看出在这个指令系统中, 只有访问主存LD, ST指令和转移指令有两个字节, 其余指令均为单字节单时钟指令。

1. 在 COP2000 软件中, 用菜单的[文件|调入指令系统/微程序]功能, 打开 COP2000 下的“RISC.INS”, 这就是 RISC 指令/微指令系统。

2. 在 COP2000 软件中, 用菜单的[文件|打开文件]功能, 打开 COP2000 下的“EX7.ASM”源程序。

3. 按快捷图标 F7, 执行“单微指令运行”功能, 观察执行每条微指令时, 寄存器的输入/输出状态, 各控制信号的状态, PC 及 uPC 如何工作。

4. 自动产生 ABEL 组合逻辑控制器, 比较非 RISC 指令系统, 可以看出 RISC 指令系统简单很多。

第五章 组合逻辑控制器

微程序控制器由微程序给出 24 位控制信号，而微程序的地址又是由指令码提供的，这就是说 24 位控制信号是由指令码确定的。如：MOV A, 12H 及 MOV A, #34H 指令的微序如下：

助记符	状态	微地址	微程序	相关控制位
MOV A, MM	T2	78	C77FFF	EMRD, PCOE, EMEN, MAREN
	T1	79	D7BFF7	EMRD, EMEN, MAROE, AEN
	T0	7A	CBFFFF	EMRD, PCOE, IREN
MOV A, #II	T1	7C	C7FFF7	EMRD, PCOE, EMEN, AEN
	T0	7D	CBFFFF	EMRD, PCOE, IREN

我们用组合逻辑的方法来写出相应的控制表达式

IR7..IR2 为指令的高六位（低两位用于选择寄存器 R0..R3）

T3,T2,T1,T0 为处于的周期

```

!EMRD =    !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2    // MOV A, MM    T2 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1    // MOV A, MM    T1 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0    // MOV A, MM    T0 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1    // MOV A, #II   T1 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0;    // MOV A, #II   T0 周期

!PCOE =    !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2    // MOV A, MM    T2 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0    // MOV A, MM    T0 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1    // MOV A, #II   T1 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0;    // MOV A, #II   T0 周期

!EMEN =    !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2    // MOV A, MM    T2 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1    // MOV A, MM    T1 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T2;    // MOV A, #II   T1 周期

!MAREN =   !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2;    // MOV A, MM    T2 周期

!MAROE =   !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1;    // MOV A, MM    T1 周期

!AEN  = #   !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1    // MOV A, MM    T1 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1;    // MOV A, #II   T1 周期

!IREN =    !IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0    // MOV A, MM    T0 周期
          # !IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0;    // MOV A, #II   T0 周期

```

上面给出的表达式仅是两条指令的表达式，而且没有化简的，不难看出 24 位控制信号是指令码及周期数的函数。增加一条指令，只要增加一些或项即可，如增加 ADD A, #11H

助记符	状态	微地址	微程序	相关控制位
ADD A, #II	T2	1C	C7FFEF	EMRD, PCOE, EMEN, WEN
	T1	1D	FFFE90	FEN, X1, X0, AEN, S2, S1, S0
	T0	1E	CBFFFF	EMRD, PCOE, IREN

EMRD 增加:

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 // ADD A, #II T2 周期

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 // ADD A, #II T0 周期

PCOE 增加:

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 // ADD A, #II T2 周期

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 // ADD A, #II T0 周期

EMEN 增加:

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 // ADD A, #II T2 周期

AEN 增加:

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

IREN 增加:

!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 // ADD A, #II T0 周期

ADD A, #II 新增加的控制信号有 WEN, FEN, X2, X1, S2, S1, S0

!WEN = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 // ADD A, #II T2 周期

!FEN = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

!X2 = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

!X1 = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

!S2 = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

!S1 = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

!S0 = !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 // ADD A, #II T1 周期

IR7..IR0 由指令寄存器提供, ABEL 表达式是:

[IR7..IR0] := [IBUS7..IBUS0];

[IR7..IR0].CE = !IREN;

[IR7..IR0].AR = !RST;

[IR7..IR0].CLK = CK;

COP2000 每条指令最多有 4 个周期 (T3, T2, T1, T0), 可由两位 D 触发器 (RT1, RT0) 表示。

T3 = RT1 & RT0;

T2 = RT1 & !RT0;

T1 = !RT1 & RT0;

T0 = !RT1 & !RT0;

RT1, RT0 构成一个带预置的减计数器, ABEL 表达式是:

```

WHEN !RT1 & !RT0 THEN {
    [RT1..RT0] := [CT1..CT0];
} ELSE {
    [RT1..RT0] := [RT1..RT0] - 1;
}
[RT1..RT0].CLK = CK;
[RT1..RT0].AR = !RST;

```

当 RT1, RT0 为 0 时, 表示现执行的是本指令的最后一个周期, 这个周期为取指周期。

在取指时将 RT1, RT0 置为下一条指令的首个周期值。

当 RT1, RT0 不为 0 时, 将周期数减一。

CT1, CT0 根据指令计算出, ABEL 表达式是:

```

TRUTH_TABLE([IBUS7, IBUS6, IBUS5, IBUS4, IBUS3, IBUS2]->[CT1, CT0])
    [ 0, 1, 1, 1, 1, 0]->[ 1, 0]; // MOV A, MM
    [ 0, 1, 1, 1, 1, 1]->[ 0, 1]; // MOV A, #II

```

下面是 COP2000 出厂时的组合逻辑控制器 ABEL 表达式:

```

module COP2000
Title 'COP2000'

Declarations
    XRD    PIN 26 istype 'COM';
    EMWR   PIN 3  istype 'COM';
    EMRD   PIN 25 istype 'COM';
    PCOE   PIN 4  istype 'COM';
    EMEN   PIN 24 istype 'COM';
    IREN   PIN 5  istype 'COM';
    EINT   PIN 23 istype 'COM';
    ELP    PIN 21 istype 'COM';

    MAREN  PIN 20 istype 'COM';
    MAROE  PIN 1  istype 'COM';
    OUTEN  PIN 19 istype 'COM';
    STEN   PIN 84 istype 'COM';
    RRD    PIN 18 istype 'COM';
    RWR    PIN 83 istype 'COM';
    CN     PIN 17 istype 'COM';

```

```

FEN    PIN 15 istype 'COM';

X2     PIN 14 istype 'COM';
X1     PIN  6 istype 'COM';
X0     PIN 13 istype 'COM';
WEN    PIN  7 istype 'COM';
AEN    PIN 12 istype 'COM';
S2     PIN  9 istype 'COM';
S1     PIN 11 istype 'COM';
S0     PIN 10 istype 'COM';

```

```

IBUS7  PIN 32;
IBUS6  PIN 34;
IBUS5  PIN 36;
IBUS4  PIN 37;
IBUS3  PIN 33;
IBUS2  PIN 35;
IBUS1  PIN 43;
IBUS0  PIN 44;

```

```

CK     PIN 39;
RST    PIN 40;

```

```

RT1    PIN 82 istype 'REG';
RT0    PIN  2 istype 'REG';

```

```

CT1    PIN 77 istype 'COM';
CT0    PIN 79 istype 'COM';

```

```

MON    PIN 80;

```

```

IR7    NODE istype 'REG';
IR6    NODE istype 'REG';
IR5    NODE istype 'REG';
IR4    NODE istype 'REG';
IR3    NODE istype 'REG';
IR2    NODE istype 'REG';
IR1    NODE istype 'REG';
IR0    NODE istype 'REG';

```

```

T3     NODE istype 'COM';
T2     NODE istype 'COM';
T1     NODE istype 'COM';
T0     NODE istype 'COM';

```

Equations

```

// XRD 1
!XRD   = IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 ;    // READ    A, MM

// EMWR 2
!EMWR  = IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 #    // MOV     @R?, A
        IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 ;    // MOV     MM, A

// EMRD 100
!EMRD  = !IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 #    // _FATCH_

```

[illegible]

- 65 -

```

!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T2 # // ADDC A, #11
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T3 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T2 # // SUB A, #11
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T3 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T2 # // SUBC A, #11
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T3 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T2 # // AND A, #11
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T2 # // OR A, #11
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // OR A, #11
!IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // MOV A, R?
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // MOV A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // MOV A, #11
!IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // MOV A, #11
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // MOV R?, A
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // MOV @R?, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // MOV MM, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // MOV MM, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // MOV R?, #11
IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // MOV R?, #11
IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // READ A, MM
IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // READ A, MM
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // WRITE MM, A
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // WRITE MM, A
IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // UNDEF
IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // JC MM
IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // JC MM
IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // JZ MM
IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // JZ MM
IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // JMP MM
IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // JMP MM
IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // UNDEF
IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // UNDEF

```

```

IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // _INT_
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T3 # // CALL MM
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // CALL MM
IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // IN
IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // OUT
IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // RET
IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // RR A
IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // RL A
IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // RRC A
IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // RLC A
IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // NOP
IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // CPL A
IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // RETI
IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0 ; // UNDEF

// EMEN 38
!EMEN = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T3 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T2 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T2 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T3 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T2 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T2 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T3 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T2 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T3 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T2 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T2 # // OR A, #11
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // MOV A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // MOV A, #11
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // MOV @R?, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // MOV MM, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // MOV MM, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // MOV R?, #11
IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // READ A, MM
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // WRITE MM, A
IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // JC MM

```

```

IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // JZ MM
IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // JMP MM
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 ; // CALL MM

// IREN 64
!IREN = !IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // _FATCH_
!IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // UNDEF
!IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
!IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // UNDEF
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // OR A, #11
!IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // MOV A, R?
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // MOV A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // MOV A, #11
!IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // MOV R?, A
!IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // MOV @R?, A
!IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // MOV MM, A
!IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // MOV R?, #11
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // READ A, MM
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // WRITE MM, A
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // UNDEF
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // UNDEF
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // JC MM
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // JZ MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // JMP MM
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // UNDEF
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // UNDEF
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // _INT_
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T0 # // CALL MM
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T0 # // IN
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T0 # // OUT
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF

```

```

IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T0 # // RET
IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T0 # // RR      A
IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T0 # // RL      A
IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T0 # // RRC     A
IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T0 # // RLC     A
IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T0 # // NOP
IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T0 # // CPL      A
IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T0 # // RETI
IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T0 # // UNDEF
IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T0 ; // UNDEF

// EINT 1
!EINT = IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 ; // RETI

// ELP 7
!ELP = IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // JC      MM
IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // JZ      MM
IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // JMP     MM
IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // _INT_
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // CALL    MM
IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // RET
IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 ; // RETI

// MAREN 19
!MAREN = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T3 # // ADD     A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T3 # // ADD     A, MM
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T3 # // ADDC    A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // ADDC    A, MM
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T3 # // SUB     A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T3 # // SUB     A, MM
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T3 # // SUBC    A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T3 # // SUBC    A, MM
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T3 # // AND     A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T3 # // AND     A, MM
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T3 # // OR      A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T3 # // OR      A, MM
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T2 # // MOV     A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // MOV     A, MM
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T2 # // MOV     @R?, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // MOV     MM, A
IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // READ    A, MM
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // WRITE   MM, A
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T3 ; // CALL    MM

// MAROE 19
!MAROE = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // ADD     A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // ADD     A, MM
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // ADDC    A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // ADDC    A, MM
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T2 # // SUB     A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // SUB     A, MM
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T2 # // SUBC    A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // SUBC    A, MM

```

```

!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // AND A, MM
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // OR A, MM
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // MOV A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // MOV A, MM
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // MOV @R?, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // MOV MM, A
IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // READ A, MM
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // WRITE MM, A
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 ; // CALL MM

// OUTEN 2
!OUTEN = IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // WRITE MM, A
IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 ; // OUT

// STEN 2
!STEN = IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // _INT_
IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T2 ; // CALL MM

// RRD 15
!RRD = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T3 # // ADD A, @R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T2 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T3 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T2 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T3 # // SUB A, @R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T2 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T3 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T3 # // AND A, @R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T2 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T3 # // OR A, @R?
!IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // MOV A, R?
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T2 # // MOV A, @R?
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T2 ; // MOV @R?, A

// RWR 2
!RWR = IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // MOV R?, A
IR7 & !IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 ; // MOV R?, #11

// CN 2
!CN = IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // RR A
IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 ; // RL A

// FEN 29
!FEN = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?

```

```

!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // OR A, #11
IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // RR A
IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // RL A
IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // RRC A
IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // RLC A
IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 ; // CPL A

// X2 7
!X2 = IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // _INT_
      IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // _INT_
      IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T3 # // CALL MM
      IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T2 # // CALL MM
      IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // IN
      IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // RET
      IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 ; // RETI

// X1 34
!X1 = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
      !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
      !IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
      !IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?
      !IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
      !IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
      !IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // SUBC A, R?
      !IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // SUBC A, @R?
      !IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // SUBC A, MM
      !IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // SUBC A, #11
      !IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // AND A, R?
      !IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // AND A, @R?
      !IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // AND A, MM
      !IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // AND A, #11
      !IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
      !IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
      !IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
      !IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // OR A, #11
      IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // MOV R?, A
      IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // MOV @R?, A

```

```

IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // MOV MM, A
IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // WRITE MM, A
IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // _INT_
IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // IN
IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // OUT
IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // RR A
IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // RRC A
IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 ; // CPL A

// X0 35
!X0 = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
      !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
      !IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
      !IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?
      !IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
      !IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
      !IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // SUBC A, R?
      !IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // SUBC A, @R?
      !IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // SUBC A, MM
      !IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // SUBC A, #11
      !IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // AND A, R?
      !IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // AND A, @R?
      !IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // AND A, MM
      !IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // AND A, #11
      !IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
      !IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
      !IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
      !IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // OR A, #11
      IR7 & !IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // MOV R?, A
      IR7 & !IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // MOV @R?, A
      IR7 & !IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // MOV MM, A
      IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // WRITE MM, A
      IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // IN
      IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // OUT
      IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // RET
      IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // RL A
      IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // RLC A
      IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // CPL A
      IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 ; // RETI

// WEN 24
!WEN = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // ADD A, R?
      !IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // ADD A, @R?
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // ADD A, MM
      !IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T2 # // ADD A, #11
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T2 # // ADDC A, R?
      !IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // ADDC A, @R?
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // ADDC A, MM
      !IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T2 # // ADDC A, #11
      !IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T2 # // SUB A, R?

```

```

!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T2 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T2 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T2 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T2 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T2 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T2 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T2 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T2 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T2 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T2 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T2 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T2 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T2 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T2 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T2 ; // OR A, #11

// AEN 35
!AEN = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 # // SUBC A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // OR A, #11
!IR7 & IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // MOV A, R?
!IR7 & IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // MOV A, @R?
!IR7 & IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // MOV A, MM
!IR7 & IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // MOV A, #11
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // READ A, MM
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // IN A
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // RR A
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // RL A
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // RRC A
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // RLC A
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 ; // CPL A

// S2 16
!S2 = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?

```

```

!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
!IR7 & IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // AND A, R?
!IR7 & IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // AND A, @R?
!IR7 & IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // AND A, MM
!IR7 & IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // AND A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 ; // OR A, #11

```

// S1 16

```

!S1 = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
!IR7 & !IR6 & IR5 & IR4 & !IR3 & !IR2 & T1 # // SUB A, R?
!IR7 & !IR6 & IR5 & IR4 & !IR3 & IR2 & T1 # // SUB A, @R?
!IR7 & !IR6 & IR5 & IR4 & IR3 & !IR2 & T1 # // SUB A, MM
!IR7 & !IR6 & IR5 & IR4 & IR3 & IR2 & T1 # // SUB A, #11
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & !IR2 & T1 # // SUBC A, R?
!IR7 & IR6 & !IR5 & !IR4 & !IR3 & IR2 & T1 # // SUBC A, @R?
!IR7 & IR6 & !IR5 & !IR4 & IR3 & !IR2 & T1 # // SUBC A, MM
!IR7 & IR6 & !IR5 & !IR4 & IR3 & IR2 & T1 ; // SUBC A, #11

```

// S0 13

```

!S0 = !IR7 & !IR6 & !IR5 & IR4 & !IR3 & !IR2 & T1 # // ADD A, R?
!IR7 & !IR6 & !IR5 & IR4 & !IR3 & IR2 & T1 # // ADD A, @R?
!IR7 & !IR6 & !IR5 & IR4 & IR3 & !IR2 & T1 # // ADD A, MM
!IR7 & !IR6 & !IR5 & IR4 & IR3 & IR2 & T1 # // ADD A, #11
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // ADDC A, R?
!IR7 & !IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // ADDC A, @R?
!IR7 & !IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // ADDC A, MM
!IR7 & !IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // ADDC A, #11
!IR7 & IR6 & IR5 & !IR4 & !IR3 & !IR2 & T1 # // OR A, R?
!IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 # // OR A, @R?
!IR7 & IR6 & IR5 & !IR4 & IR3 & !IR2 & T1 # // OR A, MM
!IR7 & IR6 & IR5 & !IR4 & IR3 & IR2 & T1 # // OR A, #11
IR7 & IR6 & IR5 & !IR4 & !IR3 & IR2 & T1 ; // CPL A

```

// Does not output when MON is high

```

XRD. OE = !MON;
EMWR. OE = !MON;
EMRD. OE = !MON;
PCOE. OE = !MON;
EMEN. OE = !MON;
IREN. OE = !MON;
EINT. OE = !MON;

```

```

ELP.OE  = !MON;
MAREN.OE = !MON;
MAROE.OE = !MON;
OUTEN.OE = !MON;
STEN.OE  = !MON;
RRD.OE   = !MON;
RWR.OE   = !MON;
CN.OE    = !MON;
FEN.OE   = !MON;
X2.OE    = !MON;
X1.OE    = !MON;
X0.OE    = !MON;
WEN.OE   = !MON;
AEN.OE   = !MON;
S2.OE    = !MON;
S1.OE    = !MON;
S0.OE    = !MON;

// Load IR register
[IR7..IRO] := [IBUS7..IBUS0];
[IR7..IRO].CE = !IREN;
[IR7..IRO].AR = !RST;
[IR7..IRO].CLK = CK;

// T counter
WHEN !RT1 & !RT0 THEN {
    [RT1..RT0] := [CT1..CT0];
} ELSE {
    [RT1..RT0] := [RT1..RT0] - 1;
}
[RT1..RT0].CLK = CK;
[RT1..RT0].AR = !RST;

// set T
T3 = RT1 & RT0;
T2 = RT1 & !RT0;
T1 = !RT1 & RT0;
T0 = !RT1 & !RT0;

// calc constant for T counter
TRUTH_TABLE ([IBUS7, IBUS6, IBUS5, IBUS4, IBUS3, IBUS2] -> [CT1, CT0])
[ 0, 0, 0, 0, 0, 0, 0] -> [ 0, 0]; // Fetch
[ 0, 0, 0, 0, 0, 0, 1] -> [ 0, 0]; // UNDEF
[ 0, 0, 0, 0, 1, 0, 0] -> [ 0, 0]; // UNDEF
[ 0, 0, 0, 0, 1, 1, 0] -> [ 0, 0]; // UNDEF
[ 0, 0, 0, 0, 1, 0, 0] -> [ 1, 0]; // ADD    A, R?
[ 0, 0, 0, 0, 1, 0, 1] -> [ 1, 1]; // ADD    A, @R?
[ 0, 0, 0, 0, 1, 1, 0] -> [ 1, 1]; // ADD    A, MM
[ 0, 0, 0, 0, 1, 1, 1] -> [ 1, 0]; // ADD    A, #11
[ 0, 0, 0, 1, 0, 0, 0] -> [ 1, 0]; // ADDC   A, R?
[ 0, 0, 0, 1, 0, 0, 1] -> [ 1, 1]; // ADDC   A, @R?
[ 0, 0, 0, 1, 0, 1, 0] -> [ 1, 1]; // ADDC   A, MM
[ 0, 0, 0, 1, 0, 1, 1] -> [ 1, 0]; // ADDC   A, #11
[ 0, 0, 0, 1, 1, 0, 0] -> [ 1, 0]; // SUB    A, R?
[ 0, 0, 0, 1, 1, 0, 1] -> [ 1, 1]; // SUB    A, @R?
[ 0, 0, 0, 1, 1, 1, 0] -> [ 1, 1]; // SUB    A, MM

```

```

[ 0, 0, 1, 1, 1, 1] -> [ 1, 0]; // SUB    A, #11
[ 0, 1, 0, 0, 0, 0] -> [ 1, 0]; // SUBC   A, R?
[ 0, 1, 0, 0, 0, 0] -> [ 1, 1]; // SUBC   A, @R?
[ 0, 1, 0, 0, 1, 1] -> [ 1, 0]; // SUBC   A, MM
[ 0, 1, 0, 1, 0, 0] -> [ 1, 0]; // AND     A, R?
[ 0, 1, 0, 1, 0, 1] -> [ 1, 1]; // AND     A, @R?
[ 0, 1, 0, 1, 1, 0] -> [ 1, 1]; // AND     A, MM
[ 0, 1, 0, 1, 1, 1] -> [ 1, 0]; // AND     A, #11
[ 0, 1, 1, 0, 0, 0] -> [ 1, 0]; // OR      A, R?
[ 0, 1, 1, 0, 0, 1] -> [ 1, 1]; // OR      A, @R?
[ 0, 1, 1, 0, 1, 0] -> [ 1, 1]; // OR      A, MM
[ 0, 1, 1, 0, 1, 1] -> [ 1, 0]; // OR      A, #11
[ 0, 1, 1, 1, 0, 0] -> [ 0, 1]; // MOV     A, R?
[ 0, 1, 1, 1, 0, 1] -> [ 1, 0]; // MOV     A, @R?
[ 0, 1, 1, 1, 1, 0] -> [ 1, 0]; // MOV     A, MM
[ 0, 1, 1, 1, 1, 1] -> [ 0, 1]; // MOV     A, #11
[ 1, 0, 0, 0, 0, 0] -> [ 0, 1]; // MOV     R?, A
[ 1, 0, 0, 0, 0, 1] -> [ 1, 0]; // MOV     @R?, A
[ 1, 0, 0, 0, 1, 0] -> [ 0, 1]; // MOV     MM, A
[ 1, 0, 0, 0, 1, 1] -> [ 0, 1]; // MOV     R?, #11
[ 1, 0, 0, 1, 0, 0] -> [ 1, 0]; // READ    A, MM
[ 1, 0, 0, 1, 0, 1] -> [ 1, 0]; // WRITE   MM, A
[ 1, 0, 0, 1, 1, 0] -> [ 0, 0]; // UNDEF
[ 1, 0, 0, 1, 1, 1] -> [ 0, 0]; // UNDEF
[ 1, 0, 1, 0, 0, 0] -> [ 0, 1]; // JC      MM
[ 1, 0, 1, 0, 0, 1] -> [ 0, 1]; // JZ      MM
[ 1, 0, 1, 0, 1, 0] -> [ 0, 0]; // UNDEF
[ 1, 0, 1, 0, 1, 1] -> [ 0, 1]; // JMP     MM
[ 1, 0, 1, 1, 0, 0] -> [ 0, 0]; // UNDEF
[ 1, 0, 1, 1, 0, 1] -> [ 0, 0]; // UNDEF
[ 1, 0, 1, 1, 1, 0] -> [ 1, 0]; // _INT_
[ 1, 0, 1, 1, 1, 1] -> [ 1, 1]; // CALL    MM
[ 1, 1, 0, 0, 0, 0] -> [ 0, 1]; // IN
[ 1, 1, 0, 0, 0, 1] -> [ 0, 1]; // OUT
[ 1, 1, 0, 0, 1, 0] -> [ 0, 0]; // UNDEF
[ 1, 1, 0, 0, 1, 1] -> [ 0, 1]; // RET
[ 1, 1, 0, 1, 0, 0] -> [ 0, 1]; // RR      A
[ 1, 1, 0, 1, 0, 1] -> [ 0, 1]; // RL      A
[ 1, 1, 0, 1, 1, 0] -> [ 0, 1]; // RRC     A
[ 1, 1, 0, 1, 1, 1] -> [ 0, 1]; // RLC     A
[ 1, 1, 1, 0, 0, 0] -> [ 0, 0]; // NOP
[ 1, 1, 1, 0, 0, 1] -> [ 0, 1]; // CPL      A
[ 1, 1, 1, 0, 1, 0] -> [ 0, 0]; // UNDEF
[ 1, 1, 1, 0, 1, 1] -> [ 0, 1]; // RETI
[ 1, 1, 1, 1, 0, 0] -> [ 0, 0]; // UNDEF
[ 1, 1, 1, 1, 0, 1] -> [ 0, 0]; // UNDEF
[ 1, 1, 1, 1, 1, 0] -> [ 0, 0]; // UNDEF
[ 1, 1, 1, 1, 1, 1] -> [ 0, 0]; // UNDEF

```

end COP2000

用 EPLD 实现运算器功能

COP2000 实验仪上的运算器由一片 XC9572-PLCC44 实现，具体的 ABEL 程序如下，用户可行自行修改以实现其它功能

Module ALU

Declarations

```
ALU1 interface ( S2,S1,S0,C1,A,B -> S,C0 ); // 一位运算器
```

```
T7..T0 functional_block ALU1;
```

```
DIn7..DIn0 PIN 25, 26, 27, 28, 29, 34, 33, 36;
```

```
A_EN pin 20;
```

```
W_EN pin 22;
```

```
F_EN pin 24;
```

```
ALU_CK pin 18;
```

```
A7..A0 node istype 'reg';
```

```
W7..W0 node istype 'reg';
```

```
O7..O0 PIN 5, 4, 3, 2, 1, 44, 43, 42 istype 'com';
```

```
X1 pin 37;
```

```
X0 pin 38;
```

```
RZ pin 13 istype 'reg';
```

```
RCy pin 14 istype 'reg';
```

```
RL0 pin 6 istype 'com';
```

```
RR7 pin 40 istype 'com';
```

```
CN PIN 39;
```

```
S2 PIN 9;
```

```
S1 PIN 11;
```

```
S0 PIN 12;
```

```
CIn node;
```

```
Cy node;
```

```
C6..C0 node;
```

Equations

```
[A7..A0] := [DIn7..DIn0];
```

```
[A7..A0].CLK = A_EN # ALU_CK;
```

```
[W7..W0] := [DIn7..DIn0];
```

```
[W7..W0].CLK = W_EN # ALU_CK;
```

```
CIn = RCy; // Cin connect to RCy output
```

```
RL0 = CN & CIn;
```

```

RR7      = CN & CIn;

RZ       := !07 & !06 & !05 & !04 & !03 & !02 & !01 & !00;
RZ.CLK   = F_EN # ALU_CLK;

RCy      := !X1 & !X0 & Cy #           // direct Cout = Cy
           X1 & X0 & Cy #             // direct Cout = Cy
           CN & !X1 & X0 & 00 #        // shift right with C
           !CN & !X1 & X0 & RCy #      // shift right without C
           CN & X1 & !X0 & 07 #        // shift left with C
           !CN & X1 & !X0 & RCy;       // shift left without C

RCy.CLK  = F_EN # ALU_CLK;

T0.S1 = S1;
T0.S0 = S0;
T0.S2 = S2;
T0.CI = S2 & (!F_OE & CIn # F_OE & PCIn);
T0.A  = A0;
T0.B  = W0;
O0    = T0.S;
C0    = T0.CO;

T1.S1 = S1;
T1.S0 = S0;
T1.S2 = S2;
T1.CI = C0;
T1.A  = A1;
T1.B  = W1;
O1    = T1.S;
C1    = T1.CO;

T2.S1 = S1;
T2.S0 = S0;
T2.S2 = S2;
T2.CI = C1;
T2.A  = A2;
T2.B  = W2;
O2    = T2.S;
C2    = T2.CO;

T3.S1 = S1;
T3.S0 = S0;
T3.S2 = S2;
T3.CI = C2;
T3.A  = A3;
T3.B  = W3;
O3    = T3.S;
C3    = T3.CO;

T4.S1 = S1;

```

```
T4.S0 = S0;
T4.S2 = S2;
T4.CI = C3;
T4.A  = A4;
T4.B  = W4;
O4    = T4.S;
C4    = T4.C0;
```

```
T5.S1 = S1;
T5.S0 = S0;
T5.S2 = S2;
T5.CI = C4;
T5.A  = A5;
T5.B  = W5;
O5    = T5.S;
C5    = T5.C0;
```

```
T6.S1 = S1;
T6.S0 = S0;
T6.S2 = S2;
T6.CI = C5;
T6.A  = A6;
T6.B  = W6;
O6    = T6.S;
C6    = T6.C0;
```

```
T7.S1 = S1;
T7.S0 = S0;
T7.S2 = S2;
T7.CI = C6;
T7.A  = A7;
T7.B  = W7;
O7    = T7.S;
Cy    = T7.C0;
End ALU
```

Module ALU1

```
Interface ( S2, S1, S0, CI, A, B -> S, C0 );
```

Declarations

```
S2, S1, S0 pin;
CI, A, B  pin;
S, C0     pin  istype 'com';
TS        node;
ADD       node;
SUB       node;
AND       node;
OR        node;
DIR       node;
CPL       node;
```

Equations

```

ADD = !S1 & !S0;
SUB = !S1 & S0;
OR  = !S2 & S1 & !S0;
AND = !S2 & S1 & S0;
CPL = S2 & S1 & !S0;
DIR = S2 & S1 & S0;

CO = B & CI #
    A & !B & CI & ADD #
    A & B & !CI & ADD #
    !A & !B & CI & !ADD #
    !A & B & !CI & !ADD ;

TS = !A & !B & CI #
    !A & B & !CI #
    A & !B & !CI #
    A & B & CI;

S = (ADD # SUB) & TS #
    OR      & (A # B) #
    AND     & (A & B) #
    DIR     & A #
    CPL     & !A;

```

End ALU1

用 EPLD 实现堆栈功能

定义:

```
ST7..ST0 NODE istype 'reg';
```

写堆栈控制:

```

[ST7..ST0]    := [DBUS7..DBUS0];
[ST7..ST0].CE = !STEN;
[ST7..ST0].CLK = CK;

```

读堆栈控制:

```

[DBUS7..DBUS0] = [R07..R00];
[DBUS7..DBUS0].OE = !X2 & X1 & !X0;

```

用 EPLD 实现 R0..R3 功能

定义:

```

R07..R00 NODE istype 'reg';
R17..R10 NODE istype 'reg';
R27..R20 NODE istype 'reg';
R37..R30 NODE istype 'reg';

```

写寄存器控制:

```
[R07..R00]    := [DBUS7..DBUS0];
[R07..R00].CE = !PWR & !IR1 & !IR0;
[R07..R00].CLK = CK;
[R17..R10]    := [DBUS7..DBUS0];
[R17..R10].CE = !PWR & !IR1 & IR0;
[R17..R10].CLK = CK;
[R27..R20]    := [DBUS7..DBUS0];
[R27..R20].CE = !PWR & IR1 & !IR0;
[R27..R20].CLK = CK;
[R37..R30]    := [DBUS7..DBUS0];
[R37..R30].CE = !PWR & IR1 & IR0;
[R37..R30].CLK = CK;
```

读寄存器控制:

```
WHEN !IR1 & !IR0 THEN
  [DBUS7..DBUS0] = [R07..R00];
ELSE WHEN !IR1 & IR0 THEN
  [DBUS7..DBUS0] = [R17..R10];
ELSE WHEN IR1 & !IR0 THEN
  [DBUS7..DBUS0] = [R27..R20];
ELSE WHEN !IR1 & IR0 THEN
  [DBUS7..DBUS0] = [R37..R30];

[DBUS7..DBUS0].OE = !PRD;
```

第六章 设计指令/微指令系统

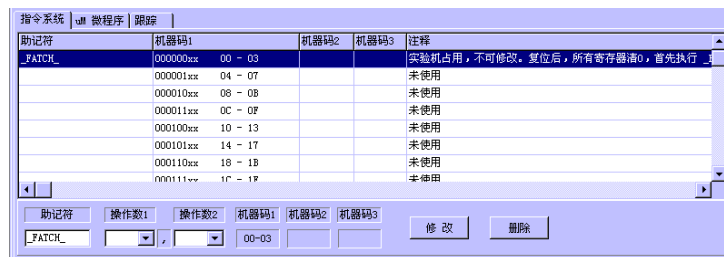
COP2000 计算机组成原理实验仪，可以由用户自己设计指令/微指令系统，前一章的“指令流水实验”就是利用另一套指令/微指令系统来实现指令的流水工作。这样用户可以在现有的指令系统上进行扩充，加上一些较常用的指令，也可重新设计一套完全不同的指令/微指令系统。COP2000 内已经内嵌了一个智能化汇编语言编译器，可以对用户设定的汇编助记符进行汇编。

做为例题，我们建立一个有如下指令的系统：

指令助记符	指令意义描述
LD A, #II	将立即数装入累加器 A
ADD A, #II	累加器 A 加立即数
GOTO MM	无条件跳转指令
OUTA	累加器 A 输出到端口

因为硬件系统需要指令机器码的最低两位做为 R0-R3 寄存器寻址用，所以指令机器码要忽略掉这两位。我们暂定这四条指令的机器码分别为 04H, 08H, 0CH, 10H。其它指令的设计，用户可参考此例，做为练习来完成。

1. 打开 COP2000 组成原理实验软件，选择[文件|新建指令系统/微程序]，清除原来的指令/微程序系统，观察软件下方的“指令系统”窗口，所有指令码都“未使用”。



2. 选择第二行，即“机器码1”为 0000 01XX 行，在下方的“助记符”栏填入数据装载功能的指令助记符“LD”，在“操作数1”栏选择“A”，表示第一个操作数为累加器 A。在“操作数2”栏选择“#II”，表示第二个操作数为立即数。按“修改”按钮确认。
3. 选择第三行，即“机器码1”为 0000 10XX 行，在下方的“助记符”栏填入加法功能的指令助记符“ADD”，在“操作数1”栏选择“A”，表示第一操作数为累加器 A，在“操作数2”栏选择“#II”，表示第二操作数为立即数。按“修改”按钮确认。
4. 选择第四行，即“机器码1”为 0000 11XX 行，在下方的“助记符”栏填入无条件跳转功能的指令助记符“GOTO”，在“操作数1”栏选择“MM”，表示跳转地址为 MM，此指令无第二操作数，无需选择“操作数2”。按“修改”按钮确认。因为硬件设计时，跳转指令的跳转控制需要指令码的第3位和第2位 IR3、IR2 来决定，无条件跳转的控制要求 IR3 必需为1，所以无条件跳转的机器码选择在此行，机器码为 000011XX。关于跳转

控制的硬件设计和实验可参考前面章节。

- 选择第五行，即“机器码 1”为 0001 00XX 行，在下方的“助记符”栏填入输出数据功能的指令助记符“OUTA”，由于此指令隐含指定了将累加器 A 输出到输出商品寄存器，所以不用选择“操作码 1”和“操作数 2”，按“修改”按钮确认。

现在我们只是输入了四条指令(见下图)，

指令系统 uM 微程序 跟踪

助记符	机器码1	机器码2	机器码3	注释
FATCH	000000xx 00 - 03			实验机占用，不可修改。复位后，所有寄存器清0，首
LD A, #II	000001xx 04 - 07	II		
ADD A, #II	000010xx 08 - 0B	II		
GOTO III	000011xx 0C - 0F	III		
OUTA	000100xx 10 - 13			
	000101xx 14 - 17			未使用

助记符

操作数1

操作数2

机器码1

机器码2

机器码3

修改

删除

GOTO

III

/

0C-0F

III

下面要做的是根据指令的功能来设计相应的微程序。

- 将窗口切换到“uM 微程序”窗口，现在此窗口中所有微指令值都是 0FFFFFFH，也就是无任何操作，我们需要在此窗口输入每条指令的微程序来实现该指令的功能。

指令系统		微程序		跟踪																																													
助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC																																								
FATCH		-> 00	FFFFFF				A输出		+1																																								
		01	FFFFFF				A输出		+1																																								
		02	FFFFFF				A输出		+1																																								
		03	FFFFFF				A输出		+1																																								
LD	A, #II	04	FFFFFF				A输出		+1																																								
		05	FFFFFF				A输出		+1																																								
		06	FFFFFF				A输出		+1																																								
		07	FFFFFF				A输出		+1																																								
<div>◀</div>																																																	
<div>▶</div>																																																	
XRD	EMWR	EMRD	PCOE	EMEN	IREN	EINT	ELP	MAREN	MAROE	OUTEN	STEN	SRD	RWR	CH	FEN	X2	X1	X0	WEN	AEN	S2	S1	S0																										
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																										
PC:00 uPC:00 A:00 W:00 C:0 Z:0										RD:00 R1:00 R2:00 R3:00										IR:00 ST:00 IA:EO MAR:00										IX:00 OUT:00										AEN:FF DBUS:FF IBUS:FF									

- 每个程序开始要执行的第一条微指令应是取指操作，因为程序复位后，PC 和 uPC 的值都为 0，所以微程序的 0 地址处就是程序执行的第一条取指的微指令。取指操作要做的工作是从程序存储器 EM 中读出下条将要执行的指令，并将指令的机器码存入指令寄存器 IR 和微程序计数器 uPC 中，读出下条操作的微指令。根据此功能，首先选中“_FATCH_”指令的第一行，观察窗口下方的各控制信号，有“勾”表示信号为高，处于无效状态，去掉“勾”信号为低，为有效状态。要从 EM 中读数，EMRD 必需有效，去掉信号下面的“勾”使其有效；读 EM 的地址要从 PC 输出，所以 PCOE 要有效，允许 PC 输出，去掉 PCOE 下面的“勾”，PCOE 有效同时还会使 PC 加 1，准备读 EM 的下一地址；IREN 是将 EM 读出的指令码存入 uPC 和 IR，所以要去掉 IREN 的“勾”使其有效。这样，取指操作的微指令就设计好了，取指操作的微指令的值为 0CBFFFFH。
- 现在我们来看把立即数装入累加器 A 要做哪些工作，首先要从 EM 中读出立即数，并送到数据总线 DBUS，再从 DBUS 上将数据打入累加器 A 中，按照这个要求，从 EM 中读数据，EMRD 应该有效，EM 的地址由 PC 输出，PCOE 必需有效，读出的数据送到 DBUS，EMEN 也应有效，要求将数据存入 A 中，AEN 也要有效，选中“LD A, #II”指令的第一行，根据前面描述，将所有有效位下面的“勾”去掉，使其有效，这条微指令的值为 0C7FFF7H。为了保证程序的连续执行，每条指令的最后必需是取指令，取出下条将要

执行的指令。选中指令的第二行（第二条微指令）填入取指操作所需的有效位，（取操作描述可见第7步）。

9. 本指令为立即数加法指令，立即数加可分两步，首先从 EM 中读出立即数，送到 DBUS，并存入工作寄存器 W 中，从 EM 中读数，EMRD 应有效，读 EM 的地址由 PC 输出，PCOE 要有效，读出的数据要送到 DBUS，EMEN 应有效，数据存入 W 中，WEN 应有效，根据描述，选中“ADD A,#II”指令的第一行，将有效信号的“勾”去掉，使其有效，这条微指令的值为 0C7FFE FH。第二步，执行加法操作，并将结果存入 A 中。执行加法操作，S2S1S0 的值应为 000(二进制)，结果无需移位直接输出到 DBUS，X2X1X0 的值就要为 100(二进制)，从 DBUS 将数据再存入 A 中，AEN 应有效。与此同时，ABUS 和 IBUS 空闲，取指操作可以并行执行，也就是以 PC 为地址，从 EM 中读出下条将要执行指令的机器码，并打入 IR 和 uPC 中，根据取指操作的说明，EMRD、PCOE、IREN 要有效，根据上面描述，选中该指令的第二行，将 EMRD、PCOE、IREN、X2X1X0、AEN、S2S1S0 都置成有效和相应的工作方式，此微指令的值为 0CBFF90H。
10. “GOTO MM”为无条件跳转，所要执行的操作为从 EM 中读出目标地址，送到数据总线 DBUS 上，并存入 PC 中，实现程序跳转。从 EM 中读数，EMRD 要有效，读 EM 的地址由 PC 输出，PCOE 有效，数据送到 DBUS，EMEN 要有效，将数据打入 PC 中，由两位决定，ELP 有效，指令寄存器 IR 的第三位 IR3 应为 1，由于本指令机器码为 0CH，存入 IR 后，IR3 为 1。选中“GOTO MM”指令的第一行，将上面的 EMRD、PCOE、EMEN、ELP 设成低，使其成为有效状态，结合指令的第三位，实现程序跳转，这条微指令的值为 0C6FFF FH。下条微指令应为取指操作，选中此指令的第二行，将 EMRD、PCOE、IREN 设成有效，微指令的值为 0CBFFFFH。
11. “OUTA”，将累加器的内容输出到输出端口。其操作为累加器 A 不做运算，直通输出，ALU 结果不移位输出到 DBUS，DBUS 上的数据存入输出商品 OUT。累加器 A 直通输出结果，S2S1S0 值要为 111(二进制)，ALU 结果不移位输出到数据总线 DBUS，X2X1X0 的值要等于 100(二进制)，DBUS 数据要打入 OUT，那么 OUTEN 应有效。与此同时，ABUS 和 IBUS 空闲，取指操作可以并行执行，也就是以 PC 为地址，从 EM 中读出下条将要执行指令的机器码，并打入 IR 和 uPC 中，根据取指操作的说明，EMRD、PCOE、IREN 要有效，综上所述，选中此指令的第一行，将 EMRD、PCOE、IREN、OUTEN、X2X1X0、S2S1S0 置成有效状态和相应的工作方式，微指令的值为 0CBDF9 FH。

指令系统	微程序	跟踪	助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC
			FETCH	T0	00	CBFFFF		指令寄存器IR	PC输出	A输出		写入	+1
					01	FFFFFF				A输出		+1	
					02	FFFFFF				A输出		+1	
					03	FFFFFF				A输出		+1	
			LD A, #II	T1	04	CTFFFF	寄存器值EM	寄存器A	PC输出	A输出		+1	+1
				T0	05	CBFFFF		指令寄存器IR	PC输出	A输出		写入	+1
					06	FFFFFF				A输出		+1	
					07	FFFFFF				A输出		+1	
			ADD A, #II	T1	08	CTFFF	寄存器值EM	寄存器W	PC输出	A输出		+1	+1
				T0	09	CBFF90	ALU直通	寄存器A 指令寄存器IR	PC输出	加运算		写入	+1
					0A	FFFFFF				A输出		+1	
XRD EMRD EMRD PCOE EMEN IREN EINT ELP MAREN MAROE OUTEN STEN ERD EWR CN FEN X2 X1 X0 WEN AEN S2 S1 S0 <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>													
PC:06 uPC:0C A:02 W:01 C:0 Z:0 R0:00 R1:00 R2:00 R3:00 IR:0C ST:00 IA:00 MAR:00 IN:00 OUT:02 ABUS:05 DBUS:02 IBUS:0C													

12. 选择菜单[文件]保存指令系统/微程序]功能，将新建的指令系统/微程序保存下来，以便以后调用。为不与已有的两个指令系统冲突，将新的指令系统/微程序保存为“INST3.INS”。



13. 在源程序窗口输入下面程序

```

LD    A, #0
LOOP:
ADD   A, #1
OUTA
GOTO  LOOP

```

14. 将程序另存为 NEW_INST.ASM，将程序汇编成机器码，观察反汇编窗口，会显示出程序地址、机器码、反汇编指令。

程序地址	机器码	反汇编指令	指令说明
00	04 00	LD A, #00	立即数 00H 存入累加器 A
02	08 01	ADD A, #01	累加器 A 值加 1
04	10	OUTA	累加器 A 输出到输出端口 OUT
05	0C 02	GOTO 02	程序无条件跳转到 02 地址

15. 按快捷图标 F7，执行“单微指令运行”功能，观察执行每条微指令时，数据是否按照设计要求流动，寄存器的输入/输出状态是否符合设计要求，各控制信号的状态，PC 及 uPC 如何工作是否正确。

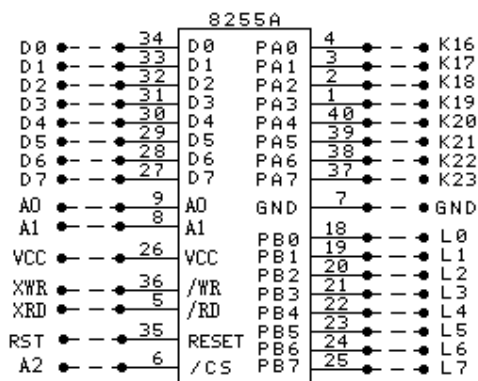
到此为止，我们利用 COP2000 软件系统已经建成了一个新的指令系统/微程序。新的指令系统从汇编助记符到指令机器码到微指令都与原来的指令系统有所不同。做为例子，我们只创建了四条指令，对于其它指令，用户可以为练习来扩充完整。成为一个真正的指令系统。可以用快捷图标上的“生成组合逻辑控制器 ABEL 程序”功能将微程序转换组合逻辑控制方式的 ABEL 语言程序，在 EDA 开发系统上编译后，编程到 CPLD 上，用组合逻辑来控制程序的运行。

第七章 扩展实验

COP2000 计算机组成原理实验仪的原型机具有对外部设备操作的指令，用这两条的输入、输出指令可以对扩展的外部设备进行操作，实现功能的扩展。例如扩展外部的 8255 来扩展输入、输出端口，扩展外部的 8253 来扩展定时器/计数器等等。

扩展实验 1：用 8255 扩展 I/O 端口实验

1. 将 8255 插在 40 芯紧锁座上，按图和接线表连接好 8255 各信号线。



连接线表

连接	8255 管脚	接入孔
1	21	L3
2	22	L4
3	23	L5
4	24	L6
5	25	L7
6	26	+5V
7	27	D7
8	28	D6
9	29	D5
10	30	D4
11	31	D3
12	32	D2
13	33	D1
14	34	D0
15	35	RST
16	36	XWR
17	37	K23
18	38	K22
19	39	K21
20	40	K20

连接	8255 管脚	接入孔
21	20	L2
22	19	L1
23	18	L0
24	17	不接
25	16	不接
26	15	不接
27	14	不接
28	13	不接
29	12	不接
30	11	不接
31	10	不接
32	9	A0
33	8	A1
34	7	GND
35	6	A2
36	5	XRD
37	4	K16
38	3	K17
39	2	K18
40	1	K19

2. 打开 COP2000 计算机组成原理实验仪电源，运行 COP2000 软件，将软件连接到实验仪硬件。输入下面程序：（或从 COP2000 目录下调入 EX8255.ASM）。

```

CONTROL EQU 03H
PORTA EQU 00H
PORTB EQU 01H

MOV A, #90H
WRITE CONTROL, A
LOOP:
  READ A, PORTA
  CPL A
  WRITE PORTB, A
  JMP LOOP

END

```

3. 将程序编译后全速执行，程序从 8255 的 PA 口读回数据，取反后输出到 PB 口，重复循环。拨动 K16-K23 开关，可以看到 L0-L7 上有相应的输出。

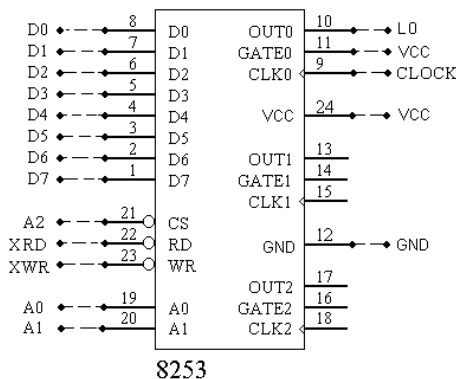
扩展试验 2：用 8253 扩展定时器试验

1. 将 8253 插在 40 芯紧锁座上，按图和接线表接好 8253 的信号线。

接线表

连接	8253 管脚	接入孔
1	13	不接
2	14	不接
3	15	不接
4	16	不接
5	17	不接
6	18	不接
7	19	A0
8	20	A1
9	21	A2
10	22	XRD
11	23	XWR
12	24	+5V

连接	8253 管脚	接入孔
13	12	GND
14	11	+5V
15	10	L0
16	9	clock
17	8	D0
18	7	D1
19	6	D2
20	5	D3
21	4	D4
22	3	D5
23	2	D6
24	1	D7



2. 打开 COP2000 计算机组成原理实验仪电源，运行 COP2000 软件，将软件连接到实验仪硬件。输入下面程序：（或从 COP2000 目录下调入 EX8253.ASM）。

```

CONTROL EQU 03h
COUNT0 EQU 00h
TIMES EQU 06h

MOV A, #00110110B
WRITE CONTROL, A
MOV A, #TIMES
WRITE COUNT0, A ;低字节
MOV A, #0
WRITE COUNT0, A ;高字节
LOOP:
  NOP
  JMP LOOP

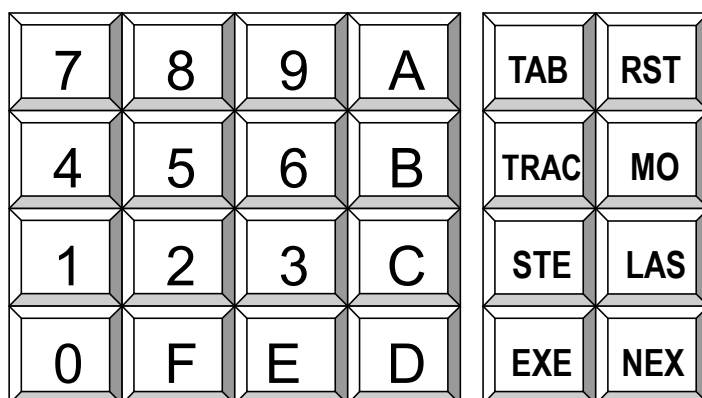
END

```

3. 将程序汇编后单步执行到循环处，再单步观察 L0 灯的翻转情况，如果全速执行 L0 翻转过快，可对 8253 计数器高字节写入一个数字，再全速执行，观察 L0 灯。

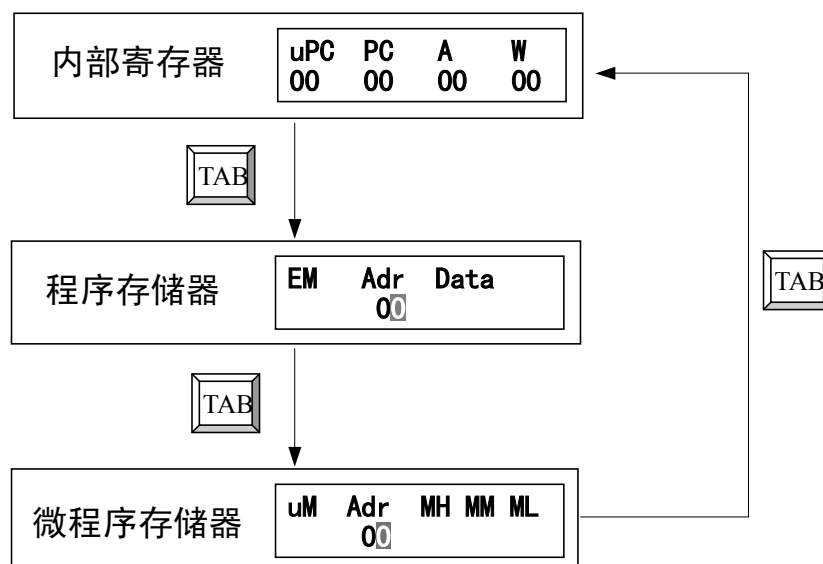
第八章 实验仪键盘使用

伟福的 COP2000 计算机组成原理实验仪除了可以连在 PC 机上调试程序，也可以用实验仪上自带的键盘输入程序及微程序，并可以单步调试程序和微程序，在显示屏上观察各内部寄存器的值，编辑修改程序和微程序存储器。



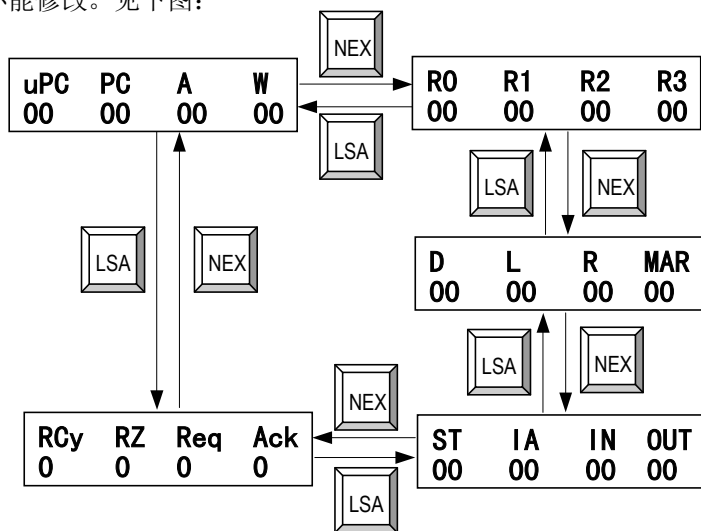
显示屏的显示内容分三个主菜单：

1. 观察内部寄存器、2. 观察和修改程序存储器、3. 观察和修改微程序存储器。三个主菜单用 TAB 键切换。如下图：



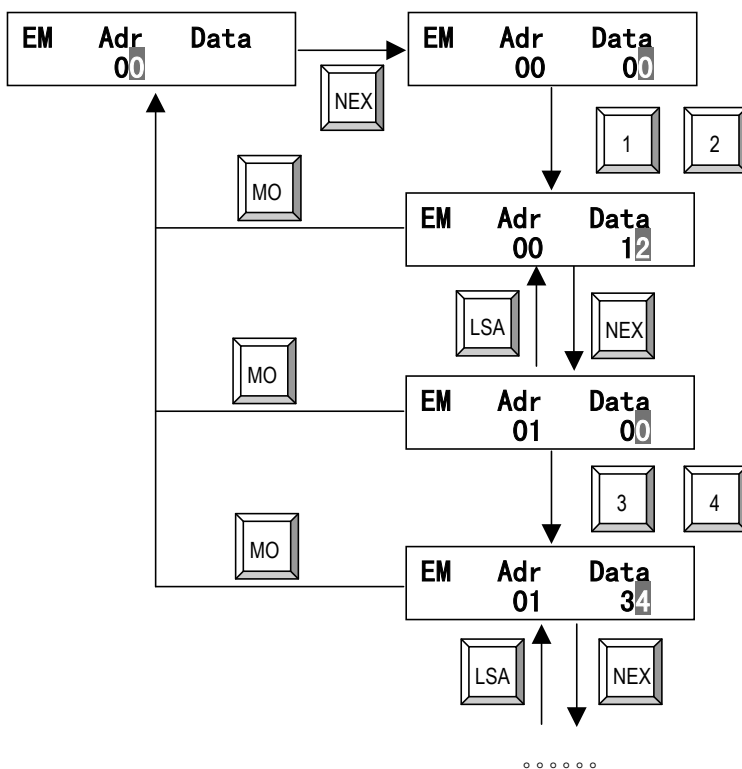
1. 观察内部寄存器：

内部寄存器的内容分五页显示，用 LAST 或 NEXT 键向前或向后翻页。内部寄存器由程序执行结果决定，不能修改。见下图：



2. 观察、修改程序存储器内容：

显示屏显示如下图，其中“Adr”表示程序存储器地址，“Data”表示该地址中数据。光标初始停在“Adr”处，此时可以用数字键输入想要修改的程序地址，也可以用 NEXT 和 LAST 键将光标移到“Data”处，输入或修改此地址中的数据。再次按 NEXT 或 LAST 键可自动将地址+1 或将地址-1，并可用数字键修改数据。按 MON 键可以回到输入地址的状态。见下图。





3. 观察、修改微程序存储器内容:


微程序存储器数据的观察、修改与上面程序存储器的观察修改方法相似，不同的是微程序要输入 3 个字节，而程序存储器的修改只要输入 1 个字节。微程序观察修改的显示屏显示如下图，其中“Adr”表示微程序地址，“MH”表示微程序的高字节，“MM”表示微程序的中字节，“ML”表示微程序的低字节。




使用实验仪键盘可以用三种方法调试程序，程序单步、微程序单步、全速执行。当用键盘调试程序时，显示屏显示寄存器第一页的内容。




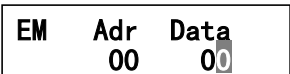


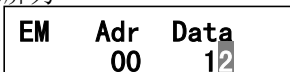


 为微程序单步执行键，每次按下此键，就执行一个微程序指令，同时显示屏显示微程序计数器、程序计数器、A 寄存器、W 寄存器的值。可以通过 NEXT 或 LAST 键翻页观察其它寄存器的值。也可以用“CLOCK”按键给出微程序执行的每个时钟，当 CLOCK 按下和松开时，观察各个寄存器的输出和输入灯的状态。

 为程序单步执行键，每次按下此键，就执行一条程序指令，同时显示屏显示微程序计数器、程序计数器、A 寄存器、W 寄存器的值。可以通过 NEXT 或 LAST 键翻页观察其它寄存器的值。



 为全速执行键，按下此键时，程序就会全速执行。显示屏显示“Running...”，按键盘任一键中止程序执行。

 复位键，按下此键，程序中止运行，所有寄存器清零，程序指针回到 0 地址。

举例：用键盘输入以下程序代码：

1. 按  键，直到显示屏显示内容为 
2. 按  键，光标移到“Data”下，显示屏为 
3. 按   两个数字键，显示屏为 
4. 按  键，地址+1，显示屏为 
5. 按   两个数字键，显示屏为 

重复 4、5 两步，直到输入所有的程序代码。

在第 1 步时，光标停在“Adr”处，可以按数字键  ...  输入要修改的程序存储器的地址，然后再按 NEXT 键输入程序代码。如果光标移到“Data”下，而此时又想改变地址，可以按 MON 键，将光标移回到“Adr”处，按数字键输入地址。

输入微程序代码的方法与此相似，不同的是程序只需输入两个数字，即一个字节，而微程序要输入 6 个数字，即三个字节。如果多于 6 个数字会自动从右向左滚动，保留最后 6 个数字。如果输入不足 6 个数字就用 NEXT 或 LAST 翻页，则只有被改动的几个数字有效，其它数字不变。

4. 用小键盘调试实验一

程序地址	机器码	反汇编指令	指令说明
00	7C 12	MOV A, #12	立即数 12H 送到累加器 A
02	70	MOV A, R0	寄存器 R0 送到累加器 A
03	74	MOV A, @R0	R0 间址的存储器内容送到累加器 A
04	78 01	MOV A, 01	存储器 01 单元内容送到累加器 A
06	C0	IN	端口 IN 内容输入到累加器 A
07	C4	OUT	累加器 A 内容输出到端口 OUT

2el

一：输入机器码

按 TAB 键选择 EM

顺序输入机器码：7C 12 70 74 78 01 C0 C4

输完机器码后按 RST 复位

二：单步执行微程序

按 RST 复位键后，PC=0，uPC=0

uM 输出 24 位微程序：CB FF FF 此微指令为取指指令

第一条微指令

按下（按住不放）CLOCK 时钟键，此时：

PC 地址输出红灯亮：表明 EM 地址由 PC 提供

EM 读红灯亮：表明 EM 输出数据

IR 打入黄灯亮：取出的指令将被写入 IR 及 uPC

放开 CLOCK 时钟键，完成一个时钟，此时：

PC 值为 01（时钟上升沿 PC+1）

IR 值为 7C，uPC 值为 7C（指令码）

uM 输出为：C7 FF F7（EM 值送 A）

第二条微指令

按下（按住不放）CLOCK 时钟键，此时：

EM 读红灯亮：表明 EM 输出数据

EM 与总线连接红灯亮：表明 EM 输出到总线

A 打入黄灯亮：总线数据（12H）将被写入 A
放开 CLOCK 时钟键，完成一个时钟，此时：
PC 值为 02（时钟上升沿 PC+1）
A 值为 12
uPC 值为 7D（时钟上升沿 uPC+1）
uM 输出为：CB FF FF（取指指令）

第三条微指令

按下（按住不放）CLOCK 时钟键，此时：
PC 地址输出红灯亮：表明 EM 地址由 PC 提供
EM 读红灯亮：表明 EM 输出数据
IR 打入黄灯亮：取出的指令将被写入 IR 及 uPC
放开 CLOCK 时钟键，完成一个时钟，此时：
PC 值为 03（时钟上升沿 PC+1）
IR 值为 70，uPC 值为 70（指令码）
uM 输出为：FF F7 F7（R? 值送 A）

第四条微指令

按下（按住不放）CLOCK 时钟键，此时：
R0 输出红灯亮：表明 R0 输出数据
A 打入黄灯亮：总线数据（00）将被写入 A
放开 CLOCK 时钟键，完成一个时钟，此时：
PC 值为 03（时钟上升沿 PC+1）
A 值为 00
uPC 值为 71（时钟上升沿 uPC+1）
uM 输出为：CB FF FF（取指指令）

第五条微指令

按下（按住不放）CLOCK 时钟键，此时：
PC 地址输出红灯亮：表明 EM 地址由 PC 提供
EM 读红灯亮：表明 EM 输出数据
IR 打入黄灯亮：取出的指令将被写入 IR 及 uPC
放开 CLOCK 时钟键，完成一个时钟，此时：
PC 值为 04（时钟上升沿 PC+1）
IR 值为 74，uPC 值为 74（指令码）
uM 输出为：FF 77 FF（R? 值送 MAR）

第六条微指令

按下（按住不放）CLOCK 时钟键，此时：
R0 输出红灯亮：表明 R0 输出数据

MAR 打入黄灯亮：总线数据（00）将被写入 MAR
放开 CLOCK 时钟键，完成一个时钟，此时：
MAR 值为 00
uPC 值为 75（时钟上升沿 uPC+1）
uM 输出为：D7 BF F7（EM 值送 A）

第七条微指令

按下（按住不放）CLOCK 时钟键，此时：
MAR 地址输出红灯亮：表明 EM 地址由 MAR 提供
EM 读红灯亮：表明 EM 输出数据
A 打入黄灯亮：总线数据将被写入 A
放开 CLOCK 时钟键，完成一个时钟，此时：
A 值为 7C
uPC 值为 76
uM 输出为：CB FF FF（取指）

用同样的方法执行余下的指令。也可以用 TRACE 或 STEP 键执行指令。

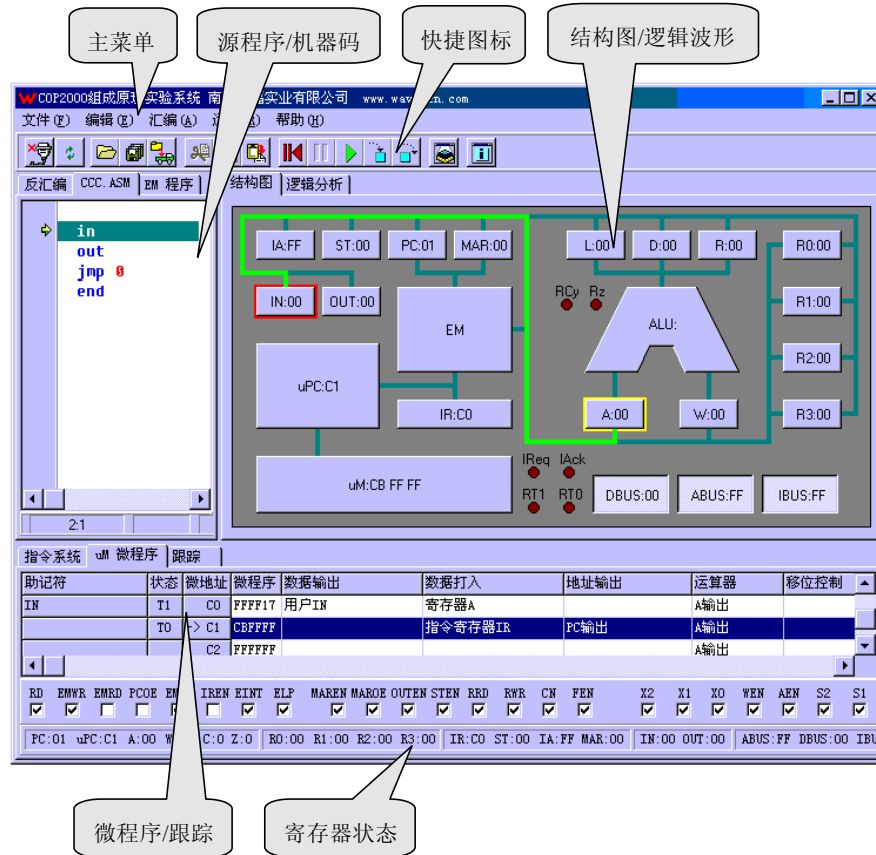
在做分部实验时，实验仪键盘不再起作用，显示屏显示内容为 8 芯电缆的连接方式及数据总线的十六进制值和二进制值。例如显示屏显示内容如下：表示 J1 通过 8 芯电缆接到 J3，当前 DBUS 的值为十六进制值 0AAH，后面为二进制值 10101010。

J1 <=> J3 DBUS:AA 10101010

COP2000 组成原理实验仪硬件的自动检测步骤

1. 将 IA 的开关拨成“11100000”，将中断地址设成 0E0H，J1 接 J2 控制开关拨到“微程序”方向。
2. 按住“RST”键不松，同时开机。在显示屏有显示后，松开“RST”键。
3. 实验仪进行自测，自测后，显示“1234”，分别按 1、2、3、4 键测试各 LED 灯的情况。
按 1：检测各寄存器的 LED，LED 从右至左逐个点亮，8 段管显示 01-80 数字。
按 2：检测 uM 输出的 LED，24 位分三段，从右至左逐个点亮。
按 3：检测各个寄存器输出 LED（红色 LED），循环点亮每个寄存器的输出 LED。
按 4：检测各个寄存器输入 LED（黄色 LED），循环点亮每个寄存器的输入 LED。
4. 手动检测键盘，将键盘每个键都按一次，显示屏会显示相应的键码。
5. 检测 24 个开关，将开关上下拨动，观察灯是否有正确地变化。
6. 将开关拨到“组合逻辑”方向，不按“RST”开机，按“EXEC”键运行程序，可以看到累加器 A 做加 1 运算，按“INT”键，产生中断，将累加器 A 的值输出。
7. 将开关拨到“微程序”方向，不按“RST”开机，按“EXEC”键运行程序，累加器 A 开始加 1，按“INT”键，将累加器 A 的值输出。

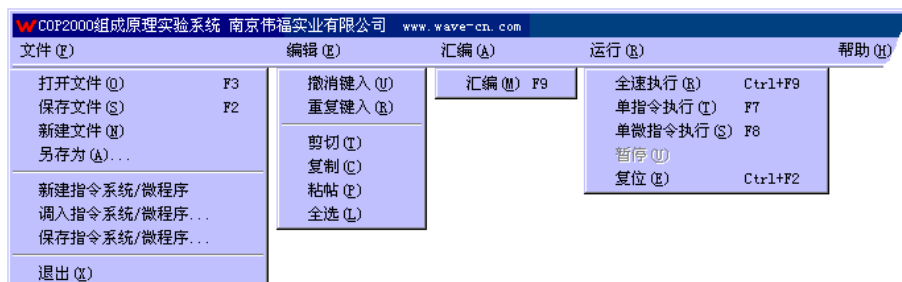
第九章 COP2000 集成开发环境使用



COP2000 集成调试软件界面分六部分：

- 1)主菜单区 实现实验仪的各项功能的菜单，包括[文件][编辑][汇编][运行][帮助]五大项，各项下面做详细介绍。
- 2)快捷图标区 快速实现各项功能按键。
- 3)源程序/机器码区 在此区域有源程序窗口、反汇编窗口、EM 程序代码窗口。源程序用于输入、显示、编辑汇编源程序；反汇编窗口显示程序编译后的机器码及反汇编的程序；EM 程序代码窗口用数据方式机器码。
- 4)结构图/逻辑波形区 结构图能结构化显示模型机的各部件，以及运行时数据走向寄存器值；逻辑波形图能显示模型机运行时所有信号的时序。
- 5)微程序/跟踪区 微程序表格用来显示程序运行时微程序的时序，及每个时钟脉冲各控制位的状态，跟踪表用来记录显示程序及微程序执行的轨迹，指令系统可以帮助你设计新的指令系统。
- 6)寄存器状态区 用来显示程序执行时各内部寄存器的值。

1) 主菜单



主菜单分[文件][编辑][汇编][运行][帮助]五部分

[文件 | 打开文件] 打开汇编程序或文本文件，若打开的是汇编程序（后缀为 ASM），会把程序放在源程序区，若是其它后缀的文本文件就把打开的文件放在结构图区。

[文件 | 保存文件] 将修改过的文件保存。不论是汇编源程序还是其它文本文件，只要被修改过，就会被全部保存。

[文件 | 新建文件] 新建一个空的汇编源程序。

[文件 | 另存为...] 将汇编源程序换名保存。

[文件 | 新建指令系统/微程序] 新建一个空的指令系统和微程序，用于自己设计指令系统。见微程序区的指令系统。

[文件 | 调入指令系统/微程序] 调入设计好的指令系统和微程序定义。

[文件 | 保存指令系统/微程序] 保存自己设计的指令系统和微程序。

[文件 | 退出] 退出集成开发环境。

[编辑 | 撤消键入] 撤消上次输入的文本。

[编辑 | 重复键入] 恢复被撤消的文本。

[编辑 | 剪切] 将选中的文本剪切到剪贴板上。

[编辑 | 复制] 将选中的文本复制到剪贴板上。

[编辑 | 粘贴] 从剪贴板上将文本粘贴到光标处。

[编辑 | 全选] 全部选中文本

[汇编 | 汇编] 将汇编程序汇编成机器码。

[运行 | 全速执行] 全速执行程序。

[运行 | 单指令执行] 每步执行一条汇编程序指令。

[运行 | 单微指令执行] 每步执行一条微程序指令。

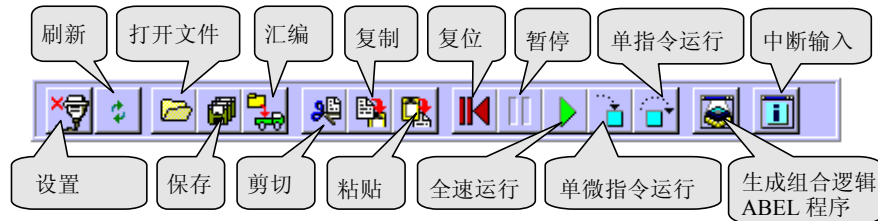
[运行 | 暂停] 暂停程序的全速执行。

[运行 | 复位] 将程序指针复位到程序起始处。

[帮助 | 关于] 有关 COP2000 计算机组成原理实验仪及软件的说明。

[帮助 | 帮助] 软件使用帮助。

2) 快捷键图标



伟福的计算机组成原理实验仪既可以带硬件实验仪进行实验，也可以用集成开发环境软件来模拟模型机的运行。图标的“设置”功能就是选择用 COP2000 硬件实验仪，还是使用软件模拟器。若是使用硬件实验仪，还要选择与实验仪通信所用串行口。

“刷新”功能就是在程序运行过程中刷新各寄存器的值。以便在程序全速执行时观察寄存器的内容。

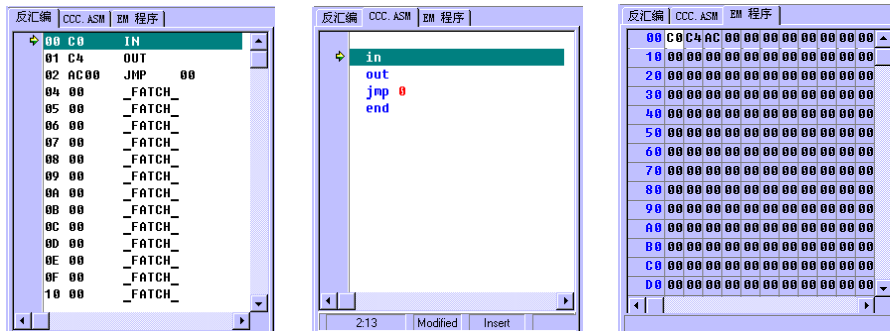
文件的“打开”、“保存”功能与主菜单的相应功能一样。文件的编辑功能，执行控制功能与主菜单也相同。

“生成组合逻辑 ABEL 程序”功能就是在你用微程序控制方式设计了一套指令系统，并且验证无误后，帮助你生成组合逻辑控制方式的 ABEL 程序。

“中断输入”功能，就是在软件模拟中断程序时，用此键来申请中断。



3) 源程序/机器码窗口



源程序/机器码区分三个窗口：反汇编窗口、源程序窗口、EM 程序窗口

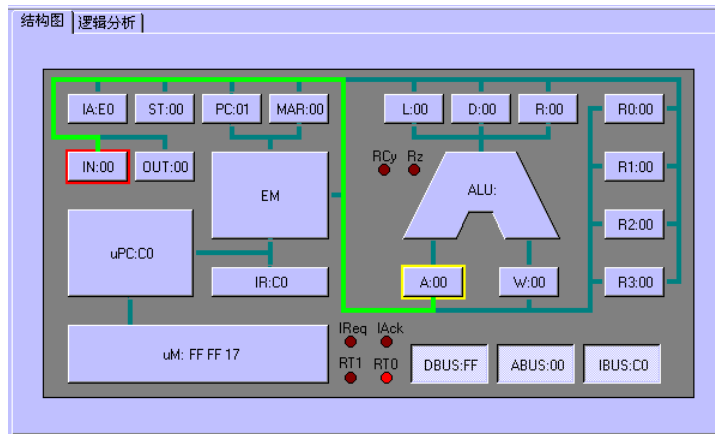
源程序窗口用于输入、修改程序。在[文件]菜单中打开一个以“*.ASM”为后缀的文件时，系统认为此文件为源程序，其内容会在源程序窗口显示，并可以修改，然后编译。若

再次打开以“*.ASM”后缀的文件，则新文件将旧文件覆盖，在源程序窗口只显示最新打开的汇编源程序。若打开其它后缀的文件，系统会将其内容显示在“结构图/逻辑分析”窗口区。在[文件]菜单中，使用“新建文件”功能，会清除源程序窗口的内容，让用户重新输入新的程序。

反汇编窗口用于显示程序地址、机器码、反汇编后的程序。对于一些双字节的指令，紧随其后的机器码、反汇编程序显示是无效的。

EM 程序窗口以十六进制数据的形式显示程序编译后的机器码。可以直接输入数值来修改机器码。

4) 结构图/逻辑分析窗口

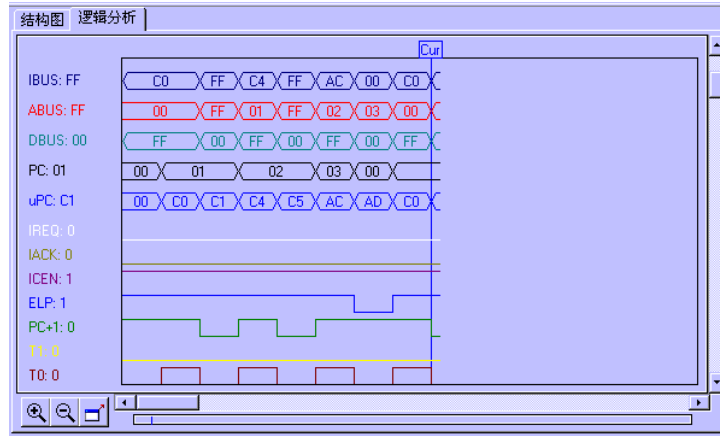


结构图/逻辑分析区分三种窗口，结构图窗口、逻辑分析窗口、其它文本显示窗口。

结构图窗口显示模型机的内部结构，包括各种寄存器（A、W、R0-R3、MAR、IR、ST、L、D、R）、运算器（ALU）、程序指针（PC）、程序存储器（EM）、微程序指针（uPC）、微程序存储器（uM）及各种状态位（RCy、Rz、IReq、IAck），在程序单步运行时，可以在结构图上看到数据的走向及寄存器的输入输出状态。当寄存器或存储器显示为红色框时，表示数据从此流出，当寄存器或存储器显示为黄色框时，表示数据流入此寄存器。此时总线上的值也可以从结构图的下方观察到。其中 DBUS 为数据总线、ABUS 为地址总线、IBUS 为指令总线。RT1、RT0 显示的将要执行的指令的第几个时钟周期。本模型机最多有四个时钟周期，用 RT1、RT0 的 11、10、01、00 四个状态表示。见上图。

逻辑分析窗口显示的是在指令执行时，各种信号的时序波形，包括所有寄存器、所有的控制信号在不同时钟状态下的值，可以直观地看到各种信号彼此之间的先后时序关系。

“Cur”光标表示当前时间，可以移动此光标来选择不同的时间，观察此时间下，各个寄存器、控制信号的逻辑状态。见下图。



在执行“打开文件”时，若打开文件不是汇编程序（后缀不是*.ASM），那么系统会在此区新建一页来显示打开的文件。若文件被修改过，那么在“保存文件”时，会将所有的修改过的文件存盘。

5) 指令/微程序/跟踪窗口

此区分三页：指令系统窗口、微程序窗口、跟踪窗口。

指令系统	uM 微程序	跟踪
助记符	机器码1	机器码2 机器码3 注释
FATCH	00000xxx 00 - 03	实验机占用，不可修改。复位后，所有寄存器清0，首先执行 _FATCH_ 指令取指
	000001xxx 04 - 07	未使用
	000010xxx 08 - 0B	未使用
	000011xxx 0C - 0F	未使用
ADD A, R?	000100xxx 10 - 13	
ADD A, 0R?	000101xxx 14 - 17	

指令系统窗口用于设计用户自己的指令系统，用户借助此窗口可以设计另外一套独立的指令系统，除了一此由于硬件关系不能改变的指令，其它指令都可由用户自己设计。各条指令相应的微程序在“uM 微程序”窗口中设计（见下图）。设计好的指令系统可以用菜单上的[文件 | 保存指令系统/微程序]功能来存盘，便于下次调用。若想为此指令系统生成一套由组合逻辑控制的控制机构，可以用“快捷图标”区的“生成组合逻辑 ABEL 程序”功能来生成 ABEL 程序，编译后编程到组合逻辑控制芯片上即可。

指令系统	uM 微程序	跟踪
助记符	状态	微地址 微程序 数据输出 数据打入 地址输出 运算器 移位控制 uPC PC
FATCH	T0	00 CFFFFFFF 指令寄存器IR PC输出 A输出 写入 +1
	01	FFFFFFF A输出 +1
	02	FFFFFFF A输出 +1
	03	FFFFFFF A输出 +1
UNDEF	T0	04 CFFFFFFF 指令寄存器IR PC输出 A输出 写入 +1
	05	FFFFFFF A输出 +1

uM 微程序窗口用于观察每条指令所对应的微程序的执行过程，以及微代码的状态。在此窗口中，可以看到数据是从何寄存器输出的、数据输入到何寄存器、地址是由 PC 输出还

是由 MAR 输出、运算器在做何种运算、如何移位、uPC 及 PC 如何工作等等。可以通过改变窗口下方的微代码的各个控制位的方式来重新设计微程序，与“指令系统”窗口的指令修改相结合，可以设计自己的指令。

指令系统		uM 微程序		跟踪																			
助记符	状态	微地址	微程序	数据输出	数据打入	地址输出	运算器	移位控制	uPC	PC													
01 OUT	T1	C4	FFDF9F ALU直通		用户OUT		A输出		+1														
	T0	C5	CBFFFF		指令寄存器IR	PC输出	A输出		写入	+1													
02 JMP 00	T1	AC	C6FFFF 存储器值EM		寄存器PC	PC输出	A输出		+1	写入													
	T0	AD	CBFFFF		指令寄存器IR	PC输出	A输出		写入	+1													
00 IN	T1	C0	FFFF17 用户IN		寄存器A		A输出		+1														
	T0	-> C1	CBFFFF		指令寄存器IR	PC输出	A输出		写入	+1													
XRD	EMVR	EMRD	PCOE	EMEN	IREN	EINT	ELP	MAREN	MAROE	OUTEN	STEN	RRD	RWR	CN	FEN	X2	X1	X0	WEN	AEN	S2	S1	S0
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

跟踪窗口显示程序执行过程的轨迹，包括每条被执行的指令、微指令，以及微指令执行时，各控制位、各个寄存器的状态。可以将鼠标移到想相应的程序行或微程序行来显示执行该指令或微指令时，各寄存器、控制位的状态。

6) 寄存器状态

PC:01	uPC:C0	A:00	W:00	C:0	Z:0	R0:00	R1:00	R2:00	R3:00	IR:C0	ST:00	IA:E0	MAR:00	IN:00	OUT:00	ABUS:00	DBUS:FF	IBUS:C0	L:00	D:00	R:00
-------	--------	------	------	-----	-----	-------	-------	-------	-------	-------	-------	-------	--------	-------	--------	---------	---------	---------	------	------	------

寄存器状态区显示程序执行时，各内部寄存器的值。

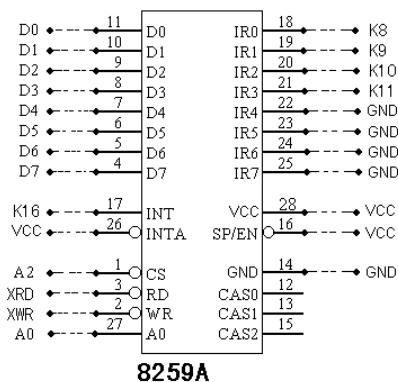
扩展试验 3：用 8259A 做中断控制试验

1. 将 8259A 插在 40 芯紧锁座上，按图和接线表接好 8259 的信号线。

接线表

连接	8259A 管脚	接 入 孔
1	15	不接
2	16	+5V
3	17	K16
4	18	K8
5	19	K9
6	20	K10
7	21	K11
8	22	GND
9	23	GND
10	24	GND
11	25	GND
12	26	+5V
13	27	A0
14	28	+5V

连接	8259A 管脚	接 入 孔
15	14	GND
16	13	不接
17	12	不接
18	11	D0
19	10	D1
20	9	D2
21	8	D3
22	7	D4
23	6	D5
24	5	D6
25	4	D7
26	3	XRD
27	2	XWR
28	1	A2



2. 将 8259A 的 17 脚中断输出脚接到 K16 上，要把 K16 下方的 74HC245 拔掉，防止输出短路，K16 通过输入端口“IN”将 8259A 的 INT 信号接入模型机。
3. 打开 COP2000 计算机组成原理实验仪电源，运行 COP2000 软件，将软件连接到实验仪硬件。输入下面程序：（或从 COP2000 目录下调入 EX8259.ASM）。

```

C8259A equ 00H
C8259B equ 01H
ICW1 equ 00010110b ;ICW1, edge, 4, single, no ICW4
ICW2 equ 00100000b ;Interrupt Addr
OCW1 equ 11111000b ;Low 3 IRQ enabled
EOI equ 00100000b ;End of Interrupt

mov a, #ICW1 ;Initialize 8259A register
write C8259A, a
mov a, #ICW2
write C8259B, a
mov a, #OCW1
write C8259B, a

LOOP:
in ;wait Interrupt
rrc a ;
jc INT ;
jmp LOOP

INT:
read a, C8259A ;read in which IRQ
mov r0, a ;save to R0

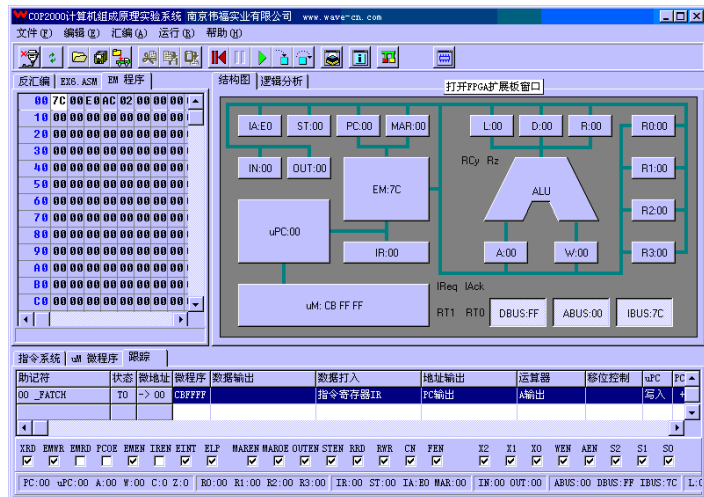
mov a, #EOI
write C8259A, a
jmp LOOP
end

```


4. 将程序汇编后，单步（F8）执行到 LOOP 标号下的“in”指令处，向上拨动 K8~K11 中任一开关，发出中断请求，8259A 产生的中断信号经 K16，到输入端口 IN，经程序读入到累加器 A 中，右移到进位标志 C 中，程序判断是否有中断，若有中断，进位标志 C 置 1，程序跳转到中断处理。此时中断已经响应，可以向下拨回 K8~K11 开关。在中断处理中，读出是哪位中断请求，放入 R0 寄存器中。程序向 8259A 发中断结束指令后，返回主程序等待再次中断。
5. 也可全速执行程序，拨动 K8~K11 开关，观察 R0 寄存器的显示，与中断请求的位是否对应，注意，在程序中，K11 对应的 IRQ4 被屏蔽，拨动 K11 不会产生中断，其它三位都产生中断，并在 R0 寄存器处显示相应中断请求位。

第十章 十六位机 (FPGA) 扩展实验板

通过选配的十六位机(FPGA)扩展板,就可以在 COP2000 计算机组成原理实验仪上做十六位、三十二位模型机的实验。FPGA 扩展板上有: 20 万门大规模 FPGA 芯片, 学生将设计好的电路下载到芯片上, 来完成模型机的功能。可以根据难易程度, 可以先设计 8 位, 再到 16 位, 再到 32 位逐步完成, 灵活多变, 开放性好。学生在设计过程中, 能充分理解模型机中各部件, 各电路的实现方法, 强化学习效果。**64Kx16 位存储器**, 能保存大容量的程序。**12 位八段数码管**, 用于显示模型机内部的寄存器、总线值, 学生在设计时, 可将需要观察的内部寄存器、总线等值接到这些八段管上, 直观地观察模型机运行时内部状态变化。**16 位发光管**, 用于显示模型机内部的状态, 例如进位标志、零标志、中断申请标志等等。**四十路开关**, 用于输入外部信号, 例如在做单步实验时, 这些开关可用来输入地址总线值、数据总线值、控制信号等。



在 COP2000 的主界面上, 按“打开 FPGA 扩展板窗口”按钮, 打开 FPGA 扩展板的界面, 此窗口有两个页面, “结构图”页面和“存储器”页面。

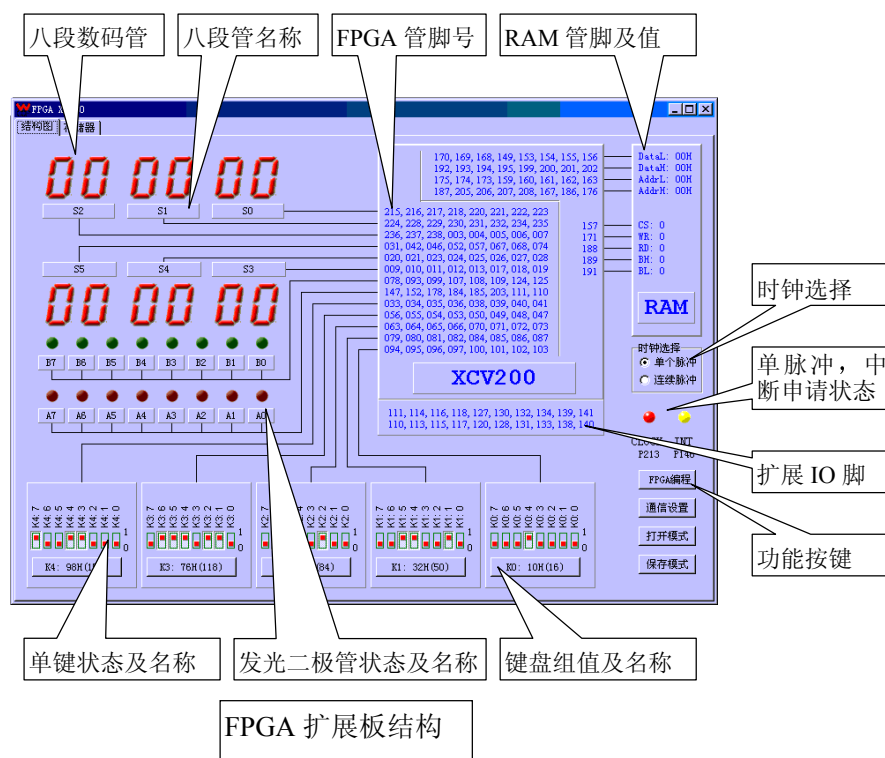
在“结构图”页面上，显示出与实际 FPGA 扩展板相同的器件及各器件与 FPGA XCV200 相连接的管脚号，在实验时，只要将想观察的值从相应的管脚输出，就会在八段数码管或发光二极管上显示出来，注意用八段数码管观察值时，只需直接在管脚上输出数据即可，无需另加译码逻辑电路将数据值译成八段码格式。需要读入键盘值时，就直接从该键盘相连的管脚读入。RAM 与 XCV200 相连的地址线数据线及读写控制线所对应的管脚也显示在图上，当有存储器读写操作时，各信号所对应的值会在图上显示。“CLOCK”“INT”状态灯显示单脉冲信号和中断申请信号的状态，在做模型机总体实验时，学生可以在 COP2000 的主板上按下单脉冲或中断申请按键，键的状态会在此显示。“时钟选择”用于选择总体实验时，模型机的时钟是单脉冲还时连续的高速时钟脉冲。XCV200 下方的管脚号表示 20 个扩展的 IO 信号，当实验中需要另外的输入输出脚时，可以使用这些扩展脚。

“结构图”页面内各器件接到 XCV200 芯片的管脚如下表

结构图内器件	连接 XCV200 的管脚（从高位到低位）
八段数码管 D0	215, 216, 217, 218, 220, 221, 222, 223
八段数码管 D1	224, 228, 229, 230, 231, 232, 234, 235
八段数码管 D2	236, 237, 238, 003, 004, 005, 006, 007
八段数码管 D3	009, 010, 011, 012, 013, 017, 018, 019
八段数码管 D4	020, 021, 023, 024, 025, 026, 027, 028
八段数码管 D5	031, 042, 046, 052, 057, 067, 068, 074
发光二极管[B7..B0]	078, 093, 099, 107, 108, 019, 124, 125
发光二极管[A7..A0]	147, 152, 178, 184, 185, 203, 111, 110
开关组 K0	094, 095, 096, 097, 100, 101, 102, 103
开关组 K1	079, 080, 081, 082, 084, 085, 086, 087
开关组 K2	063, 064, 065, 066, 070, 071, 072, 073
开关组 K3	056, 055, 054, 053, 050, 049, 048, 047
开关组 K4	033, 034, 035, 036, 038, 039, 040, 041
存储器数据线低 8 位	170, 169, 168, 149, 153, 154, 155, 156
存储器数据线高 8 位	192, 193, 194, 195, 199, 200, 201, 202
存储器地址线低 8 位	175, 174, 173, 159, 160, 161, 162, 163
存储器地址线高 8 位	187, 205, 206, 207, 208, 167, 186, 176
存储器控制线	157(CS), 171(WR), 188(RD), 189(BH), 191(BL)
扩展 IO 端口[E18..E0](偶数)	111, 114, 116, 118, 127, 130, 132, 134, 139, 141
扩展 IO 端口[E19..E1](奇数)	110, 113, 115, 117, 120, 128, 131, 133, 138, 140

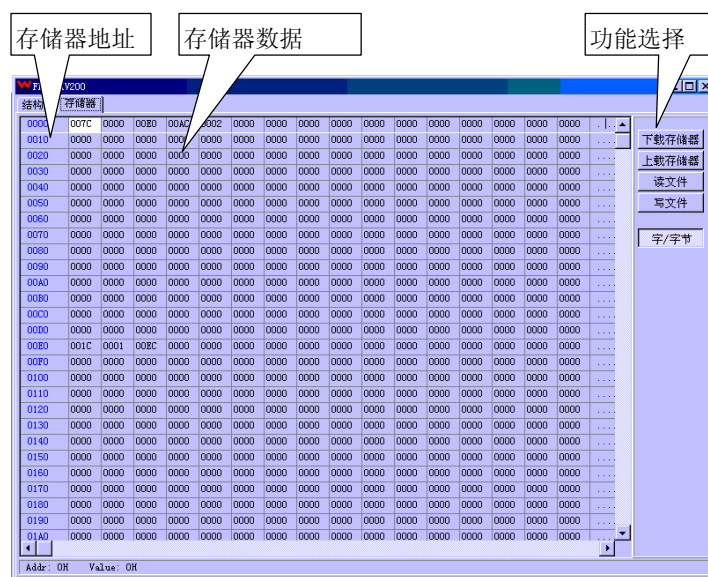
上表中，存储器控制线的 CS 表示片选信号，WR 为写信号，RD 为读信号，BH 为数据高八位选择信号，BL 为数据低八位选择信号。注意 E18、E19 两个脚同时还接到 A0、A1 两个发光二极管上，当这两个扩展 IO 脚状态有所变化时，会在发光二极管上显示出来。

每个器件的名称都可以修改，以适合于不同的应用。单击名称，系统就会弹出改变名称的对话框，让你输入想要定义的名称。关于键盘，即可以定义键盘组（8 个键）的名称，也可定义单个键的名称。这些名称可以保存在文件中，下次实验时无需重新定义，直接从相应文件中读入即可。



“结构图”窗口内有四个功能按键：“FPGA 编程”、“通信设置”、“打开模式”、“保存模式”。“FPGA 编程”是将 XILINX 开发环境中设计生成的*.bit 格式程序文件下载到 XCV200 芯片中，在下载过程，有进度条显示下载进度，如果 COP2000 实验仪上没有插 FPGA 扩展板，或在下载过程出错，系统会显示“FPGA 出错”信息，这时需要插上 FPGA 扩展板或重新编程。如果 COP2000 实验仪没有连接到计算机上，会显示“串口未连接”信息，这时用“通信设置”功能选择好串行口，并连接实验仪。如果实验仪与计算机通信成功，会显示“通信成功”信息，否则会显示“与实验仪通信错误”的信息，这时要检查串口通信电缆是否连接，实验仪是否加电等。“保存模式”是将定义好器件名称保存到文件中，这样下次做相同实验时，就不需要再次定义这些名称，只要用“打开模式”功能，直接打开相应的文件即可。

在“存储器”页面上，显示出 64K x 16 位存储器的内容，你可以在此页面上对存储器中的数据进行修改、下载、上载、从文件中读入、写到文件中等操作，也可以对数据进行填充、移动。在第一次打开“FPGA 扩展板窗口”时，系统会自动将主界面上的“EM 窗口”内容复制到此窗口内。所以在做模型机总体实验时，可以在主界面内调入源程序，并编译，生成机器码，在“EM 窗口”内可以看到程序的机器码，这时在主界面上打开 FPGA 扩展板窗口，转到“存储器”页面，前面生成的机器码已经复制到此窗口内，通过“下载存储器”将程序下载到扩展板的存储器内，就可运行程序了。



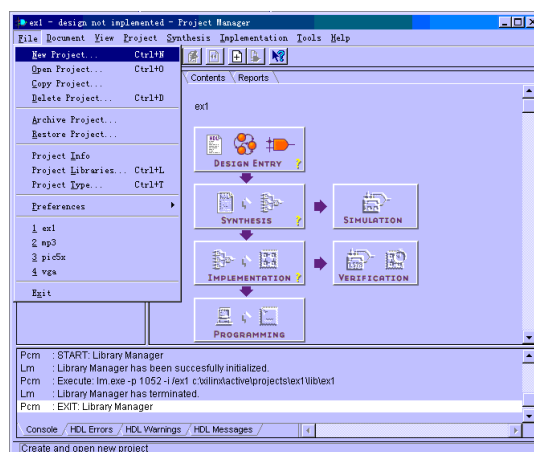
存储器窗口

“存储器”页面中有五个功能键：“下载存储器”、“上载存储器”、“读文件”、“写文件”、“字/字节”。“下载存储器”是将存储器窗口中的数据下载到 FPGA 扩展板上的 64K x 16 位存储器中。在弹出的对话框中输入要下载的起始地址和结束地址，程序会自动将指定地址的数据下载到扩展板上的 RAM 中。“上载存储器”是从扩展板上的 RAM 中读出数据并写到此窗口中，在弹出的对话框中输入要上载数据的起始地址和结束地址，程序会将指定地址的 RAM 数据上载到窗口中。“读文件”从磁盘上读入数据文件，在此窗口中显示。“写文件”是将存储器窗口中的数据写到磁盘上。“字/字节”选择数据显示方式是以‘字’的方式（16 位）显示还是以‘字节’方式（8 位）显示。注意，不论数据显示方式是 16 位方式还是 8 位方式，前面介绍的所有的数据操作（上载、下载、读文件、写文件）都是以 16 位方式来完成的。

如果要改变窗口内数据的值，可以用鼠标选中要修改的单元，直接用键盘输入十六进制数据即可，也可以在数据窗口内按鼠标右键，在弹出菜单中选择“修改”功能，在弹出对话框中可以输入十六进制数据或十进制数据，若在数据最后加上“H”表示输入十六进制数据。在弹出菜单中还可以选择“转到指定地址”、“数据块填充”、“数据块移动”、“读文件”、“写文件”、“改变显示列数”等功能。

第十一章 XILINX 开发环境使用入门

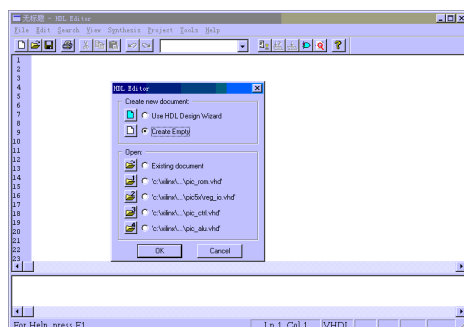
十六位机（FPGA）扩展实验板上用的 20 万门大规模可编程芯片 XCV200 是 XILINX 公司产品。XILINX 公司是世界上最大的 FPGA/EPLD 生产商之一，其 Xilinx Foundation 开发环境也是电子工程师最常用的软件之一。它以流程图的方式引导用户一步一步完成设计。在 Foundation 开发环境中，可以完成设计输入（原理图、VHDL）、逻辑综合、逻辑功能仿真、逻辑编译、功能验证、时序分析、编程下载等 EDA 设计的所有步骤。以 Foundation F4.2i 为例，我们来了解在 XILINX 开发环境下，EDA 设计的流程。



1. 建立设计项目。启动软件打开 EDA 开发环境，选择菜单 [File] 的 [New Project] 功能，出现如图对话框，在“Name”框内填上项目名，例如“EX1”等。项目的设计数据可以保存在软件缺省指定的目录下，软件缺省的设计目录为“C:\XILINX\ACTIVE\PROJECTS”，也可以自己指定设计目录。“Flow”指定设计输入的方法，“Schematic”表示用原理图方式进行设计，“HDL”表示用 VHDL 方式进行设计，这里选中“HDL”。按“OK”确认退出。



2. 建立空设计文件，输入 VHDL 程序。在流程图窗口内，选择文本输入图标，打开 VHDL 输入窗口。如图，选择打开方式为“Create empty”，在空的窗口中输入 VHDL 语言，

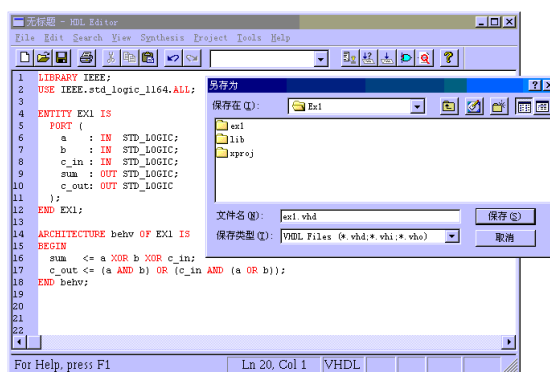


其中“a”和“b”表示全加器的“输入 1”和“输入 2”，“c_in”表示“前级进位输入”，“sum”表示全加器的“和”，“c_out”表示全加器的进位。

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

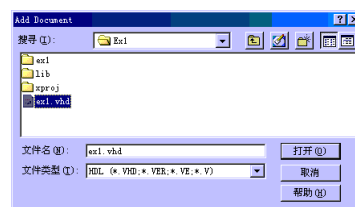
ENTITY EX1 IS
    PORT (
        a      : IN  STD_LOGIC;
        b      : IN  STD_LOGIC;
        c_in   : IN  STD_LOGIC;
        sum    : OUT STD_LOGIC;
        c_out  : OUT STD_LOGIC
    );
END EX1;

ARCHITECTURE behv OF EX1 IS
BEGIN
    sum    <= a XOR b XOR c_in;
    c_out <= (a AND b) OR (c_in AND (a OR b));
END behv;
```

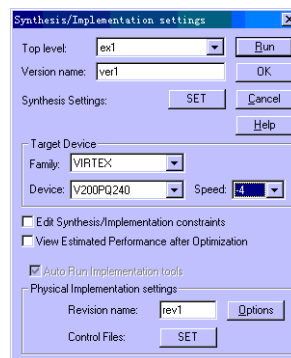
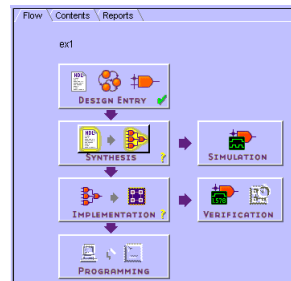


3. 保存 VHDL 文本。在文本窗口内，选择[File]的[Save as...]功能，将 VHDL 保存起来，保存的路径软件会自动给出，一般不要改动，保存类型选择“VHDL File”。文件名为 EX1.VHD。保存程序后，可以对 VHDL 文件进行“语法检查”，选择文本编辑窗口的菜单[Synthesis]的[Check Syntax]功能对 VHDL 进行检查。

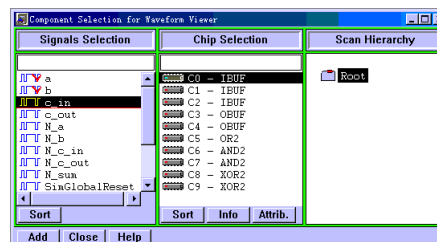
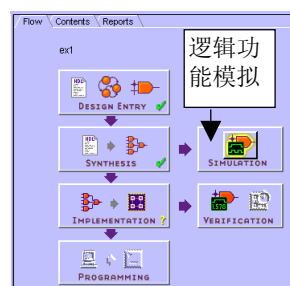
4. 将 VHDL 文件加入项目中。前面建立的项目为空项目，没有设计文件，这时需要加入设计文件，可以是原理图，也可以是 VHDL 文件。选择主菜单[Project]的[Add Source File(s)]功能，出现如图对话框，在窗口中选择刚才保存的 EX1.VHD 文件。



5. 对项目中的设计进行综合。在流程图的窗口内，选择综合功能的图标，软件对设计进行综合。如果在进行综合之前，没有设置过项目的属性，这时软件会自动弹出综合/编译属性设置对话框，让用户对属性进行设置，见下图。设置时主要是选择 FPGA/EPLD 芯片的类型和速度，“Target Device”框内的“Family”栏内选择芯片的系列，这里选“VIRTEX”系列，“Device”栏内选择具体的芯片型号，为“V200PQ240”，“Speed”栏内选择芯片速度，我们选“-4”，按“Run”按钮启动综合/编译程序。用主菜单[Project]的[Create Version]功能也可以弹出综合/编译属性设置对话框，让用户设置项目属性。如果综合过程中发现错误，程序会停止，并提示错误，也可以用文本编辑窗口的菜单[Synthesis]的[View Report]功能查看详细错误信息。改正设计中的错误后，重新进行综合，直到完成。如果综合前没有进行语法检查，程序会自动先做检查，然后再综合。

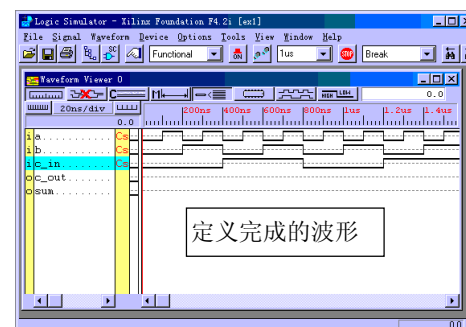
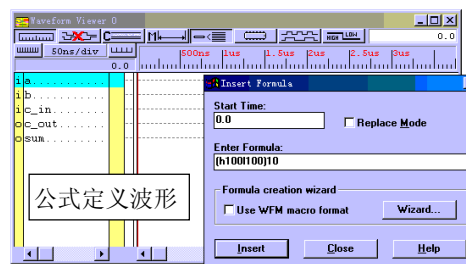


6. 逻辑功能模拟。综合完成后，就可以对用软件对设计进行逻辑功能的模拟。用鼠标选择流程图窗口内的逻辑功能模拟图标，打开软件逻辑模拟窗口，用户将想要观察的信号和需要驱动的信号添加到此窗口内，定义驱动波形，然后启动软件模拟逻辑功能，在窗口内就可以看到各信号的状态，此时的波形是在理想状态，信号无延时情况下的逻辑运行结果。



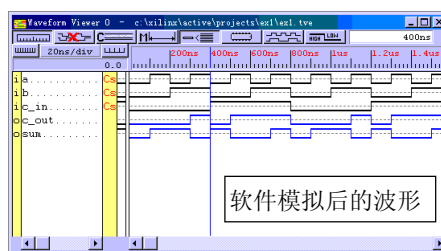
7. 在窗口内添加信号。新打开的波形窗口没有信号，用户需要将要观察的信号和驱动信号加入窗口内，选择软件模拟窗口的菜单[Signal]的[Add Signals]功能，系统弹出添加信号窗口，在窗口的“Signals Selection”框内双击想要观察信号名和驱动信号名，该会自动添加到观察窗口内，或选择好信号名后，按“Add”钮，也可将信号加入观察窗口。窗口内有很多中间信号，我们选择“a”、“b”、“c_in”这三个输入的驱动信号和“c_out”、“sum”这两个是想观察的输出信号，完成信号添加后，按“Close”钮关闭添加信号窗口。

8. 定义驱动信号波形。信号加入观察窗口后，我们需要对驱动信号定义，来模拟外部的输入时序。在本例中，全加器的三个输入端共有 8 种组合状态，我们将三路输入信号定义成不同频率的时钟，就可组合成 8 个状态。首先，用鼠标点击信号名选中信号“a”，在该信号右边的波形区内（红线右边）按鼠标右键，选择弹出菜单的[Insert Formula]功能，弹出公式输入波形的对话框，“Start Time”表示定义的起始时间，这里填入‘0.0’；“Replace Mode”表示是覆盖原有信号还是在该处插入定义的信号，这里可以不选中；“Enter Formula”栏是让用户输入描述波形的公式，我们要将“a”信号定义成 100ns 高 100ns 低，这样重复 10 次，公式为‘(H100L100)10’，具体公式如何定义，可按“Help”钮寻求帮助。公式写好后，按“Insert”钮确认。定义好“a”信号后，不需要退出公式对话框，直接在波形窗口的信号名选择“b”信号，在公式输出栏中填入公式‘(H200L200)4’，表



示“b”信号定义为 200ns 高 200ns 低重复 4 次。如此我们再将“c_in”定义成 400ns 高 400ns 低重复 2 次的信号，公式为 ‘(H400L400) 2’。全部输入信号定义完成后的波形窗口如图。

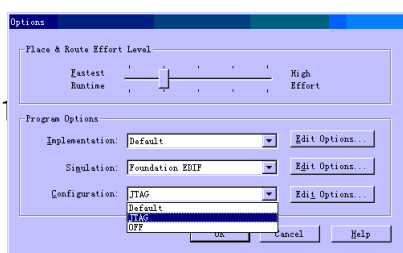
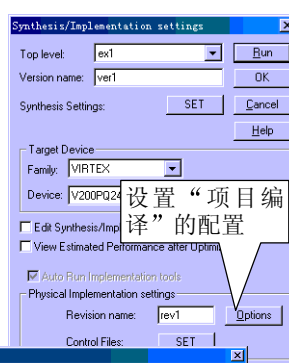
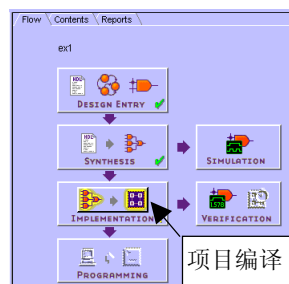
9. 运行软件逻辑仿真。当所有输入的波形定义好了之后，就可以进行软件逻辑功能模拟仿真了，按下逻辑功能模拟窗口工具栏里的“单步模拟”图标，软件模拟器每次仿真一步，每次所走的时间可由用户设定。模拟器每走一步，输出信号的波形就会输出一步的信号，若仿真过程中发现输出信号不是设计要求的，可以重新定义输入信号的波形，按下逻辑模拟窗口工具栏里的“加电”图标，软件模拟器复位，再按“单步模拟”，重新开始逻辑功能模拟。软件模拟后的波形如图。在波形窗口中，点击鼠标左键，出现蓝色标尺，用此标尺可比较信号的延时情况，因为是逻辑功能的模拟，为理想状态，从图上可以看出输出信号与输入信号之间没有延时。



10. 保存仿真波形。仿真完成后的波形可以保存下来，定义的输入波形在下次仿真时能调出使用。选择仿真窗口主菜单[File]的[Save WaveForm]功能，出现如图对话框，填上文件名，波形文件后缀为“*.tve”，按“确定”保存文件。



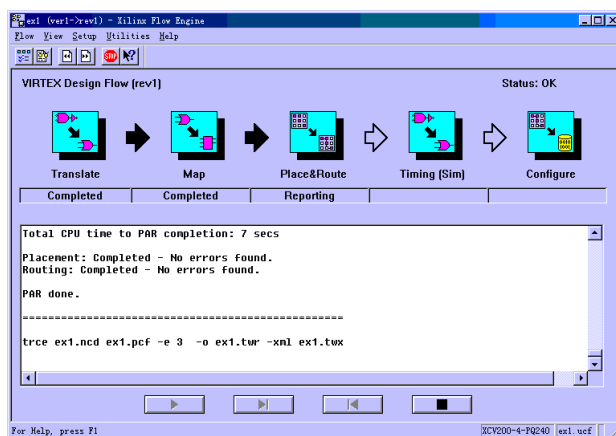
11. 项目编译。在逻辑功能仿真完成后，就要将项目中的设计电路编译生成目标文件，此目标文件下载到具体的芯片中，来实现项目中设计的逻辑功能。用鼠标点击流程图窗口中的编译图标启动编译程序。如果是项目第一次编译，软件会弹出综合/编译设置对话框，对编译属性进行设置，具体的设置方法和技巧可参考软件的帮助，按对话框中“Options”钮，在弹出的“配置选项”对话框中选择“Configuration”下拉框的“JTAG”，要求产生“*.BIT”格式文件，再按下“Configuration”右边的“Edit Options”钮，对 JTAG 文件进行配置，在弹出的“VIRTEX 配置选项”对话框的最下方，把“Enable .bit File Compression”选中，打上勾，这样编译时会产生压缩的*.bit 文件，以减少编程时间。按“确定”键确认退出此对话框，再按“OK”确认退到“综合/编译设置”对话框，按“Run”开始编译即可。编译时，软件显示编译过程的窗口，一步一步显示编译的过程，如果有错，程序会给出提示，用户可观察出错报告找出错误所在，解决错误后再次综合/编译。



配置选项对话框

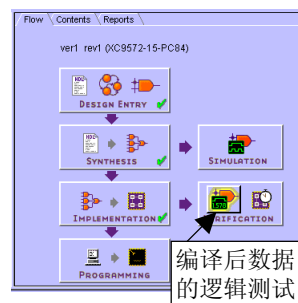


VIRTEX 配置选项对话框

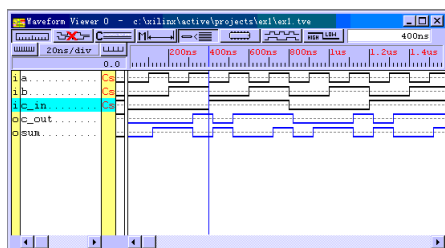


编译流程

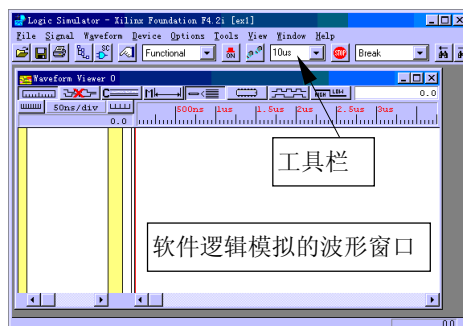
12. 逻辑功能验证。当项目中设计电路按具体芯片编译后，要对产生的数据是否正确进行验证，前面的逻辑功能仿真是正对所设计的逻辑功能进行仿真，现在的验证是对编译产生的数据进行测试，因为编译产生的数据是正对某个具体的芯片内部的逻辑电路，与理想的逻辑电路有所不同。用鼠标点击流程图窗口中的逻辑验证图标，软件打开软件逻辑功能仿真窗口，波形窗口中为空白，用仿真窗口菜单[File]的[Load WaveForm]功能打开前面保存的波形，这里主要是用前面定义的“a”、“b”、“c_in”三个输入信号，保存输出信号“c_out”、“sum”可以不考虑，在仿真窗口的工具栏中，选择仿真功能为“Timing”即时序仿真功能，按工具栏中的“加电”图标复位电路，按工具栏中的“单步模拟”单步时序仿真，仿真完成后，在波形窗口内，输入信号有变化的地方点击鼠标左键，可以看到在输入信号有变化时，输出信号没有立即变化，而是有一些延时。



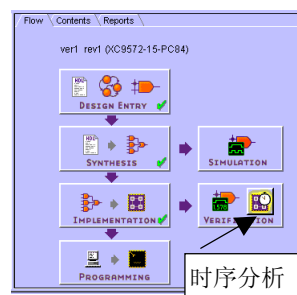
编译后数据的逻辑测试



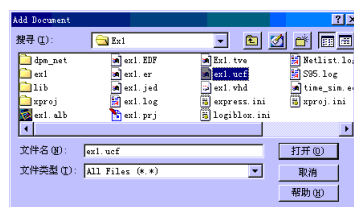
时序仿真后的波形



13. 时序分析。时序分析程序可以精确地计算出输入信号到输出信号之间的延时，让用户了解所设计的电路最大的工作频率。用鼠标点击流程图窗口中的时序分析图标，软件打开时序分析窗口，在时序窗口中，按照项目所选择的芯片型号，可以看到从输入信号经过几级中间信号再到输出信号，信号之间的延时，及总的延时。



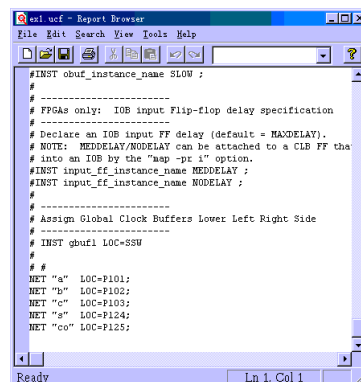
14. 将信号锁定到芯片的管脚。在前面的仿真过程中，信号都是随意分配到芯片的管脚上，在将编译后目标文件下载到芯片上之前，要将设计中的输入、输出信号锁定到芯片的管脚，在 Xilinx 的 Foundation 开发环境中，信号到管脚的锁定是用文件描述的。选择开发环境的主菜单 [Project] 的 [Add Source Files] 功能，出现如图对话框，“文件类型”设为 ‘All Files (*.*)’，在当前目录下选中 “ex1.ucf” 文件加入到项目中，此文件就是管脚锁定文件，为文本文件，可以编辑输入管脚的描述。在项目管理窗口中双击此文件名打开文件。在文件的最后加上以下信号到芯片管脚锁定的描述语句：



加入管脚锁定文件

```
NET "a"      LOC = P101;  
NET "b"      LOC = P102;  
NET "c_in"   LOC = P103;  
NET "sum"    LOC = P124;  
NET "c_out"  LOC = P125;
```

关于信号与管脚锁定描述的方法可参考文件内说明和开发环境的帮助。参考 FPGA 扩展板和软件的“结构图”窗口，我们用扩展板上开关组 K0 的第 0 个键做为“c_in”输入，第 1 个键做为“b”输入，第 2 个键做为“a”输入，用发光二极管 B0 显示“c_out”，B1 显示“sum”，按照“结构图”的定义，将信号 a 锁定到芯片的 101 号脚上，将信号 b 锁定到芯片 102 号脚上，将信号 c_in 锁定到芯片的 103 号管脚，sum 信号锁定到芯片的 124 号管脚，c_out 信号锁定到芯片 125 号管脚上。

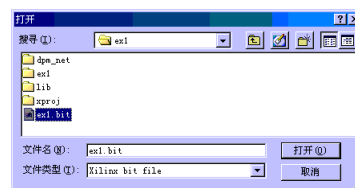


15. 重新综合/编译。在 Xilinx 的 Foundation 开发环境中，加入或修改了信号与管脚锁定的描述后，要全部重新综合/编译。选择主菜单 [Project] 的 [Clear Implementation Data] 功能，将综合/编译产生的数据全部清除，然后重新综合编译项目，产生新的包含管脚信号的数据文件，才可以编程下载到芯片上。注意：在 Xilinx 开发环境中，每次修改了信号与

管脚锁定描述后，都要清除旧的综合/编译数据，重新编译产生新数据，否则可能会导致错误。注意，清除综合/编译数据时，也会清除项目中一些编译配置数据，所以在重新编译这前，要按第 5 步和第 11 步重新设置综合/编译配置。

16. 编程下载。当用软件仿真验证设计的电路工作正常。就可以将编译产生的位图文件编程下载到芯片，用芯片来工作了。在使用 FPGA 扩展板时，我们不用 Xilinx 的开发环境来编程 FPGA 芯片，这里用 COP2000 实验仪的软件来下载 FPGA 程序，在 FPGA 扩展窗口的“结构图”页面上选择

“FPGA 编程”功能，在“打开”对话框中，选择刚才生成的“EX1.BIT”文件所在路径，选中该文件，按“打开”钮，程序就会对 FPGA 进行编程，并显示编程进度。在选择“FPGA 编程”功能时，若 COP2000 实验仪与计算机连接不正确，程序会显示“未连接串口”，若 FPGA 扩展板没有接到实验仪上或扩展板有错，程序会显示“FPGA 出错”。

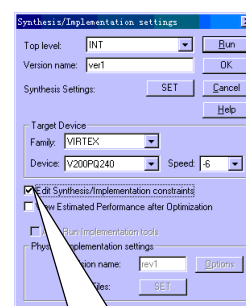


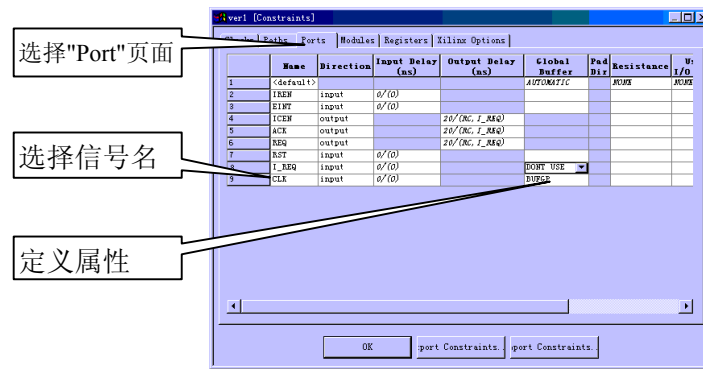
17. 设置/选择实验仪的工作模式。当程序下载到芯片上后就可以用实验仪进行实验来验证我们所做设计是否正确，在进行实验之前，最好能对扩展板的模式进行设置，也就重新定义用于的键盘、八段数码管、发光二极管的名称，方便观察信号。定义好的名称，可以用软件上的“保存模式”保存到盘上，下次再做同样的实验时，用“打开模式”打开定义的文件即可。

18. 在实验仪验证设计。设计电路已经下载到实验仪的适配板上，实验仪的工作模式也选择好，下面就可对设计进行硬件上验证。分别拨动开关组 K0 的第 0 键、第 1 键、第 2 键，改变其状态，表示三个输入信号“a”“b”“c_in”的状态的改变，观察发光管 B1 和 B0，显示是否按照设计的逻辑做相应变化。

至此，用户已经一步一步地学会了在 Xilinx 的 Foundation 开发环境中，从最初的新建项目直到最后用硬件来实现设计思想的各个主要步骤，为了易于学习，中间有一些环节没有介绍，这需要用户在以后的学习和开发过程中逐步了解，逐步提高。在开发过程中，也可以参考 EDA 开发环境的说明和软件中的帮助。

注意：在 Xilinx 的 Foundation 开发环境中，如果用户在设计中使用了时钟信号、复位信号、输出允许信号等可以全局使用的信号，在综合/编译时，编译器会自动处理，如果不是按照设计的要求进行处理，就有可能会出现错误，这时就要求你自己对这些信号的属性进行定义，自行定义属性的方法是：在进行综合/编译前，进行综合/编译属性设置时，以弹出的对话框，选中“Edit Synthesis/ Implementation constraints”前的选中勾，这样在随后的综合/编译过程中，会弹出如下设置框，选择“Port”页面，选择要设定属性的信号名，在“Global Buffer”栏中，做相应的设置，选择“DONT USE”使不做为全局信号。选择“BUFGP”表示要做为全局信号。按“OK”钮确认退出。





第十二章 十六位机扩展实验

分部实验一、十六位 ALU 实验

实验要求：用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，用 VHDL 语言编写程序，下载到 XCV200FPGA 中，实现十六位模型机的 ALU 功能。

实验目的：了解十六位模型机中算术、逻辑运算单元（ALU）的工作原理和实现方法。学习用 VHDL 语言描述硬件逻辑。学习使用 EDA 开发环境。

实验说明：下为十六位 ALU 的 VHDL 语言：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ALU IS
PORT (
  A      : IN    STD_LOGIC_VECTOR(15 DOWNTO 0); -- K3, K2
  W      : IN    STD_LOGIC_VECTOR(15 DOWNTO 0); -- K1, K0
  S0     : IN    STD_LOGIC;                      -- K4.0
  S1     : IN    STD_LOGIC;                      -- K4.1
  S2     : IN    STD_LOGIC;                      -- K4.2
  D      : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0); -- D1, D0
  CIn    : IN    STD_LOGIC;                      -- K4.3
  COut   : OUT   STD_LOGIC                       -- A0
);
END ALU;

ARCHITECTURE behv OF ALU IS
  SIGNAL T: STD_LOGIC_VECTOR(16 DOWNTO 0);
BEGIN
  T <= ('0' & A) + ('0' & W)          WHEN S2 = '0' AND S1 = '0' AND S0 = '0' ELSE
        ('0' & A) - ('0' & W)          WHEN S2 = '0' AND S1 = '0' AND S0 = '1' ELSE
        ('0' & A) OR ('0' & W)         WHEN S2 = '0' AND S1 = '1' AND S0 = '0' ELSE
        ('0' & A) AND ('0' & W)        WHEN S2 = '0' AND S1 = '1' AND S0 = '1' ELSE
        ('0' & A) + ('0' & W) + CIn    WHEN S2 = '1' AND S1 = '0' AND S0 = '0' ELSE
        ('0' & A) - ('0' & W) - CIn    WHEN S2 = '1' AND S1 = '0' AND S0 = '1' ELSE
        NOT ('0' & A)                  WHEN S2 = '1' AND S1 = '1' AND S0 = '0' ELSE
        ('0' & A);

  D  <= T(15 DOWNTO 0);
  COut <= T(16);
END behv;
```

在上面程序中，A 为累加器，W 为工作寄存器，均为十六位寄存器，W 的值由 K1、K0 两组开关共十六位输入，A 的值由 K2、K3 两组开关输入，S0、S1、S2 为运算控制位，接在 K4 开关组的第 0、1、2 个开关上，根据 S0、S1、S2 的不同，ALU 实现的功能参见下表，D 为运算结果输出，显示在四位八段管 D0、D1 上，CIn 为进位输入，由 K4 开关组的第 3 个开关

输入，C0ut 为进位输出，用发光二极管 A0 显示其状态。

S2	S1	S0	ALU 实现的功能
0	0	0	$D = A + W$ 运算结果为 A 加 W
0	0	1	$D = A - W$ 运算结果为 A 减 W
0	1	0	$D = A W$ 运算结果为 A 逻辑或 W
0	1	1	$D = A \& W$ 运算结果为 A 逻辑与 W
1	0	0	$D = A + W + C_{in}$ 运算结果为 A 加 W 加进位
1	0	1	$D = A - W - C_{in}$ 运算结果为 A 减 W 减进位
1	1	0	$D = \sim A$ 运算结果为 A 取反
1	1	1	$D = A$ 运算结果为 A (A 直接输出)

实验步骤：

1. 打开 Xilinx 的 EDA 开发环境，选择“Open Project ...”打开“\COP2000\XCV200\”下的 ALU 项目（Xilinx 的 EDA 开发环境的使用可参见第十一章）。
2. 充分理解 ALU.VHD。了解模型机中 ALU 的实现原理。
3. 对 ALU 项目进行综合/编译，生成 ALU.BIT 文件。
4. 打开伟福的 COP2000 开发环境，打开 FPGA 扩展板的界面。
5. 按“打开模式”键，打开“\COP2000\XCV200\ALU\”目录下的 ALU.MOD 文件。
6. 按“通信设置”键，将实验仪连接到计算机上。
7. 按“FPGA 编程”键，将“\COP2000\XCV200\ALU\”目录下的 ALU.BIT 文件下载到 XCV200 芯片上，
8. 拨动 K0、K1 输入 W 的值，拨动 K2、K3 输入 A 的值，拨动 K4 的 0、1、2 位，设置运算方式，拨动 K4 的第 3 位，设置进位，观察八段管 D1、D0 的运算结果，观察发光二极管 A0 是否有进位输出。

本实验可参考 COP2000 实验仪使用说明的分部实验的第 14 页“运算器实验”和综合实验的第 48 页“实验 2 数据运算实验”

分部实验二、十六位寄存器实验

实验要求：用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，用 VHDL 语言编写程序，下载到 XCV200FPGA 中，实现十六位模型机的寄存器输入输出功能。

实验目的：了解十六位模型机中寄存器的工作原理和实现方法。学习用 VHDL 语言描述硬件逻辑。学习使用 EDA 开发环境。

实验说明：下为十六位寄存器的 VHDL 语言：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY REG IS
PORT (
    D : IN  STD_LOGIC_VECTOR(15 DOWNTO 0); -- 开关组 K1, K0
    R : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -- 八段管 D1, D0
    EN : IN  STD_LOGIC;                    -- 开关 K4.0
    RST: IN  STD_LOGIC;                    -- 开关 K4.7
    CLK: IN  STD_LOGIC                     -- CLOCK
);
END REG;

ARCHITECTURE behv OF REG IS

BEGIN
    PROCESS (CLK, RST, EN)
    BEGIN
        IF RST = '1' THEN
            R <= (OTHERS=>'0');
        ELSIF CLK' EVENT AND CLK = '1' THEN
            IF (EN = '0') THEN
                R <= D;
            END IF;
        END IF;
    END PROCESS;
END behv;
```

在上面程序中，D 为输入数据，R 为寄存器，均为十六位寄存器，D 的值由 K1、K0 两组开关共十六位输入，R 为内部寄存器，用八段管 D1、D0 显示，EN 为寄存器选通信号，接在 K4 开关组的第 0 个开关上，RST 为复位信号，接在开关组 K4 的第 7 个开关上，CLK 为时钟脉冲，由 COP2000 实验仪主机上的“CLOCK”按键提供。

实验步骤：

1. 打开 Xilinx 的 EDA 开发环境，选择“Open Project...”打开“\COP2000\XCV200\”下的 REG 项目（Xilinx 的 EDA 开发环境的使用可参见第十一章）。
2. 充分理解 REG.VHD。了解寄存器的实现原理。
3. 对 REG 项目进行综合/编译，生成 REG.BIT 文件。
4. 打开伟福的 COP2000 开发环境，打开 FPGA 扩展板的界面。
5. 按“打开模式”键，打开“\COP2000\XCV200\REG\”目录下的 REG.MOD 文件。界面上“时钟选择”应设成“单脉冲”，这样 CLK 信号就由 COP2000 实验仪主板上的“CLOCK”按键提供。

6. 按“通信设置”键，将实验仪连接到计算机串行口上。
7. 按“FPGA 编程”键，将“\COP2000\XCV200\REG\”目录下的 REG.BIT 文件下载到 XCV200 芯片上，
8. 拨动 K4 的第 7 位到“1”的位置，输出“复位”信号，观察八段管 D1、D0（R 寄存器的输出显示）是否清零，再将 K4 的 7 位回到“0”位，拨动 K0、K1 输入 D 的值，拨动 K4 的 0 位，设置寄存器选通信号“EN”为有效状态（“0”有效），按动 COP2000 实验仪上的“CLOCK”按键，产生一个时钟信号，观察八段管 D1、D0，看看是否将 D 的值存入寄存器 R 中并显示出来，拨动开关组 K1、K0，改变 D 的值，再将“EN”置于无效状态（“1”位置），按“CLOCK”产生时钟信号，观察八段管 D1、D0 是否会随着改变。

本实验可参考 COP2000 实验仪使用说明的分部实验的第 8 页“寄存器实验”。

分部实验三、十六位寄存器组实验

实验要求：用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，用 VHDL 语言编写程序，下载到 XCV200FPGA 中，实现十六位模型机的多个寄存器输入输出功能。

实验目的：了解十六位模型机中寄存器组的工作原理和实现方法。学习用 VHDL 语言描述硬件逻辑。学习使用 EDA 开发环境。

实验说明：下为十六位模型机中四个寄存器的 VHDL 语言：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY REGS IS
PORT (
    D : IN  STD_LOGIC_VECTOR(15 DOWNTO 0); -- 开关组 K1, K0
    R : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -- 八段管 D1, D0
    SA : IN  STD_LOGIC; -- 开关 K4.0
    SB : IN  STD_LOGIC; -- 开关 K4.1
    RD : IN  STD_LOGIC; -- 开关 K4.2
    WR : IN  STD_LOGIC; -- 开关 K4.3
    RST: IN  STD_LOGIC; -- 开关 K4.7
    CLK: IN  STD_LOGIC -- CLOCK
);
END REGS;

ARCHITECTURE behv OF REGS IS

SIGNAL R0, R1, R2, R3: STD_LOGIC_VECTOR(15 DOWNTO 0);

BEGIN

-- R0
PROCESS(CLK, RST, WR, SA, SB)
BEGIN
    IF RST = '1' THEN
        R0 <= (OTHERS=>'0');
    ELSIF CLK'EVENT AND CLK = '1' THEN
        IF (WR = '0') AND (SB = '0') AND (SA = '0') THEN
            R0 <= D;
        END IF;
    END IF;
END PROCESS;

-- R1
PROCESS(CLK, RST, WR, SA, SB)
BEGIN
    IF RST = '1' THEN
        R1 <= (OTHERS=>'0');
    ELSIF CLK'EVENT AND CLK = '1' THEN
        IF (WR = '0') AND (SB = '0') AND (SA = '1') THEN
            R1 <= D;
        END IF;
    END IF;
END PROCESS;

-- R2
PROCESS(CLK, RST, WR, SA, SB)
BEGIN
```

```

IF RST = '1' THEN
  R2 <= (OTHERS=>'0');
ELSIF CLK' EVENT AND CLK = '1' THEN
  IF (WR = '0') AND (SB = '1') AND (SA = '0') THEN
    R2 <= D;
  END IF;
END IF;
END PROCESS;

-- R3
PROCESS (CLK, RST, WR, SA, SB)
BEGIN
  IF RST = '1' THEN
    R3 <= (OTHERS=>'0');
  ELSIF CLK' EVENT AND CLK = '1' THEN
    IF (WR = '0') AND (SB = '1') AND (SA = '1') THEN
      R3 <= D;
    END IF;
  END IF;
END PROCESS;

R <= R0 WHEN (SB = '0') AND (SA = '0') AND (RD = '0') ELSE
  R1 WHEN (SB = '0') AND (SA = '1') AND (RD = '0') ELSE
  R2 WHEN (SB = '1') AND (SA = '0') AND (RD = '0') ELSE
  R3 WHEN (SB = '1') AND (SA = '1') AND (RD = '0') ELSE
  (OTHERS=>'0');

END behv;

```

在上面程序中，D 为输入数据，为十六位寄存器，D 的值由 K1、K0 两组开关共十六位输入，R0..R3 为内部寄存器，R 用做内部寄存器显示输出，用八段管 D1、D0 显示其值，SA、SB 为寄存器选择控制信号，接在 K4 开关组的第 0、1 个开关上，RD 为寄存器读信号，接 K4 的第 2 个开关上，WR 为寄存器写信号，接在 K4 的第 3 个开关上，RST 为复位信号，接在开关组 K4 的第 7 个开关上，CLK 为时钟脉冲，由 COP2000 实验仪主机上的“CLOCK”按键提供。

实验步骤：

1. 打开 Xilinx 的 EDA 开发环境，选择“Open Project...”打开“\COP2000\XCV200\”下的 REGS 项目（Xilinx 的 EDA 开发环境的使用可参见第十一章）。
2. 充分理解 REGS.VHD。了解模型中寄存器组的实现原理。
3. 对 REGS 项目进行综合/编译，生成 REGS.BIT 文件。
4. 打开伟福的 COP2000 开发环境，打开 FPGA 扩展板的界面。
5. 按“打开模式”键，打开“\COP2000\XCV200\REGS\”目录下的 REGS.MOD 文件。界面上“时钟选择”应设成“单脉冲”，这样 CLK 信号就由 COP2000 实验仪主板上的“CLOCK”提供。
6. 按“通信设置”键，将实验仪连接到计算机串行口上。
7. 按“FPGA 编程”键，将“\COP2000\XCV200\REGS\”目录下的 REGS.BIT 文件下载到 XCV200 芯片上。
8. 拨动 K4 的第 7 位到‘1’的位置，输出“复位”信号，观察八段管 D1、D0（R 寄存器的输出显示）是否清零，然后将 K4 的第 2（RD 信号）、3（WR 信号）位置成‘1’

状态，使读写信号都处于无效状态。

9. 寄存器组写实验：将 K4 的 7 位回到时“0”位，拨动 K0、K1 输入 D 的值，拨动 K4 的 0、1 位设成‘00’，选择寄存器 R0，拨动 K4 的 3 位，设置寄存器写信号“WR”为有效状态（“0”有效），按动 COP2000 实验仪上的“CLOCK”按钮，产生一个时钟信号，将 D 写入寄存器 R0 中。拨动 K0、K1 开关组，改变 D 值，再改变 K4 的第 0、1 位设成‘01’，选择寄存器 R1，按“CLOCK”给出时钟信号，将 D 存入寄存器 R1 中。如此将不同的十六位数据分别存 R0..R3 寄存器中。
10. 寄存器组读实验：将 K4 的第 3 位（WR 信号）设成‘1’使其无效，将 K4 的 0、1 两位拨成‘00’，选择寄存器 R0，再将 K4 的第 2 位（RD 信号）设成‘0’，读出 R0 中的数据并输出到八段管 D1、D0 上显示。拨动 K4 的第 2 位，使 RD 信号无效，改变 K4 的 0、1 两位选择寄存器 R1，再拨动 K4 的第 2 位使 RD 信号有效，读出 R1 的值并显示在八段管 D1、D0 上。如此，读出寄存器 R2、R3 的值，并观察与写入的数据是否相同。

本实验可参考 COP2000 实验仪使用说明的分部实验的第 8 页“寄存器实验”。

分部实验四、十六位指令计数器 PC 实验

实验要求： 用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，用 VHDL 语言编写程序，下载到 XCV200FPGA 中，实现十六位模型机的指令计数器功能。

实验目的： 了解十六位模型机中指令计数器（PC）的工作原理和实现方法。学习用 VHDL 语言描述硬件逻辑。学习使用 EDA 开发环境。

实验说明： 下为十六位模型机中 PC 的 VHDL 语言：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY pc IS
PORT (
    D : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- K1, K0
    PC : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -- D1, D0
    C : IN STD_LOGIC; -- K4.0
    Z : IN STD_LOGIC; -- K4.1
    ELP : IN STD_LOGIC; -- K4.2
    PC1 : IN STD_LOGIC; -- K4.3
    IR2 : IN STD_LOGIC; -- K4.4
    IR3 : IN STD_LOGIC; -- K4.5
    RST : IN STD_LOGIC; -- K4.7
    CLK : IN STD_LOGIC -- CLOCK
);
END PC;

ARCHITECTURE behv OF PC IS
SIGNAL RPC: STD_LOGIC_VECTOR(15 DOWNTO 0);

BEGIN

    PC <= RPC;

    PROCESS(CLK, RST, IR3, IR2, C, Z, ELP)
    VARIABLE LDPC: STD_LOGIC; -- 转移控制
    BEGIN
        IF RST = '1' THEN
            RPC <= (OTHERS=>'0');
        ELSIF CLK'EVENT AND CLK = '1' THEN
            IF (ELP = '0') AND (
                ( (IR3 = '0') AND (IR2 = '0') AND (C = '1') ) OR
                ( (IR3 = '0') AND (IR2 = '1') AND (Z = '1') ) OR
                ( IR3 = '1' )
            ) THEN LDPC := '0'; ELSE LDPC := '1';
            END IF;

            IF LDPC = '0' THEN
                RPC <= D;
            ELSIF (PC1 = '1') THEN
                RPC <= RPC + 1;
            END IF;
        END IF;
    END PROCESS;

END behv;
```

在上面程序中，D 为十六位输入数据，用于表示跳转条件满足时，跳转的目标地址。D 的值由 K1、K0 两组开关输入。PC 为指令计数器（PC），用八段数码管 D1、D0 显示其值。C、Z 接在 K4 开关组的第 0、1 个开关上，用于模拟模型机中的进位标志和零标志信号，ELP 接 K4 的第 2 个开关上，为程序跳转控制信号，为‘1’时不允许预置 PC，为‘0’时，根据指令码的第 3、2 位和 C、Z 状态来控制程序是否跳转（见下表说明）。PC1 接在 K4 的第 3 个开关上，表示 PC 加 1 控制信号。IR2、IR3 接在 K4 的第 4、5 个开关上，表示程序指令的第 2 位和第 3 位，在本模型机实验中，这两用于控制程序的跳转（见下表说明）。RST 为复位信号，接在开关组 K4 的第 7 个开关上，CLK 为时钟脉冲，由 COP2000 实验仪主机上的“CLOCK”按键提供。

ELP	IR3	IR2	C	Z	LDPC
1	x	x	x	x	1
0	0	0	1	x	0
0	0	0	0	x	1
0	0	1	x	1	0
0	0	1	x	0	1
0	1	x	x	x	0

上表中，LDPC 为内部信号，用于控制 PC 是否能被预置。

当 ELP = 1 时，LDPC = 1，不允许 PC 被预置。

当 ELP = 0 时，LDPC 由 IR3、IR2、C、Z 确定

当 IR3、IR2 = 1X 时，LDPC = 0，D 的值在 CLK 上升沿打入 PC，实现程序的 JMP（直接跳转）功能。

当 IR3、IR2 = 00 时，LDPC = C 取反，当 C = 1 时，D 的值在 CLK 上升沿打入 PC，实现程序的 JC（有进位跳转）功能

当 IR3、IR2 = 01 时，LDPC = Z 取反，当 Z = 1 时，D 的值在 CLK 上升沿打入 PC，实现程序的 JZ（累加器为零跳转）功能。

本实验中，RST = 1 时，指令计数器 PC 被清 0，当 LDPC = 0 时，在 CLK 上升沿 D 的值打入 PC，当 PC1 = 1 时，在 CLK 上升沿 PC 加 1。

实验步骤：

1. 打开 Xilinx 的 EDA 开发环境，选择“Open Project ...”打开“\COP2000\XCV200\”下的 PC 项目（Xilinx 的 EDA 开发环境的使用可参见第十一章）。
2. 充分理解 PC.VHD，了解模型机中指令计数器的实现原理。
3. 对 PC 项目进行综合/编译，生成 PC.BIT 文件。
4. 打开伟福的 COP2000 开发环境，打开 FPGA 扩展板的界面。
5. 按“打开模式”键，打开“\COP2000\XCV200\PC\”目录下的 PC.MOD 文件。界面上“时钟选择”应设成“单脉冲”，这样 CLK 信号就由 COP2000 实验仪主板上的“CLOCK”提供。
6. 按“通信设置”键，将实验仪连接到计算机串行口上。
7. 按“FPGA 编程”键，将“\COP2000\XCV200\PC\”目录下的 PC.BIT 文件下载到 XCV200 芯片上，

8. 拨动 K4 的第 7 位到 ‘1’ 的位置，输出 “复位” 信号，使电路处于复位状态。观察八段管 D1、D0（PC 输出显示）是否清零，
9. PC+1 实验：将 K4 的第 2（ELP 信号）、3（PC1 信号）位置成 ‘1’ 状态，使跳转控制信号处于 PC+1 状态。再将 K4 的 7 位回到时 “0” 位，退出 “复位” 状态，按动 COP2000 实验仪上的 “CLOCK” 按钮，产生一个时钟信号，观察八段管 D1、D0 的显示，看看 PC 是否加 1，再给出单脉冲，观察 PC 是否再次加 1，
10. 直接跳转实验：使拨动 K0、K1 输入 D 的值，设置跳转的目标地址，拨动 K4 的第 2 位设成 ‘0’，使 ELP 信号为低，拨动 K4 的 5 位，将其设成 ‘1’，使 IR3 为高，将跳转控制设成 “直接跳转” 方式，按动 COP2000 实验仪上的 “CLOCK” 按钮，产生一个时钟信号，将 D 写入 PC 中，观察八段管 D1、D0，看看 PC 是否转到目标地址。
11. 条件跳转实验 1：拨动 K4 的第 4、5 位，使 IR2、IR3 置成 ‘00’，将跳转控制设置成 “判进位跳转” 方式。将 K4 的 0 位设置成 ‘1’，表示有进位，按动 COP2000 实验仪上的 “CLOCK” 按钮，产生一个时钟信号，观察八段管 D1、D0，看看 PC 是否转到 D 所指定的目标地址。再将 K4 的 0 拨成 ‘0’，表示无进位，给出一个单脉冲，观察 D1、D0，看看 PC 是否加 1。
12. 条件跳转实验 2：拨动 K4 的第 5、4 位，使 IR3、IR2 置成 ‘01’，将跳转控制设置成 “判零跳转” 方式。将 K4 的 1 位设置成 ‘1’，表示累加器为零，按动 COP2000 实验仪上的 “CLOCK” 按钮，产生一个时钟信号，观察八段管 D1、D0，看看 PC 是否转到 D 所指定的目标地址。再将 K4 的 1 拨成 ‘0’，表示累加器不为零，给出一个单脉冲，观察 D1、D0，看看 PC 是否加 1。

本实验可参考 COP2000 实验仪使用说明的分部实验的第 22 页 “PC 实验” 和综合实验的第 51 页 “实验 4 转移实验”

分部实验五、中断控制实验

实验要求： 用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，用 VHDL 语言编写程序，下载到 XCV200FPGA 中，实现十六位模型机中断控制功能。

实验目的： 了解十六位模型机中中断控制的工作原理和实现方法。学习用 VHDL 语言描述硬件逻辑。学习使用 EDA 开发环境。

实验说明： 下为十六位模型机串实现中断控制的 VHDL 语言：

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY INT IS
PORT (
    IREN : IN  STD_LOGIC; -- K4.0
    EINT : IN  STD_LOGIC; -- K4.1
    ICEN : OUT STD_LOGIC; -- A2
    ACK  : OUT STD_LOGIC; -- A1
    REQ  : OUT STD_LOGIC; -- A0
    RST  : IN  STD_LOGIC; -- K4.7
    I_REQ: IN  STD_LOGIC; -- INT
    CLK  : IN  STD_LOGIC; -- CLOCK
);
END INT;

ARCHITECTURE behv OF INT IS

    SIGNAL R_REQ, R_ACK: STD_LOGIC;
    SIGNAL FATCH_INT: STD_LOGIC;

BEGIN

    REQ <= R_REQ;
    ACK <= R_ACK;
    ICEN <= FATCH_INT;

    PROCESS(I_REQ, RST, EINT)
    BEGIN
        IF (EINT = '0') OR (RST = '1') THEN
            R_REQ <= '0';
        ELSIF I_REQ'EVENT AND I_REQ = '1' THEN
            R_REQ <= '1';
        END IF;
    END PROCESS;

    PROCESS(CLK, RST, EINT, IREN, R_REQ, R_ACK, FATCH_INT)
    BEGIN
        IF (IREN = '0') AND (R_REQ = '1') AND (R_ACK = '0') THEN
            FATCH_INT <= '0';
        ELSE
            FATCH_INT <= '1';
        END IF;

        IF (EINT = '0') OR (RST = '1') THEN
            R_ACK <= '0';
        ELSIF CLK'EVENT AND CLK = '1' THEN
            IF FATCH_INT = '0' THEN
                R_ACK <= R_REQ;
            END IF;
        END IF;
    END PROCESS;

```

```
END PROCESS;  
  
END behv;
```

在上面程序中，IREN 接 K4 的第 0 位，表示程序执行过程中的取指令操作，中断请求信号只有在此信号有效时（取指令时）才会被响应。IENT 接 K4 的第 1 位，用于在中断返回时，清除中断请求寄存器和中断响应寄存器。ICEN 为输出信号接发光二极管 A2，此信号用于控制读中断指令。ACK 接发光二极管 A1，显示中断响应信号。REQ 接发光二极管 A0，显示中断请求信号。RST 为复位信号，接在开关组 K4 的第 7 个开关上，I_REQ 为中断申请输入信号，由 COP2000 实验仪主板上的“INT”按键输出到 FPGA 扩展板上，CLK 为时钟脉冲，由 COP2000 实验仪主机上的“CLOCK”按键提供。

实验步骤：

1. 打开 Xilinx 的 EDA 开发环境，选择“Open Project ...”打开“\COP2000\XCV200\”下的 INT 项目（Xilinx 的 EDA 开发环境的使用可参见第十一章）。
2. 充分理解 INT.VHD。了解寄存器的实现原理。
3. 对 REG 项目进行综合/编译，生成 INT.BIT 文件。
4. 打开伟福的 COP2000 开发环境，打开 FPGA 扩展板的界面。
5. 按“打开模式”键，打开“\COP2000\XCV200\INT\”目录下的 INT.MOD 文件。界面上“时钟选择”应设成“单脉冲”，这样 CLK 信号就由 COP2000 实验仪主板上的“CLOCK”提供。
6. 按“通信设置”键，将实验仪连接到计算机串行口上。
7. 按“FPGA 编程”键，将“\COP2000\XCV200\INT\”目录下的 INT.BIT 文件下载到 XCV200 芯片上，
8. 拨动 K4 的第 7 位到“1”的位置，输出“复位”信号，将 K4 的第 0、1 位拨到‘1’的位置，使 IREN 和 EINT 都处于无效状态，将内部的中断请求寄存器，中断响应寄存器都清零，使其能响应中断。
9. 中断申请：将 K4 的第 7 位回到时“0”位，使电路正常工作。按下 COP2000 实验仪主板上的“INT”按键申请中断，FPGA 扩展板上的 A0 发光二极管变亮，表示有中断申请。
10. 中断响应：将 K4 的第 0 位拨成‘0’，也就使 IREN 有效，表示取指令操作，扩展板上的 A1 发光二极管变亮，表示已经响应中断。
11. 中断处理：按动 COP2000 实验仪上的“CLOCK”按钮，产生一个时钟信号，扩展板上 A2 发光二极管不亮，表示取指操作取出中断处理指令来执行。
12. 中断退出：将 K4 的第 0 位拨成‘1’，IREN 置成无效，将 K4 的第 1 位拨成‘0’，将 EINT 置成有效，发光二极管 A2 变亮，A1、A0 变暗，中断申请寄存器和中断响应寄存器清零，表示可以接受下一次中断申请。

本实验可参考 COP2000 实验仪使用说明的分部实验的第 32 页“中断实验”和综合实验的第 55 页“实验 6 中断实验”

实验六 十六位模型机的总体实验

实验要求： 用 COP2000 的扩展实验板上的开关做为输入、八段数码管做为输出，以上面介绍的分部实验为基础，用 VHDL 语言编写程序，下载到 XCV200FPGA 芯片中，实现一个十六位模型机，模型机要有完整的指令系统，中断处理系统、输入输出系统。

实验目的： 了解十六位模型机整机的工作原理和实现方法。用 VHDL 语言描述硬件逻辑，让 FPGA 能实现复杂的运算、处理功能。

实验说明： 伟福的 COP2000 软件已经提供了一个完整的十六位模型机的 VHDL 程序，学生也可以用 COP2000 软件来自动生成这个程序，方法是：在软件的主界面上按“生成组合逻辑控制器 VHDL，ABEL 程序”快捷按钮，程序会生成三个文件：“COP2000.ABL”为实验仪主板上组合逻辑控制器的 ABEL 程序，“EDA2000.VHD”为在 EDA 实验仪上做 8 位 CPU 实验的程序，“COP2000.VHD”为此十六位模型机的 VHDL 程序。下为十六位模型机完整的 VHDL 语言：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY COP2000 IS
  GENERIC (
    ALL_ZERO : STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000000000000"; -- 十六位机
    INT_ENTER: STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000011100000";
    INT_CODE : STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000010111000";
    DataWidth: integer := 16
  ) -- ALL_ZERO : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000"; -- 八位机
  -- INT_ENTER: STD_LOGIC_VECTOR(7 DOWNTO 0) := "11100000";
  -- INT_CODE : STD_LOGIC_VECTOR(7 DOWNTO 0) := "10111000";
  -- DataWidth: integer := 8
);

PORT (
  clk      : IN      STD_LOGIC;           --主时钟输入
  rst      : IN      STD_LOGIC;           --复位输入
  keyin    : IN      STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); --键输入
  portout  : OUT     STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); --端口输出

  mem_d    : INOUT   STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); --存储器数据线
  mem_a    : OUT     STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); --存储器地址线
  -- mem_ah : OUT     STD_LOGIC_VECTOR(15 DOWNTO 8);           --八位机时，高 8 位地址为 0

  mem_rd   : OUT     STD_LOGIC;           --存储器读信号
  mem_wr   : OUT     STD_LOGIC;           --存储器写信号

  mem_bh   : OUT     STD_LOGIC;           --存储器高 8 位选择信号
  mem_bl   : OUT     STD_LOGIC;           --存储器低 8 位选择信号
  mem_cs   : OUT     STD_LOGIC;           --存储器片选信号

  i_req    : IN      STD_LOGIC           --中断请求信号
);
END COP2000;

ARCHITECTURE behv OF COP2000 IS
```

```

-- 寄存器定义
SIGNAL A : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL W : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL R0 : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL R1 : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL R2 : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL R3 : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL PC : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL MAR : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL ST : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL IA : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);
SIGNAL IR : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0);

-- 标志定义
SIGNAL R_CY: STD_LOGIC; -- 进位标志寄存器
SIGNAL R_Z : STD_LOGIC; -- 零标志寄存器
SIGNAL CY : STD_LOGIC; -- 本次运算进位标志
SIGNAL Z : STD_LOGIC; -- 本次运算零标志

-- 中断定义
SIGNAL R_REQ: STD_LOGIC; -- 中断请求寄存器
SIGNAL R_ACK: STD_LOGIC; -- 中断响应寄存器
SIGNAL ICEN : STD_LOGIC; -- 取中断指令

-- 24 位微控制信号
SIGNAL S0 : STD_LOGIC;
SIGNAL S1 : STD_LOGIC;
SIGNAL S2 : STD_LOGIC; -- 运算器功能选择
SIGNAL AEN : STD_LOGIC; -- A 写允许
SIGNAL WEN : STD_LOGIC; -- W 写允许
SIGNAL X0 : STD_LOGIC;
SIGNAL X1 : STD_LOGIC;
SIGNAL X2 : STD_LOGIC; -- 寄存器输出控制

SIGNAL FEN : STD_LOGIC; -- 标志寄存器写允许
SIGNAL CN : STD_LOGIC; -- 移位时是否带进位
SIGNAL RWR : STD_LOGIC; -- 寄存器(R0..R3)写允许
SIGNAL RRD : STD_LOGIC; -- 寄存器(R0..R3)读允许
SIGNAL STEN : STD_LOGIC; -- ST 寄存器写允许
SIGNAL OUTEN: STD_LOGIC; -- OUT 寄存器写允许
SIGNAL MAROE: STD_LOGIC; -- MAR 寄存器地址输出允许
SIGNAL MAREN: STD_LOGIC; -- MAR 寄存器写允许

SIGNAL ELP : STD_LOGIC; -- PC 寄存器写允许
SIGNAL EINT : STD_LOGIC; -- 中断结束
SIGNAL IREN : STD_LOGIC; -- IR 寄存器写允许
SIGNAL EMEN : STD_LOGIC; -- EM 存储器与数据总线(D_BUS)相通控制位
SIGNAL PCOE : STD_LOGIC; -- PC 寄存器地址输出允许
SIGNAL EMRD : STD_LOGIC; -- 主存储器读允许
SIGNAL EMWR : STD_LOGIC; -- 主存储器写允许
SIGNAL XRD : STD_LOGIC; -- 外部 I/O 读允许

-- ALU 运算器定义
SIGNAL T : STD_LOGIC_VECTOR(DataWidth DOWNTO 0); -- 运算结果
SIGNAL D : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 直通门
SIGNAL R : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 右移门
SIGNAL L : STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 左移门

-- 总线定义
SIGNAL D_BUS: STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 数据总线
SIGNAL I_BUS: STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 指令总线
SIGNAL A_BUS: STD_LOGIC_VECTOR(DataWidth-1 DOWNTO 0); -- 地址总线

```

```

-- 指令周期
SIGNAL RT: STD_LOGIC_VECTOR(1 DOWNTO 0); -- 当前周期数
SIGNAL CT: STD_LOGIC_VECTOR(1 DOWNTO 0); -- 下一条指令总周期数

BEGIN
  IA <= INT_ENTER; -- 中断向量, 定义为 "E0"

  -- 寄存器 A
  PROCESS(clk, rst, AEN)
  BEGIN
    IF rst = '1' THEN
      A <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
      IF AEN = '0' THEN
        A <= D_BUS;
      END IF;
    END IF;
  END PROCESS;

  -- 寄存器 W
  PROCESS(clk, rst, WEN)
  BEGIN
    IF rst = '1' THEN
      W <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
      IF WEN = '0' THEN
        W <= D_BUS;
      END IF;
    END IF;
  END PROCESS;

  -- 寄存器 R0
  PROCESS(clk, rst, RWR, IR)
  BEGIN
    IF rst = '1' THEN
      R0 <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
      IF (RWR = '0') AND (IR(1) = '0') AND (IR(0) = '0') THEN
        R0 <= D_BUS;
      END IF;
    END IF;
  END PROCESS;

  -- 寄存器 R1
  PROCESS(clk, rst, RWR, IR)
  BEGIN
    IF rst = '1' THEN
      R1 <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
      IF (RWR = '0') AND (IR(1) = '0') AND (IR(0) = '1') THEN
        R1 <= D_BUS;
      END IF;
    END IF;
  END PROCESS;

  -- 寄存器 R2
  PROCESS(clk, rst, RWR, IR)
  BEGIN
    IF rst = '1' THEN
      R2 <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN

```

```
IF (RWR = '0') AND (IR(1) = '1') AND (IR(0) = '0') THEN
    R2 <= D_BUS;
END IF;
END IF;
END PROCESS;

-- 寄存器 R3
PROCESS(clk, rst, RWR, IR)
BEGIN
    IF rst = '1' THEN
        R3 <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF (RWR = '0') AND (IR(1) = '1') AND (IR(0) = '1') THEN
            R3 <= D_BUS;
        END IF;
    END IF;
END PROCESS;

-- 寄存器 OUT
PROCESS(clk, rst, OUTEN)
BEGIN
    IF rst = '1' THEN
        portout <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF OUTEN = '0' THEN
            portout <= D_BUS;
        END IF;
    END IF;
END PROCESS;

-- 寄存器 ST
PROCESS(clk, rst, STEN)
BEGIN
    IF rst = '1' THEN
        ST <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF STEN = '0' THEN
            ST <= D_BUS;
        END IF;
    END IF;
END PROCESS;

-- 寄存器 MAR
PROCESS(clk, rst, MAREN)
BEGIN
    IF rst = '1' THEN
        MAR <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF MAREN = '0' THEN
            MAR <= D_BUS;
        END IF;
    END IF;
END PROCESS;

-- 寄存器 IR
PROCESS(clk, rst, IREN)
BEGIN
    IF rst = '1' THEN
        IR <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF IREN = '0' THEN
            IR <= I_BUS;
        END IF;
    END IF;
END PROCESS;
```

```

    END IF;
    END IF;
END PROCESS;

-- 寄存器 PC
PROCESS(clk, rst, IR, R_CY, R_Z, ELP)
VARIABLE LDPC: STD_LOGIC; -- 转移控制
BEGIN
    IF rst = '1' THEN
        PC <= (OTHERS=>'0');
    ELSIF clk'EVENT AND clk = '1' THEN
        IF (ELP = '0') AND (
            ( (IR(3) = '0') AND (IR(2) = '0') AND (R_CY = '1') ) OR
            ( (IR(3) = '0') AND (IR(2) = '1') AND (R_Z = '1') ) OR
            (IR(3) = '1')
        ) THEN LDPC := '0'; ELSE LDPC := '1';
    END IF;

    IF LDPC = '0' THEN
        PC <= D_BUS;
    ELSIF (PCOE = '0') AND (ICEN = '1') THEN -- 在转入中断时 PC 不加一
        PC <= PC + 1;
    END IF;
    END IF;
END PROCESS;

-- 标志位
PROCESS(clk, rst, FEN)
BEGIN
    IF rst = '1' THEN
        R_CY <= '0';
        R_Z <= '0';
    ELSIF clk'EVENT AND clk = '1' THEN
        IF FEN = '0' THEN
            R_CY <= CY;
            R_Z <= Z;
        END IF;
    END IF;
END PROCESS;

-- ALU 运算器, 第八位为进位标志
T <= ('0' & A) + ('0' & W) WHEN S2 = '0' AND S1 = '0' AND SO = '0' ELSE
    ('0' & A) - ('0' & W) WHEN S2 = '0' AND S1 = '0' AND SO = '1' ELSE
    ('0' & A) OR ('0' & W) WHEN S2 = '0' AND S1 = '1' AND SO = '0' ELSE
    ('0' & A) AND ('0' & W) WHEN S2 = '0' AND S1 = '1' AND SO = '1' ELSE
    ('0' & A) + ('0' & W) + R_CY WHEN S2 = '1' AND S1 = '0' AND SO = '0' ELSE
    ('0' & A) - ('0' & W) - R_CY WHEN S2 = '1' AND S1 = '0' AND SO = '1' ELSE
    NOT ('0' & A) WHEN S2 = '1' AND S1 = '1' AND SO = '0' ELSE
    '0' & A;

-- 直通门
D <= T(DataWidth-1 DOWNTO 0);

-- 右移门
R(DataWidth-2 DOWNTO 0) <= T(DataWidth-1 DOWNTO 1);
R(DataWidth-1) <= R_CY WHEN CN = '1' ELSE '0';

-- 左移门
L(DataWidth-1 DOWNTO 1) <= T(DataWidth-2 DOWNTO 0);
L(0) <= R_CY WHEN CN = '1' ELSE '0';

CY <= T(0) WHEN (X1 = '0') AND (X0 = '1') AND (CN = '1') ELSE --不带进位右移

```

```

        R_CY          WHEN (X1 = '0') AND (X0 = '1') AND (CN = '0') ELSE --带进位右移
        T(DataWidth-1) WHEN (X1 = '1') AND (X0 = '0') AND (CN = '1') ELSE --不带进位左移
        R_CY          WHEN (X1 = '1') AND (X0 = '0') AND (CN = '0') ELSE --带进位左移
        T(DataWidth); --直通

Z <= '1' WHEN (R = ALL_ZERO) AND (X1 = '0') AND (X0 = '1') ELSE -- 右移门
'1' WHEN (L = ALL_ZERO) AND (X1 = '1') AND (X0 = '0') ELSE -- 左移门
'1' WHEN (D = ALL_ZERO) ELSE -- 直通门
'0';

-- 中断处理
PROCESS(i_req, rst, EINT)
BEGIN
    IF (EINT = '0') OR (rst = '1') THEN
        R_REQ <= '0';
    ELSIF i_req'EVENT AND i_req = '1' THEN
        R_REQ <= '1';
    END IF;
END PROCESS;

PROCESS(clk, rst, EINT, IREN, R_REQ, R_ACK, ICEN)
BEGIN
    IF (IREN = '0') AND (R_REQ = '1') AND (R_ACK = '0') THEN
        ICEN <= '0';
    ELSE
        ICEN <= '1';
    END IF;

    IF (EINT = '0') OR (rst = '1') THEN
        R_ACK <= '0';
    ELSIF clk'EVENT AND clk = '1' THEN
        IF ICEN = '0' THEN
            R_ACK <= R_REQ;
        END IF;
    END IF;
END PROCESS;

-- 地址总线
A_BUS <= PC WHEN PCOE = '0' ELSE
        MAR; -- WHEN MAROE = '0'

-- 指令总线
I_BUS <= INT_CODE WHEN ICEN = '0' ELSE mem_d;

-- 数据总线
PROCESS(X2, X1, X0, RRD, keyin, IA, ST, PC, D, R, L, R0, R1, R2, R3, IR, I_BUS)
BEGIN
    IF (X2 = '0') AND (X1 = '0') AND (X0 = '0') THEN
        D_BUS <= keyin;
    ELSIF (X2 = '0') AND (X1 = '0') AND (X0 = '1') THEN
        D_BUS <= IA;
    ELSIF (X2 = '0') AND (X1 = '1') AND (X0 = '0') THEN
        D_BUS <= ST;
    ELSIF (X2 = '0') AND (X1 = '1') AND (X0 = '1') THEN
        D_BUS <= PC;
    ELSIF (X2 = '1') AND (X1 = '0') AND (X0 = '0') THEN
        D_BUS <= D;
    ELSIF (X2 = '1') AND (X1 = '0') AND (X0 = '1') THEN
        D_BUS <= R;
    ELSIF (X2 = '1') AND (X1 = '1') AND (X0 = '0') THEN
        D_BUS <= L;
    ELSIF (RRD = '0') AND (IR(1) = '0') AND (IR(0) = '0') THEN

```



```

    D_BUS <= R0;
    ELSIF (RRD = '0') AND (IR(1) = '0') AND (IR(0) = '1') THEN
        D_BUS <= R1;
    ELSIF (RRD = '0') AND (IR(1) = '1') AND (IR(0) = '0') THEN
        D_BUS <= R2;
    ELSIF (RRD = '0') AND (IR(1) = '1') AND (IR(0) = '1') THEN
        D_BUS <= R3;
    ELSE
        D_BUS <= I_BUS; -- (EMRD = '0') AND (EMEN = '0')
    END IF;
END PROCESS;

-- 主存储器 EM
mem_bh <= '0';
mem_bl <= '0';
mem_cs <= '0';

mem_a(DataWidth-1 DOWNT0 0) <= A_BUS;
-- mem_ah <= (OTHERS=>'0'); -- 八位机时, 高位地址为 0
mem_rd <= '0' WHEN (EMRD = '0') AND (ICEN = '1') ELSE '1';
mem_wr <= NOT clk WHEN EMWR = '0' ELSE '1';
mem_d <= D_BUS WHEN EMWR = '0' ELSE (OTHERS=>'Z');

-- 指令周期
PROCESS(clk, rst)
BEGIN
    IF rst = '1' THEN
        RT <= "00";
    ELSIF clk'EVENT AND clk = '1' THEN
        IF RT = "00" THEN
            RT <= CT; -- 取下一条指令时同时算出下一条指令的周期数 CT
        ELSE
            RT <= RT - 1;
        END IF;
    END IF;
END PROCESS;

-- 24 位微控制信号
PROCESS(IR, RT)
VARIABLE STATUS: STD_LOGIC_VECTOR(7 DOWNT0 0);
BEGIN
    STATUS(7 DOWNT0 2) := IR(7 DOWNT0 2);
    STATUS(1 DOWNT0 0) := RT(1 DOWNT0 0);

    -- XRD 1
    CASE STATUS IS
        WHEN "10010001" => XRD <= '0'; -- READ    A, MM
        WHEN OTHERS      => XRD <= '1';
    END CASE;

    -- EMWR 2
    CASE STATUS IS
        WHEN "10000101" => EMWR <= '0'; -- MOV    @R?, A
        WHEN "10001001" => EMWR <= '0'; -- MOV    MM, A
        WHEN OTHERS      => EMWR <= '1';
    END CASE;

    -- EMRD 100
    CASE STATUS IS
        WHEN "00000000" => EMRD <= '0'; -- _FATCH_
        WHEN "00000100" => EMRD <= '0'; -- UNDEF
        WHEN "00001000" => EMRD <= '0'; -- UNDEF
    END CASE;

```

```

WHEN "00001100" => EMRD <= '0'; -- UNDEF
WHEN "00010000" => EMRD <= '0'; -- ADD      A, R?
WHEN "00010110" => EMRD <= '0'; -- ADD      A, @R?
WHEN "00010100" => EMRD <= '0'; -- ADD      A, @R?
WHEN "00011011" => EMRD <= '0'; -- ADD      A, MM
WHEN "00011010" => EMRD <= '0'; -- ADD      A, MM
WHEN "00011000" => EMRD <= '0'; -- ADD      A, MM
WHEN "00011110" => EMRD <= '0'; -- ADD      A, #11
WHEN "00011100" => EMRD <= '0'; -- ADD      A, #11
WHEN "00100000" => EMRD <= '0'; -- ADDC     A, R?
WHEN "00100110" => EMRD <= '0'; -- ADDC     A, @R?
WHEN "00100100" => EMRD <= '0'; -- ADDC     A, @R?
WHEN "00101011" => EMRD <= '0'; -- ADDC     A, MM
WHEN "00101010" => EMRD <= '0'; -- ADDC     A, MM
WHEN "00101000" => EMRD <= '0'; -- ADDC     A, MM
WHEN "00101110" => EMRD <= '0'; -- ADDC     A, #11
WHEN "00101100" => EMRD <= '0'; -- ADDC     A, #11
WHEN "00110000" => EMRD <= '0'; -- SUB      A, R?
WHEN "00110110" => EMRD <= '0'; -- SUB      A, @R?
WHEN "00110100" => EMRD <= '0'; -- SUB      A, @R?
WHEN "00111011" => EMRD <= '0'; -- SUB      A, MM
WHEN "00111010" => EMRD <= '0'; -- SUB      A, MM
WHEN "00111000" => EMRD <= '0'; -- SUB      A, MM
WHEN "00111110" => EMRD <= '0'; -- SUB      A, #11
WHEN "00111100" => EMRD <= '0'; -- SUB      A, #11
WHEN "01000000" => EMRD <= '0'; -- SUBC     A, R?
WHEN "01000110" => EMRD <= '0'; -- SUBC     A, @R?
WHEN "01000100" => EMRD <= '0'; -- SUBC     A, @R?
WHEN "01001011" => EMRD <= '0'; -- SUBC     A, MM
WHEN "01001010" => EMRD <= '0'; -- SUBC     A, MM
WHEN "01001000" => EMRD <= '0'; -- SUBC     A, MM
WHEN "01001110" => EMRD <= '0'; -- SUBC     A, #11
WHEN "01001100" => EMRD <= '0'; -- SUBC     A, #11
WHEN "01010000" => EMRD <= '0'; -- AND      A, R?
WHEN "01010110" => EMRD <= '0'; -- AND      A, @R?
WHEN "01010100" => EMRD <= '0'; -- AND      A, @R?
WHEN "01011011" => EMRD <= '0'; -- AND      A, MM
WHEN "01011010" => EMRD <= '0'; -- AND      A, MM
WHEN "01011000" => EMRD <= '0'; -- AND      A, MM
WHEN "01011110" => EMRD <= '0'; -- AND      A, #11
WHEN "01011100" => EMRD <= '0'; -- AND      A, #11
WHEN "01100000" => EMRD <= '0'; -- OR       A, R?
WHEN "01100110" => EMRD <= '0'; -- OR       A, @R?
WHEN "01100100" => EMRD <= '0'; -- OR       A, @R?
WHEN "01101011" => EMRD <= '0'; -- OR       A, MM
WHEN "01101010" => EMRD <= '0'; -- OR       A, MM
WHEN "01101000" => EMRD <= '0'; -- OR       A, MM
WHEN "01101110" => EMRD <= '0'; -- OR       A, #11
WHEN "01101100" => EMRD <= '0'; -- OR       A, #11
WHEN "01110000" => EMRD <= '0'; -- MOV      A, R?
WHEN "01110101" => EMRD <= '0'; -- MOV      A, @R?
WHEN "01110100" => EMRD <= '0'; -- MOV      A, @R?
WHEN "01111010" => EMRD <= '0'; -- MOV      A, MM
WHEN "01111001" => EMRD <= '0'; -- MOV      A, MM
WHEN "01111000" => EMRD <= '0'; -- MOV      A, MM
WHEN "01111101" => EMRD <= '0'; -- MOV      A, #11
WHEN "01111100" => EMRD <= '0'; -- MOV      A, #11
WHEN "10000000" => EMRD <= '0'; -- MOV      R?, A
WHEN "10000100" => EMRD <= '0'; -- MOV      @R?, A
WHEN "10001010" => EMRD <= '0'; -- MOV      MM, A
WHEN "10001000" => EMRD <= '0'; -- MOV      MM, A
WHEN "10001101" => EMRD <= '0'; -- MOV      R?, #11

```

```

WHEN "10001100" => EMRD <= '0'; -- MOV      R?, #11
WHEN "10010010" => EMRD <= '0'; -- READ      A, MM
WHEN "10010000" => EMRD <= '0'; -- READ      A, MM
WHEN "10010110" => EMRD <= '0'; -- WRITE     MM, A
WHEN "10010100" => EMRD <= '0'; -- WRITE     MM, A
WHEN "10011000" => EMRD <= '0'; -- UNDEF
WHEN "10011100" => EMRD <= '0'; -- UNDEF
WHEN "10100001" => EMRD <= '0'; -- JC         MM
WHEN "10100000" => EMRD <= '0'; -- JC         MM
WHEN "10100101" => EMRD <= '0'; -- JZ         MM
WHEN "10100100" => EMRD <= '0'; -- JZ         MM
WHEN "10101000" => EMRD <= '0'; -- UNDEF
WHEN "10101101" => EMRD <= '0'; -- JMP        MM
WHEN "10101100" => EMRD <= '0'; -- JMP        MM
WHEN "10110000" => EMRD <= '0'; -- UNDEF
WHEN "10110100" => EMRD <= '0'; -- UNDEF
WHEN "10111000" => EMRD <= '0'; -- _INT_
WHEN "10111101" => EMRD <= '0'; -- CALL      MM
WHEN "10111100" => EMRD <= '0'; -- CALL      MM
WHEN "11000000" => EMRD <= '0'; -- IN
WHEN "11000100" => EMRD <= '0'; -- OUT
WHEN "11001000" => EMRD <= '0'; -- UNDEF
WHEN "11001100" => EMRD <= '0'; -- RET
WHEN "11010000" => EMRD <= '0'; -- RR         A
WHEN "11010100" => EMRD <= '0'; -- RL         A
WHEN "11011000" => EMRD <= '0'; -- RRC        A
WHEN "11011100" => EMRD <= '0'; -- RLC        A
WHEN "11100000" => EMRD <= '0'; -- NOP
WHEN "11100100" => EMRD <= '0'; -- CPL        A
WHEN "11101000" => EMRD <= '0'; -- UNDEF
WHEN "11101100" => EMRD <= '0'; -- RETI
WHEN "11110000" => EMRD <= '0'; -- UNDEF
WHEN "11110100" => EMRD <= '0'; -- UNDEF
WHEN "11111000" => EMRD <= '0'; -- UNDEF
WHEN "11111100" => EMRD <= '0'; -- UNDEF
WHEN OTHERS      => EMRD <= '1';
END CASE;

```

-- PCOE 86

CASE STATUS IS

```

WHEN "00000000" => PCOE <= '0'; -- _FATCH_
WHEN "00000100" => PCOE <= '0'; -- UNDEF
WHEN "00001000" => PCOE <= '0'; -- UNDEF
WHEN "00001100" => PCOE <= '0'; -- UNDEF
WHEN "00010000" => PCOE <= '0'; -- ADD      A, R?
WHEN "00010100" => PCOE <= '0'; -- ADD      A, @R?
WHEN "00011011" => PCOE <= '0'; -- ADD      A, MM
WHEN "00011000" => PCOE <= '0'; -- ADD      A, MM
WHEN "00011110" => PCOE <= '0'; -- ADD      A, #11
WHEN "00011100" => PCOE <= '0'; -- ADD      A, #11
WHEN "00100000" => PCOE <= '0'; -- ADDC     A, R?
WHEN "00100100" => PCOE <= '0'; -- ADDC     A, @R?
WHEN "00101011" => PCOE <= '0'; -- ADDC     A, MM
WHEN "00101000" => PCOE <= '0'; -- ADDC     A, MM
WHEN "00101110" => PCOE <= '0'; -- ADDC     A, #11
WHEN "00101100" => PCOE <= '0'; -- ADDC     A, #11
WHEN "00110000" => PCOE <= '0'; -- SUB      A, R?
WHEN "00110100" => PCOE <= '0'; -- SUB      A, @R?
WHEN "00111011" => PCOE <= '0'; -- SUB      A, MM
WHEN "00111000" => PCOE <= '0'; -- SUB      A, MM
WHEN "00111110" => PCOE <= '0'; -- SUB      A, #11
WHEN "00111100" => PCOE <= '0'; -- SUB      A, #11

```

```

WHEN "01000000" => PCOE <= '0'; -- SUBC    A, R?
WHEN "01000100" => PCOE <= '0'; -- SUBC    A, @R?
WHEN "01001011" => PCOE <= '0'; -- SUBC    A, MM
WHEN "01001000" => PCOE <= '0'; -- SUBC    A, MM
WHEN "01001110" => PCOE <= '0'; -- SUBC    A, #11
WHEN "01001100" => PCOE <= '0'; -- SUBC    A, #11
WHEN "01010000" => PCOE <= '0'; -- AND     A, R?
WHEN "01010100" => PCOE <= '0'; -- AND     A, @R?
WHEN "01011011" => PCOE <= '0'; -- AND     A, MM
WHEN "01011000" => PCOE <= '0'; -- AND     A, MM
WHEN "01011110" => PCOE <= '0'; -- AND     A, #11
WHEN "01011100" => PCOE <= '0'; -- AND     A, #11
WHEN "01100000" => PCOE <= '0'; -- OR      A, R?
WHEN "01100100" => PCOE <= '0'; -- OR      A, @R?
WHEN "01101011" => PCOE <= '0'; -- OR      A, MM
WHEN "01101000" => PCOE <= '0'; -- OR      A, MM
WHEN "01101110" => PCOE <= '0'; -- OR      A, #11
WHEN "01101100" => PCOE <= '0'; -- OR      A, #11
WHEN "01110000" => PCOE <= '0'; -- MOV     A, R?
WHEN "01110100" => PCOE <= '0'; -- MOV     A, @R?
WHEN "01111010" => PCOE <= '0'; -- MOV     A, MM
WHEN "01111000" => PCOE <= '0'; -- MOV     A, MM
WHEN "01111101" => PCOE <= '0'; -- MOV     A, #11
WHEN "01111100" => PCOE <= '0'; -- MOV     A, #11
WHEN "10000000" => PCOE <= '0'; -- MOV     R?, A
WHEN "10000100" => PCOE <= '0'; -- MOV     @R?, A
WHEN "10001010" => PCOE <= '0'; -- MOV     MM, A
WHEN "10001000" => PCOE <= '0'; -- MOV     MM, A
WHEN "10001101" => PCOE <= '0'; -- MOV     R?, #11
WHEN "10001100" => PCOE <= '0'; -- MOV     R?, #11
WHEN "10010010" => PCOE <= '0'; -- READ    A, MM
WHEN "10010000" => PCOE <= '0'; -- READ    A, MM
WHEN "10010110" => PCOE <= '0'; -- WRITE   MM, A
WHEN "10010100" => PCOE <= '0'; -- WRITE   MM, A
WHEN "10011000" => PCOE <= '0'; -- UNDEF
WHEN "10011100" => PCOE <= '0'; -- UNDEF
WHEN "10100001" => PCOE <= '0'; -- JC      MM
WHEN "10100000" => PCOE <= '0'; -- JC      MM
WHEN "10100101" => PCOE <= '0'; -- JZ      MM
WHEN "10100100" => PCOE <= '0'; -- JZ      MM
WHEN "10101000" => PCOE <= '0'; -- UNDEF
WHEN "10101101" => PCOE <= '0'; -- JMP     MM
WHEN "10101100" => PCOE <= '0'; -- JMP     MM
WHEN "10110000" => PCOE <= '0'; -- UNDEF
WHEN "10110100" => PCOE <= '0'; -- UNDEF
WHEN "10111000" => PCOE <= '0'; -- _INT_
WHEN "10111111" => PCOE <= '0'; -- CALL    MM
WHEN "10111100" => PCOE <= '0'; -- CALL    MM
WHEN "11000000" => PCOE <= '0'; -- IN
WHEN "11000100" => PCOE <= '0'; -- OUT
WHEN "11001000" => PCOE <= '0'; -- UNDEF
WHEN "11001100" => PCOE <= '0'; -- RET
WHEN "11010000" => PCOE <= '0'; -- RR      A
WHEN "11010100" => PCOE <= '0'; -- RL      A
WHEN "11011000" => PCOE <= '0'; -- RRC     A
WHEN "11011100" => PCOE <= '0'; -- RLC     A
WHEN "11100000" => PCOE <= '0'; -- NOP
WHEN "11100100" => PCOE <= '0'; -- CPL     A
WHEN "11101000" => PCOE <= '0'; -- UNDEF
WHEN "11101100" => PCOE <= '0'; -- RETI
WHEN "11110000" => PCOE <= '0'; -- UNDEF
WHEN "11110100" => PCOE <= '0'; -- UNDEF

```

```

    WHEN "11111000" => PCOE <= '0'; -- UNDEF
    WHEN "11111100" => PCOE <= '0'; -- UNDEF
    WHEN OTHERS      => PCOE <= '1';
END CASE;

-- EMEN 38
CASE STATUS IS
    WHEN "00010110" => EMEN <= '0'; -- ADD      A, @R?
    WHEN "00011011" => EMEN <= '0'; -- ADD      A, MM
    WHEN "00011010" => EMEN <= '0'; -- ADD      A, MM
    WHEN "00011110" => EMEN <= '0'; -- ADD      A, #11
    WHEN "00100110" => EMEN <= '0'; -- ADDC     A, @R?
    WHEN "00101011" => EMEN <= '0'; -- ADDC     A, MM
    WHEN "00101010" => EMEN <= '0'; -- ADDC     A, MM
    WHEN "00101110" => EMEN <= '0'; -- ADDC     A, #11
    WHEN "00110110" => EMEN <= '0'; -- SUB      A, @R?
    WHEN "00111011" => EMEN <= '0'; -- SUB      A, MM
    WHEN "00111010" => EMEN <= '0'; -- SUB      A, MM
    WHEN "00111110" => EMEN <= '0'; -- SUB      A, #11
    WHEN "01000110" => EMEN <= '0'; -- SUBC     A, @R?
    WHEN "01001011" => EMEN <= '0'; -- SUBC     A, MM
    WHEN "01001010" => EMEN <= '0'; -- SUBC     A, MM
    WHEN "01001110" => EMEN <= '0'; -- SUBC     A, #11
    WHEN "01010110" => EMEN <= '0'; -- AND      A, @R?
    WHEN "01011011" => EMEN <= '0'; -- AND      A, MM
    WHEN "01011010" => EMEN <= '0'; -- AND      A, MM
    WHEN "01011110" => EMEN <= '0'; -- AND      A, #11
    WHEN "01100110" => EMEN <= '0'; -- OR       A, @R?
    WHEN "01101011" => EMEN <= '0'; -- OR       A, MM
    WHEN "01101010" => EMEN <= '0'; -- OR       A, MM
    WHEN "01101110" => EMEN <= '0'; -- OR       A, #11
    WHEN "01110101" => EMEN <= '0'; -- MOV      A, @R?
    WHEN "01111010" => EMEN <= '0'; -- MOV      A, MM
    WHEN "01111001" => EMEN <= '0'; -- MOV      A, MM
    WHEN "01111101" => EMEN <= '0'; -- MOV      A, #11
    WHEN "10000101" => EMEN <= '0'; -- MOV      @R?, A
    WHEN "10001010" => EMEN <= '0'; -- MOV      MM, A
    WHEN "10001001" => EMEN <= '0'; -- MOV      MM, A
    WHEN "10001101" => EMEN <= '0'; -- MOV      R?, #11
    WHEN "10010010" => EMEN <= '0'; -- READ     A, MM
    WHEN "10010110" => EMEN <= '0'; -- WRITE    MM, A
    WHEN "10100001" => EMEN <= '0'; -- JC       MM
    WHEN "10100101" => EMEN <= '0'; -- JZ       MM
    WHEN "10101101" => EMEN <= '0'; -- JMP      MM
    WHEN "10111101" => EMEN <= '0'; -- CALL     MM
    WHEN OTHERS      => EMEN <= '1';
END CASE;

-- IREN 64
CASE STATUS IS
    WHEN "00000000" => IREN <= '0'; -- _FATCH_
    WHEN "00000100" => IREN <= '0'; -- UNDEF
    WHEN "00001000" => IREN <= '0'; -- UNDEF
    WHEN "00001100" => IREN <= '0'; -- UNDEF
    WHEN "00010000" => IREN <= '0'; -- ADD      A, R?
    WHEN "00010100" => IREN <= '0'; -- ADD      A, @R?
    WHEN "00011000" => IREN <= '0'; -- ADD      A, MM
    WHEN "00011100" => IREN <= '0'; -- ADD      A, #11
    WHEN "00100000" => IREN <= '0'; -- ADDC     A, R?
    WHEN "00100100" => IREN <= '0'; -- ADDC     A, @R?
    WHEN "00101000" => IREN <= '0'; -- ADDC     A, MM
    WHEN "00101100" => IREN <= '0'; -- ADDC     A, #11

```

```

WHEN "00110000" => IREN <= '0'; -- SUB      A, R?
WHEN "00110100" => IREN <= '0'; -- SUB      A, @R?
WHEN "00111000" => IREN <= '0'; -- SUB      A, MM
WHEN "00111100" => IREN <= '0'; -- SUB      A, #11
WHEN "01000000" => IREN <= '0'; -- SUBC     A, R?
WHEN "01000100" => IREN <= '0'; -- SUBC     A, @R?
WHEN "01001000" => IREN <= '0'; -- SUBC     A, MM
WHEN "01001100" => IREN <= '0'; -- SUBC     A, #11
WHEN "01010000" => IREN <= '0'; -- AND      A, R?
WHEN "01010100" => IREN <= '0'; -- AND      A, @R?
WHEN "01011000" => IREN <= '0'; -- AND      A, MM
WHEN "01011100" => IREN <= '0'; -- AND      A, #11
WHEN "01100000" => IREN <= '0'; -- OR       A, R?
WHEN "01100100" => IREN <= '0'; -- OR       A, @R?
WHEN "01101000" => IREN <= '0'; -- OR       A, MM
WHEN "01101100" => IREN <= '0'; -- OR       A, #11
WHEN "01110000" => IREN <= '0'; -- MOV      A, R?
WHEN "01110100" => IREN <= '0'; -- MOV      A, @R?
WHEN "01111000" => IREN <= '0'; -- MOV      A, MM
WHEN "01111100" => IREN <= '0'; -- MOV      A, #11
WHEN "10000000" => IREN <= '0'; -- MOV      R?, A
WHEN "10000100" => IREN <= '0'; -- MOV      @R?, A
WHEN "10001000" => IREN <= '0'; -- MOV      MM, A
WHEN "10001100" => IREN <= '0'; -- MOV      R?, #11
WHEN "10010000" => IREN <= '0'; -- READ     A, MM
WHEN "10010100" => IREN <= '0'; -- WRITE    MM, A
WHEN "10011000" => IREN <= '0'; -- UNDEF
WHEN "10011100" => IREN <= '0'; -- UNDEF
WHEN "10100000" => IREN <= '0'; -- JC       MM
WHEN "10100100" => IREN <= '0'; -- JZ       MM
WHEN "10101000" => IREN <= '0'; -- UNDEF
WHEN "10101100" => IREN <= '0'; -- JMP      MM
WHEN "10110000" => IREN <= '0'; -- UNDEF
WHEN "10110100" => IREN <= '0'; -- UNDEF
WHEN "10111000" => IREN <= '0'; -- _INT_
WHEN "10111100" => IREN <= '0'; -- CALL    MM
WHEN "11000000" => IREN <= '0'; -- IN
WHEN "11000100" => IREN <= '0'; -- OUT
WHEN "11001000" => IREN <= '0'; -- UNDEF
WHEN "11001100" => IREN <= '0'; -- RET
WHEN "11010000" => IREN <= '0'; -- RR      A
WHEN "11010100" => IREN <= '0'; -- RL      A
WHEN "11011000" => IREN <= '0'; -- RRC     A
WHEN "11011100" => IREN <= '0'; -- RLC     A
WHEN "11100000" => IREN <= '0'; -- NOP
WHEN "11100100" => IREN <= '0'; -- CPL      A
WHEN "11101000" => IREN <= '0'; -- UNDEF
WHEN "11101100" => IREN <= '0'; -- RETI
WHEN "11110000" => IREN <= '0'; -- UNDEF
WHEN "11110100" => IREN <= '0'; -- UNDEF
WHEN "11111000" => IREN <= '0'; -- UNDEF
WHEN "11111100" => IREN <= '0'; -- UNDEF
WHEN OTHERS      => IREN <= '1';
END CASE;

-- EINT 1
CASE STATUS IS
  WHEN "11101101" => EINT <= '0'; -- RETI
  WHEN OTHERS      => EINT <= '1';
END CASE;

-- ELP 7

```

```

CASE STATUS IS
  WHEN "10100001" => ELP <= '0'; -- JC      MM
  WHEN "10100101" => ELP <= '0'; -- JZ      MM
  WHEN "10101101" => ELP <= '0'; -- JMP      MM
  WHEN "10111001" => ELP <= '0'; -- _INT_
  WHEN "10111101" => ELP <= '0'; -- CALL    MM
  WHEN "11001101" => ELP <= '0'; -- RET
  WHEN "11101101" => ELP <= '0'; -- RETI
  WHEN OTHERS      => ELP <= '1';
END CASE;

-- MAREN 19
CASE STATUS IS
  WHEN "00010111" => MAREN <= '0'; -- ADD      A, @R?
  WHEN "00011011" => MAREN <= '0'; -- ADD      A, MM
  WHEN "00100111" => MAREN <= '0'; -- ADDC     A, @R?
  WHEN "00101011" => MAREN <= '0'; -- ADDC     A, MM
  WHEN "00110111" => MAREN <= '0'; -- SUB      A, @R?
  WHEN "00111011" => MAREN <= '0'; -- SUB      A, MM
  WHEN "01000111" => MAREN <= '0'; -- SUBC     A, @R?
  WHEN "01001011" => MAREN <= '0'; -- SUBC     A, MM
  WHEN "01010111" => MAREN <= '0'; -- AND      A, @R?
  WHEN "01011011" => MAREN <= '0'; -- AND      A, MM
  WHEN "01100111" => MAREN <= '0'; -- OR       A, @R?
  WHEN "01101011" => MAREN <= '0'; -- OR       A, MM
  WHEN "01110110" => MAREN <= '0'; -- MOV      A, @R?
  WHEN "01111010" => MAREN <= '0'; -- MOV      A, MM
  WHEN "10000110" => MAREN <= '0'; -- MOV      @R?, A
  WHEN "10001010" => MAREN <= '0'; -- MOV      MM, A
  WHEN "10010010" => MAREN <= '0'; -- READ     A, MM
  WHEN "10010110" => MAREN <= '0'; -- WRITE    MM, A
  WHEN "10111111" => MAREN <= '0'; -- CALL     MM
  WHEN OTHERS      => MAREN <= '1';
END CASE;

-- MAROE 19
CASE STATUS IS
  WHEN "00010110" => MAROE <= '0'; -- ADD      A, @R?
  WHEN "00011010" => MAROE <= '0'; -- ADD      A, MM
  WHEN "00100110" => MAROE <= '0'; -- ADDC     A, @R?
  WHEN "00101010" => MAROE <= '0'; -- ADDC     A, MM
  WHEN "00110110" => MAROE <= '0'; -- SUB      A, @R?
  WHEN "00111010" => MAROE <= '0'; -- SUB      A, MM
  WHEN "01000110" => MAROE <= '0'; -- SUBC     A, @R?
  WHEN "01001010" => MAROE <= '0'; -- SUBC     A, MM
  WHEN "01010110" => MAROE <= '0'; -- AND      A, @R?
  WHEN "01011010" => MAROE <= '0'; -- AND      A, MM
  WHEN "01100110" => MAROE <= '0'; -- OR       A, @R?
  WHEN "01101010" => MAROE <= '0'; -- OR       A, MM
  WHEN "01110101" => MAROE <= '0'; -- MOV      A, @R?
  WHEN "01111001" => MAROE <= '0'; -- MOV      A, MM
  WHEN "10000101" => MAROE <= '0'; -- MOV      @R?, A
  WHEN "10001001" => MAROE <= '0'; -- MOV      MM, A
  WHEN "10010001" => MAROE <= '0'; -- READ     A, MM
  WHEN "10010101" => MAROE <= '0'; -- WRITE    MM, A
  WHEN "10111101" => MAROE <= '0'; -- CALL     MM
  WHEN OTHERS      => MAROE <= '1';
END CASE;

-- OUTEN 2
CASE STATUS IS
  WHEN "10010101" => OUTEN <= '0'; -- WRITE    MM, A

```

```

    WHEN "11000101" => OUTEN <= '0'; -- OUT
    WHEN OTHERS      => OUTEN <= '1';
END CASE;

-- STEN 2
CASE STATUS IS
    WHEN "10111010" => STEN <= '0'; -- _INT_
    WHEN "10111110" => STEN <= '0'; -- CALL    MM
    WHEN OTHERS      => STEN <= '1';
END CASE;

-- RRD 15
CASE STATUS IS
    WHEN "00010010" => RRD <= '0'; -- ADD      A, R?
    WHEN "00010111" => RRD <= '0'; -- ADD      A, @R?
    WHEN "00100010" => RRD <= '0'; -- ADDC     A, R?
    WHEN "00100111" => RRD <= '0'; -- ADDC     A, @R?
    WHEN "00110010" => RRD <= '0'; -- SUB      A, R?
    WHEN "00110111" => RRD <= '0'; -- SUB      A, @R?
    WHEN "01000010" => RRD <= '0'; -- SUBC     A, R?
    WHEN "01000111" => RRD <= '0'; -- SUBC     A, @R?
    WHEN "01010010" => RRD <= '0'; -- AND      A, R?
    WHEN "01010111" => RRD <= '0'; -- AND      A, @R?
    WHEN "01100010" => RRD <= '0'; -- OR       A, R?
    WHEN "01100111" => RRD <= '0'; -- OR       A, @R?
    WHEN "01110001" => RRD <= '0'; -- MOV      A, R?
    WHEN "01110110" => RRD <= '0'; -- MOV      A, @R?
    WHEN "10000110" => RRD <= '0'; -- MOV      @R?, A
    WHEN OTHERS      => RRD <= '1';
END CASE;

-- RWR 2
CASE STATUS IS
    WHEN "10000001" => RWR <= '0'; -- MOV      R?, A
    WHEN "10001101" => RWR <= '0'; -- MOV      R?, #11
    WHEN OTHERS      => RWR <= '1';
END CASE;

-- CN 2
CASE STATUS IS
    WHEN "11010001" => CN <= '0'; -- RR       A
    WHEN "11010101" => CN <= '0'; -- RL       A
    WHEN OTHERS      => CN <= '1';
END CASE;

-- FEN 29
CASE STATUS IS
    WHEN "00010001" => FEN <= '0'; -- ADD      A, R?
    WHEN "00010101" => FEN <= '0'; -- ADD      A, @R?
    WHEN "00011001" => FEN <= '0'; -- ADD      A, MM
    WHEN "00011101" => FEN <= '0'; -- ADD      A, #11
    WHEN "00100001" => FEN <= '0'; -- ADDC     A, R?
    WHEN "00100101" => FEN <= '0'; -- ADDC     A, @R?
    WHEN "00101001" => FEN <= '0'; -- ADDC     A, MM
    WHEN "00101101" => FEN <= '0'; -- ADDC     A, #11
    WHEN "00110001" => FEN <= '0'; -- SUB      A, R?
    WHEN "00110101" => FEN <= '0'; -- SUB      A, @R?
    WHEN "00111001" => FEN <= '0'; -- SUB      A, MM
    WHEN "00111101" => FEN <= '0'; -- SUB      A, #11
    WHEN "01000001" => FEN <= '0'; -- SUBC     A, R?
    WHEN "01000101" => FEN <= '0'; -- SUBC     A, @R?
    WHEN "01001001" => FEN <= '0'; -- SUBC     A, MM

```



```

WHEN "01001101" => FEN <= '0'; -- SUBC    A, #11
WHEN "01010001" => FEN <= '0'; -- AND      A, R?
WHEN "01010101" => FEN <= '0'; -- AND      A, @R?
WHEN "01011001" => FEN <= '0'; -- AND      A, MM
WHEN "01011101" => FEN <= '0'; -- AND      A, #11
WHEN "01100001" => FEN <= '0'; -- OR        A, R?
WHEN "01100101" => FEN <= '0'; -- OR        A, @R?
WHEN "01101001" => FEN <= '0'; -- OR        A, MM
WHEN "01101101" => FEN <= '0'; -- OR        A, #11
WHEN "11010001" => FEN <= '0'; -- RR        A
WHEN "11010101" => FEN <= '0'; -- RL        A
WHEN "11011001" => FEN <= '0'; -- RRC       A
WHEN "11011101" => FEN <= '0'; -- RLC       A
WHEN "11100101" => FEN <= '0'; -- CPL        A
WHEN OTHERS      => FEN <= '1';
END CASE;

-- X2 7
CASE STATUS IS
WHEN "10111010" => X2 <= '0'; -- _INT_
WHEN "10111001" => X2 <= '0'; -- _INT_
WHEN "10111111" => X2 <= '0'; -- CALL    MM
WHEN "10111110" => X2 <= '0'; -- CALL    MM
WHEN "11000001" => X2 <= '0'; -- IN
WHEN "11001101" => X2 <= '0'; -- RET
WHEN "11101101" => X2 <= '0'; -- RETI
WHEN OTHERS      => X2 <= '1';
END CASE;

-- X1 34
CASE STATUS IS
WHEN "00010001" => X1 <= '0'; -- ADD      A, R?
WHEN "00010101" => X1 <= '0'; -- ADD      A, @R?
WHEN "00011001" => X1 <= '0'; -- ADD      A, MM
WHEN "00011101" => X1 <= '0'; -- ADD      A, #11
WHEN "00100001" => X1 <= '0'; -- ADDC     A, R?
WHEN "00100101" => X1 <= '0'; -- ADDC     A, @R?
WHEN "00101001" => X1 <= '0'; -- ADDC     A, MM
WHEN "00101101" => X1 <= '0'; -- ADDC     A, #11
WHEN "00110001" => X1 <= '0'; -- SUB      A, R?
WHEN "00110101" => X1 <= '0'; -- SUB      A, @R?
WHEN "00111001" => X1 <= '0'; -- SUB      A, MM
WHEN "00111101" => X1 <= '0'; -- SUB      A, #11
WHEN "01000001" => X1 <= '0'; -- SUBC     A, R?
WHEN "01000101" => X1 <= '0'; -- SUBC     A, @R?
WHEN "01001001" => X1 <= '0'; -- SUBC     A, MM
WHEN "01001101" => X1 <= '0'; -- SUBC     A, #11
WHEN "01010001" => X1 <= '0'; -- AND      A, R?
WHEN "01010101" => X1 <= '0'; -- AND      A, @R?
WHEN "01011001" => X1 <= '0'; -- AND      A, MM
WHEN "01011101" => X1 <= '0'; -- AND      A, #11
WHEN "01100001" => X1 <= '0'; -- OR        A, R?
WHEN "01100101" => X1 <= '0'; -- OR        A, @R?
WHEN "01101001" => X1 <= '0'; -- OR        A, MM
WHEN "01101101" => X1 <= '0'; -- OR        A, #11
WHEN "10000001" => X1 <= '0'; -- MOV      R?, A
WHEN "10000101" => X1 <= '0'; -- MOV      @R?, A
WHEN "10001001" => X1 <= '0'; -- MOV      MM, A
WHEN "10010101" => X1 <= '0'; -- WRITE    MM, A
WHEN "10111001" => X1 <= '0'; -- _INT_
WHEN "11000001" => X1 <= '0'; -- IN
WHEN "11000101" => X1 <= '0'; -- OUT

```

```

WHEN "11010001" => X1 <= '0'; -- RR      A
WHEN "11011001" => X1 <= '0'; -- RRC     A
WHEN "11100101" => X1 <= '0'; -- CPL     A
WHEN OTHERS      => X1 <= '1';
END CASE;

-- X0 35
CASE STATUS IS
WHEN "00010001" => X0 <= '0'; -- ADD      A, R?
WHEN "00010101" => X0 <= '0'; -- ADD      A, @R?
WHEN "00011001" => X0 <= '0'; -- ADD      A, MM
WHEN "00011101" => X0 <= '0'; -- ADD      A, #11
WHEN "00100001" => X0 <= '0'; -- ADDC     A, R?
WHEN "00100101" => X0 <= '0'; -- ADDC     A, @R?
WHEN "00101001" => X0 <= '0'; -- ADDC     A, MM
WHEN "00101101" => X0 <= '0'; -- ADDC     A, #11
WHEN "00110001" => X0 <= '0'; -- SUB      A, R?
WHEN "00110101" => X0 <= '0'; -- SUB      A, @R?
WHEN "00111001" => X0 <= '0'; -- SUB      A, MM
WHEN "00111101" => X0 <= '0'; -- SUB      A, #11
WHEN "01000001" => X0 <= '0'; -- SUBC     A, R?
WHEN "01000101" => X0 <= '0'; -- SUBC     A, @R?
WHEN "01001001" => X0 <= '0'; -- SUBC     A, MM
WHEN "01001101" => X0 <= '0'; -- SUBC     A, #11
WHEN "01010001" => X0 <= '0'; -- AND      A, R?
WHEN "01010101" => X0 <= '0'; -- AND      A, @R?
WHEN "01011001" => X0 <= '0'; -- AND      A, MM
WHEN "01011101" => X0 <= '0'; -- AND      A, #11
WHEN "01100001" => X0 <= '0'; -- OR       A, R?
WHEN "01100101" => X0 <= '0'; -- OR       A, @R?
WHEN "01101001" => X0 <= '0'; -- OR       A, MM
WHEN "01101101" => X0 <= '0'; -- OR       A, #11
WHEN "10000001" => X0 <= '0'; -- MOV      R?, A
WHEN "10000101" => X0 <= '0'; -- MOV      @R?, A
WHEN "10001001" => X0 <= '0'; -- MOV      MM, A
WHEN "10010101" => X0 <= '0'; -- WRITE    MM, A
WHEN "11000001" => X0 <= '0'; -- IN
WHEN "11000101" => X0 <= '0'; -- OUT
WHEN "11001101" => X0 <= '0'; -- RET
WHEN "11010101" => X0 <= '0'; -- RL       A
WHEN "11011101" => X0 <= '0'; -- RLC      A
WHEN "11100101" => X0 <= '0'; -- CPL      A
WHEN "11101101" => X0 <= '0'; -- RETI
WHEN OTHERS      => X0 <= '1';
END CASE;

-- WEN 24
CASE STATUS IS
WHEN "00010010" => WEN <= '0'; -- ADD      A, R?
WHEN "00010110" => WEN <= '0'; -- ADD      A, @R?
WHEN "00011010" => WEN <= '0'; -- ADD      A, MM
WHEN "00011110" => WEN <= '0'; -- ADD      A, #11
WHEN "00100010" => WEN <= '0'; -- ADDC     A, R?
WHEN "00100110" => WEN <= '0'; -- ADDC     A, @R?
WHEN "00101010" => WEN <= '0'; -- ADDC     A, MM
WHEN "00101110" => WEN <= '0'; -- ADDC     A, #11
WHEN "00110010" => WEN <= '0'; -- SUB      A, R?
WHEN "00110110" => WEN <= '0'; -- SUB      A, @R?
WHEN "00111010" => WEN <= '0'; -- SUB      A, MM
WHEN "00111110" => WEN <= '0'; -- SUB      A, #11
WHEN "01000010" => WEN <= '0'; -- SUBC     A, R?
WHEN "01000110" => WEN <= '0'; -- SUBC     A, @R?

```

```

WHEN "01001010" => WEN <= '0'; -- SUBC    A, MM
WHEN "01001110" => WEN <= '0'; -- SUBC    A, #11
WHEN "01010010" => WEN <= '0'; -- AND     A, R?
WHEN "01010110" => WEN <= '0'; -- AND     A, @R?
WHEN "01011010" => WEN <= '0'; -- AND     A, MM
WHEN "01011110" => WEN <= '0'; -- AND     A, #11
WHEN "01100010" => WEN <= '0'; -- OR      A, R?
WHEN "01100110" => WEN <= '0'; -- OR      A, @R?
WHEN "01101010" => WEN <= '0'; -- OR      A, MM
WHEN "01101110" => WEN <= '0'; -- OR      A, #11
WHEN OTHERS      => WEN <= '1';
END CASE;

```

-- AEN 35

CASE STATUS IS

```

WHEN "00010001" => AEN <= '0'; -- ADD     A, R?
WHEN "00010101" => AEN <= '0'; -- ADD     A, @R?
WHEN "00011001" => AEN <= '0'; -- ADD     A, MM
WHEN "00011101" => AEN <= '0'; -- ADD     A, #11
WHEN "00100001" => AEN <= '0'; -- ADDC    A, R?
WHEN "00100101" => AEN <= '0'; -- ADDC    A, @R?
WHEN "00101001" => AEN <= '0'; -- ADDC    A, MM
WHEN "00101101" => AEN <= '0'; -- ADDC    A, #11
WHEN "00110001" => AEN <= '0'; -- SUB     A, R?
WHEN "00110101" => AEN <= '0'; -- SUB     A, @R?
WHEN "00111001" => AEN <= '0'; -- SUB     A, MM
WHEN "00111101" => AEN <= '0'; -- SUB     A, #11
WHEN "01000001" => AEN <= '0'; -- SUBC    A, R?
WHEN "01000101" => AEN <= '0'; -- SUBC    A, @R?
WHEN "01001001" => AEN <= '0'; -- SUBC    A, MM
WHEN "01001101" => AEN <= '0'; -- SUBC    A, #11
WHEN "01010001" => AEN <= '0'; -- AND     A, R?
WHEN "01010101" => AEN <= '0'; -- AND     A, @R?
WHEN "01011001" => AEN <= '0'; -- AND     A, MM
WHEN "01011101" => AEN <= '0'; -- AND     A, #11
WHEN "01100001" => AEN <= '0'; -- OR      A, R?
WHEN "01100101" => AEN <= '0'; -- OR      A, @R?
WHEN "01101001" => AEN <= '0'; -- OR      A, MM
WHEN "01101101" => AEN <= '0'; -- OR      A, #11
WHEN "01110001" => AEN <= '0'; -- MOV     A, R?
WHEN "01110101" => AEN <= '0'; -- MOV     A, @R?
WHEN "01111001" => AEN <= '0'; -- MOV     A, MM
WHEN "01111101" => AEN <= '0'; -- MOV     A, #11
WHEN "10010001" => AEN <= '0'; -- READ    A, MM
WHEN "11000001" => AEN <= '0'; -- IN      A
WHEN "11010001" => AEN <= '0'; -- RR      A
WHEN "11010101" => AEN <= '0'; -- RL      A
WHEN "11011001" => AEN <= '0'; -- RRC     A
WHEN "11011101" => AEN <= '0'; -- RLC     A
WHEN "11100101" => AEN <= '0'; -- CPL     A
WHEN OTHERS      => AEN <= '1';
END CASE;

```

-- S2 16

CASE STATUS IS

```

WHEN "00010001" => S2 <= '0'; -- ADD     A, R?
WHEN "00010101" => S2 <= '0'; -- ADD     A, @R?
WHEN "00011001" => S2 <= '0'; -- ADD     A, MM
WHEN "00011101" => S2 <= '0'; -- ADD     A, #11
WHEN "00110001" => S2 <= '0'; -- SUB     A, R?
WHEN "00110101" => S2 <= '0'; -- SUB     A, @R?
WHEN "00111001" => S2 <= '0'; -- SUB     A, MM

```

```

    WHEN "00111101" => S2 <= '0'; -- SUB      A, #11
    WHEN "01010001" => S2 <= '0'; -- AND      A, R?
    WHEN "01010101" => S2 <= '0'; -- AND      A, @R?
    WHEN "01011001" => S2 <= '0'; -- AND      A, MM
    WHEN "01011101" => S2 <= '0'; -- AND      A, #11
    WHEN "01100001" => S2 <= '0'; -- OR       A, R?
    WHEN "01100101" => S2 <= '0'; -- OR       A, @R?
    WHEN "01101001" => S2 <= '0'; -- OR       A, MM
    WHEN "01101101" => S2 <= '0'; -- OR       A, #11
    WHEN OTHERS      => S2 <= '1';
END CASE;

-- S1 16
CASE STATUS IS
    WHEN "00010001" => S1 <= '0'; -- ADD      A, R?
    WHEN "00010101" => S1 <= '0'; -- ADD      A, @R?
    WHEN "00011001" => S1 <= '0'; -- ADD      A, MM
    WHEN "00011101" => S1 <= '0'; -- ADD      A, #11
    WHEN "00100001" => S1 <= '0'; -- ADDC     A, R?
    WHEN "00100101" => S1 <= '0'; -- ADDC     A, @R?
    WHEN "00101001" => S1 <= '0'; -- ADDC     A, MM
    WHEN "00101101" => S1 <= '0'; -- ADDC     A, #11
    WHEN "00110001" => S1 <= '0'; -- SUB      A, R?
    WHEN "00110101" => S1 <= '0'; -- SUB      A, @R?
    WHEN "00111001" => S1 <= '0'; -- SUB      A, MM
    WHEN "00111101" => S1 <= '0'; -- SUB      A, #11
    WHEN "01000001" => S1 <= '0'; -- SUBC     A, R?
    WHEN "01000101" => S1 <= '0'; -- SUBC     A, @R?
    WHEN "01001001" => S1 <= '0'; -- SUBC     A, MM
    WHEN "01001101" => S1 <= '0'; -- SUBC     A, #11
    WHEN OTHERS      => S1 <= '1';
END CASE;

-- S0 13
CASE STATUS IS
    WHEN "00010001" => S0 <= '0'; -- ADD      A, R?
    WHEN "00010101" => S0 <= '0'; -- ADD      A, @R?
    WHEN "00011001" => S0 <= '0'; -- ADD      A, MM
    WHEN "00011101" => S0 <= '0'; -- ADD      A, #11
    WHEN "00100001" => S0 <= '0'; -- ADDC     A, R?
    WHEN "00100101" => S0 <= '0'; -- ADDC     A, @R?
    WHEN "00101001" => S0 <= '0'; -- ADDC     A, MM
    WHEN "00101101" => S0 <= '0'; -- ADDC     A, #11
    WHEN "01100001" => S0 <= '0'; -- OR       A, R?
    WHEN "01100101" => S0 <= '0'; -- OR       A, @R?
    WHEN "01101001" => S0 <= '0'; -- OR       A, MM
    WHEN "01101101" => S0 <= '0'; -- OR       A, #11
    WHEN "11100101" => S0 <= '0'; -- CPL      A
    WHEN OTHERS      => S0 <= '1';
END CASE;
END PROCESS;

-- 计算指令周期数
PROCESS(I_BUS)
VARIABLE INST: STD_LOGIC_VECTOR(7 DOWNTO 2);
BEGIN
    INST := I_BUS(7 DOWNTO 2);
    CASE INST IS
        WHEN "000100" => CT <= "10"; -- ADD      A, R?
        WHEN "000101" => CT <= "11"; -- ADD      A, @R?
        WHEN "000110" => CT <= "11"; -- ADD      A, MM
        WHEN "000111" => CT <= "10"; -- ADD      A, #11
    
```

```

WHEN "001000" => CT <= "10"; -- ADDC    A, R?
WHEN "001001" => CT <= "11"; -- ADDC    A, @R?
WHEN "001010" => CT <= "11"; -- ADDC    A, MM
WHEN "001011" => CT <= "10"; -- ADDC    A, #11
WHEN "001100" => CT <= "10"; -- SUB     A, R?
WHEN "001101" => CT <= "11"; -- SUB     A, @R?
WHEN "001110" => CT <= "11"; -- SUB     A, MM
WHEN "001111" => CT <= "10"; -- SUB     A, #11
WHEN "010000" => CT <= "10"; -- SUBC    A, R?
WHEN "010001" => CT <= "11"; -- SUBC    A, @R?
WHEN "010010" => CT <= "11"; -- SUBC    A, MM
WHEN "010011" => CT <= "10"; -- SUBC    A, #11
WHEN "010100" => CT <= "10"; -- AND     A, R?
WHEN "010101" => CT <= "11"; -- AND     A, @R?
WHEN "010110" => CT <= "11"; -- AND     A, MM
WHEN "010111" => CT <= "10"; -- AND     A, #11
WHEN "011000" => CT <= "10"; -- OR      A, R?
WHEN "011001" => CT <= "11"; -- OR      A, @R?
WHEN "011010" => CT <= "11"; -- OR      A, MM
WHEN "011011" => CT <= "10"; -- OR      A, #11
WHEN "011100" => CT <= "01"; -- MOV     A, R?
WHEN "011101" => CT <= "10"; -- MOV     A, @R?
WHEN "011110" => CT <= "10"; -- MOV     A, MM
WHEN "011111" => CT <= "01"; -- MOV     A, #11
WHEN "100000" => CT <= "01"; -- MOV     R?, A
WHEN "100001" => CT <= "10"; -- MOV     @R?, A
WHEN "100010" => CT <= "10"; -- MOV     MM, A
WHEN "100011" => CT <= "01"; -- MOV     R?, #11
WHEN "100100" => CT <= "10"; -- READ    A, MM
WHEN "100101" => CT <= "10"; -- WRITE   MM, A
WHEN "101000" => CT <= "01"; -- JC      MM
WHEN "101001" => CT <= "01"; -- JZ      MM
WHEN "101011" => CT <= "01"; -- JMP     MM
WHEN "101110" => CT <= "10"; -- _INT_
WHEN "101111" => CT <= "11"; -- CALL    MM
WHEN "110000" => CT <= "01"; -- IN
WHEN "110001" => CT <= "01"; -- OUT
WHEN "110011" => CT <= "01"; -- RET
WHEN "110100" => CT <= "01"; -- RR      A
WHEN "110101" => CT <= "01"; -- RL      A
WHEN "110110" => CT <= "01"; -- RRC     A
WHEN "110111" => CT <= "01"; -- RLC     A
WHEN "111001" => CT <= "01"; -- CPL     A
WHEN "111011" => CT <= "01"; -- RETI
WHEN OTHERS => CT <= "00";

END CASE;
END PROCESS;

END behv;

```

在总体实验中，rst 为复位信号，接在开关组 K4 的第 7 个开关上。i_req 为中断申请输入信号，由 COP2000 实验仪主板上的“INT”按键输出到 FPGA 扩展板上。clk 为时钟脉冲，由 COP2000 实验仪主机上的“CLOCK”按键提供。keyin 输入端口，接在扩展板的 K1、K0 开关组上，用于输入外部信号。portout 为输出端口，接到 D1、D0 四个八段数码管上，用于显示输出信号。mem_d 为十六位存储器的数据线，mem_a 为十六位存储器的地址线，mem_rd 为存储器读控制信号，mem_wr 为存储器写控制信号，mem_cs 为存储器片选信号，mem_bh 为存储器高 8 位数据选择信号，mem_bl 为存储器低 8 位数据选择信号。

实验步骤:

1. 打开 Xilinx 的 EDA 开发环境, 选择 “Open Project ...” 打开 “\COP2000\XCV200\” 下的 COP2000 项目 (Xilinx 的 EDA 开发环境的使用可参见第十一章)。
2. 充分理解 COP2000.VHD。了解十六位模型机的实现原理。
3. 对 COP2000 项目进行综合/编译, 生成 COP2000.BIT 文件。
4. 打开伟福的 COP2000 开发环境, 暂时不打开 FPGA 扩展板的界面, 在开发环境的主界面中, 打开 “\COP2000\XCV200\COP2000\” 目录下 “TEST.ASM” 程序, 编译 “TEST.ASM”。在主界面中的 EM 窗口中应有 TEST 程序的机器码。
5. 打开 FPGA 扩展板的界面。切换到 “存储器” 页面, 可以看到 EM 的机器码, 也就 TEST 程序已经被自动复制到存储器窗口中, 再将窗口切换到 “结构图” 页面。
6. 按 “打开模式” 键, 打开 “\COP2000\XCV200\COP2000\” 目录下的 COP2000.MOD 文件。界面上 “时钟选择” 应设成 “单脉冲”, 这样 CLK 信号就由 COP2000 实验仪主板上的 “CLOCK” 提供。
7. 按 “通信设置” 键, 将实验仪连接到计算机串行口上。
8. 按 “FPGA 编程” 键, 将 “\COP2000\XCV200\COP2000\” 目录下的 COP2000.BIT 文件下载到 XCV200 芯片上。
9. 切换到 “存储器” 页面, 按 “下载存储器” 键, 将 TEST 程序的机器码下载到 FPGA 扩展板上的存储器中。
10. 拨动 K4 的第 7 位到 “1” 的位置, 输出 “复位” 信号, 再将 K4 的第 7 位拨到 ‘0’ 的位置, 使得十六位模型机正常工作。
11. 按动 COP2000 实验仪上的 “CLOCK” 按键, 产生多个时钟信号, 观察存储器的读写是否符合设计要求, 数据线、地址线、控制线是否正确。
12. 拨动 K4 的第 7 位, 使模型机置于复位状态。在 “结构图” 页面上将 “时钟选择” 设成 “连续脉冲” 方式。这样模型机的时钟为高速的连续脉冲, 模型机运行于全速状态。
13. 在模型机全速运行时, 按动 COP2000 实验仪上的 “INT” 按键, 产生中断请示信号, 让模型机中断, 在中断处理程序中, 我们将累加器 A 带进位右移一位。中断返回后输出累加器中的内容。重复按 “INT” 按键, 产生多个中断请示信号, 让模型机多次中断, 累加器多次移位输出, 观察模型机输出是否满足设计要求。
14. 改变主界面上程序, 以完成不同的功能, 将程序编译, 将编译的机器码复制到 “扩展板” 界面的 “存储器” 页面内。将模型机置于复位状态, 并用 “下载存储器” 功能下载到扩展板的存储器中, 退出模型机的复位状态, 使其正常工作, 检查程序运行是否满足程序要求。

本综合实验可参考 COP2000 实验仪使用说明的第三章 “COP2000 模型机”、第四章 “模型机综合实验”、第五章 “组合逻辑控制器” 等章节。